Supporting Data Decomposition in Parallel Programming

Anne Meade

This thesis is submitted to the University of Limerick in fulfilment of the requirements for the degree of Doctor of Philosophy

Department of Computer Science and Information Systems

Supervisors: Mr. J.J. Collins & Dr. Jim Buckley

Submitted to the University of Limerick, May, 2014
To my parents, Leo and Marie,

for all their support through the years.
Abstract

Parallelising serial software systems presents many challenges. In particular, the task of decomposing large, data-intensive applications for execution on distributed architectures is described in the literature as error-prone and time-consuming. The Message Passing Interface (MPI) specification is the de facto industry standard to program for such architectures, but requires low level knowledge of data distribution details as programmers must explicitly invoke inter-process communication routines. This research reports the findings from empirical studies conducted in industry, to explore and characterise the challenges associated with performing data decomposition. Findings from these studies culminated in a list of derived requirements for tool support, encompassing automation of grid indexing, generation of data structures and communication calls, and provision of assistance when changing from an implemented decomposition strategy. Additional requirements include the need for a tool to be MPI focused, initially target structured grids and have a low impact on the application code. These requirements were subsequently buttressed to address gaps in the state-of-the-art and provided motivation for the development of a tool named MPIGen.

MPIGen provides an abstraction for MPI, encapsulating the low level details involved in decomposing data and exchanging messages between processors. Users can express the parallel intent of their application through input parameters and then generate code containing wrapper functions that encompass the MPI functionality. The wrapper functions can then be invoked within the serial code resulting in a semi-automated parallelised solution. The programmer is relieved of the burden of deciphering memory locations when exchanging data between processors. The tool was evaluated in two studies involving both students and High Performance Computing (HPC) practitioners as subjects. The findings concluded that MPIGen provides an efficient abstraction for performing data decomposition and that it satisfies the list of empirically derived requirements.

Parallel programming is a difficult skill that software developers need to learn, yet the low level nature of specifications such as MPI is an adverse factor to its adoption. MPIGen makes it easier to adopt this skill-set as it offers effective support to parallel programmers when undertaking decomposition and communication.
Statement of Original Authorship

I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the title page, and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

Anne Meade
Acknowledgements

First and foremost, I would like to thank my advisors – J.J., who held my hand through the problem-child phase and spent many an hour strategising, guiding and encouraging; Jim, who very generously jumped on board a rocky ship, adding stability through the unlimited provision of advice and constructive feedback.

I’m also grateful to all the people who participated in the studies conducted, in particular those who willingly took time out of their working schedules to facilitate this research. Thanks in particular to my industrial collaborators; Michael Johnston, Prashobh Balasundaram, and the late Brendan Boulter in IBM and Rob Lamb, Mandy Crossley, and Ross Bryant in JBA.

Thanks also to my examiners, Dr. Norah Power and Dr. Sebastian Nanz for their time and feedback.

I’m grateful to all my colleagues and friends in Lero and Limerick, particularly to Deva and Mick who endured many a ranting session. Thanks as well to Jo for the coffee-break distractions, to Lili and Clare for being my daytime exercise buddies, to Sarah for her uplifting pep-talks, and to Howell, Anila and Pádraig for the whiteboard advisory sessions. Thanks also to Joe for his diligence in ensuring my chocolate-intake was plentiful.

Special thanks are due to my family, particularly Mam (Marie) and Dad (Leo), who taught me at a young age that accomplishments are founded on the will to achieve. I’m also extremely grateful for the unflagging support and encouragement from Leona, Eoin, Edel, Niamh, and Alan. You all have contributed more than you know.

Heartfelt thanks to Klaas, for keeping me motivated, reassuring me it’s worth it and being such a solid support system.

Finally, I gratefully acknowledge the financial support of the Lero Graduate School of Software Engineering.
# Contents

Abstract i

Statement of Original Authorship iii

Acknowledgements v

List of Figures xiii

List of Tables xvii

List of Abbreviations xix

1 Introduction 1
   1.1 Research Background 1
   1.2 Research Objectives and Questions 3
   1.3 Summary of Contributions 6
   1.4 Structure of Thesis 8

2 Data Decomposition within the Parallelisation Process 11
   2.1 Introduction 11
   2.2 The Drive to Parallelise Software 12
   2.3 Parallelisation Concepts 14
      2.3.1 Parallel Architectures 14
4.2.1 Research Design ................................................. 51
4.2.2 Findings ......................................................... 52
4.3 Stage II: Participant-Observer Study .............................. 57
  4.3.1 Research Design .............................................. 57
  4.3.2 The Porting Task ............................................. 58
  4.3.3 Findings ......................................................... 60
4.4 Stage III: Focus Group Study ..................................... 63
  4.4.1 Research Design .............................................. 64
  4.4.2 Findings ......................................................... 65
  4.4.3 Limitations ..................................................... 69
4.5 Stage IV: Survey Study ............................................ 70
  4.5.1 Research Design .............................................. 70
  4.5.2 Findings ......................................................... 71
  4.5.3 Threats to Validity ........................................... 79
4.6 Derived Requirements ............................................. 79
4.7 Conclusion ......................................................... 84

5 Exploration of Tool Support Available for Data Decomposition 87
  5.1 Introduction ...................................................... 87
  5.2 Pattern-Based Approaches ..................................... 90
    5.2.1 PASM ......................................................... 91
    5.2.2 MAP3S ....................................................... 93
  5.3 DSL Approaches ................................................ 96
    5.3.1 Promoter ..................................................... 96
    5.3.2 Hi-PaL ......................................................... 99
  5.4 Generic Template Approaches .................................. 101
    5.4.1 Janus ......................................................... 101
    5.4.2 C++ components as part of ISTL module ............... 103
    5.4.3 TACO .......................................................... 106
  5.5 PSE Approaches ................................................ 108
    5.5.1 POOMA ....................................................... 108
5.5.2 PETSc ........................................... 110
5.6 Other Approach ........................................ 113
  5.6.1 DDTool ........................................ 113
5.7 Additional Requirements .............................. 115
5.8 Comparative Analysis ................................. 116
5.9 Conclusion ............................................ 120

6 Design and Implementation of MPIGen ............. 121
  6.1 Introduction ........................................ 121
  6.2 MPIGen Overview .................................... 122
  6.3 Addressing the Derived Requirements ............ 123
  6.4 Key Design Considerations ........................ 126
  6.5 MPIGen Output ..................................... 129
  6.6 Test Cases ......................................... 132
    6.6.1 Game of Life .................................. 133
    6.6.2 Applying MPIGen .............................. 134
    6.6.3 Performance of Game of Life ................. 137
  6.7 Functional Limitations ............................. 137
  6.8 Conclusion ......................................... 138

7 Evaluation of MPIGen .................................. 139
  7.1 Introduction ...................................... 139
  7.2 Evaluation Overview ................................ 140
  7.3 Study Design – Phase I .............................. 143
    7.3.1 Participants .................................... 146
  7.4 Evaluation of the Requirements – Experts ....... 148
    7.4.1 R1: MPI Focused .............................. 148
    7.4.2 R2: Automated Grid Indexing ................. 149
    7.4.3 R3: Data Structure & Communication Generation .... 150
    7.4.4 R4: Ease of Strategy Conversion .............. 151
    7.4.5 R5: Low Application Code Impact ............ 152
7.4.6 R8: Minimal Learning Curve ........................................ 153
7.5 Evaluation of the Usability Components – Experts ................. 155
  7.5.1 Satisfaction .................................................. 155
  7.5.2 Errors ......................................................... 155
  7.5.3 Efficiency ...................................................... 156
7.6 Other Insights from Phase I ............................................. 157
7.7 Study Design – Phase II ............................................... 159
  7.7.1 Participants ................................................... 162
  7.7.2 Parallelisation Exercises ....................................... 164
  7.7.3 Ethical Considerations ......................................... 166
7.8 Evaluation of the Usability Components – Students ................. 166
  7.8.1 Learnability .................................................. 166
  7.8.2 Satisfaction .................................................. 168
  7.8.3 Errors ......................................................... 169
  7.8.4 Efficiency ...................................................... 174
  7.8.5 Analysis According to Student Cohorts ......................... 180
  7.8.6 Summary of Statistical Tests Performed in Phase II .......... 183
7.9 Usability Findings Aggregated ......................................... 184
7.10 Discussion .......................................................... 185
7.11 Threats to Validity .................................................... 186
  7.11.1 External Validity ............................................. 186
  7.11.2 Learning effect .............................................. 188
  7.11.3 Reliability .................................................... 188
7.12 Conclusion .......................................................... 189

8 Conclusion .................................................................. 191
  8.1 Introduction .......................................................... 191
  8.2 Review of Research Objectives ...................................... 192
    8.2.1 Research Question 1 ......................................... 192
    8.2.2 Research Question 2 ......................................... 193
    8.2.3 Research Question 3 ......................................... 194
# List of Figures

1.1 Research Questions and Associated Contributions .......................... 7
1.2 Thesis Structure ......................................................... 10
2.1 Intel Processor Range .................................................. 13
2.2 Data Decomposition .................................................... 18
2.3 Block Based Distributions .............................................. 21
2.4 Matrix Decomposition .................................................. 22
2.5 9-point Stencil ........................................................... 23
2.6 Ghost Cell Example ..................................................... 24
3.1 Overview of Research Questions ........................................ 32
3.2 Research Process ........................................................ 47
4.1 Stages of Empirical Derivation ........................................ 50
4.2 Label Count assigned during Coding Process ....................... 53
4.3 Sketches to decipher Decomposition and Communication ........ 61
4.4 Breakdown of Project Type ............................................. 72
4.5 Breakdown of Grid Type ................................................ 73
4.6 Breakdown of Decomposition Type ................................... 74
4.7 Average Difficulty Rating of Parallelisation Phases ................ 75
4.8 Considerations when Performing Data Decomposition ............ 77
5.1 PASM Example 1 .......................................................... 91
5.2 PASM Example 2 .......................................................... 92
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Research Stages and Types of Questions</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Relevant Research Questions, Stages and Types</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Participants’ Details</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Identified Challenges</td>
<td>54</td>
</tr>
<tr>
<td>4.3</td>
<td>Mentors’ Details</td>
<td>58</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary of Findings in Stage II</td>
<td>62</td>
</tr>
<tr>
<td>4.5</td>
<td>Study Participants for Stage III</td>
<td>64</td>
</tr>
<tr>
<td>4.6</td>
<td>Summary of Tooling Proposals</td>
<td>69</td>
</tr>
<tr>
<td>4.7</td>
<td>Participant Experience for Stage IV</td>
<td>71</td>
</tr>
<tr>
<td>4.8</td>
<td>Summary of Findings in Stage IV</td>
<td>78</td>
</tr>
<tr>
<td>4.9</td>
<td>Summary of Derived Requirements</td>
<td>84</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of Tools</td>
<td>89</td>
</tr>
<tr>
<td>5.2</td>
<td>PASM - Requirements Analysis</td>
<td>93</td>
</tr>
<tr>
<td>5.3</td>
<td>MAP₃S - Requirements Analysis</td>
<td>95</td>
</tr>
<tr>
<td>5.4</td>
<td>Promoter - Requirements Analysis</td>
<td>98</td>
</tr>
<tr>
<td>5.5</td>
<td>Hi-PaL - Requirements Analysis</td>
<td>100</td>
</tr>
<tr>
<td>5.6</td>
<td>Janus - Requirements Analysis</td>
<td>103</td>
</tr>
<tr>
<td>5.7</td>
<td>ISTL library - Requirements Analysis</td>
<td>105</td>
</tr>
<tr>
<td>5.8</td>
<td>TACO - Requirements Analysis</td>
<td>108</td>
</tr>
<tr>
<td>5.9</td>
<td>POOMA library - Requirements Analysis</td>
<td>110</td>
</tr>
<tr>
<td>5.10</td>
<td>PETSc library - Requirements Analysis</td>
<td>112</td>
</tr>
</tbody>
</table>
List of Abbreviations

API  Application Programming Interface
BG/P  IBM’s Blue Gene® /P Supercomputer
CFD  Computational Fluid Dynamics
CPU  Central Processing Unit
CSIS  Computer Science and Information Systems
CUDA  Compute Unified Device Architecture
DSM  Distributed Shared Memory
FraSPA  Framework for Synthesising Parallel Applications
GPGPU  General Purpose computing on Graphics Processing Units
GPU  Graphics Processing Unit
GUI  Graphical User Interface
Hi—PaL  High Level Parallelisation Language
HPC  High Performance Computing
HPCS  High Productivity Computing Systems
ICHEC  Irish Centre for High End Computing
JBA  Jeremy Benn Associates
JSP  Java Server Page
LoC  Lines of Code
MAP₃S  MPI/C Advanced Pattern-based Parallel Programming System
MMP  Massively Parallel Processing
MPI  Message Passing Interface
MPIGen  MPI Generator
OpenCL  Open Computing Language
OPL  Our Pattern Language
PASM  Parallel Architectural Skeleton Model
PETSc  Portable,Extensible, Toolkit for Scientific Computation
PGAS  Partitioned Global Address Space
PLPP  Pattern Language for Parallel Programming
POOMA  Parallel Object-Oriented Methods and Applications
PSE  Problem Solving Environment
RMA  Remote Method Access
RMI  Remote Method Invocation
SCEC  Southern California Earthquake Centre
SIMD  Single Instruction Multiple Data
SLR  Systematic Literature Review
SMP  Symmetric Multiprocessing
SPMD  Single Program Multiple Data
STL  Standard Template Library
TAAAPoS  Towards Automated Architecture Aware Parallelisation of Systems
1 Introduction

1.1 Research Background

Lehman’s first law of software evolution states a program must go through continual change or become less useful (Lehman 1980). An important driver in software evolution is the exploitation of new technological advances (Hinchey and Coyle 2010). As conventional scaling of clock speeds reaches a plateau, a technological advancement that resolves the resultant performance bottleneck is the advent of multicore processors. This has sparked the move from single processor architectures to high performance computing (HPC) environments. To utilise these multicore architectures and join in on what has been coined the ‘concurrency revolution’ (Sutter and Larus 2005), programs must be successfully engineered in order to support concurrent execution. Programmers must adopt a different style of programming and focus their efforts on creating software that can exploit parallelism to avail of the performance gains on offer (Pankratius 2010; Sangani 2007).

The parallelisation process has been documented in the literature as error prone and difficult (Massingill et al. 2007; Vandierendonck and Mens 2011). Developing parallel software within dedicated timeframes without impacting the quality of the code, is a much more complex task than developing serial software.
1.1 Research Background

under similar constraints (Dig 2011; Patterson 2010). O’Hanlon (2006) further emphasises this:

*Parallel programming has proven to be a really hard concept … as hard as any that computer science has faced.*

The focus of this research is on the parallelisation of data-intensive applications for execution on homogeneous architectures. The end goal of porting to such architectures is increased application performance and so the *distributed* memory model is pertinent to this research as it facilitates scalability and enables high performance gains. As the *shared* memory model has limited scalability, it is not considered. Latency is an important factor in computationally-intense applications as messages need to be communicated during execution and so heterogeneous architectures such as CPU-GPU systems are not applicable.

When designing a parallelisation strategy, decomposition and the subsequent distribution of data or tasks to processes running on available cores is a critical consideration (Massingill et al. 2007). As decomposition by data is more commonly implemented in large scale applications than decomposition by task (Okur and Dig 2012; Grama et al. 2003), the effort involved in designing and implementing data decomposition is a core focus of this thesis.

When designing a data decomposition for a grid-based structure, the developer must give prior consideration to the performance impact of the potential data exchanges between the allocated processes. Messages typically need to be communicated at various intervals between neighbouring processes in order to exchange data at boundaries. Given that this communication can be quite complex over a large number of processes, this is a difficult task (Kennedy et al. 2003, p. 43). To implement this, the parallel programmer must manually invoke message passing calls correctly to express the parallelisation within their sequential application, while ensuring synchronisation of the distributed data. A further challenge involves the determination of an optimal decomposition strategy for a specific application, as implementing an alternative decomposition involves invasive rewriting of the parallelised code.
Chapter 1. Introduction

The Message Passing Interface (MPI) is the message-passing protocol of choice for many parallel programmers (Basili et al. 2008). MPI is a set of library routines and has become the de facto standard programming model for distributed memory architectures (Gropp 2011; Mattson et al. 2004, p. 13). Parallelising MPI applications can be laborious as programmers must be cognisant of the low-level details of the data structures and explicitly handle the complexities of message passing (Keutzer et al. 2010). This involves complex mapping and management of indices as the programmer must be aware of index locations in the global problem domain as well as those visible to each process which has been allocated a data partition.

Appropriate tool support is necessary to address these challenges and assist the efficient data decomposition of programs. Dongarra et al. (2003) claim:

*The goal of parallel computing software systems should be to make parallel programming easier and the resulting applications more portable while achieving the highest possible performance.*

The research performed in this thesis suggests that this has not been achieved. The onus is on the software development community to create tool support to allow parallel programmers to work more effectively.

1.2 Research Objectives and Questions

Feng and Balaji (2009) claim most programmers are ‘ill-equipped to produce proper parallel programs’ as they lack sufficient tool-support. Yet, Mattson and Wrinn (2008) claim hundreds of parallel programming technologies have been created (from environments to languages to API) and so this begs the question — Why is there a low adoption rate of these technologies? Nanz et al. (2013a) argue that compelling results reporting usability and performance of existing tooling approaches is lacking and so the development community has difficulty deciding which tools address their needs. Basili et al. (2008) state MPI is still the dominant model as many of the new technologies that claimed productivity (predominantly new languages), are no longer available and so programmers have grown
1.2 Research Objectives & Questions

sceptical of new offerings. However, failure to deviate from using MPI results in higher programming burden as developers are exposed to low level details of data structures and their memory locations. An abstracted view of the operations required to perform tasks such as data decomposition, is necessary to exploit the technological advances offered by multicore architectures without losing gains in the productivity of programmers (Goodale et al. 2003).

This thesis aims to characterise the problems faced by practitioners, when performing data decomposition and communication when porting applications from serial to multicore environments. One goal is to empirically derive a solution that abstracts the low-level details involved, reducing the adoption barriers faced by previous tooling options by meeting practitioners’ requirements. The high-level objective of this research is as follows:

To investigate the specification and implementation of data decomposition as an issue in parallel programming and to address problems programmers face in these tasks.

This high-level objective can be broken down into two concrete objectives:

1. To evaluate data decomposition and communication as a key challenge in the parallelisation process.

2. To derive an empirically based approach to support effective data decomposition and communication.

In order to achieve the aforementioned objectives, the following research questions were derived in the context of parallelising applications in a HPC environment:

- **RQ1: Is data decomposition and communication a real and prevalent challenge that practitioners face?** While a previous exploratory
literature review\(^1\) outlined parallelisation-related challenges in the state-of-the-art, this question aims to gain a comprehensive understanding of the prevalence of the challenge of data decomposition and the subsequent communication operations.

- **RQ2: What do practitioners use to support their efforts when performing data decomposition and communication?** The methods/techniques/tools used by practitioners to support this task are not documented and require investigation.

- **RQ3: What tooling requirements are necessary to support practitioners when performing data decomposition and communication?** The requirements for support need to be investigated before undertaking the design of a tool to assist this task.

- **RQ4: What tooling support exists to assist the task of data decomposition and communication?** The available tooling support needs to be identified and analysed to investigate if the current offerings meet the requirements derived in RQ3.

- **RQ5: Does a tool that fulfils requirements in RQ3, effectively support the task of data decomposition and communication?** A tool derived from the requirements in RQ3 requires a thorough evaluation to determine whether fulfilment of these requirements does satisfy practitioners’ needs.

These research questions (including additional sub-questions) are described in detail in Chapter 3. This research was conducted as part of the TAAAPoS (Towards Automated Architecture Aware Parallelisation of Systems) project in collaboration

with two industrial partners, namely JBA Consulting and IBM\textsuperscript{2}. Three doctoral researchers were involved in the project, each working towards achieving different research goals.\textsuperscript{3} A number of the empirical studies described in this thesis were conducted as a group, yet each researcher followed his and her own research avenue exclusively.

### 1.3 Summary of Contributions

The contributions of this research can be summarised as follows:

- **C1**: A list of empirically derived requirements to facilitate data decomposition and communication when parallelising applications.

- **C2**: A characterisation of existing tool support.

- **C3**: A prototype tool shown to support data decomposition and communication by meeting practitioners’ requirements.

Figure 1.1 depicts these contributions and the associated research questions. As the researcher is unaware of any empirical studies specifically focusing on the tasks of performing data decomposition and communication in a HPC context, the findings from four empirical studies conducted to address RQ1 and RQ2, feed into the first contribution. These studies involved exploratory interviews, a participant-observer study, a focus group interview and a globally distributed survey. The findings shed light on practices and show that practitioners are unaware of tooling to support the task of decomposing data and devising the applicable communication calls when parallelising applications. Contribution C1 encompasses requirements empirically derived to address RQ3. These include the

\textsuperscript{2}A trademark of International Business Machines Corporation, registered in many jurisdictions worldwide.

\textsuperscript{3}The interests of the peer-researchers involved optimising/tuning already parallelised applications and using program transformations to parallelise code automatically.
need for a tool to have an MPI focus, generate the data structures and communication calls, automate grid indexing, provide support for switching between decomposition strategies and have a low code impact. These requirements provide input to the development of a prototype tool to assist programmers.

The knowledge gained from the review of the state-of-the-art conducted to address RQ4, in which approaches are characterised and tools outlined, embodies the second research contribution (C2). The approaches included patterns, generic templates, domain specific languages and problem solving environments. Analysis of the tooling options available showed that no tool meets all the derived requirements from C1.

**Figure 1.1:** Research Questions and Associated Contributions
The design and implementation of a prototype tool named MPIGen, enables a solution to the limitations of existing tooling support and encompasses the third research contribution (C3). MPIGen was evaluated to address RQ5, using the derived requirements as a baseline and can be considered effective as it was found to meet all these requirements. The prototype itself can be considered a design-science contribution as it demonstrates the feasibility of using the derived requirements to develop an effective artifact to support data decomposition (Hevner et al. 2004).

## 1.4 Structure of Thesis

The remainder of the thesis is structured as follows:

- **Chapter 2** presents relevant background information and introduces the context of the research. The parallelisation process is described and the role of decomposition is outlined. An overview of data decomposition is presented and the issues associated are indicated. The communication operations involved in message-passing applications are also explored.

- **Chapter 3** outlines the research process in detail. The research questions presented in this thesis are discussed and the design of each study conducted is justified. The details involving the approach taken to answer each research question are presented.

- **Chapter 4** describes an empirical investigation of the problem of data decomposition. The properties of the problem are characterised through four empirical studies. The findings from each of these are outlined and a list of requirements for potential tool support is empirically derived from these data.

---

• **Chapter 5** reports on the literature review of the existent tool support where different approaches are characterised. A comparative analysis of available tools is presented based on the requirements identified in Chapter 4.

• **Chapter 6** describes the design and implementation of a prototype tool named MPIGen based on the requirements derived to address RQ3. The inputs, outputs and functionality of the tool are outlined and its use in parallelising grid-based applications is detailed.

• **Chapter 7** outlines a two-phased study used to assess the effectiveness of the MPIGen prototype tool support. The experimental procedure used to gather quantitative data and the statistical analysis applied to these data are presented. A qualitative evaluation of the tool using expert feedback is also discussed.

• **Chapter 8** concludes this thesis by discussing how the research conducted fulfils the stated research objectives. The contributions of the research are also outlined and a view to future work is presented.

Figure 1.2 provides an illustration of the chapters in this thesis, detailing the contributions associated.
1.4 Structure of Thesis

Figure 1.2: Thesis Structure
2

Data Decomposition within the Parallelisation Process

2.1 Introduction

Parallelising serial software systems in order to run in high performance computing (HPC) environments presents many challenges to parallel programmers. A key issue facing programmers is the task of decomposing an application into subcomponents which can execute in parallel (Kennedy et al. 2003, p. 43). This chapter introduces concepts associated with parallelising applications and outlines the main issues concerning decomposition and communication in the parallelisation process.

In order to comprehend the low-level issues of decomposition, an introductory knowledge of parallel machine architecture is necessary. To this end distributed memory systems are described in this chapter. Data decomposition (the core focus of this research), involves the distribution of data to different processors in order to run in parallel. It is often essential to share data at the boundaries of each processor involved in the parallelisation and so the programmer is charged with the task of exchanging this data accurately and efficiently. To implement a decomposition strategy, the parallel programmer must choose a programming model that suits the application; that is the message-passing model in this research. The Message
2.2 The Drive to Parallelise Software

Passing Interface (MPI) is the de facto industry standard implementation for the message-passing model and the challenges associated are outlined in this chapter. Prior to these discussions, the importance of the field of HPC needs to be contextualised and so this chapter firstly presents a brief history of parallelisation.

The chapter is structured as follows: Section 2.2 describes the software crisis motivating the drive to parallelise software. An overview is provided of parallelisation concepts in Section 2.3 and a brief overview of the parallelisation process is introduced in Section 2.4. Decomposition is discussed in Section 2.5 and communication with a focus on MPI is presented in Section 2.6. A conclusion to this chapter is provided in Section 2.7.

2.2 The Drive to Parallelise Software

The evolution of software has been, until now, associated with the expectation of faster rates of execution. In 1965, Moore (1965) predicted that the number of transistors on a chip would double roughly every 18 months into the near future. This prediction, now referred to as Moore’s law, has borne fruit as Intel chips reached 2GHz in 2001 and consequently, software has experienced exponential hikes in performance. In keeping with Moore’s law, clock speeds should have continued to increase and have surpassed 15GHz by 2010. However, this has not been the case (Asanovic et al. 2009). In Figure 2.1, the blue dots represent the clock frequency for Intel x86 processors from 1970 to 2010. The frequency-increases came to a halt circa 2005, as it became unfeasible to dissipate the heat generated from the amount of transistors that can fit on a single chip (Catanzaro and Keutzer 2010). Clock speed increases can no longer support software performance gains, as their technical limit has been reached (Goth 2009). Rather than increasing the clock speeds, CPU manufacturers have started to increase the number of processor cores; this is represented by the red dots in Figure 2.1. In order to sustain ever-increasing computing performance, developers are faced with the task of evolving systems that traditionally ran on single power inefficient processors to
alternatively utilise multiple processors (Asanovic et al. 2009). Microsoft software architect and Chair of the C++ standards committee Herb Sutter claimed ‘the free lunch was over’ for programmers riding the wave of exponential growth in processor performance predicted by Moore (Sutter 2005). Sutter emphasised the need to change development practice and incorporate parallelisation.

However, the task of parallelisation is not an easy feat as Massingill et al. (2007) outline:

*Creating parallel software is difficult, time-consuming and error-prone. It is especially challenging for legacy applications.*

The concept of parallel programming is not novel (Sutter 2005) and has a strong track record in both the academic and commercial sector. However, the increasing availability of parallel hardware through affordable and continuously evolving de facto standard multicore architectures, generates an increasing requirement in the field of parallelisation, that is support for the evolution of the code from a serial to a parallel paradigm. This is particularly true of computationally intense applications which continuously demand faster execution rates driving developers to utilise multicore environments to meet these performance needs (Sutter 2005). This particular type of evolution is challenging, and is often referred to as a ‘new software engineering crisis’ (Vandierendonck and Mens 2011). Solutions are
2.3 Parallelisation Concepts

required to address the challenge of switching to multicore to avail of continuous performance gains without losing the gains in programmer productivity.

2.3 Parallelisation Concepts

2.3.1 Parallel Architectures

A parallel architecture can be defined as follows (Duncan 1990):

An explicit high level framework for the development of parallel programming solutions by providing multiple processors, whether simple or complex, that cooperate to solve problems through concurrent execution.

Parallel architectures can be structured using shared, distributed or a combination of both memories. Within a shared memory mode, all processors share a single address space and are required to read and write shared variables in order to communicate with each other (Mattson et al. 2004, p. 10). An example of this is Symmetric Multiprocessor (SMP) machines where each CPU (Central Processing Unit) shares a connection to a single address space and has equal access and access time (Lin and Snyder 2009). SMP systems however, do not scale well as bandwidth of the data buses is limited.

In a distributed memory mode, each CPU has its own address space and communicates with other processors by sending and receiving messages. There is no concept of coherency (sharing the same cached data) as each processor has its own local memory. To communicate between processors, the programmer must explicitly define the details regarding communication. An example of a distributed system is a Massively Parallel Processing (MPP) system. Within MPP systems, nodes\(^1\) are physically housed in the same chassis. These systems typically use highly optimised interconnect networks such as IBM’s Blue Gene®/Q supercomputer, which houses a five-dimensional torus interconnect network.

\(^1\)In HPC terms, a node is a physical collection of CPUs, memory and interfaces to other nodes and devices.
Alternative parallel architectures include hybrid systems (combination of both shared and distributed memory) and heterogeneous systems which utilise the capability of CPUs (multicore) whilst also harnessing the capabilities of GPUs (many-core).

### 2.3.2 Parallel Programming Models

A parallel programming model acts as an abstraction above the hardware and memory architecture (Mattson et al. 2004, p. 12). It is not specific to a particular type of machine or memory architecture. A message passing memory model exists where tasks use their own local memory during computation and exchange data through message passing (Duncan 1990). This memory model is a natural fit for distributed memory systems. The communication typically involves cooperative operations where a send operation must have a matching receive operation. This memory model has the potential to scale well, with some applications in MPP environments able to scale up to thousands of processors. When programming in a distributed memory environment, the entire dataset used by the application is physically distributed among the processors. The implications of this is that parallelisation must be introduced to all functionality utilising this data. This adds complexity and burdens the parallel programming as the parallelisation cannot be implemented and tested incrementally in the application, but must be implemented for all the required functionality and the parallelised application tested as a whole. Message Passing Interface (MPI-2 2014) has evolved to be the de facto industry standard implementation for message passing memory models (Diaz et al. 2012; Kasim et al. 2008). As this research targets distributed message passing systems, Section 2.6 will provide more insight into the MPI specification.

Other models include shared-memory (OpenMP v.4.0 2013; POSIX.1 2008), heterogeneous-memory (CUDA v.5.5 2013; OpenCL v.2.0 2013), or Partitioned Global Address Space (PGAS) (Dongarra et al. 2008). Shared-memory models are not applicable to this research due to their scalability limitations; heterogeneous-memory models are not applicable as GPUs are not suitable for applications
2.4 Parallelisation Process

where latency is more important than throughput (Owens et al. 2009) and PGAS models are not applicable as their use in legacy applications involves invasively reengineering the entire application.

2.4 Parallelisation Process

Kennedy et al. (2003, ch. 3) describe four tasks involved in converting a serial application to parallel. These are as follows:

- **Identify parallelism:** the code must firstly be profiled to ascertain which parts should run in parallel. Having located performance bottlenecks, the parallel programmer may need to analyse the code for further details such as data dependencies. Data dependencies add to the complexity of the parallelisation process and if not managed effectively, the resultant code may produce inaccurate results (Geer 2007; Hwu et al. 2008).

- **Perform decomposition:** a strategy needs to be devised for decomposing the program. Decomposition can be divided into two core areas — data and task. Data decomposition in the context of HPC occurs when the same operation is applied to different data in a parallel program. Task decomposition occurs when the program is decomposed according to the tasks the program must carry out, i.e. multiple independent code segments are run in parallel on the same data (McCool 2008). As data decomposition is more commonly used when parallelising applications (Grama et al. 2003; Kennedy et al. 2003) and is the focus of this research, this strategy will be further explored in Section 2.5.

- **Choose the programming model:** this involves determining which memory model is appropriate for the application constrained by the available hardware. The chosen model will determine the necessary communication. Section 2.3.2 discussed the options available to a parallel programmer.
Chapter 2. Data Decomposition

- **Implement parallel program**: the programmer must determine the best implementation that will work with the chosen memory model. In the case of a message-passing memory model, the programmer must write the message-passing calls. Details regarding this task are provided in Section 2.6.

Once a serial application is converted to a parallel format, the programmer may decide to enhance the implemented solution to increase performance. This can be done through the process of performance optimisation and tuning. In a sequential code scenario, the compiler takes the responsibility for optimisation, however, in a parallel context, the programmer is charged with this responsibility (Hwu et al. 2008).

When parallelising an application, choosing an efficient decomposition is critical to minimise communication overheads (Geer 2007; Lee and Kedem 2002). Changing the decomposition strategy is thus one approach used by programmers to obtain better performance. As this thesis is concerned with abstracting decomposition and communication when designing and implementing a parallel solution, these tasks will be discussed in more detail in the next sections.

### 2.5 Data Decomposition

Decomposition typically occurs in the design phase of the parallelisation process when the parallel developer has identified the computationally intensive parts of the program and has captured an understanding of the data structures and how the data are used (Mattson et al. 2004, p. 34). Data decomposition corresponds to Single Instruction, Multiple Data (SIMD) in Flynn’s taxonomy (Flynn 1972). It is often performed in two steps — firstly, the data on which the computations are performed are partitioned and secondly, tasks are assigned the work of computing those data (Grama et al. 2003). Data decomposition of an *embarrassingly* parallel program (a program where little effort is required to separate the problem into a number of parallel tasks, e.g. little or no data dependencies exist (Crandall and
2.5 Data Decomposition

Quinn (1993)) requires minimal effort. In these applications, communication is necessary on program completion, where the resultant data is gathered and sent to the master process. However, in more complicated applications, where data dependencies do exist, decomposing the program requires extensive knowledge of the program’s low level details, as dealing with issues such as data layout and communication is extremely complex and error-prone (Chan et al. 2003). Massingill et al. (2007) emphasise the complexity involved, stating that:

\[ A \text{ good data decomposition balances the competing forces of flexibility, efficiency, and simplicity} \]

In a simplified example, data decomposition involves dividing the data into N parts depending on the degree of concurrency. These N parts can be mapped onto N processors and run in parallel if there are no data dependencies. Figure 2.2 shows how a matrix-vector multiplication problem can be decomposed using this technique. In this example, we are calculating \( A \cdot b = y \). Data are partitioned into 4 parts (N=4) corresponding to 4 independent tasks. The grey shaded partition in Figure 2.2 shows the data accessed by task 1 (Sinnen 2007). Each process has an allocated slice of the global data which is stored locally, enabling it to calculate the computation relevant to its local data in parallel with other processes.

Data decomposition can be qualified based on when the partitions are set. Pautasso and Alonso (2006) break this into three categories:

![Figure 2.2: Data Decomposition applied to matrix-vector Multiplication (Sinnen 2007)](image-url)
Chapter 2. Data Decomposition

- **Static**: the number of partitions is known in advance (at design time) and is fixed.

- **Dynamic**: the number of partitions is set at deployment or runtime and should be determined based on the size of the dataset and the amount of resources available during execution.

- **Adaptive**: the number of partitions can be manually controlled or automatically determined during the execution.

Mohammed et al. (2011) analysed data decomposition characteristics of three scientific applications (all based on bioinformatics and medical image processing), in order to identify patterns. These characteristics are as follows:

- **Structure acquiring method** where the decomposition can be *manual* or *predefined*. *Manual* requires expert domain knowledge as the programmer determines the decomposition whereas *predefined* adds restrictions such as infrastructure demands.

- **Mutually exclusive** where produced data units are either mutually exclusive or partially overlapping, i.e. have *ghost* or *halo* regions (these regions are described in Section 2.5.2).

- **Exhaustive decomposition** determines whether all the data elements in an input dataset are needed to complete the decomposition or whether this can be done progressively.

- **Results combination** where the developer needs to be aware of the combination/reduction step as this is influenced by the decomposition method, e.g. the reduction of computations from all the processes could potentially result in a significantly sized data structure, too large to fit in the memory of the master processor.

Awareness of these characteristics helps to increase efficiency and enable reusability when performing data decomposition across applications (Mohammed
2.5 Data Decomposition

et al. 2011). For example, if the programmer is faced with a results combination which will not fit in memory, he/she can consider using a parallelised format (such as MPI-IO) to gather these results. This technique can then be reused if another application has the same characteristic. Similarly, a programmer may recognise when an exhaustive decomposition is not required such as a multi-stage computation where the output of one stage is the input to the next. The whole dataset does not need to be decomposed at the outset which can save programmer effort. If time has been expended programming a partially overlapping distributed application, the programmer may be able to reuse the same data exchange operations in a similar application. In the same way, a predefined decomposition for one application may be reused for another. Cognisance of these characteristics has the potential to increase the efficiency of the development process.

2.5.1 Grid-based Data Decomposition

In many scientific domains, structured multidimensional grids are commonly used to provide visual representations of the problem space (Crandall and Quinn 1993; Hendrickson 1998). Asanovic et al. (2006) describe a structured grid as a type of dwarf\(^2\) which can be applied to general purpose computing applications such as fluid dynamics, finite element methods, magneto hydrodynamics, general relativity and quantum chromodynamics. The data involved in these applications are depicted as points in cells of the grid, but are represented in memory using linear structures such as arrays. These structures can be decomposed into contiguous substructures. Such decompositions can be along a number of dimensions and the substructures are referred to as blocks (Mattson et al. 2004). It is typical that an array is divided into as many equally sized blocks as there are processes allocated to the execution of the application. Different shapes of the blocks are possible, e.g. a block can be a row in the array, creating a row-based topology, a column in the array creating a column-based topology, or a rectangular region spanning multiple rows and columns (where the elements are unit squares or

\(^2\)An algorithmic method that captures a pattern of computation and communication
unit cubes) creating a *Cartesian* topology. A block-cyclic distribution strategy is employed when work is not distributed equally. More blocks are created than the amount of processors and a round-robin algorithm maps these blocks to the processors (Grama et al. 2003). In a randomised block distribution, blocks are mapped randomly to processors; this strategy is used when the amount of work is spread irregularly across the array (Grama et al. 2003). Figure 2.3 depicts a row topology on the left and a Cartesian topology on the right. The numbers in the sub-domains represent the processor number. The bottom left and right of the figure represent block-cyclic distributions and span the same processor count as the block representation.

### 2.5.2 Data Decomposition Issues to Consider

Decomposing data so that they are efficiently distributed over a multidimensional grid is a challenging process (Dovolnov et al. 2003). Massingill et al. (2007) claim
the process of finding a suitable decomposition is a balance of competing forces. This section will discuss some of these forces.

When decomposing a problem the programmer needs to consider how fine he/she wishes to partition the problem as the granularity of the data to be distributed can significantly alter the communication overhead in a message passing application. A fine granularity will result in smaller data pieces but a higher number of messages. This in turn will result in more communication that may decrease overall performance. In a coarse grained decomposition, on the other hand, the data are divided into larger pieces resulting in a smaller number of messages and hence lower communication overhead. Mattson et al. (2004, p. 82) suggest the decision process is ad-hoc and claim programmers typically:

*experiment with a range of chunk sizes to empirically determine the best size for a given system.*

They present an example that demonstrates the implications of choosing different decomposition mappings as depicted in Figure 2.4. An $N \times N$ matrix is decomposed into four pieces using either a column (left of figure) or a block/Cartesian (right of figure) decomposition. In the case of the column decomposition, the data are divided into columns of area $N \times \frac{N}{4}$. The perimeter (addition of all four sides) where data will be exchanged/communicated for each distributed column is $N + N + \frac{N}{4} + \frac{N}{4} (= 2\frac{1}{2}N)$. In the Cartesian decomposition, the data are
divided into blocks of area $\frac{N}{2} \times \frac{N}{2}$ and the perimeter for each distributed block is $\frac{N}{2} + \frac{N}{2} + \frac{N}{2} + \frac{N}{2} (= 2N)$. From this we can see that the amount of data to be exchanged would be less for the Cartesian decomposition $O(N^2)$ than for column decomposition ($2N$). However, in certain computations, operations are only performed on data in cells above and below the data cell in question, and not left and right. In such a scenario, column decomposition would be more efficient as the full perimeter is not involved in the data exchange.

When this data are decomposed, each element in the global array (representing the grid as a whole) is mapped to an element in a locally distributed array (a subcomponent of the global array distributed to each process) (Midkiff 2011). Each element (point in the grid) must be identified by either its global or local index. *Stencil* operations are commonly used to map the data involved in the exchanges. Each point in a grid is associated with a subset of neighbours from which it receives updates (Datta et al. 2008). Figure 2.5 depicts a two-dimensional 9-point stencil. The cell drawn as a red point in the centre must communicate with each of its eight neighbours depicted as black points. This is also known as a *star* stencil and requires updating of corner cells as well as row and column borders. A 5-point stencil is known as a *cross* and requires interaction with four neighbours where corners do not need to be exchanged.

The challenge of data exchange is apparent when a point is located on the border of the distributed chunk and does not have a full set of neighbouring points within its local data set. A solution to this challenge is the *Ghost-Cell*
2.5 Data Decomposition

Pattern (Kjolstad and Snir 2010), where each process stores a replication of the dependent data in a buffered region on the border of its allocated local data chunk. In each iteration, neighbouring processes then update these buffer data locations which are referred to as halo regions containing ghost cells (Hager and Wellein 2010, p.118). Kjolstad and Snir (2010) illustrate this exchange in Figure 2.6 where two processes (1 and 2) each containing a local two-dimensional structure are depicted. The halo regions are outlined in an amber colour. When an iteration occurs, Process 1 sends a copy of its local data in column 4 which is saved to the halo region of Process 2. Process 2, similarly sends a copy of the data in its local column 5 and updates the halo region in Process 1. This scenario is referred to as a type of Geometric decomposition by Keutzer et al. (2010). Each process does not need to know the whole data distribution but rather how the entries of its local partition corresponds to the entries in the global structure. The relation between the indices in local data structures and the indices in the global data distribution needs to be expressed for communication to be possible. Keutzer et al. (2010) claim that ‘complex bookkeeping’ is required to map indices in this manner, adding further complexity to the decomposition process.

The programming language of the application can also impact the cache performance in certain decomposition scenarios. In the C language, the method used for storing multidimensional arrays in linear memory is row-major order

![Figure 2.6: Ghost Cell Example (Kjolstad and Snir 2010)](image-url)
while in the Fortran language is column-major order. Having a row-major order means that rows are numbered as the first index of a two-dimensional array and columns are the second index. This is reversed in column-major order. This should be considered when choosing a row or column based topology as array elements that are contiguous in memory are faster to access due to caching which may result in better performance.

2.6 Message-Passing Communication

A process is a program performing a task on a processor and each process in a message passing program runs an instance of the program and communicates by passing messages via send and receive routines in two-sided communication. MPI is a standardised message passing library specification and is portable with Fortran and C/C++ interfaces (MPI-2 2014). In order for this communication to happen, each MPI process has a rank or an identification number. Communication may be synchronous or asynchronous and can occur as a point-to-point or a collective routine. In point-to-point communication, data are exchanged in a buffer which is an array of a number of elements of some particular MPI datatype. Data can be exchanged through send and receive operations. Messages are identified by their envelope and can only be exchanged if the sender and receiver specify the correct envelope. For two-sided communication to succeed the following requirements must be adhered to:

- Sender must specify a valid destination rank.
- Receiver must specify a valid source rank.
- The communicator (an object that defines which collection of processes may communicate with each other) must be the same.
- Identifier tags must match.
- Message data types must match.
- Receiver’s buffer must be large enough.
2.6 Message-Passing Communication

One sided communication is also possible in MPI. The programmer must set up a memory window which is a block of memory defining the area of memory that can be used for Remote Method Access (RMA) operations. Data are moved through put and get operations. Synchronisation calls are required to ensure operation completion.

The MPI library provides process mapping functionality through the creation of virtual topologies. In MPI, a topology is a mechanism for associating different addressing schemes with the processes belonging to a group. In a Cartesian topology, each process is connected to its neighbours in a virtual grid where each process is identified through Cartesian coordinates. This abstract topology is then mapped onto the natural topology of the problem domain. MPI provides mapping functions to compute the process coordinates in the Cartesian topology by the given ranks and vice versa. MPI also manages system memory that is not directly accessible to the user. For this reason, objects stored in this memory are described as opaque. A communicator is an example of an opaque object and its function is to determine the scope of the communication in which a point-to-point or a collective operation is to operate. A new communicator is necessary when creating a Cartesian topology.

The advantages of using MPI include high performance, scalability and portability. However, the low-level programming requirements associated with MPI negatively impact its adoption on a general scale. Skillicorn and Talia (1998) devised a classification to assess the suitability of parallel programming models for realistic parallel programming based on the level of abstraction offered; MPI was placed in the everything explicit category as details of decomposition and communication must be explicitly managed resulting in software that is ‘extremely difficult to build’. In a report produced by the United States President’s Information Technology Advisory Committee (PITAC 2005), MPI was likened to ‘programming in assembly language’. The report described MPI as a model requiring developers to:
provide deep knowledge of application software behaviour and its interaction with the underlying computing hardware.

MPI was further described as putting intellectual burden on HPC developers, thus limiting the usability of systems in this area. In order to convert an application from serial to parallel using MPI, the programmer must rewrite the application to incorporate the necessary MPI routines. These routines must be invoked explicitly and the programmer must be aware of the lower level details of the parallel program as well as the underlying architecture of the machine the application is intended to execute on. Basili et al. (2008) conducted a study focussed on the HPC community by surveying practitioners involved in the High Productivity Computing Systems program (HPCS (DARPA 2013)). To explain the popularity of MPI, despite ‘constant grumbling about its difficulty’, they found that scientists are slow to change from trusted technologies and so are reluctant to deviate from MPI. A finding from their study concluded:

well-supported technologies such as MPI have fewer adoption barriers than a new language.

The practitioners expressed trepidation at the idea of rewriting their applications (which were long-lived legacy codes) in a new language. This is perhaps an explanation for the lack of mass adoption of new technologies, the subsequent unavailability of such technologies and hence the dominance of MPI in the HPC world.

2.7 Conclusion

This chapter sets the background scene for the research program that is undertaken in this thesis. An overview of the parallel landscape was provided, from the rise of parallel programming to the architectures and memory models associated. Distributed and shared memory systems were described and various parallel pro-
2.7 Conclusion

Programming models were outlined. These concepts impact a parallel programmer’s decision-making process when parallelising an application.

Choosing an appropriate decomposition strategy was described as a key step in the parallelisation process. Data decomposition was introduced and the issues associated were described. The boundary exchanges made necessary by existent data dependencies in decomposed applications were illustrated and the ghost-cell pattern explained. The message-passing model specification of MPI was also introduced due to its suitability to implement parallel solutions for execution on distributed systems.

The literature shows that many issues should be considered when performing data decomposition. Knowledge of global and local data structures is essential when designing the communication exchanges. A programmer must be prepared to manually write the low-level calls to fit the chosen decomposition strategy. This is described as an error-prone activity, yet programmers are slow to move from MPI and reimplement their applications using new languages.
3

Research Process

3.1 Introduction

This chapter presents an overview of the research process followed for the research presented in this thesis. This consists of a number of studies, all of which were conducted in the context of the research objective stated in Chapter 1. The research objective led to the generation of five research questions, each of which is addressed by one or more research studies. As research methods should follow research questions in a way that best allows the researcher to obtain accurate and useful answers (Johnson and Onwuegbuzie 2004), each research question is addressed through a specific research approach; all of which are presented and justified.

The chapter is organised as follows: Section 3.2 refines the research objective, and defines and justifies the research questions. Section 3.3 presents an overview of the research design by clarifying the type of each research question and briefly outlining the philosophical stance taken by the researcher. The research approach and method used for each research question is outlined in Sections 3.4 to 3.8. Section 3.9 summarises the chapter.
3.2 Research Objective

Chapter 2 highlighted the difficulties documented in the state-of-the-art associated with porting serial applications to execute on high performance architectures. The task of accurately determining data dependencies and devising efficient communication is considered a contributing factor to the difficulty involved in parallelising message-passing applications. This research is concerned with investigating these challenges and so the following research objective was set:

To investigate the specification and implementation of data decomposition as an issue in parallel programming and to address problems programmers face in these tasks.

In order to investigate this area, the pervasiveness of the challenge of data decomposition had to be explored. This led the researcher to define the first research question:

**RQ1**: Is data decomposition and communication a real and prevalent challenge that practitioners face?

This research question is addressed through studies which are described in detail in Chapter 4.

Basili et al. (2008) and Hochstein and Basili (2008) have conducted studies in the field of HPC that empirically assess the state-of-practice. While these studies proffer insights regarding project organisation, software usage, testing, tuning and debugging, these studies do not examine the challenges posed when undertaking data decomposition. Eccles and Stacey (2006) conducted a study to understand the software needs of novice parallel programmers and again did not focus attention on the area of decomposition. To the researcher’s knowledge, there are no documented empirical studies specifically focusing on data decomposition and so the state-of-practice when performing this task is ambiguous. The following research question was defined to determine this knowledge, the results of which are presented in Chapter 4:
RQ2: What do practitioners use to support their efforts when performing data decomposition and communication?

Pancake and Cook (1994) argue that a systematic approach is required for HPC tool development, and so this research subscribes to the view that empirical data should be captured on the extent and impact of the problem and the efficiency and efficacy of current tool support, prior to embarking on tool development. In order to have a base-line on which to assess current tool support, the following research question was asked:

RQ3: What tooling requirements are necessary to support practitioners when performing data decomposition and communication?

Research Question 3 is addressed in Chapter 4 where a list of derived requirements is outlined based on studies conducted. Following these studies, it was necessary to assess the state-of-the-art to determine whether current offerings could fulfil the empirically derived list of requirements. An exploration was conducted of the existent approaches and tooling options to examine the abstractions provided to aid the task of decomposing data and performing communication. The requirements gathered from Research Question 3 were used as a comparison parameters to assess the efficacy of tooling support on offer. Hence Research Question 4 and a sub-question, Research Question 4.1, were defined as:

RQ4: What tooling support exists to assist the task of data decomposition and communication?

RQ4.1: Does the currently offered tooling support meet the derived requirements outlined in RQ3?

Research Question 4 is addressed in Chapter 5 where the results of a comparative analysis of the current tooling support is outlined.

As the state-of-the-art was found to be limited with regards to fulfilling the derived requirements, a prototype tool named MPIGen (MPI Generator) was designed and implemented to support these requirements.
3.3 Research Objective

(1995) claim that empirical studies are key to evaluating the benefits of methods and tools. In order to focus an evaluation of the prototype tool, the following research question was defined:

**RQ5:** Does a tool that fulfills the derived requirements, effectively support the task of data decomposition and communication?

Research Question 5 is addressed in two studies described in Chapter 7. An overview of the five research questions is shown in Figure 3.1.

![Figure 3.1: Overview of Research Questions](image)
3.3 Research Design

3.3.1 Research Question Type

Easterbrook et al. (2008) provide guidance on which research methods suit which problems and claim the first step in choosing a research method is clarifying the type of research question. The classification of question-type devised by Easterbrook et al. is used to assist the choice of research methods in this thesis and is summarised in Table 3.1 where sample questions are listed to demonstrate their form. The research questions defined in Section 3.2 are discussed and categorised according to these types. Exploratory questions are described as important in the early stages of a research program, in order to understand the phenomena.

<table>
<thead>
<tr>
<th>Research stage</th>
<th>Type of question</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploratory</td>
<td>Existence questions</td>
<td>Does X exist?</td>
</tr>
<tr>
<td></td>
<td>Description and Classification questions</td>
<td>What are the properties of X?</td>
</tr>
<tr>
<td></td>
<td>Descriptive-Comparative questions</td>
<td>How does X differ from Y?</td>
</tr>
<tr>
<td>Base-rate</td>
<td>Frequency and Distribution questions</td>
<td>How often does X occur?</td>
</tr>
<tr>
<td></td>
<td>Descriptive-Process questions</td>
<td>How does X normally work?</td>
</tr>
<tr>
<td>Relationship</td>
<td>Relationship questions</td>
<td>Do occurrences of X correlate with the occurrences of Y?</td>
</tr>
<tr>
<td>Causality</td>
<td>Causality questions</td>
<td>Does X cause Y? What effect does X have on Y?</td>
</tr>
<tr>
<td></td>
<td>Causality-Comparative questions</td>
<td>Is X better at preventing Y than is Z?</td>
</tr>
<tr>
<td></td>
<td>Causality-Comparative Interaction</td>
<td>Does X or Z cause more Y under one condition but not others?</td>
</tr>
<tr>
<td>Design</td>
<td>Design questions</td>
<td>What’s an effective way to achieve X?</td>
</tr>
</tbody>
</table>
being researched. Exploratory questions are broken down into the following types: *Existence*, *Description and Classification* and *Descriptive-Comparative*. The result of exploratory questions is a clearer understanding of the phenomena under study. RQ1 aims to explore the prevalence of the challenge of performing data decomposition in practice and so can be classified as an *Existence* question. RQ2 aims to determine the manner in which practitioners perform data decomposition which is an exploration of the properties of the challenge and so falls within the exploratory category as a type of *Description and Classification* question. Research methods typically used to answer exploratory questions are those that offer qualitative results which help to formulate theories about the concept under study.

Once a clear understanding of the phenomena is gained, base-rate questions need to be asked to determine patterns related to the occurrence of the phenomena. Two types of base-rate questions exist: *Frequency and Distribution* and *Descriptive Process*. RQ4 is of the style *Descriptive-Process* as the researcher needs to determine how the problem of data decomposition has been previously addressed. The tool support/approaches documented in the state-of-the-art need to be investigated.

While RQ1, RQ2 and RQ4 are knowledge-gathering questions focusing on current practice, RQ3 focuses instead on designing a better way to do things. For this reason, it is categorised as a *Design* question as it seeks to list requirements for a tool which will better enable a programmer to perform data decomposition and write communication calls. This question can be posed after the necessary information is gathered portraying the characteristics of the problem to be solved.

*Relationship* questions study the relationship between two different phenomena, and specifically whether the occurrence of one is related to the occurrence of the other. This research does not compare or contrast any phenomena and so does not pose any *Relationship* type of questions.

*Causality* questions attempt to determine a cause and an effect. RQ5 can be classified as a *Causality* type of question, as the abstraction implemented in the prototype tool must be evaluated to determine whether it causes an improvement.
Table 3.2 outlines the research questions defined in this thesis stating their type and the research stage associated. This categorisation by type helps to determine the appropriate methods to use.

### 3.3.2 Philosophical Stance

The philosophical stance adopted by a researcher determines the methods deemed appropriate to yield acceptable evidence in answer to the research questions posed. The first research paradigm considered was positivism. A positivist stance bases knowledge on logical inference from a set of observable facts such as quantifiable measures of variables and hypothesis testing (Easterbrook et al. 2008; Klein and Myers 1999). Myers (1997) views positivism as that reality which can only be described by measurable properties that are independent of the researcher (including the researcher’s instruments). Robson (2002, p. 22) critiques positivism by rejecting the view that science should deal only with observable phenomena. As this research involves interaction with participants within their working environment, the findings are not studied in isolation from their context and are therefore not based only on observable phenomena. For this reason, positivism alone is not the best approach for this research.

An interpretivist stance insists scientific knowledge must also be associated with its human context such as language, consciousness, shared meanings etc. (Easterbrook et al. 2008; Klein and Myers 1999). Elements of the empirical stud-
ies conducted involved shared understanding between the researcher and the participants and so this approach is applicable in certain cases. However, the evaluation conducted in this research program is partially based on hypotheses testing within a controlled environment basing knowledge on a set of observable facts. This contradicts the interpretivist view, instead reflecting that of a positivist reality. For this reason, the research cannot be classified as following the interpretivist paradigm alone.

A middle ground philosophy exists in that of pragmatism. Johnson and Onwuegbuzie (2004) describe a pragmatist stance as that which offers:

*a practical and outcome-oriented method of inquiry that is based on action and leads, iteratively, to further action and the elimination of doubt; and it offers a method for selecting methodological mixes that can help researchers better answer many of their research questions.*

Pragmatism can be viewed as a joining point between both positivist and interpretivist views, equating to a middle chair with positivism sitting on one side and interpretivism sitting on the other. The researcher views knowledge by studying problems using a mixture of research methods. Robson (2002, p. 43) describes this as a potential solution to such ‘paradigm wars’ as the researcher is free to use whatever method is considered most suited to the particular research problem at the time. This paradigm is the best fit for the research described in this thesis as a whole.

This pluralistic approach complements the use of mixed methods when conducting research. Mixed methods combine both qualitative and quantitative forms (Creswell and Miller 1997). It allows the use of both techniques in tandem so that the overall strength of a study is increased. A qualitative approach allows the researcher to be innovative within a self-designed framework while a quantitative approach provides a means for testing theories by examining relationships among variables (Creswell 2009). Using a mix of both qualitative and quantitative methods allows for extra flexibility and provides breadth for the researcher to
focus on the end goal of solving a problem rather than facing the limitations of being confined to a certain approach.

Part of this research also follows a design-science paradigm where knowledge and understanding of a problem and its solution are achieved through the building of an artifact such as a prototype software system (Hevner et al. 2004; Niederman and March 2012). Design-science not only aims to ‘create things that serve human purposes’ (March and Smith 1995) but also to ‘improve existing things to serve human purposes better’ (Wieringa 2009). The development of a prototype tool can provide insights into the problem being solved as well as the knowledge-base on which the software is founded. A pragmatist view of design-science does not separate the theory from the utility as the design contribution of a utility (e.g. an effective prototype tool), cannot be stated without a thorough evaluation. The accumulation of theory is thus fed into the design of the tool and the resultant tool is then evaluated based on theoretical hypotheses. In this manner, theory and design represent a ‘virtuous cycle’ (Niederman and March 2012).

3.4 Research Approach for RQ1: Exploration of Challenge

3.4.1 Exploratory Interviews

Within the software engineering community, empirical studies are used to improve processes, methods and tools, and so can be used to extract knowledge to address exploratory research questions (Sjøberg et al. 2002). Exploratory interviews seek new insights and generate ideas and hypotheses for new research (Runeson and Höst 2009). To address RQ1, an exploratory study was carried out in two different organisations, where four parallel programmers of varying expertise were interviewed. The researcher gathered qualitative data, as such data provide rich and informative results when used in empirical studies in technical fields (Seaman 1999). The interviews were semi-structured and the data were collected in
the participants’ setting where the researcher had the responsibility of interpreting the findings. The findings from this study are outlined in Section 4.2.2.

The interviews were transcribed verbatim and the data collected were analysed using coding techniques influenced by Grounded Theory (Glaser and Strauss 1967; Strauss and Corbin 1998). The researcher thoroughly read all interview transcripts and phrases of interest were extracted that were relevant to the research question. Each phrase was recorded and annotated with a code, or label, that reflected the contents of the entry. After this, the data were sorted and grouped based on the labels. Section 4.2.2 provides details of the labels used in this study. In this manner, the researcher developed her understanding of the practical problems of data decomposition as experienced by the participants.

3.4.2 Participant-Observer Study

To add to the exploratory insights required by RQ1, a Participant-Observer study was conducted in IBM where a serial flood risk management application was ported to a HPC environment over a period of three months. Yin (2003) describes a Participant-Observer study as:

\[ a \text{ special mode of observation in which you are not merely a passive observer. Instead, you may assume a variety of roles within a case study situation and may actually participate in the events being studied.}\]

This study was conducted as a mentor-driven exercise; three researchers carried out the study and three experienced parallel programmers from IBM’s HPC group guided their activities. An advantage of using such an approach is the ability to gain access to the expertise offered by parallel programmers in situ. As events occur in real-time, the researchers understand the context of the event and can gain insights to behaviour and motives which otherwise would not have been observed (Yin 2003). A disadvantage is the potential for bias as the investigators may have a vested interest and may manipulate events (Kitchenham et al. 2010). As this study involved three mentors who directed the porting activity, the researchers
relinquished full control over events and so the potential for bias was minimised. This control did not affect the observations drawn from the study, as these were derived by the researchers without guidance from the mentors and so can be treated as external observations. As this study was conducted as a group, all three colleague-researchers ensured these observations were accurately observed through regular meetings held to discuss notes taken and to raise questions about the events. The insights gained from this study contribute to the depth of exploratory findings gathered in answer to RQ1. Details regarding the work conducted in this study and the findings are described in Section 4.3.

Findings from both a focus group study and a survey study also provided insights and contributed to addressing this question. As these were primarily used to address RQ2, these are described in the following Section (3.5).

3.5 Research Approach for RQ2: State-of-Practice

3.5.1 Focus Group Study

To address RQ2, a focus group study was conducted using three participants from IBM’s HPC group. A focus group session can produce candid information (Shull et al. 2010) and so was deemed an appropriate method to provide answers to a Description and Classification type of research question. Kontio et al. (2008) view a focus group as that which involves between three and twelve participants and further elaborate that:

\[ \text{the group setting enables the participants to build on the responses of and ideas of other participants, which increases the richness of the information gained.} \]

This response building is a benefit of using such a method, however, a disadvantage is that it can often be difficult to generalise the results. To cater for this limitation,
3.5 Research Approach for RQ2

the focus group was conducted as part of a mixed method approach where more than one approach was used. Creswell (2009) describes a technique of using sequential mixed methods where the researcher expands or elaborates on the findings from one method, i.e. the focus group, with another method. The focus group gathered qualitative data which was analysed using the technique described in Section 3.4.1. A survey was then conducted which gathered both qualitative and quantitative data, building on the findings from the focus group. A survey method with a large sample can help a researcher to generalise results to a population (Creswell 2009).

3.5.2 Survey

A survey can be described as (Runeson and Höst 2009):

*the collection of standardised information from a specific population or some sample from one, usually, but not necessarily by means of a questionnaire or interview.*

Survey research provides insight into the attitudes or opinions of a population by studying a sample of that population. The population targeted in this survey was HPC practitioners, specifically those who had experience parallelising applications. Typically, surveys are carried out for descriptive purposes (Robson 2002) which supports this choice of method to further answer a Description and Classification type of research question. The survey collected both quantitative and qualitative data concurrently which according to Creswell (2009) is a form of concurrent triangulation. The researcher is then free to either merge the two types of data or compare the data side by side. The survey consisted of a number of ranking style questions followed by descriptive questions determining information such as tooling support used. An advantage to using a survey is that it offers a straightforward approach to study practitioners’ attitudes and experiences. A disadvantage of using a survey is that the data can be affected by the characteristics of the respondents such as their personality or memory. Sometimes respondents
want to be shown in a good light and so a socially desirable response bias can be present (Robson 2002). A more detailed description of the survey and the findings is outlined in Section 4.5.

3.6 Research Approach for RQ3: Requirements

RQ3 is a Design type of question. The design of a prototype cannot commence without the articulation of the goals or requirements of that tool (Niederman and March 2012). This question required the researcher to analyse the data gathered from the empirical studies conducted as part of RQ1 and RQ2 in order to devise a list of requirements for tool support. The researcher used the constant comparison technique (Seaman 1999) to identify common themes between the topics for answering the research question. Data recorded from three empirical studies: exploratory interviews, focus group session and qualitative data collected in the survey, were read thoroughly and phrases that were considered relevant to the research question were assigned labels in the field notes. Passages of text were then grouped into patterns according to the codes. The researcher created field memos to record observations from the coded data. Seaman (1999) describes field memos as ‘the vehicle by which the researcher first articulates the findings’. The following example is an excerpt from a field memo documented to record observations from the focus group session; the codes assigned in the field notes are interspersed and referenced in bold font:

A tool was suggested to assist the conversion of sequential code to parallel [Tooling Need]. This would focus on decomposition of data as this was described as a laborious and time-consuming part of the parallelisation process [Decomp. Challenges]. As one participant outlined: ‘The amount of times we sat there counting indices and rows was incredible’ [Indexing Issue]. The aim of the tool would be to map what the programmer intends the code to do and create the
data structures to carry out this mapping [Data Struct. Generation]. This tool could assist with the indexing or ensure indexing in the code [Indexing Issue] reflects some type of notation that has been specified or diagrams specified by the expert programmer [Programmer Intent].

The data were collected iteratively in these phases with the intention of building on the previous stage, for example, the survey was designed to add further details (supportive or otherwise) to the propositions developed from the focus group session. The feasibility of the observations from an earlier stage (interviews, focus group) were then checked against the data collected in the later stages (focus group, survey). For example the following field memo was recorded from the survey:

When asked what tools/media are used, one respondent answered: ‘There is no tool that really helps any parallel programmer to effectively understand how to decompose data’ [Decomp. Challenges]. Custom tools were mentioned: ‘Custom tests to make sure indexing is working’ [Indexing Issue]

The challenge surrounding indexing, derived in the focus group, was again stated in the survey responses and so a requirement to support this issue materialised. Additional requirements for tooling support emerged from the analysis and are outlined in Section 4.6.

3.7 Research Approach for RQ4: Existent tooling support (state-of-art)

RQ4 is a base-rate type of question and required a review of the literature to investigate the existent tooling support available to perform data decomposition in a HPC context. Two main approaches exist to conduct literature reviews; that is the traditional literature review and a Systematic Literature Review (SLR). A SLR is described by Kitchenham and Charters (2007) as:
a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest.

SLRs are used as a method to conduct secondary research and have become a popular approach to review existing literature, offering benefits such as repeatability due to transparency of the process (Kitchenham and Charters 2007). Traditional literature reviews rely on ad-hoc methods and can result in more biased findings as the researcher is not strictly using a systematic protocol. A researcher instead starts with a number of well-known or highly cited papers and follows a trail of references which point to other relevant publications.

The researcher conducted a scoping study where a number of terms relevant to the research question were searched in the following digital libraries: IEEE Xplore, ACM Digital Library, SpringerLink, and Google Scholar. The terms used included the words *data decomposition*, *communication*, *abstraction* as well as *tool support*, *high performance computing* and various forms of parallel (*parallelisation*, *parallelization*, *parallel programming*). The researcher was aware of the work conducted by Arora (Arora and Bangalore 2009; Arora et al. 2010) in this field where a framework for abstracting message passing applications was created. However, the scoping study conducted using search strings with different AND/OR Boolean logic did not yield any of the Arora (and Bangalore; et al.) papers. Another set of papers which would not have appeared in an SLR presents the Parallel Object-Oriented Methods and Applications (POOMA) framework (Reynders et al. 1996). Data decomposition support is not foremost in the titles or abstract descriptions in the publications associated and so would not have appeared in the findings in an SLR. For these reasons the researcher decided to conduct a traditional literature review.

The findings of the review are outlined in Chapter 5 where the tool support is categorised according to the approach used.
3.8 Research Approach for RQ5: Evaluation

RQ5 is a *Causality* type of question as the goal is to answer causality questions regarding the effectiveness of the tool. While the previous research questions sought to develop and evolve an understanding of the problem, this research question is addressed through the design-science research perspective (see Section 3.3.2 for a discussion on this perspective). Two main activities form the basis of design-science: *build* and *evaluate* (March and Smith 1995). The efficacy of the software artifact needs to be demonstrated through a well-executed evaluation (Hevner et al. 2004). A design artifact is only complete when it satisfies the requirements it was meant to solve. To strengthen an evaluation of MPIGen, two sources of evidence are studied; firstly an experiment was conducted using students as subjects and secondly, interviews were carried out using experts as subjects.

3.8.1 Student Experiment

Conducting experiments in empirical studies is important to validate products created for the development and maintenance of software and hence advance knowledge (Basili 1996). An experiment was designed where 64 student participants were required to use the prototype tool (MPIGen) to parallelise an application in a laboratory setting. This experiment was also conducted where the same participants performed a similar parallelisation exercise but did not use any tool support, i.e. they were required to manually parallelise the application using MPI. As it was not possible to recruit professionals to conduct a programming exercise of six hours duration, students from both undergraduate and postgraduate courses in computer related fields were recruited to partake. A number of studies have been conducted using students as subjects in empirical studies (Höst et al. 2000; Runeson 2003; Svahnberg et al. 2008) where only minor differences were found between their performance and that of professionals and so the researcher
Chapter 3. Research Process

is confident the findings from the experiment contribute to the validity of the tool support, at least from the perspective of a novice practitioner.

Quantitative evaluation methods identify a measurable property or set of properties that are expected to change as a consequence of using the tool (Kitchenham 1996). Quantitative data measuring the time spent working on the solution, the number of accurate submissions and the lines of code typed, were collected for both the manual and the tool-assisted exercises. Hypotheses were formed and tested and the results were analysed using SPSS version 21 (SPSS 2014). A survey was also distributed to the students, following their participation in both exercises, which captured qualitative data to gain further insight. A concurrent mixed methods approach (Creswell 2009) was used as the survey findings were integrated with the quantitative data in the interpretation of the overall results. A more detailed description of the experimental setup and analysis as well as the survey findings are presented in Sections 7.7 and 7.8.

3.8.2 Expert Interviews

Five expert parallel programmers (those who have parallelised a minimum of three applications and have at least one year’s experience in a HPC industrial context) were recruited to evaluate the prototype tool. A demonstration of the tool was firstly carried out and the participants were instructed to browse the generated code and pose any questions. As three of the participants were based in one organisation (the Irish Centre for High End Computing, ICHEC), a focus group session was conducted onsite with all three participants present. Two separate interviews were conducted for the remaining two participants, one was located remotely and so the session was conducted using video conferencing technology. A survey was distributed following the demonstration and the participants were required to provide both qualitative and quantitative data. To further the researcher’s understanding, a further interview/focus group session was conducted to seek clarification on any of the answers provided.
3.9 Conclusion

As described in Section 3.5.2, a concurrent triangulation method was used where the researcher was free to merge the qualitative and quantitative data gathered in order to attain richer insight from the answers provided. An example of a quantitative question asked is one where the participant was required to tick a box representing options to answer the question: *Would this tool be applicable for typical parallelisation projects in your organisation?* A qualitative style question which adds depth and potential reasoning to the answer given in the previous quantitative style question is the following: *What are the advantages/disadvantages of using the MPIGen tool?* This merging of data allowed the researcher to better interpret the answers. A full description of the participants, survey details and findings are presented in Sections 7.3 and 7.4.

3.9 Conclusion

This chapter presents an overview of the research process followed in this thesis. Five research questions are addressed in this research program, for which six studies were conducted. A variety of research methods were used, such as interviews, participant-observer, focus group, survey, traditional literature review and an experiment. Data were collected from different sources and a mixed method approach was employed where both qualitative and quantitative data were compared. A graphical overview of the research phases conducted in this thesis, including the methods employed in each phase is presented in Figure 3.2. The research questions are outlined and the phase in which each question is addressed is illustrated. A justification is also provided in this chapter for each of the research methods chosen to address the five research questions. Chapters 4, 5 and 7 address the research questions in more depth and provide extensive details regarding the research designs for each of the studies, as well as discussions of their limitations.
Figure 3.2: Research Process
4

Empirical Characterisation
of the Challenge
of Data Decomposition

4.1 Introduction

Data decomposition was a problematic phase of the parallelisation process in the literature outlined in Chapter 2. This chapter seeks to empirically derive the characteristics of this challenge by gathering data related to practitioners’ experience. Figure 4.1 is an expansion of the phase Investigation and Characterisation in Figure 3.2 (on page 47) and illustrates in more detail the four stages involved in this phase of the research. Firstly, an exploratory study was conducted in IBM and ICHEC where participants were interviewed to ascertain the prevalence of data decomposition as a challenge in the parallelisation process (Stage I). Following this, a participant-observer study was conducted, to gain deeper insight regarding the issues experienced by practitioners when decomposing and porting a data-intensive application to a HPC environment (Stage II). The aim of this study was to build a characterisation of the problem. As the task of decomposition and the subsequent communication emerged as real and prevalent issues in stages I and II, the next stage of this derivation involved an investigation of the state-of-practice
4.1 Introduction

Figure 4.1: Stages of Empirical Derivation

of partitioners when performing data decomposition. To this end, a focus group and follow-up interviews were conducted with members of the HPC group in IBM (Stage III). Finally, a survey was distributed to a wider audience to evaluate the problem of data decomposition on a more global scale (Stage IV). As can be seen from Figure 4.1, each stage provided input to the next until the magnitude of the challenges surrounding data decomposition in a HPC environment was fully scoped. The findings from these stages are used in the comparative analysis study which is described in Chapter 5. The following research questions previously outlined in Chapter 3, are addressed in this chapter:

**RQ1**: Is data decomposition and communication a real and prevalent challenge that practitioners face?

**RQ2**: What do practitioners use to support their efforts when performing data decomposition and communication?
RQ3: What tooling requirements are necessary to support practitioners when performing data decomposition and communication?

This chapter is organised as follows: Section 4.2 describes the exploratory study while Section 4.3 outlines the design and findings of the participant-observer study. The focus group study is discussed in Section 4.4 and an overview of the survey study is provided in Section 4.5. This is followed by a discussion in Section 4.6 which includes an overview of empirically derived tool support requirements. Section 4.7 then concludes this chapter.

4.2 Stage I: Exploratory Interviews

To assess whether data decomposition emerged as a core prevalent issue in practice, an empirical study was conducted in both ICHEC (Irish Centre for High End Computing) and IBM. ICHEC is a state-sponsored HPC centre whose core mission is to provide resources, support, training and education for both researchers and Irish industries in the HPC field. The core business of IBM’s HPC centre lies in taking applications, porting and tuning these so that they run efficiently on their Blue Gene® supercomputers as well as providing evolution support for existing applications. Both centres employ parallel programmers who agreed to participate in this study.

4.2.1 Research Design

Interviews were conducted in both ICHEC and IBM with two senior parallel programmers from each, all having industrial experience in a HPC environment. The daily tasks of these participants involved evolution, maintenance and porting of systems to HPC environments. Table 4.1 lists details and background experience of these participants. In order to protect their privacy, they are referred to as P1 through to P4.

The aim of this study was to explore prevalent issues experienced and so the interviewees were asked to discuss parallelisation in general, incorporating the
4.2 Stage I: Exploratory Interviews

Table 4.1: Participants’ Details

<table>
<thead>
<tr>
<th>Org.</th>
<th>ID</th>
<th>Current Title</th>
<th>Experience in Computer Science</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICHEC</td>
<td>P1</td>
<td>Senior Software Developer</td>
<td>10 years (2 years HPC)</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>Computational Scientist</td>
<td>11 years (11 years HPC)</td>
</tr>
<tr>
<td>IBM</td>
<td>P3</td>
<td>HPC Specialist</td>
<td>4 years (3 years HPC)</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>Software Engineer</td>
<td>3 years (3 years HPC)</td>
</tr>
</tbody>
</table>

challenges they encountered. The approach used was an open-questioning format so as not to guide their answers, that is, no emphasis was placed on decomposition. Questions included:

- **Could you give a brief synopsis of how you parallelise systems?**

- **What are the main aspects of the system that the programmer should focus on during the parallelisation task?**

The interviews were not limited to only those questions prepared in the interview guide but were instead semi-structured (Hove and Anda 2005) as such questions enable a researcher to explore a topic in more depth, and to discuss unanticipated topics that emerge during the interview. Two researchers conducted all interviews. The interviews lasted approximately ninety minutes, and were digitally recorded with the participants’ consent. The interviews were transcribed verbatim and were analysed using qualitative data analysis methods (Seaman 1999). All interview transcripts were thoroughly read and phrases of particular interest were coded with labels to reflect the topic of that phrase (see Section 3.4 for more details on coding techniques).

4.2.2 Findings

In the interviews concentrated on challenges, the participants were free to speak of any issues encountered in a work-related context. The interviewees spoke at
length about non embarrassingly-parallel applications and specifically recounted issues faced when dealing with MPI and distributed memory.

Table 4.2 outlines these reported challenges (referenced Ch1-Ch8), along with the corresponding participants. The labels assigned when performing the coding during the data analysis process are also listed. The amount of times a label was assigned was counted to determine the prevalence of the topics in the interview responses. This count is illustrated in Figure 4.2.

As can be seen, DECOM, which was labelled when the participants spoke of the challenges surrounding data decomposition\(^1\) is the most prevalent. All four participants expressed their frustration surrounding data dependencies, data decomposition and the management of data exchanges in parallelisation projects. This corresponds to challenge Ch1. Participant P4 recounted that the majority of projects he has worked on were data-driven, with specific emphasis on how the data were distributed. He further described the task of data decomposition as:

\(^1\)including data dependencies, data partitioning and data splitting

![Label Count assigned during Coding Process](image)

**Figure 4.2:** Label Count assigned during Coding Process
4.2 Stage I: Exploratory Interviews

Table 4.2: Identified Challenges

<table>
<thead>
<tr>
<th>Ref. Coding Label</th>
<th>Challenge</th>
<th>Description</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ch1 DECOM</td>
<td>Decomposition of Data</td>
<td>Decomposing and managing data when writing parallel code is challenging</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ch2 LIMTOOLS</td>
<td>Limitation of Tools</td>
<td>There are insufficient tools available to aid parallel programmers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ch3 DOM_KNL</td>
<td>Lack of domain knowledge</td>
<td>Parallel programmers lack the domain knowledge necessary to fully parallelise an application and need to to communicate with domain experts</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ch4 TUNING</td>
<td>Tuning</td>
<td>Getting 100% of application performance is extremely difficult</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ch5 H/W_VAR</td>
<td>Hardware variance</td>
<td>Migrating applications between machines is problematic as architectural configurations vary</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ch6 TESTING</td>
<td>Testing</td>
<td>Testing parallel programs during development is challenging</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ch7 COM_DEB</td>
<td>Complex Debugging</td>
<td>Debugging a parallel program is a challenge</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ch8 LACK_DOC</td>
<td>Lack of documentation</td>
<td>Different architectures require specific documentation and often this is not recorded</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One of the major issues to solve at the beginning of a parallelisation project.

Deciphering the division of the data was expressed as ‘the most complicated part’ by participant P2, who continued by describing the steps involved:
Chapter 4. Empirical Characterisation

[...] you want to split data across several machines; [you] want to understand what kind of information you need to exchange between process a and process b for it to work; [you] want to figure out if you need just global data or local data ...

Participant P1 agreed that the problem is based on data locality and elaborated that once the data is distributed, the developer has to perform MPI exchanges to access any remote data:

*The main problem is that once you start splitting data, you don’t have the whole data anymore, just a part and so you need to do exchanges.*

However, this is not a trivial task as the developer needs to carefully consider data dependencies when exchanging messages as outlined by P1:

*Need to know what is connected, modifying some data could modify another.*

The granularity of the messages in a parallelised program can greatly impact performance and so an efficient data decomposition strategy plays an important role as stated by P3:

*Reducing the total amount of communication can get higher performance but this is dependent on the split.*

Participant P4 outlined the informal process followed to assess the best performing data decomposition strategy when porting applications to a high performance architecture:

*To find which is the best partition, use a trial and error method, try on few sets rather than full set ... make a guess in advance as to best partition then try that one first.*

This informal method seems to be a non-optimal approach. The challenges surrounding data decomposition were unanimously expressed by all participants,
4.2 Stage I: Exploratory Interviews

thereby providing evidence that this is indeed a real challenge experienced by practitioners.

The label LIM_TOOLS representing the challenge of tool limitations was next in order of prevalence as depicted in Figure 4.2. This challenge was also reported by all participants (Ch2 in Table 4.2). Participants P3 and P4 described adequate profiling tool support to determine bottlenecks, as well as tracing tools to analyse performance once the parallel solution is executed. However, neither participant mentioned any tool support to assist the task of data decomposition or communication when parallelising an application. Tools were described as useful on parallelised codebases but not helpful during the parallelisation process:

*Tools give insights once code is parallelised. You start with sequential code and then do work; in between, nothing is working so you have nothing. You are crossing the desert at some point.* —P2

The challenges surrounding the lack of domain knowledge (Ch3) and the tuning of an application (Ch4) were mentioned by three participants. Participant P2 reported liaising with domain experts to help determine the data dependencies in a codebase:

*By just reading the code, you won’t figure out all the dependencies. You must work with scientists, knowledge of science helps you understand.*

Participant P3 spoke of analysing the efficiency of the data decomposition and the resultant communication as an approach to increase an application’s performance claiming a typical technique is to:

*Reduce the area of communication between processes.*

The challenges of dealing with hardware variance (Ch5) and the issues surrounding the testing of parallel applications (Ch6) were reported by two participants, while one participant outlined complex debugging (Ch7) and the lack of documentation (Ch8) as problematic.
The findings suggest that data decomposition and subsequent communication are perceived as a prevalent challenge among experienced practitioners. This, allied with a perceived lack of tool support during parallelisation implies the need for a tool-supported data decomposition and communication approach.

4.3 Stage II: Participant-Observer Study

While the participants were experienced practitioners in the field, suggesting high external validity, the number of participants and companies involved implies that these can be considered provisional findings. To buttress these findings, the researcher took on the roles of participant and observer simultaneously in a participant-observer study in IBM, to further verify and understand the data decomposition and communication issues faced by parallel programmers. This research method is described in detail in Chapter 3. The study was conducted in collaboration with two industrial partners: JBA Consulting and IBM’s HPC centre. JBA provide consultancy in the flood risk management domain and have developed a performance-demanding application, JFLOW®, based on Computational Fluid Dynamics (CFD) equations. The researcher ported a non proprietary version of the JFLOW® software to the Blue Gene® architecture in order to experience and subsequently evaluate the issues presented when decomposing and parallelising a data-intensive application.

4.3.1 Research Design

As the researcher was unfamiliar with the flood risk management domain, an initial industrial placement was carried out in JBA over a period of three months. JBA’s core software system, JFLOW®, is a two-dimensional (2D) application used for simulation of overland flooding based on the 2D shallow water equations (Crossley et al. 2010; Lamb et al. 2009). Having attended an introductory hydrological workshop, an architecture evaluation of JFLOW® was planned. A three month industrial placement was subsequently carried out in IBM’s HPC centre
4.3 Stage II: Participant-Observer Study

by a team of three researchers (all with software engineering backgrounds but considered to be novice parallel programmers). The aim of this placement was to port a non-proprietary version of JFLOW\textsuperscript{®}, namely \textit{JFLOW\textsuperscript{®} Reduced Engine} (JFLOW RE), to the Blue Gene\textsuperscript{®} architecture. The port of JBA’s codebase to Blue Gene\textsuperscript{®}/P (BG/P) was guided by three assigned mentors whose profile information is listed in Table 4.3. The mentors are referred to as M1 to M3. The number of applications parallelised and optimised was not provided by M3 and so this information is omitted from the table.

Table 4.3: Mentors’ Details

<table>
<thead>
<tr>
<th>Profile Information</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Years industrial experience in HPC</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td># serial applications converted to parallel</td>
<td>5</td>
<td>7</td>
<td>n/a</td>
</tr>
<tr>
<td># parallel applications optimised</td>
<td>16</td>
<td>2</td>
<td>n/a</td>
</tr>
</tbody>
</table>

4.3.2 The Porting Task

To facilitate domain understanding in preparation for the porting task, the researchers mapped JFLOW’s\textsuperscript{®} functionality to the source code while in JBA. The codebase was then reverse engineered in order to provide high-level abstractions. Object Oriented (OO) representations such as class, component and deployment diagrams were extracted from the code in conjunction with software quality metrics gathered using NDEPENDS (NDEPENDS 2013).

As the researchers’ lacked knowledge in computational science, an initial activity in IBM involved tutorial sessions to introduce the concepts associated with numerical knowledge. These tutorials were organised by the mentors and conducted as six two-hour sessions in advance of any porting tasks. Topics covered included Taylor’s theorem, Newton’s method, MacLaurin series, Linear Algebra and Graph theory. Once the port commenced, the mentors provided the researchers with feedback on their intended work approach and reviewed the
code implementation at weekly intervals. All three mentors had prior experience converting grid-based applications from serial to parallel, but were unfamiliar with the flood risk management domain.

The parallel architecture of BG/P is structured as a hybrid (both shared and distributed) memory model (Gara 2011) and so for scalability reasons, an MPI implementation of JFLOW RE was agreed. Devising an efficient decomposition strategy involved careful analysis of the data dependencies and resultant communication overhead. As the application was based on a 2D grid comprising of a large data structure, the researchers (with guidance from mentors) decided to use two different data decomposition strategies, the first using a row-based topology and the second using a Cartesian-based topology.

As part of the data decomposition phase, each processor was allocated a subsection of the global grid including ghost edges (also known as halo regions) containing duplicates of data at the boundaries of the neighbouring blocks. The application contained thirteen different data structures which required decomposition. After each time-step in the computation, the ghost cells in each of these data structures had to be populated and so the researcher had to ensure valid and efficient message passing calls by determining the correct array indices in order to send data to and receive data from the correct locations. Depending on the decomposition strategy, more or less communication was required from different locations in the data structure. Two implementations of the parallelised codebase were required to reflect both data decomposition strategies.

Tools used in the porting process were focused on either pre-parallelisation activities such as determining bottlenecks, or post-parallelisation activities for tuning and optimisation. Doxygen (Doxygen 2013) was used by the mentors to generate call graph abstractions from the source code to identify the control-flow in the application. The GProf profiler (GProf 2014) was used in the performance characterisation of the parallelised application to determine which parts of the program consumed most time and which functions were called during execution. A lightweight MPI profiler, mpiP (mpiP 2014) was used to collect statistical tracing
4.3 Stage II: Participant-Observer Study

information about MPI functions and Scalasca (Scalasca 2014) was used to measure runtime behaviour to identify performance bottlenecks.

The parallelised JFLOW RE application was executed on BG/P and performance characterised. Metrics were collected from executions across different core counts (ranging from 128 to 1024) and various grid sizes (ranging from $1028 \times 1028$ to $16388 \times 16388$). Two implementations of the application reflecting both the row and Cartesian decomposition strategies were characterised. The results are illustrated in Appendix A. The performance characterisation showed scaling occurred for all executed grid sizes and the row based implementation outperformed the Cartesian based implementation.

4.3.3 Findings

A core bottleneck when porting JFLOW RE to BG/P was the task of decomposing the data and determining the correct function calls to implement the message passing. This task proved to be extremely tedious and time consuming validating the finding in Stage I, where the challenge of data decomposition was found to be real and prevalent. The tools used in the porting process were described in the previous section (4.3.2). No tools were available to assist the writing of MPI operations or to help devise the data decomposition. The researcher had to keep mental records of the data structures that were distributed, which was an error prone process. The researcher was compelled to draw diagrams by hand to visualise the decomposition strategy chosen so as to help determine the correct indices when sending and receiving data between processors. Figure 4.3 depicts some of these hand-drawn diagrams. As can be seen in all four diagrams, the researcher drew grid representations and modelled decomposition scenarios.

Only one test case was available to ensure correctness and that involved comparing the final parallelised output against the serial output by executing the UNIX command `diff` on the resultant text files. Any incorrect index in the data structures being communicated resulted in an incorrect final solution. In
such cases, the researcher had to manually check all the communication calls to determine which index was invalid.

As two different decomposition strategies were chosen, the coding effort was significantly increased. The researchers had to implement a row-based topology and then rewrite all the decomposition and communication calls to implement the Cartesian topology. Both efforts were necessary as it was unknown in advance which solution would scale and provide better performance. These findings provide the foundations for an initial characterisation of the properties of the challenge of decomposition and are summarised in Table 4.4 with references F1-II - F5-II.
### 4.3 Stage II: Participant-Observer Study

Table 4.4: Summary of Findings in Stage II

<table>
<thead>
<tr>
<th>Theme</th>
<th>Ref.</th>
<th>Finding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Challenge of data decomposition</td>
<td>F1.II</td>
<td>Calculating indices was tedious and time consuming</td>
</tr>
<tr>
<td></td>
<td>F2.II</td>
<td>Keeping mental records of data structures was error prone</td>
</tr>
<tr>
<td>Limitations of tool support for data decomposition</td>
<td>F3.II</td>
<td>Informal sketches were necessary as no support provided</td>
</tr>
<tr>
<td></td>
<td>F4.II</td>
<td>Difficulties were experienced tracing communication related errors</td>
</tr>
<tr>
<td></td>
<td>F5.II</td>
<td>Significant effort was required to implement two solutions to determine which decomposition strategy would better scale</td>
</tr>
</tbody>
</table>

Other findings not directly relevant to the task of data decomposition but insightful to the parallelisation process as a whole were also recorded as follows:

- Before a program is parallelised, effort can be applied to optimise the serial codebase such as performing loop unrolling. However, such optimisations can make the potential decomposition of the data structures less clear for an MPI implementation.

- When parallelising code, it was advised to get the easiest (even if not optimal) parallel version working and plan to optimise the working version. As the process of parallelising an application is similar to ‘crossing a desert’, as was described by a participant in Stage I, getting an initial version parallelised correctly provides a testable working base.

- When writing MPI calls for complex applications, it is useful to use a wrapper class to encapsulate the MPI operations that will be invoked rather than weaving the MPI operations through the codebase.

- A compiler flag which switches the application from serial to parallelised mode is a helpful mechanism to implement, so that the parallelised code
can be run in serial at any point, providing a basic test to ensure correctness is maintained.

The participant-observer study allowed the researcher to perceive reality from inside a HPC environment rather than as someone external to it (Yin 2003, p. 94). The experience provided insights the researcher would not have gained through interviews with experts alone. This added knowledge was utilised in Stage III where the predefined questions posed in the focus group session were more relevant to real problems experienced by practitioners in a HPC setting.

A participant-observer study however, has limitations and can result in bias when evaluating an issue the researcher has a vested interest in (Kitchenham et al. 2010). As this study was mentor-driven by experienced programmers in HPC, this bias was minimised as these experts guided the participants on the steps taken to port the software. Another limitation relates to the time spent ‘participating’ in the study which can negatively impact the time spent ‘observing’ (Yin 2003, p. 94). The study was carried out in conjunction with two peer researchers and all observations were frequently discussed in a group setting in order to seek clarification about experiences that may not have been fully observed by a solo researcher.

The external validity of a study is concerned with assessing if the results from this parallel port can be applicable to another parallel port (Yin 2003, p. 37). The researchers enquired if the issues encountered were those that expert parallel programmers encounter. The mentors confirmed that the issues experienced by the researchers were those they experience when porting applications to a HPC environment. While this is a limited study, this confirmation goes towards stronger external validity.

4.4 Stage III: Focus Group Study

In Stage I, data decomposition emerged as a real challenge experienced by practitioners and the difficulties involved were experienced first hand by the researcher
4.4 Stage III: Focus Group Study

in Stage II. The objective of Stage III of the research was to gain insights into the core issues concerning parallel developers when performing data decomposition. Tool support options available to assist this task were also explored.

4.4.1 Research Design

A focus group study was chosen in this stage of the research. Due to their interactive nature, focus groups can encourage and prompt participants to react to points during discussion often leading to the discovery of issues that were unknown to the researchers in advance (Kontio et al. 2008). Participants in IBM’s HPC centre in Dublin were approached to take part in this study which aimed to specifically explore the area of data decomposition and communication in message passing applications. Three experts\(^2\) in parallel development were involved in the focus group session which lasted approximately two hours. The researcher acted as a moderator and introduced the the goals of the research to the participants before conducting the session. All participants were assured that the findings would be reported anonymously. The moderator facilitated discussion using predefined questions that provided the opportunity for open discussion, (see Appendix B), but did not allow her opinions or experiences from Stages I or II to influence the discussion. Table 4.5 lists title and background experience of these participants. Participants are numbered so as to protect their privacy (e.g. P1). A second round of interviews was subsequently carried out where two thirty-minute

\(^2\)One had partaken in the interviews conducted in Stage I.

<table>
<thead>
<tr>
<th>ID</th>
<th>Current Title</th>
<th>Experience in Computer Science</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Senior software architect</td>
<td>21 years (5 years HPC)</td>
</tr>
<tr>
<td>P2</td>
<td>HPC specialist</td>
<td>4 years (3 years HPC)</td>
</tr>
<tr>
<td>P3</td>
<td>Software engineer</td>
<td>4 years (4 years HPC)</td>
</tr>
</tbody>
</table>
semi-structured interviews were conducted based on questions derived from the findings of the focus group. This allowed the researchers to clarify issues raised in the focus group session and ask more detailed questions. Another objective was to gain insights into potential tooling support the participants would deem useful. All interviews were conducted by at least two researchers (all three researchers were present in the focus group session) and all sessions were digitally recorded with the participants’ consent. The interviews were transcribed verbatim in order to allow for in-depth analysis and were analysed using qualitative data analysis methods (Seaman 1999). All interview transcripts were thoroughly read and phrases of particular interest were coded with labels to reflect the topic of that phrase. As this analysis was carried out by a research team of three, independent researchers analysed certain sections of the transcripts and their interpretations were then aggregated and discussed among the team.

Three tooling proposals to address the challenges surrounding data decomposition were derived from the empirical data gathered or directly proffered by the participants in this study. These tooling proposals are hypothetical in nature and will be referred to as TP1-TP3. All findings go towards a broader characterisation of properties of the challenge of data decomposition and the associated needs of practitioners.

4.4.2 Findings

The distribution and management of large data structures was discussed in the context of parallelisation projects and in particular, the participants spoke of the bottlenecks experienced when decomposing the data and performing MPI exchanges. Certain complex codebases (a molecular dynamics application was provided as an example) were described as a ‘nightmare to decompose’ due to the limitations of available tool support when implementing a chosen data decomposition strategy. Participant P3 expressed his need for lightweight tool support when mapping data structures:
4.4 Stage III: Focus Group Study

If you have a simple tool that helps you to map what you want to do into whatever language you are doing or data structure you want to have in the program . . . would save a lot of time.

To implement a send and receive operation in MPI, the parallel programmer must explicitly define the data to be exchanged, which involves calculation of the array indices. Deciphering these index locations was reported as a tedious process, often involving the manual counting of elements in an array. Frustration was apparent when the participants described this process as Participant P3 pointed out:

*The amount of times we sat there counting indices and rows was incredible . . . If I can type in, my data structure is called this and my indices are called this, my global index is this and my local one is this, it should tell you that your loop should be 0 to nx to send this row to here. Simple things like that would save a LOT of time.*

Participant P2 also acknowledged the lack of support when dealing with data and arrays:

*There is nothing to tell you how the data is offset in the global array.*

A tooling proposal to address this challenge is derived from the empirical evidence presented and will be referred to as TP1. The aim of TP1 is to provide support for automated indexing, as well as a notation (e.g., a diagram) to facilitate experts to specify indices. For example, if the programmer wished to decompose a grid-based solution using a row topology, the user could express this through a notation and the tool would be able to map the data structures appropriately and provide the user with this mapping. The programmer would then be free to use these predefined structures in their MPI operations rather than explicitly stating the index locations. The participants were queried about any known abstractions to assist the coding of message exchanges; no abstractions were acknowledged.

Participant P2 spoke of the repercussions of incorrectly exchanging data:
Chapter 4. Empirical Characterisation

There is nothing to tell you if you do something and get your exchange wrong.

Participant P3 discussed the benefits of a tool which could not only map the data structures, but also generate the MPI calls required to exchange the data:

When implementing MPI, if I can get some aid to help me manage what I want to happen, and what MPI calls are necessary to do it, anything that helps you see the calls you need to do at a low level would be helpful.

While the derived tooling proposal TP1, could abstract the creation of data structures, encapsulating the array indices, the parallel programmer would still be required to code MPI operations. This problem is a driving factor behind the derivation of a second tooling proposal — TP2, the aim of which is to provide an easy interface to not only specify the decomposition strategy, but also to offer a notation to outline the required communication. The programmer would specify the communication necessary to implement the required parallelisation, providing this information as input to the tool. The tool would then auto-generate MPI functionality based on this input. The programmer could then invoke this MPI functionality in their serial code resulting in a parallelised version.

Once a decomposition strategy is implemented, determining its efficiency is a typical procedure. Participant P1 referred to an optimal decomposition strategy as:

the best bang for the buck which will get you the extra couple of [performance] percent.

However, the participants indicated various issues associated with finding an efficient decomposition, characterising the process as very ad-hoc; participant P3 explained:

What people do is try a block distribution, benchmark and see what you get, then try a cyclic and see what you get but it’s hit and miss, more miss than hit.
4.4 Stage III: Focus Group Study

The participants spoke of the tool MPITrace which they use to monitor the MPI calls sent to and received from processors during execution. However, the data returned from such profiling tools outline the calls a parallel program executes, but do not explain the underlying reasoning for particular calls. This is left to the programmer to decipher:

*You have MPITrace data ... you just see a list of reduce, barrier, sum, this and that ... you've no idea if it is reducing a grid or exchanging sides here and here ... to understand what it is doing, you have to go into the code, read through routines and see what the MPI is doing and what data is moving around.* —P3

An optimal decomposition, while yielding a large return in performance, can be very time consuming to determine, as the programmer is required to re-write the parallel implementation for each potential strategy:

*Maybe you're doing some reconfiguring of the decomposition because you'd prefer different results, it takes some time.* —P1

While TP1 maps data structures and TP2 generates the code, the programmer still does not have support to determine whether the chosen decomposition strategy is optimal for his/her application. A third tooling proposal TP3, is therefore derived to assist the process of evaluating decomposition strategies to assess for optimality. This proposal could be used by the programmer to profile, inspect and visualise alternative decomposition strategies and their consequences (e.g. dependencies created). This would facilitate a practical approach to evaluate the application’s efficiency (e.g., in terms of performance) for different decomposition topologies.

TP1-TP3 are empirically derived light-weight tooling proposals which centre on hypothetical abstractions that would target the bottlenecks encountered when performing data decomposition and communication in the parallelisation process. Table 4.6 summarises these proposals.
Table 4.6: Summary of Tooling Proposals

<table>
<thead>
<tr>
<th>Tooling Proposal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP1</td>
<td>Map the decomposition a developer intends to implement and to create the data structures to carry this out.</td>
</tr>
<tr>
<td>TP2</td>
<td>Generate MPI in order to implement the communication calls based on the decomposition.</td>
</tr>
<tr>
<td>TP3</td>
<td>Assist the selection of an optimal decomposition strategy.</td>
</tr>
</tbody>
</table>

4.4.3 Limitations

Focus group studies often produce biased results due to group dynamics (Kontio et al. 2008). The researcher attempted to balance the discussion by activating the less active participants where possible. Social acceptability is another limitation associated with focus groups (Kontio et al. 2008). A participant may be shy or too embarrassed to volunteer information that may be deemed incorrect by another participant. The researcher again tried to mitigate this problem by conducting follow up interviews to seek clarification where a participant did not elaborate sufficiently on a point. Secrecy can also limit focus group sessions where relevant information can often be withheld due to proprietary reasons (Kontio et al. 2008). This situation was avoided by guaranteeing the anonymity and confidentiality of any results that could reveal proprietary details and by allowing the participants the right-to-edit of any dissemination/publication derived from the sessions. The short time allocated for the focus group (two hours) may be considered a limitation as complex issues raised by the participants may not be understood by the researchers (Kontio et al. 2008). As the participant-observer study was conducted prior to the focus group session, the researcher had some experience in the field of HPC and so did not approach the session as a complete novice. Also, the follow-on interviews helped to clarify any misunderstood issues raised after a period of reflection.
4.5 Stage IV: Survey Study

The number of participants may also be considered a limitation in this study, however, the focus group study is conducted in conjunction with the other research methods (Stages I, II and IV) and so is not the principal method in the research. An initial study report was sent to the participants of the focus group as a measure to ensure the findings (encapsulated in the tooling proposals TP1-TP3) were correctly captured and interpreted. This is a form of member checking which is a tactic used to assess a study’s validity (Runeson et al. 2012, p. 73). The participants confirmed the findings as accurate.

4.5 Stage IV: Survey Study

This study involved a survey targeting experts in the field to gain insights into the problems of data decomposition experienced by participants across a broader population. Specifically it aimed to generalise the findings obtained in Stages I and II: that data decomposition and communication are difficult tasks, lacking tool support. It also aimed to assess the prevalence of these tasks in parallelisation. Finally, it aimed to assess representative scenarios where data decomposition posed a core challenge, in order to guide potential tool developers towards those scenarios.

4.5.1 Research Design

The survey was presented as 13 questions divided into three sections. The first section gathered information regarding the profile of the participants including their experience in a HPC context (3 questions). The second section related to general parallelisation experience and collected data related to past projects parallelised (4 questions). The third focused on data decomposition specifically (6 questions). Appendix C presents a complete listing of these questions. The survey was compiled using the online tool SurveyMonkey and circulated through avenues such as the STEM-Trek webpage (StemTrek 2013) and LinkedIn HPC forums. Purposive and snowball sampling (Oates 2006) were both used to choose these
Chapter 4. Empirical Characterisation

Table 4.7: Participant Experience for Stage IV

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industrial experience (years)</td>
<td>3.16</td>
<td>0.00</td>
<td>16.00</td>
</tr>
<tr>
<td>Academic experience (years)</td>
<td>5.52</td>
<td>0.00</td>
<td>27.00</td>
</tr>
<tr>
<td>Combined experience (years)</td>
<td>8.67</td>
<td>1.00</td>
<td>27.00</td>
</tr>
<tr>
<td>No. of applications parallelised</td>
<td>7.90</td>
<td>4.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Typical duration of parallelisation projects (months)</td>
<td>7.91</td>
<td>0.50</td>
<td>36.00</td>
</tr>
</tbody>
</table>

populations. Purposive sampling occurs when the researcher deliberately picks the sample from people likely to produce the data. The targeted participants were those working or researching in HPC environments such as US-XSEDE (Extreme Science and Engineering Discovery Environment) and EU-PRACE (Partnership for Advanced Computing in Europe). A method of snowball sampling was used where people in these fields passed on the survey link to those who they expected to be also relevant to the research area.

The summary of the participants’ experience (gathered from the first section of the survey) is outlined in Table 4.7. In all, 32 participants completed the survey. Of these, 26 had industrial experience. Six participants had academic experience only, but all had significant experience in parallelising applications (10 applications on average). Therefore, it can be stated with confidence that the academic participants also have extensive knowledge of this domain, possibly similar to that of the industry participants.

4.5.2 Findings

The findings gathered from the second section of the survey are elaborated on in the next sub-section (General Parallelisation Findings), followed by the findings gathered from the third section of the survey where the focus was on decomposition (Decomposition Findings).
4.5 Stage IV: Survey

*General Parallelisation Findings:* Figure 4.4 illustrates the breakdown of the past largest five projects the participants worked on. As can be seen, pure MPI based projects (using MPI as the only parallelisation technology), totalled the highest number at 31.51%. Pure MPI combined with a hybrid of MPI and other technologies (OpenMP+MPI, CUDA+MPI) totalled 52.05% of all listed projects. Pure OpenMP (using OpenMP as the only parallelisation technology), totalled 14.38% while pure OpenMP combined with a hybrid of OpenMP and other technologies (OpenMP+MPI, OpenMP+CUDA) amounted to 32.87%. Heterogeneous projects totalled 10.96%, however, other options such as a hybrid involving CUDA also featured. PGAS projects amounted to less than 1%, while the *Other* category in the survey question amounted to 19.18% which featured unlisted technology options such as CUDA+MPI+OpenMPI, MATLAB, and PGAS using Global Arrays toolkit.

The participants were asked to list the grid structures encountered when parallelising message passing applications. Figure 4.5 depicts the percentage of participants who listed using *Rectangular* (78%), *Adaptive* (31%) and *Irregular* grids (44%) when parallelising two or three dimensional grid structures. The data

![Figure 4.4: Breakdown of Project Type](image-url)
were further broken down into the frequency of use. Fifty-three percent of the participants claimed structured rectangular grids constituted at least 50% of all message passing applications. Thirty-eight percent of the participants put this figure at over 70% of all their message passing applications. Thirty-four percent of the participants reported using irregular grids in a maximum of 30% of their projects, while 56% of the participants did not use irregular grids at all. With regards to adaptive grids, 31% reported working with these in a maximum of 30% of applications. Sixty-nine percent of participants did not work with adaptive grids at all. The participants listed using a number of other structures, such as Yin-Yang, butterfly (two circles connecting in a common point), and spectral (wave-space). The data gathered from this survey show that MPI based projects with data centred on structured rectangular grids represent a large proportion of parallelisation scenarios.

**Decomposition Findings:** As was reported in Chapter 2, decomposition can be broken down into two core areas: data and task. The prevalence of decomposition type across projects was questioned in the survey and the findings show that 49.82% of decomposition is based on distributing the data (see Figure 4.6). The remainder 50.18% of projects is based on task decomposition as well as a hybrid

![Breakdown of Grid Type](image_url)

**Figure 4.5:** Breakdown of Grid Type
of task and data decomposition. In total, data decomposition (be it hybrid or pure) occurs in approximately 75% of parallelisation projects.

Figure 4.6: Breakdown of Decomposition Type

The parallelisation process can be divided into the following separate phases as identified in Chapter 2:

- Profiling (pre-parallelisation)
- Task Decomposition
- Data Decomposition
- Communication Design
- Performance Characterisation (post-parallelisation)

The participants were asked to rate the difficulty of these phases in the parallelisation process, using a five-level Likert scale, from very easy to very difficult. Figure 4.7 presents the results. Data decomposition was rated as most difficult. This is consistent with the findings in Stages II and III of the study. Performance characterisation, communication and task decomposition were next on the difficulty rating. Profiling was considered to be less problematic with a neutral score. Respondents were asked to estimate the time and effort spent performing data decomposition in the parallelisation process. On average, programmers reported spending 40.38% of time and 40.59% of effort performing data decomposition, showing that not only is it a difficult activity but that it also consumes a lot of their effort.
When asked if and what tools or media the participants used to assist data decomposition, 72% of respondents answered that no tools were used. Those who answered Yes, listed ‘pen and paper,’ ‘profiling to test results’ and ‘my imagination’ as the tools used. The latter response particularly reflects the exasperation experienced. Some participants claimed they used custom tools suggesting the lack of alternatives; the following quote also indicates the challenge associated with accurate indexing:

*Usually custom tools; Matlab scripts to plan out layouts; custom tests (not quite unit tests) to make sure indexing is working.*

To investigate the participants’ cognisance of any potential support (even if not adopted), they were queried regarding their awareness of existent tools or media that could facilitate data decomposition. From the set of respondents, 88% were not aware of any tools. One participant elaborated by stating:

*There is no tool that really helps any parallel programmer to effectively understand how to decompose data. Most of the experience comes from every application specific domain.*
4.5 Stage IV: Survey

Two participants who answered ‘yes’ listed ParMetis, a graph partitioning tool (ParMETIS 2013), as the only tool they were aware of. Another participant answered ‘yes’, followed by the comment ‘but not useful’ and a fourth participant answered ‘yes’ but did not elaborate. Following this question, the participants were asked whether they use any notations during the data decomposition phase; the consensus was that no formal notations are in use. However, participants indicated a number of informal means to help them in the decomposition activity:

- Hand-made diagrams.
- Drawings of the decomposed domain to identify overlapping regions.
- Diagrams of 2D/3D data blocks with communication halos marked.
- Representations as provided by customers (scientists), e.g., matrix, grids.
- Colour-coding sections of a grid based on process number.

The participants were asked what considerations they deemed important during data decomposition. The characteristics presented are outlined in Figure 4.8 and are based on the initial interviews in Stage I of this study where the participants spoke of challenges encountered when parallelising applications. As the question was explicitly asked in relation to data decomposition, scalability refers to the scalability opportunities achieved by choosing an efficient decomposition strategy. Ease of programming refers to the extent to which the code can be programmed to achieve data decomposition with efficiency and satisfaction. Communication overheads refers to the potential reduction in performance due to the overhead caused by the setting up and subsequent sending and receiving of messages between processors. The findings showed all three considerations were somewhat to very important. A participant remarked that no single consideration is most important for a good data decomposition:

*I rank everything as a top priority. I guess the point is that when jumping a canyon one needs to get 100% of the way across, 20%, 50%, and 90% just don’t cut it.*
This reinforces the sentiment expressed by Massingill et al. (2007) who argued it is a balance of competing forces. Table 4.8 provides a summary of the key findings from this survey referenced as F1_IV - F8_IV. The relevance of the tooling proposals derived in Stage III to these findings is described.
Table 4.8: Summary of Findings in Stage IV

<table>
<thead>
<tr>
<th>Theme</th>
<th>Ref.</th>
<th>Finding</th>
<th>Relevance of Tooling Proposals (T1-T3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prevalence of data decomposition</td>
<td>F1_IV</td>
<td>Data decomposition (be it hybrid or pure) occurs in approximately 75% of parallelisation projects.</td>
<td>TP1 - TP3 indicate the task of data decomposition needs support.</td>
</tr>
<tr>
<td>Challenge of data decomposition</td>
<td>F2_IV</td>
<td>Data decomposition was rated as most difficult from a list of parallelisation phases.</td>
<td>TP1 - TP2 suggest the need to assist the developer when performing data decomposition.</td>
</tr>
<tr>
<td></td>
<td>F3_IV</td>
<td>Programmers reported spending approximately 40% of time and effort performing data decomposition.</td>
<td>TP1 - TP3 promote the reduction of time and effort.</td>
</tr>
<tr>
<td>Limitations of tool support for data decomposition</td>
<td>F4_IV</td>
<td>No formal tools are used by the participants to assist data decomposition. Custom tests used to check indexing.</td>
<td>TP1 suggests the need for a notation to map what the developer wants to do.</td>
</tr>
<tr>
<td></td>
<td>F5_IV</td>
<td>Notations are seldom used and when used, are predominantly informal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F6_IV</td>
<td>Majority of participants (88%) are unaware of existent tooling support.</td>
<td></td>
</tr>
<tr>
<td>Tooling Considerations</td>
<td>F7_IV</td>
<td>Scalability, management of communication overheads and ease of programming all ranked closely as important issues to consider in tooling support.</td>
<td>TP1 and TP2 advocate ease of programming. TP3 indicates the need to support the selection of data decomposition strategy for scalability reasons.</td>
</tr>
<tr>
<td></td>
<td>F8_IV</td>
<td>MPI based projects (52%) with data centred on rectangular grids (78%) represent a large proportion of parallelisation scenarios.</td>
<td>TP2 suggests the need to generate MPI code.</td>
</tr>
</tbody>
</table>
4.5.3 Threats to Validity

Although the survey is based on the experiences of a significant number of experienced HPC programmers, certain precautions were taken to increase its external validity. To eliminate one element of bias, the options in multiple choice questions appeared in a randomised format to minimise any influencing orderings. The survey was trialled in a pilot study with two colleague researchers\(^4\) to check that the questions were understandable before distribution. Questions were rephrased based on the data collected in this pilot study. The first section of the survey collected profiling data as it was crucial to ensure only respondents with sufficient knowledge completed the survey in full (Kitchenham and Pfleeger 2008). It was after Question 4 (\textit{From the five largest parallelisation projects you have worked on, please state the HPC technology used per project}) that 13 of the 45 initial sample discontinued their submission. This resulted in 32 fully completed surveys, suggesting that only experienced respondents continued to submit information. The preliminary results of the survey were also distributed to the participants who had requested feedback (in the survey questionnaire) as a form of member checking.

4.6 Derived Requirements

Stages I-IV represent what Wieringa et al. (2006) referred to as the \textit{problem investigation} phase. Within the requirements engineering field, this phase is concerned with gaining an understanding of the problem. Other phases are concerned with proposing and validating solutions. Stages I-IV provide evidence that the challenge of data decomposition is very real for many HPC experts. The difficulty associated is particularly evidential in the informal sketches captured in the participant observer study (F3.II) as well as the ranking of \textit{most difficult phase} furnished by the participants in the survey (F2.IV). Given that respondents stated that 40% of the parallelisation process is consumed by this phase (F3.IV), it

\(^4\)Members of the research centre but not associated with the project group.
is remarkable that available tools have gained little traction in industry (F4_ IV - F6_ IV).

Requirements elicitation involves activities which result in the identification of a number of requirements which a proposed software solution must satisfy (Cheng and Atlee 2007). Stages I-IV described above applied a subset of the requirements elicitation techniques outlined in the classification devised by Nuseibeh and Easterbrook (2000):

- **Traditional** - this involves using generic data-gathering techniques such as interview and surveys.
- **Group elicitation** - this involves the exploitation of group dynamics using techniques such as focus group studies.
- **Prototyping** - this involves building a prototype in order to receive feedback from stakeholders.
- **Model-driven** - this technique requires the use of a model to represent the information to be gathered.
- **Cognitive** - this involves using knowledge-based techniques such as recording participants as they speak their thoughts whilst conducting tasks.
- **Contextual** - this involves the use of ethnographic-style studies such as participant-observer.

Stage I involved exploratory interviews, thus employing a *traditional* technique. Stage II centred on a participant-observer study and so is considered a *contextual* technique. Stage III involved the use of a focus group interview and so applied a *group elicitation* technique. Stage IV again employed a *traditional* technique through a survey.

A prototype tool was designed and implemented in a later phase of this research as presented in Chapter 6. However, this was not created to elicit any additional requirements and so does not represent the *prototyping* technique as envisaged by Nuseibeh and Easterbrook (2000). As the requirements sought were on a high-level, a model was not considered necessary to represent the information and so *model-driven* techniques were not utilised. *Contextual* techniques are
described as an alternative to cognitive techniques (Nuseibeh and Easterbrook 2000). A contextual technique in the form of a participant-observer study, was chosen as an alternative to a cognitive technique, as the three month placement in IBM was deemed adequate exposure to the attitudes and thoughts of practitioners.

Nuseibeh and Easterbrook (2000) suggest that the use of different types of techniques can be complementary. In order to ensure that the problems surrounding data decomposition were investigated thoroughly and a representative list of requirements for future tool support were derived, the researcher applied a combination of three different techniques, which can be considered a form of triangulation. These requirements, which are referred to as R1-R6 (not ranked in any manner), are as follows:

- **R1: MPI Focused.** The findings of the exploratory study in Stage I (Section 4.2.2) and the survey in Stage IV (Section 4.5.2) clearly indicate that MPI is one of the most used technologies in parallelisation projects. Given that MPI is so prevalent throughout industry and HPC practitioners are slow to deviate from trusted technologies (Basili et al. 2008), there is a clear need for more and better tool support for this technology. The MPI-focus of the tooling proposal TP2, derived from the empirical evidence in Stage III (Section 4.4.2), provides further evidence that this is a necessary requirement. If MPI code is generated by a tool, such code will be maintainable by HPC practitioners.

- **R2: Automated Grid Indexing.** The tooling proposal TP1 in Stage III of this study (Section 4.4.2), advocates the automation of indexing when calculating the communication calls associated with data decomposition. The survey findings (Section 4.5.2) show that developers use manual drawings to determine the necessary low level details reflecting the researcher’s experience in the participant-observer study (Section 4.3.3). These results suggest that the effort involved in manually calculating indices and sketching grid diagrams is a core issue for programmers in this field.
4.6 Derived Requirements

- **R3: Data Structure & Communication Generation.** The tooling proposal TP2 from Stage III of this study (Section 4.4.2) focused on providing assistance when coding message passing communication calls and their required data structures. As *Management of Communication Overheads* was cited as an important characteristic when performing data decomposition in Stage IV (Section 4.5.2) ensuring both accuracy and efficiency is crucial when writing MPI calls. Keeping track of the data structures and devising the communication routines were also reported as bottlenecks in Stage II (Section 4.3.3). This suggests the need to facilitate a higher level of abstraction when **generating the data structures and communication calls required.**

- **R4: Ease of Strategy Conversion.** Findings from Stage I of this study (Section 4.2.2) identified that trial and error methods were employed when deciding the most suitable decomposition strategy and that the effort required to convert one decomposition scheme to another (e.g., from a Cartesian to a row-based) is significant. This effort was experienced by the researcher when participating in the port described in Stage II (Section 4.3.3), as two different implementations of the application were coded to reflect two decomposition strategies. TP3, a tooling proposal derived in Stage III of this research (Section 4.4.2), focused on this challenge reiterating the need to **minimise developer effort when implementing alternative decomposition designs.**

- **R5: Low Application Code Impact.** Ease of programming was listed as an important consideration in Stage IV (Section 4.5.2) when performing data decomposition. This characteristic provides empirical evidence for a tool which would **not require invasive rewriting of the serial code to achieve data decomposition**, i.e. a tool that would have a low application code impact. As HPC applications have a long life span, developers may be reluctant to rewrite any of their code in an unfamiliar language (Basili
et al. 2008). Implementing in a new language or API can thus act as a deterrent for tool adoption and is looked on negatively with regards to future maintenance. Tool support requiring few code modifications would have a greater chance of adoption.

- **R6: Targeting Structured Grids.** As the results in the survey in Stage IV (Section 4.5.2) presented rectangular grids as the dominant structure in parallelisation scenarios, tool support should initially be focused on applications providing solutions for problems modelled on structured grids.

These requirements are summarised in Table 4.9 and the stages/findings which provided empirical justification are also listed.
4.7 Conclusion

Table 4.9: Summary of Derived Requirements

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Requirement</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>MPI focused</td>
<td>• Findings from Stage I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage III - TP2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage IV - F8.IV</td>
</tr>
<tr>
<td>R2</td>
<td>Automated grid indexing</td>
<td>• Findings from Stage II - F1.II</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage III - TP1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage IV - F4.IV</td>
</tr>
<tr>
<td>R3</td>
<td>Data structure &amp; communication generation</td>
<td>• Findings from Stage II - F2.II, F4.II</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage III - TP2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage IV - F3.IV, F7.IV</td>
</tr>
<tr>
<td>R4</td>
<td>Ease of strategy conversion</td>
<td>• Findings from Stage I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage II - F5.II</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage III - TP3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Findings from Stage IV- F7.IV</td>
</tr>
<tr>
<td>R5</td>
<td>Low application code impact</td>
<td>• Findings from Stage IV - F7.IV</td>
</tr>
<tr>
<td>R6</td>
<td>Targeting structured grids</td>
<td>• Findings from Stage IV - F8.IV</td>
</tr>
</tbody>
</table>

4.7 Conclusion

Research Question RQ1 (restated in Section 4.1) is answered in all four stages of this study. The reports of practitioners in Stage I and the experience of the researcher in Stage II provide evidence to support the reality of the challenge of data decomposition. The properties associated with this challenge were explored in more detail in Stages III and IV where the participants described data decomposition as the most difficult phase of the parallelisation process, consum-
ing approximately 40% of time and effort. RQ2 was addressed in both the focus group and the survey studies, as practitioners were explicitly asked what tooling support they use to assist their efforts. A key finding was that the participants do not use tooling to assist this phase of the parallelisation process and instead resort to manual efforts. Of the survey respondents, the majority were unaware of available tooling support for decomposing data in grid based applications.

A list of requirements for potential tool support was derived in Section 4.6 and are outlined in Table 4.9. These requirements were based on data gathered in all four stages described in this chapter and go towards answering the research question RQ3. The next stage of this research involves a thorough investigation of available tooling support and an assessment based on the derived requirements. This investigation is described in Chapter 5. Chapter 6 outlines the design and implementation of a tool which meets these requirements.
5.1 Introduction

Practitioners in the HPC field were surveyed regarding the tooling available and support used when performing data decomposition. A key finding was that tooling support was not used by the experts and in fact, a majority of the participants were unaware of existing offerings. Within the state-of-the-art, several approaches are documented to abstract the task of performing decomposition and communication when parallelising applications. This chapter explores these offerings and presents a critique using the derived requirements as a comparison framework. The following research question, previously outlined in Chapter 3, is addressed in this chapter:

**RQ4:** What tooling support exists to assist the task of data decomposition and communication?
5.1 Introduction

Six requirements for tool support were described in Chapter 4 and so the following sub-question is defined to assess the level of support offered for these:

**RQ4.1:** Does the currently offered tooling support meet the derived requirements outlined in RQ3?

To address these research questions, the researcher investigated support available in the state-of-the-art and organised the findings under four approaches as follows:

- Pattern-based (Section 5.2)
- Domain Specific Languages (Section 5.3)
- Generic Template (Section 5.4)
- Problem Solving Environments (Section 5.5)

An *Other* category is provided to incorporate any tools not associated with the listed approaches, one tool fits in this category and is discussed in Section 5.6. All described tooling options require the user to reengineer the serial codebase using libraries or API to create the parallel version. Approaches which do provide abstractions, but involve a complete rewrite of the serial code include High Performance Fortran (HPF) (Kennedy et al. 2007) and languages designed for Partitioned Global Address Space (PGAS) memory models such as those created under the the High Productivity Computing Systems (HPCS) initiative from DARPA ((Dongarra et al. 2008)). As these require the user to rewrite their code completely rather than a refactoring step, they are more suitable to parallel applications written from scratch in greenfield development, rather than those evolving from legacy systems. For this reason, these approaches are beyond the scope of this research and not included.

An overview of the described tooling options is provided in Table 5.1, which classifies each option and lists the supported languages. To put the accessibility of the tools in context, the current availability for public download is listed, as well as the year of the last academic publication regarding the development and/or use of the tool. Section 3.7 provides an overview of the digital libraries
Table 5.1: Overview of Tools

<table>
<thead>
<tr>
<th>Appr.</th>
<th>Tool</th>
<th>Last known publication</th>
<th>Language support</th>
<th>Avail. for download</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>PASM</td>
<td>Akon et al. (2005)</td>
<td>C++</td>
<td>✗</td>
<td>Extended to support the design of skeletons.</td>
</tr>
<tr>
<td>DSL</td>
<td>Promoter</td>
<td>Besch et al. (1998)</td>
<td>C++</td>
<td>✗</td>
<td>C++ was standardised during the project’s timeframe.</td>
</tr>
<tr>
<td></td>
<td>Hi-PaL</td>
<td>Arora et al. (2013)</td>
<td>C/C++</td>
<td>✗</td>
<td>Extended to include command-line interface.</td>
</tr>
<tr>
<td></td>
<td>TACO</td>
<td>Rotta et al. (2012)</td>
<td>C++</td>
<td>✓</td>
<td>(TACO 2012)</td>
</tr>
<tr>
<td>PSE</td>
<td>POOMA</td>
<td>Cummings et al. (1999)</td>
<td>C++</td>
<td>✓**</td>
<td>FreePOOMA is based on POOMA. (FreePOOMA. 2004).</td>
</tr>
<tr>
<td></td>
<td>PETSc</td>
<td>Balay et al. (2013)</td>
<td>C/C++/Fortran</td>
<td>✓</td>
<td>(PETSc 2013)</td>
</tr>
<tr>
<td>Other</td>
<td>DDTool</td>
<td>Leng et al. (2001)</td>
<td>C</td>
<td>✓**</td>
<td>Project has been suspended. (Vipar 2002)</td>
</tr>
</tbody>
</table>

**Not compilable due to outdated software dependencies.

used in the literature search. Each tooling option is discussed in the following sections using varying examples demonstrating the decomposition of a grid-based application and the steps necessary to perform data exchanges between neighbouring processes. An analysis of the support provided by each tool is also presented with respect to the following derived requirements as outlined in Chapter 4:

- R1: MPI focused.
5.2 Pattern-Based Approaches

- **R2**: Automated grid indexing.
- **R3**: Data structure and communication generation.
- **R4**: Ease of strategy conversion.
- **R5**: Low application code impact.
- **R6**: Targeting structured grids.

Additional requirements extracted from the tool review are also taken into consideration and are described in Section 5.8 where a comparative analysis across the ten tools is tabulated.

5.2 Pattern-Based Approaches

A design pattern is described by Buschmann et al. (1996, p. 5) as that which:

*addresses a recurring design problem that arises in specific design situations and presents a solution to it.*

In the parallel computing domain, design patterns incorporate both design and implementation patterns and often consist of data parallel patterns (Goswami et al. 2002a). Keutzer et al. (2010) describe a pattern-based language which they initially named *Pattern Language for Parallel Programming* (PLPP), and later evolved to become *Our Pattern Language* (OPL). OPL consists of layers corresponding to the steps involved when mapping a problem to a parallel computation in order to run in a HPC environment. The patterns are ordered in categories such as *Structural and computational* where a structured grid is a provided example, *Parallel algorithm strategies* such as geometric decomposition (discussed in Section 2.5.2), *Implementation strategies* such as distributed array and *Parallel execution* such as Single Instruction Multiple Data (SIMD) devised by Flynn (1972). While this pattern language is useful for a software developer to make decisions regarding the parallelisation process, it does not provide methods or techniques to describe how to perform geometric decomposition or execute a particular implementation strategy. To this end, the pattern-based approaches of interest to this research are
those which abstract decomposition and communication tasks when parallelising grid based applications to run in message passing environments. The following subsections (5.2.1 and 5.2.2) outline two pattern-based techniques.

### 5.2.1 PASM

The Parallel Architectural Skeleton Model (PASM) (Goswami et al. 2002b) is a pattern-based system which targets the message-passing paradigm and hides the low-level details of decomposition and communication. PASM is implemented as a C++ template library consisting of skeletons which the user must extend by specifying application-dependent parameters. To parallelise grid based applications with consideration for boundary conditions, a data-parallel class (DataParallelSkeleton) can be extended.

An optional textual interface is provided by PASM, which the user can use instead of C++ as it abstracts certain steps in the development process. This is parsed and expanded to produce front-end C++ code that is then compiled and linked with the skeleton library to produce the executable. The examples in Figures 5.1 and 5.2 illustrate the use of this textual interface. Line 6 in Figure 5.1 defines a module called FrontEnd which extends the PASM data parallel skeleton class. Protocols are chosen for internal and external communication within a module. The DataParallelSkeleton supports different protocols.

```c++
5 // Front-end of the 1-D mesh
6 FrontEnd EXTENDS DataParallelSkeleton
7 {
8     CHILDREN=MeshElement;
9     PROTOCOL=PROT_1DMesh;
10
11       Rep{
12           ....
13           int N=SetMeshWidth(4); // A member of PROT_1D Mesh
14           Grid A(1000,1000); // A 1000 x 1000 grid
15           ReadIn(A);
16           PartitionGrid(A,N); // Partition the grid row-wise
17           // among N=4 children
18           ....
19       }
```

**Figure 5.1:** PASM Example 1
and PROT_1DMesh is explicitly chosen as seen on line 9. This module has associated children of type MeshElement as defined on line 8. Rep (line 11) is the representation of the module and must be filled in with application code such as configuration of the grid and decomposition. In this case, the grid is equally partitioned among the mesh-elements according to the protocol member function SetMeshWidth defined on line 13 and the PartitionGrid function on line 16. Figure 5.2 outlines the MeshElement modules that extend the SingletonSkeleton (line 2) as they do not have their own children. The PROT_1DMesh protocol is chosen by default for these modules and enables external communication among processes. A member function of this protocol is ReceiveFromRep as seen on line 8. The grid A in this case is the distributed chunk of grid. Nearest neighbour communication occurs at the mesh boundaries as can be seen by the PROT_1DMesh member function calls on lines 13 and 14.

While PASM abstracts the data distribution and communication, it does not meet requirement R1, as the MPI implementation is hidden from the user. Requirements R4 and R5 are also not fulfilled as can be seen in the list of derived requirements in Table 5.2.
Table 5.2: PASM - Requirements Analysis

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user. User can add MPI to code.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing, however, these are dependent on the available protocol member functions and more complicated boundary exchanges would need to be written by hand using MPI.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>User required to reengineer serial code and implement extensions of skeletons.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>

5.2.2 MAP₃S

MPI/C Advanced Pattern-based Parallel Programming System (MAP₃S) is a pattern-based development framework used to parallelise applications (Mehta et al. 2006). MAP₃S is based on the Correct Object Oriented Pattern-based Parallel Programming System (CO₂P₃S) which is a parallel programming system for implementing parallel programs in Java (Anvik et al. 2003). The programmer interacts with the system by providing customisation and tuning parameters and the system responds by generating design pattern templates. For parallelising iterative computations in grid based applications, MAP₃S provides a Mesh pattern template. Figure 5.3 illustrates these parameters for CO₂P₃S, on which MAP₃S is based. The user interfaces with the system through a GUI and provides customisation parameters including topology details (cyclic or noncyclic), the number of neighbouring nodes, synchronisation details, dimensions of the grid as well as dimensions of the component blocks of the grid. The system automatically uses these parameters to perform the data decomposition and to determine process neighbours for each of the ghost regions. Tuning parameters can also be input including the maximum size of a packet transmitted between neighbouring pro-
5.2 Pattern-Based Approaches

The system then generates customised template code according to the provided parameters.

To represent elements in the grid, the programmer must define a C data structure. The grid dimensions and the dimensions of the distributed blocks of the grid (i.e. the decomposition parameters) are also defined by the user. The framework then automatically implements the decomposition and allocates a ProcessBlock containing a subset of elements (a chunk of the grid) to each process. Each ProcessBlock contains information to enable communication with distributed blocks. Such information includes its index with respect to other local blocks, its coordinates within the entire grid and information relating to the process it is allocated to. This information is stored in an nb_info structure. Information regarding neighbouring processes such as number of elements to communicate is also stored. The block has a pointer which can be used by the parallel programmer as a reference. The user is required to add their application specific code through preprocessor macros which the system inserts into the proper place in the framework. As the user must be careful not to cause any conflicts with locally scoped variables, these macros can be error-prone. Before any communication occurs, elements of the mesh are packed into MPI packets. The user must also define the macros to perform this packing. The maximum packet size (expressed in bytes) must be expressed before the program executes.
In a more recently added pattern (Simulation pattern), a data-dependency-specification language is used to free the pattern user from being limited to predefining the data dependencies in the graphical user interface (Niewiadomski et al. 2008). Data dependencies are instead set by the user in a pattern-instance specification file using a condition shape-list. This consists of the condition which is a C language expression evaluating to true or false and a shape which is a list of shapes (similar to a stencil) defined by a shape descriptor. Figure 5.4 illustrates this condition shape-list. As the grid in the example is two-dimensional, the shape descriptor uses the variable x for the x axis and the variable y to represent the y axis. The condition includes the expressions $y > x$ and $y \leq x$ and the shape descriptor defines a rectangular area of the grid limited by the specified range boundaries. The system uses this file and determines which chunks of data to assign to which nodes depending on the data dependencies.

The dependency specification file provides more flexibility to the programmer and the communication is abstracted from the pattern user. However, the require-

### Table 5.3: MAP$_3$S - Requirements Analysis

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>As the user must work within the generated code templates, complete reengineering of the original application is necessary.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>
ment to provide low-level details such as the packet size expressed in bytes, lessens the advantages of using an abstracted approach. Table 5.3 provides an overview of the tool’s ability to meet the derived requirements. The MPI implementation is hidden from the user and so Requirement R1 is not met. Requirements R4 and R5 are also not fulfilled.

5.3 DSL Approaches

Domain Specific Languages (DSLs) facilitate the mapping of domain concepts to an implementation and then constructing solutions based on these concepts. A DSL supports only those features that are needed to support its domain (Fowler 2010, p. 28). DSLs can be classified as external or internal. An external DSL has its own custom syntax and requires its own parser generator. An internal DSL uses a general-purpose language as its host. An external DSL gives more freedom of syntax whereas an internal DSL limits the user to the host language’s rules (Fowler 2009). External DSLs exist for a range of domains which are targeted at HPC environments. Liszt is one such DSL used for constructing mesh-based Partial Differential Equation (PDE) solvers which can execute on multiple HPC platforms (DeVito et al. 2011). This research focuses on DSLs whereby the domain is that of parallel programming and so exploration of approaches such as Liszt (domain is numerical methods) is not applicable to this research. Hi-PaL and Promoter are external and internal DSLs respectively and are used to provide abstractions for decomposition and communication when parallelising applications.

5.3.1 Promoter

Promoter (PROgramming MOdel To Enable Real world computing) is an internal DSL designed by Fraunhofer FIRST\(^1\) to abstract data and communication structures in a parallel programming environment (Giloì and Schramm 1993; Giloì et al.

\(^1\)Since July 1st 2012 the institute FIRST works under the name Fraunhofer FOKUS.
Chapter 5. Tool Support

1 topology Grid: 0:M, 0:N {}; // M X N grid
2
topology InnerGrid: Grid {
3  $a:1:(M-1), $b:1:(N-1);
4};  //Interior of the grid
5
Figure 5.5: Promoter Example 1

Promoter is realised as a language extension of C++ and the domain is that of parallel programming. A compiler transforms the Promoter program to a C++ program and incorporates calls to the Promoter Runtime Library (PRL) (Besch et al. 1997). Promoter introduces two specifications of distributed types: a data topology and a communication topology. A topology is defined as an index space. The combination of a data topology and a communication topology provides a description of the grid structure (represented internally as a graph) for a parallel application. The compiler maps the grid structure to the parallel machine and automatically generates any communication between distributed processes. The user is required to specify these topologies through the DSL. The concept of a topology was first introduced by Promoter but has been used in other tools since (see TACO in section 5.4.3).

Figure 5.5 is a code extract outlining a data topology definition for an \(M \times N\) sized grid. The dimensions of Grid are defined on line 1 by the range 0:M, 0:N. Line 3 shows a topology InnerGrid which is a specialisation (contains part) of the topology Grid defined on line 1. The expressions in curly brackets are called constraints of the topology, where all the valid indices for the range are defined. A ’$’ sign in the body of a topology introduces a formal index identifier. As can be seen on line 4, the indices for the InnerGrid are 1:(M-1), 1:(N-1). Distributed types are thus indexable structures and can be indexed by the defined data topology. For example, the statement `Grid<T> g` defines g as a distributed object of class T over the topology Grid. In contrast, a communication topology defines a relation between data points. Figure 5.6 declares a communication

---

\(^2\)This project was part of the Japanese Real World Computing (RWC) partnership which came to an end in 2002.
5.3 DSL Approaches

topology Laplace_5 : Grid, Grid { 
  $a$, $b$, $a+1$, $b$; //right neighbour
  $a$, $b$, $a-1$, $b$; //left neighbour
  $a$, $b$, $a$, $b+1$; //upper neighbour
  $a$, $b$, $a$, $b-1$; //lower neighbour
};

... 

Grid<double> g, h;

h[ [InnerGrid] ] = (g ! Laplace_5);

Figure 5.6: Promoter Example 2

topology (Laplace_5) that supports a five-point discretisation of the Laplace operator on a grid. The neighbours which will be involved in the communication are defined on lines 4 to 7. Lines 12 and 13 perform a transfer and reduction defined by the '!' operator. The expression g!Laplace_5 transfers one data element of g to its corresponding four neighbours of h (the InnerGrid of h). This is how the Promoter DSL performs nearest neighbour exchanges.

The PRL provides mapping strategies such as block and cycle_block (equivalent to cyclic-block) and different distributions are then defined based on the chosen topology and mapping strategy. The corresponding communications are then automatically generated by the compiler. The Promoter Abstract Machine (PAM) is responsible for the communication and synchron-

Table 5.4: Promoter - Requirements Analysis

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>Reengineering of the original application is necessary.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>
isolation but is platform dependent. Table 5.4 provides an analysis of the derived requirements with respect to this DSL, illustrating that Requirements R1, R4 and R5 are not met.

5.3.2 Hi-PaL

High Level Parallelisation Language (Hi-PaL) is an external DSL created for use in the Framework for Synthesising Parallel Applications (FraSPA) (Arora et al. 2010). The aim of this DSL is to reduce the effort involved in writing parallel applications by using a higher level of abstraction than MPI. In this approach, the user is required to express the parallel intent of the application through DSL code, which is then translated into rules for a source-to-source compiler. These rules, design templates and the original sequential application are passed as inputs to the FraSPA compiler. The compiler then copies the original abstract syntax tree and generates a parallel version. A more recent addition to this framework is a command-line interface used to express parallel intent, however, this is only suitable to parallelise small-scale applications with a few hundred lines of code due to the lack of support for parallelisation beyond loops (Arora et al. 2013). C and C++ are the currently supported languages in the FraSPA framework.

When using Hi-PaL, the user must specify the hooks which act as transformation points in the serial program where the parallel operations need to take effect. The user has a choice of specifying the location before, after and around the particular statement in the serial application. In Figure 5.7 line 1, the code

```
1. Parallel section begins after("SEED=atoi(argv[4]);") mapping is linear{
2. ParDistribute2DArrayInt(life, M, N) around statement
3. ("initMatrix<int>(life, M, N, value);") && in function ("main");
4. ParExchange2DArrayInt(life, M, N) before statement
5. ("printMatrix<int>(life, M, N);") && in function ("main");
6. AllReduceSumInt(count, multiple) after statement
7. ("count = cellsAlive(life, M, N);") && in function ("main");
8. ParExchange2DArrayInt(temp, M, N) before statement
9. ("ptr = temp;") && in function ("main");
}
```

**Figure 5.7:** Hi-PaL Example
5.3 DSL Approaches

fragment Parallel section begins identifies where to place (hook) the parallel code; in this example it is to be inserted after the atoi(argv[4]) function. The decomposition of the data is also expressed on line 1 through the mapping is code fragment. The user can choose linear, cyclic or block cyclic. Linear is chosen in this example. A set of Hi-PaL APIs exist for parallel tasks including that of data distribution. Line 2 of Figure 5.7 expresses the intent that the two-dimensional integer-type matrix life with M rows and N columns should be distributed amongst different processors. The parallel code for this distribution needs to be inserted around the statement initMatrix<int> within the function main. Lines 3 and 5 illustrate the Hi-PaL code to exchange the border cells of the blocks of matrices life and main. A reduction seen on line 4 is performed on the variable count and this is to occur multiple times whenever the search pattern count=cellsAlive(life, M, N) is encountered.

The DSL code shown in Figure 5.7 is then fed into the FraSPA system as well as the serial code and a parallel version of the codebase is generated with MPI statements embedded. Hi-PaL eliminates the need to directly write lower level MPI code, however, the user is required to learn the DSL to generate the parallelised version. The command-line interface addition to this framework is a

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✓</td>
<td>MPI is generated and accessible to the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>The user must use snippets of the serial code as hooks within the DSL code.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>
step towards eliminating this learning curve, however, this is not currently suitable for industrially-sized applications. Table 5.5 highlights Hi-PaL’s ability to support the derived requirements, illustrating the tool’s failure to meet Requirements R4 and R5.

5.4 Generic Template Approaches

The C++ Standard Template Library (STL) provides a set of container classes (list, vector, map, etc.), and template algorithms (sort, find, merge, etc.) designed to work together (Musser et al. 2001). Generic algorithms in STL are defined in terms of iterators which are pointer-like objects used to traverse the sequence of objects in a container. STL also provides function objects which are objects that can be called as if they were functions. A number of approaches to abstract data decomposition and communication have been designed as C++ template libraries and will be discussed in this section.

5.4.1 Janus

Janus is a C++ template library designed for the domain of data-parallel programming and provides an abstraction for parallelising grid based applications (Gerlach 2002; Gerlach et al. 2001). Janus utilises three concepts — domain, relation and domain function. A domain is a finite set of elements that is given as a sequence and can be used to represent the elements in a grid using global indices starting at 0. To execute in a parallel environment, the domain is distributed over mutually disjoint subdomains according to the number of processes available. Each process has an index set representing each locally mapped object. A domain function describes data associated with a domain and allows random access to this data.

Janus offers a template class called grid<N> that models the concept of domain. This template describes the Cartesian product of N integer intervals. For processes to communicate, the user must describe the relation between domains. A relation between two domains is the description of data dependencies
5.4 Generic Template Approaches

between these domains. This concept is based on the mathematical concept of a relation between two sets. Janus provides a relation template called stencil for communication on regular grids. For example, the domain model grid<N> provides a representation of an N-dimensional rectangular grid and the relation stencil<N,D> represents a D-point stencil between two N-dimensional grids.

Figure 5.8 provides a coding fragment to show how to program using the Janus library. The grid is defined using the grid template on line 5. The template parameter 2 represents the dimensions of the grid. The variables lower and upper (defined on lines 2 and 3) describe the range (0, m) × (0, n) and are passed as arguments to the grid g on line 5. Temporary variables lower1 and upper1 defined on lines 7 and 8 are used in the stencil relation, narrowing the range to avoid the boundary cells where neighbours may not exist. The eight point stencil object s, on the grid g, is defined on line 10. The stencil template parameter 2 represents the dimensions of the stencil and the parameter 8 represents the number of elements. The expression neighbour_stencil() is an inline function where the offsets of the stencil are user-defined. Communication is expressed through operations on the relation objects such as pull and pull_reduce. In a distributed relation, pull transfers non-local data and pull_reduce performs a transfer and a reduction. Using the same example as in Figure 5.8, the statement s.pull.reduce(...) would be used to perform a reduction on all the points of the grid g. After declaring and instantiating a stencil for a particular grid, all communication is handled automatically through such relation operations. The
Janus software architecture includes a package called Janus Distributed Engine (JADE), which is implemented on top of MPI and is necessary for execution in a distributed memory environment. A parallel programmer needs to be familiar with the C++ STL as well as the Janus library to use this tool. Table 5.6 provides an overview of the derived requirements and their application in the Janus system illustrating the tool’s failure to meet Requirements R1, R4 and R5.

Table 5.6: Janus - Requirements Analysis

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>Reengineering of the serial version is necessary.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>

5.4.2 C++ components as part of ISTL module

The Iterative Solver Template Library (ISTL) was developed as part of the Distributed and Unified Numerics Environment (DUNE 2013) to provide solvers based on generic sparse matrix/vector classes. In this approach, C++ generic template classes are used to abstract the data decomposition and communication when parallelising applications (Blatt and Bastian 2009). Data is stored in existent sequential data structures where each entry corresponds to an entry in the virtual global view.

The template class ParallelIndexSet is used to realise a mapping of a global index set onto local indices. The class takes two parameters, the first is the type of global index and the second is the type of local index. Figure 5.9
5.4 Generic Template Approaches

demonstrates how to set up an index set on the root process where an overlap of one cell exists with neighbouring processes. As can be seen on line 4, \texttt{PIndexSet} is of class type \texttt{ParallelIndexSet} and takes the parameters \texttt{int} and \texttt{LocalIndex}. The second parameter which represents the local index, is defined as a type of \texttt{ParallelLocalIndex} (line 3). This takes the enumeration \texttt{Flags}, defined on line 2 as its parameter, which distinguishes between the indices owned by the process (\texttt{owner}) and those that are in the ghost regions and belong to neighbouring processes (\texttt{ghost}). An index set of type \texttt{PIndexSet} is declared on line 5 as \texttt{sis} and the function \texttt{beginResize} is called on line 6, indicating the start of the resize phase where the global and local indices will be mapped as seen in the loop on lines 7-10. The index pairs (global, local) are mapped using the \texttt{add} function and a distinction is made between the indices 0-5 on line 8, which have the attribute \texttt{owner} and index 6 on line 9 which has the attribute \texttt{ghost}.

A communication template class \texttt{Interface} is then used to select subsets of indices for data exchange. This takes a custom type of \texttt{ParallelIndexSet} as a parameter. This is depicted in Figure 5.10 on line 13, where the index set \texttt{PIndexSet} is passed as a parameter. \texttt{BufferedCommunicator} as seen on line 15, is a template class which performs the communication, sets up buffers, gathers the data and scatters this after the communication step. It uses the information provided by the interface when setting up the buffers. A build function must be called before communication can start as seen on line 16, taking the data.
source, target arrays and interface \((s, s, \text{infS})\) as arguments. The \texttt{forward} method on line 19, sends data from the source to the target and the \texttt{backward} method on line 21, communicates in the opposite direction. The template parameter \texttt{CopyData} copies data to the required location and \texttt{AddData} copies from the source location but adds the received data items to the target entries.

This approach does provide a higher level of abstraction than that of MPI where communication is automated. However, the user needs to be cognisant of the lower level details of the chosen decomposition strategy and manually set up index sets as seen in Figure 5.9. Table 5.7 illustrates how this library meets the derived requirements R1-R6, highlighting the failure to meet Requirements R1, R4 and R5.

**Figure 5.10: ISTL Example 2**

```cpp
13 Interface<PIndexSet> infS;
14 ...
15 BufferedCommunicator<PIndexSet> bComm;
16 bComm.build (s, s, infS);
17 ...
19 bComm.forward<CopyData<Container>>(s,s);
20 ...
21 bComm.backward<AddData<Container>>(s,s);
```

**Table 5.7: ISTL library - Requirements Analysis**

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>Reengineering of the serial version is necessary.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>
5.4 Generic Template Approaches

5.4.3 TACO

TACO (Topologies and Collections) is a distributed object framework based on C++ templates (Nolte et al. 2000; Nolte et al. 2001). TACO is based on topologies inspired by those used in the Promoter model (see section 5.3.1). Data distribution and communication are abstracted through the use of both distributed object groups and collective operations which are generically implemented through topology classes and C++ function templates. Each process can access objects in its own local space using C++ access methods and can access objects in other processes through Remote Method Invocation (RMI) templates.

TACO’s collections are based on distributed graphs and have a user-defined topology that is dependent on the application being parallelised. Topology classes reveal the internal structure of a collection; an N-dimensional grid being an example of a topology. Topologies are implemented as distributed linked object sets using global pointers. A global pointer stores the local (process number) and global address (pointer to memory location) of an object and is implemented by a generic ObjectPtr class. A leader object (root node) represents the entire collection and collective operations are performed by addressing this leader object. The objects in a collection are accessed through an STL style iterator.

Figures 5.11 and 5.12 demonstrate how to use the TACO library when parallelising a two dimensional grid-based application. The class HeatStripe in Figure 5.11 (lines 4-11) is derived from both a HeatGrid class that provides the serial implementation of the application and TACO’s DblList topology class that implements a distributed linked list. A Cyclic mapping strategy is

```
class HeatStripe: public HeatGrid, 
public DblList<HeatStripe>
{
  public:
    ....
    //exchange data with neighbours
    void exchange () {....};
}
```

*Figure 5.11: TACO Example 1*
specified on line 14 in Figure 5.12 and a collection of 256 HeatStripe instances are distributed in a cyclic fashion by use of the GroupOf template class as seen on line 15. The taco::Cyclic class maps group members’ ranks (starting at 0) to computing nodes whereas taco::Random distributes the members randomly across all nodes. It is possible to define a custom mapping and specify node numbers explicitly. A synchronous global step operation is provided by TACO to initiate data parallel computations on line 19. This is used to invoke the exchange method from the HeatStripe class (see line 10 in Figure 5.11). The taco::m2f function on line 19 is a convenience function provided by TACO to convert ordinary methods including their parameters to the corresponding function objects so that they can be transferred to distributed nodes and applied to compatible objects. The data transfer is performed using remote memory write operations.

Collective operations in TACO use single-sided communication; method invocations are explicitly sent but their execution is implicitly performed by the framework (i.e. a receive is not explicitly performed). To achieve synchronisation, TACO provides Mutex and Lock classes similar to Java’s synchronised methods. It is the responsibility of the programmer to ensure concurrency control. TACO abstracts the decomposition and communication code required when parallelising grid-based applications, however the user is required to learn the TACO template library. Table 5.8 provides an overview of the support provided by TACO for the derived requirements and illustrates its failure to satisfy Requirements R1, R4 and R5.
5.5 PSE Approaches

Table 5.8: TACO - Requirements Analysis

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>Reengineering of the serial version is necessary.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>

5.5 PSE Approaches

Gallopoulos et al. (1994) describe a Problem Solving Environment (PSE) as a programming environment that:

> provides all the computational facilities necessary to solve a target class of problems.

PSEs have not been widely accepted and this is potentially due to their narrow focus on a subset of problems, which acts as a deterrent to their adoption by a wide user base (Mattson et al. 2004, p. 215). This Section explores PSEs which provide abstractions for data decomposition and communication for grid-based applications. Two of these (POOMA and PETSc) will be discussed in the following subsections.

5.5.1 POOMA

The Parallel Object-Oriented Methods and Applications (POOMA) Framework (Reynders et al. 1996) is a C++ class library designed for writing parallel PDE solvers using finite-difference and particle methods. POOMA provides an abstraction for performing data decomposition and communication when working with data-parallel applications (Karmesin et al. 1998). The framework is composed of
classes such as the generic \texttt{Array} class which provides an interface to a data representation. POOMA also provides objects which can be used to specify sub-sections of \texttt{Array} objects, for example \texttt{Loc<N>}, which represents a single point and \texttt{Interval<N>}, which represents a section of an array.

Figures 5.13 and 5.14 illustrate extracts of code used when parallelising a 2D grid application modelling the Laplace Equation (FreePOOMA 2000). A stencil is used in this application and can be represented as a class as seen in Figure 5.13, where the \texttt{LaplaceStencil} is defined (line 3). The methods \texttt{lowerExtent} and \texttt{upperExtent} on lines 7 and 8, tell the POOMA framework the extent of the border that the stencil needs to perform boundary exchanges. In this case, the border is one unit along both directions of the x and y axes. This customised stencil class is then passed as a template parameter to the generic stencil class \texttt{Stencil} as seen on line 8 in Figure 5.14. Line 4 in Figure 5.14 initialises the POOMA environment and line 7 creates the array representing the data in the 2D grid. The number of dimensions of the array (2) is passed as a template parameter
5.5 PSE Approach

to Array, while the actual dimensions are given as constructor parameters. An Interval class is used on lines 10 and 11. This specifies a contiguous range of index values and the constructor arguments passed into interior.1 and interior.2 specify the low and high ends of the interval’s value. The Interval template argument specifies the number of array dimensions. On line 15, the framework reads from the source array world(interior.2) but only writes to a subset of elements as defined by the left and right extents returned by the stencil. The blockandEvaluate call in line 16 ensures that the evaluation is completed before the whole array is overwritten.

POOMA was assessed based on the support provided for the derived requirements and the results are reported in Table 5.9. Requirements R1, R4 and R5 were not satisfied by this PSE.

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Automates grid indexing.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>Reengineering of the serial version is necessary.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>

5.5.2 PETSc

Portable, Extensible, Toolkit for Scientific Computation (PETSc) is a programming environment used for the set of problems in the numerical domain of partial differential equations (Balay et al. 2013). PETSc provides an abstraction for MPI including the tasks of performing data decomposition and communication. Although the user is free to incorporate MPI throughout PETSc programs, much of
the MPI functionality is abstracted from the user. PETSc offers library interfaces, which a programmer can use if applicable to their application. A Distributed Array Interface is available (DMDA) to perform data decomposition and communication of boundary cells between neighbouring processes using local and global indexing.

In the case of an application based on a two-dimensional grid, the programmer must instantiate a DMDA object by calling DMDACreate2d as seen in Figure 5.15 on line 3. The Communicator, grid dimensions and decomposition distribution are among the arguments passed into this. The periodic settings are also passed in for x and y coordinates, as well as the communication stencil width to determine the ghost regions. Exchanges can then be communicated using a box-type stencil DA_STENCIL_BOX, which includes diagonal corners and is equivalent to a 9-point stencil or a star-type stencil, DA_STENCIL_STAR, which corresponds to a 5-point stencil. Line 3 in Figure 5.15 illustrates the creation of a two-dimensional distributed array object named grid (last argument) with dimensions of nx \times ny, with a process decomposition of px \times py. PETSc uses its own communicator which is similar to MPI’s COMM_WORLD as seen in the first argument, PETSC_COMM_WORLD. Communication on the x axis of the grid is not periodic as can be seen by the second argument DMDA_BOUNDARY_NONE but is periodic on the y axis due to the third argument DMDA_BOUNDARY_PERIODIC. A star stencil will be created as seen in the fourth argument DMDA_STENCIL_STAR.

Each DMDA object defines the layout of both a distributed global vector and a local vector. Line 6 shows the creation of a global vector using the information from grid by calling DMCreateGlobalVector(). The global vector will be stored in x. DMCreateLocalVector() can also be called to create a local

```c
DMDACreate2d(PETSC_COMM_WORLD, DMDA_BOUNDARY_NONE, DMDA_BOUNDARY_PERIODIC, 
             DMDA_STENCIL_STAR, . . . nx, ny, . . . px, py, . . . &grid);

DMCreateGlobalVector(grid,&x);

DMDAGetCorners(grid,&xs,&ys,NULL,&xm,&ym,NULL);
```

Figure 5.15: PETSc Example
5.6 PSE Approach

vector. To overlap communication between a local and a global object, the user can scatter data from a global vector into its local parts. Information about the global indices of the local portion of an array can be accessed through the method DMDAGetCorners as seen on line 8. The output values include xs and ys representing the bottom left corner grid indices and xm and ym, the widths of the grid in x and y directions. The NULL value is used in the place of the z dimension as this grid is not three-dimensional. In order to extend the returned data to include ghost regions, the function DMDAGetGhostCorners is required.

PETSc works with any system supporting MPI and MPI calls can be inserted in the code where necessary. This environment is targeted at applications which solve linear and non-linear systems of equations in parallel. While this is applicable to a large set of scientific solutions, it limits the user-base outside the scientific computing domain. The PETSc API is extensive and knowledge of numerical systems is required for a lot of the offered functionality. The support provided by PETSc for the derived requirements is outlined in Table 5.10 and the failure to satisfy Requirements R1, R2, R4 and R5 is illustrated.

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user, however a user is free to add MPI code.</td>
</tr>
<tr>
<td>R2</td>
<td>✗</td>
<td>User needs awareness of data indices.</td>
</tr>
<tr>
<td>R3</td>
<td>✓</td>
<td>Generates data structures and communication operations.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires rewrite of parallelised code to change decomposition strategy.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>Reengineering of the serial version is necessary.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>
5.6 Other Approach

Visualisation systems provide techniques to explore multi-dimensional data and can contain visualisation modules which run in parallel. The Visualisation in Parallel (Vipar) project created one such system and aimed to support users developing parallel modules for use within applications. This system provided a prototype tool (DDTool) to guide the user when inputting data decomposition related parameters to the system (Larkin et al. 1996; Leng et al. 2001). DDTool is discussed in the following subsection.

5.6.1 DDTool

The design behind Vipar allows serial computationally expensive modules to be replaced with parallel versions. This is done by spawning processes or threads from the serial Modular Visualisation Environment (MVE). Vipar is structured as a set of libraries which provide interfaces between the visualisation system and the message passing system which is based on MPI. A data decomposition tool named DDTool, was developed as a graphical user interface to the AVS/Express visualisation system (Upson et al. 1989) to allow a user to describe the data decomposition strategy. Two libraries exist to support the DDTool: VPRdd, a system dependent library which distributes, manages and collects the data and VPRidd, a library independent of the system which contains routines to calculate data decomposition patterns. These libraries provide the mechanism for the DDTool to define the data decomposition.

The user interacts with the DDTool by inputting values to design the required data decomposition. Inputs include the decomposition strategy such as block or cyclic, the application’s datatype, number of dimensions and neighbourhood data to enable boundary processing. The DDTool outputs a template parallel module which contains four components. The first of these components is a distributer, which uses the VPRidd routines and handles the input to a parallel module. It calculates the portions of data to be sent to each process. A harness is the second
5.7 Other Approach

component which acts as a control process for the parallel module and sends data to and receives data from the worker processes and so uses the \texttt{VPRdd} library. Example routines include \texttt{VPRdd_SendReg} and \texttt{VPRdd_RecvReg} used to send and receive data portions and the collective \texttt{VPRdd_SwapNbr}, used to swap data between processes. A compositor is the third component which handles the output from a parallel module, i.e. it collects data into a single array again. Lastly, the fourth component is the worker which processes a data portion. The user must provide the application code to process the data within the worker component.

The DDTool was designed as a prototype and not extended to be aware of other parallel modules in the network. The data to be computed are distributed to a module and processed using spawned processes within that module. This module does not have the ability to communicate with other modules running in parallel except to send back the resultant computation in a recomposed state. The system does not have the ability to perform synchronisation. Table 5.11 presents the findings from the analysis of DDTool with respect to the derived requirements and illustrates the lack of support for Requirements R1, R3, R4 and R5.

<table>
<thead>
<tr>
<th>Req.</th>
<th>Req. met</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>✗</td>
<td>Hides the MPI implementation from the user.</td>
</tr>
<tr>
<td>R2</td>
<td>✓</td>
<td>Data indices are automatically generated.</td>
</tr>
<tr>
<td>R3</td>
<td>✗</td>
<td>Can not communicate with modules running in parallel.</td>
</tr>
<tr>
<td>R4</td>
<td>✗</td>
<td>Requires regeneration of parallel template module and recoding of worker component.</td>
</tr>
<tr>
<td>R5</td>
<td>✗</td>
<td>Parallel module template is generated but user is required to add code.</td>
</tr>
<tr>
<td>R6</td>
<td>✓</td>
<td>Targets structured grids.</td>
</tr>
</tbody>
</table>
5.7 Additional Requirements

When reviewing the capabilities of the tooling options available in the state-of-the-art, four additional requirements for support in this area emerged. As these new requirements are appended to the list derived from the empirical studies in Chapter 4, these will be referred to as R7-R10. Support for performing local boundary exchanges was investigated in all ten tooling options, yet support for exchanges where global boundary conditions need to be considered, was not documented. Requirement R7 emerged from this observation. As each tool required some level of prerequisite knowledge, the need for a minimal learning curve gave rise to Requirement R8. The portability of the code output by the tools inspired Requirement R9, while the finding showing only one of the reviewed tools supports an alternative language to C and C++, provided the motivation for Requirement R10. These additional requirements are outlined in more detail as follows:

- **R7: Flexibility with regards to Global Boundaries.** In certain applications, data are modelled using structured grids containing global boundaries which must be considered during parallelisation. For example, in CFD applications, the behaviour of a fluid may vary at an external boundary and so correct wall boundary conditions need to be applied for accurate results (Li et al. 2010). Specific processing, not applicable to interior cells, may be necessary for boundary-bordering cells and so the programmer must be able to determine whether a distributed data chunk includes a global boundary. All listed tools provide a mechanism to handle local boundary processing, i.e. exchanges between neighbouring cells to populate overlapping (ghost) regions, however, no tool offers any *in-built abstractions for more complex global boundary operations*. Two tools allow the user to manually implement MPI calls during the parallelisation process and thus allow the user to write MPI code to cater for such conditions, however, the global boundary exchanges must be coded manually. The tools which...
5.8 Comparative Analysis

do not allow any manual modification of the MPI when generating the parallelised versions are restrictive as the user has no flexibility to code more complicated functionality beyond the API/DSL/Libraries offered.

- **R8: Minimal Learning Curve.** A majority of tools outlined in this chapter offer a type of language/API to abstract data decomposition and communication when parallelising applications. A learning curve is thus associated with these approaches as the user must use a new programming paradigm. This highlighted an additional requirement, i.e. the need for a minimal learning curve, so that *a programmer does not need to invest significant time learning a tool before using.*

- **R9: Non-proprietary Runtime.** Four of the reviewed tools generate code which can run in an MPI-compatible environment without the aid of any proprietary runtime. Six of the tools generate code which require a proprietary runtime to execute. This can impact a programmer’s decision to use the tool as it adds dependencies and can limit portability. Thus the requirement of non-proprietary runtime emerged as another requirement.

- **R10: Language Extensibility.** C, C++ and Fortran are widely used to develop HPC software and Python is used to a lesser extent (Basili et al. 2008). The tools reviewed in this chapter predominantly support C++, PETSc being the only tool offering an abstraction for Fortran. Those tools reliant on the C++ standard template library are tied to the C++ language and so extensibility to support another language is not possible. An additional requirement emerged to support multiple languages or to have the ability to be extended to support multiple languages.

5.8 Comparative Analysis

Table 5.12 provides a comparative analysis of the tools reviewed based on the empirically derived requirements R1-R6, as well as the four additional requirements
R7-R10, which emerged from the literature review. Nine of the ten tools fail to support Requirement R1 as MPI operations are encapsulated and unavailable to the user due to the output of tool-specific code. Hi-PaL is the one tool in the review which generates MPI based code. Nine of the ten tools listed meet Requirement R2 with the exception of PETSc, which does not automate the indices in the code. Nine of the ten tools also fulfil Requirement R3, by generating the data structures and communication routines with the exception of DDTool, which does not allow for communication between parallel modules. No reviewed tool meets Requirement R4, as all implementations require the user to rewrite code to convert from an implemented decomposition strategy to another. Similarly, the tools fail to meet Requirement R5 as parallelisation of an application involves some level of application code impact. The ten tools do provide support for Requirement R6 as all target structured grids. With regards Requirement R7, two tools allow the user to manually implement MPI (PASM and PETSc) while one provides the option to manually create a new pattern (MAPS). While these tools make it possible to code for global boundaries, they do not provide specific support to assist this task. Requirement R8 specifies the need for a minimal learning curve, however this requirement is not met in any tooling option, as the user is required to adopt an unfamiliar environment or become proficient in a new API or language. Four of the ten tools do provide non-proprietary runtime support and so meet Requirement R9. Six of the ten tools also provide the potential for language extensibility. Those tools based on generic templates are tied to the C++ language and so fail to meet this requirement. Of the ten tools, not one provides support for every requirement outlined. A viable combination of any number of the surveyed tools does not suffice to meet the requirements listed, as support is not provided for performing complex boundary operations or for changing a decomposition strategy with minimal code impact.
**Table 5.12: Decomposition Tool Support**

<table>
<thead>
<tr>
<th>Appr.</th>
<th>Tool</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PASM</td>
<td>✗✓✓✗✗✓✓✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAP₃S</td>
<td>✗✓✓✗✗✓✓✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Promoter</td>
<td>✗✓✓✗✗✓✓✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hi-PaL</td>
<td>✓✓✓✗✗✓✓✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JANUS</td>
<td>✗✓✓✗✗✓✓✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Continued on next page*
## Continued from previous page

<table>
<thead>
<tr>
<th>Appr.</th>
<th>Tool</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
</tr>
</thead>
<tbody>
<tr>
<td>STL</td>
<td>STL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Must learn template library</td>
<td>Integrated into DUNE</td>
<td>Tied to C++</td>
</tr>
<tr>
<td>TACO</td>
<td>TACO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Must learn template library</td>
<td>✓</td>
<td>Tied to C++</td>
</tr>
<tr>
<td>PSE</td>
<td>POOMA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Must learn POOMA API</td>
<td>Requires POOMA environ.</td>
<td>Potential for extension</td>
</tr>
<tr>
<td>PSE</td>
<td>PETSc</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Can manually implement MPI</td>
<td>Must learn PETSc API</td>
<td>Requires PETSc environ.</td>
</tr>
<tr>
<td>Other</td>
<td>DDTool</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Must learn VIPAR system</td>
<td>Requires suite of VIPAR libraries &amp; AVS</td>
<td>Potential for extension</td>
</tr>
</tbody>
</table>
5.9 Conclusion

This chapter reviewed tooling support available in the state-of-the-art to assist the task of performing data decomposition and communication when parallelising grid-based applications. Ten tools were reviewed and grouped according to five approaches. A comparative analysis was conducted based on the requirements derived from the empirical data outlined in Chapter 4 and the additional requirements which emerged from the analysis of the available offerings. No reviewed tool was found to meet all the requirements. Chapter 6 describes a prototype tool designed to meet all ten requirements and an evaluation of this tool is outlined in Chapter 7.
6

Design and Implementation of MPIGen

6.1 Introduction

Requirements for a tool to support data decomposition and communication in high performance environments were empirically derived from a review of the state-of-practice outlined in Chapter 4. The state-of-the-art was explored in Chapter 5 and gaps were found in the existing support available to assist the task of decomposing applications and devising the subsequent messages. This chapter presents the design and implementation of a tool named MPIGen (MPI Generator) which rests on the knowledge gained from both the state-of-practice and the gaps identified in the state-of-the-art. This tool was prototypically implemented with the aim of validating the derived requirements.

This chapter has the following layout: Section 6.2 describes a high-level overview of MPIGen while Section 6.3 addresses the design of the tool with respect to the derived requirements. Design considerations are discussed in Section 6.4 and an overview of the output files is provided in Section 6.5. MPIGen was tested using three applications and a detailed overview of one of these test cases is provided in Section 6.6 as an illustration of the tool’s functionality and
adherence to the requirements. The functional limitations of the prototype are discussed in Section 6.7.

6.2 MPIGen Overview

MPIGen is designed to assist developers when performing decomposition and communication when converting serial applications to parallel and supports two-dimensional grid based applications. The user is required to input configuration values and the tool then generates C (C++ compatible) code which contains MPI functionality. In order to parallelise the serial application, the user must invoke the generated functions which will be referred to as wrapper functions as they wrap the MPI functionality, isolating it from the application code. Figure 6.1

![Diagram of MPIGen Overview](image)

**Figure 6.1:** MPIGen High-Level Diagram
illustrates a high-level view of the functionality of MPIGen and the steps involved in using the tool are as follows:

1. The user accesses the tool which is implemented as a web-based interface and enters values to express the desired parallelisation. For demonstration purposes, decomposition type, ghost row count and ghost column count input parameters are illustrated. A prerequisite for this step is that the user has knowledge of the existing data structures as well as the parallel intent of the serial application.

2. The input values are recorded by the tool’s front-end logic and are subsequently fed to the business logic of the tool (referred to as MPIGen backend in Figure 6.1).

3. The MPIGen backend generates code containing MPI functionality reflecting the intent expressed in the user-selected input values. A demonstration/test file with template code fragments is also generated.

4. The user edits the serial application and invokes the MPIGen function calls (defined in the generated code) in the appropriate places in the serial application. The user has the option of taking coding fragments from the demonstration file for this step.

5. The user compiles the serial application containing the MPIGen function invocations as well as the generated code using an MPI compiler (e.g. mpicc), resulting in an executable that can be run in parallel.

MPIGen is accessible through the following URL: http://mpigen.lero.ie

6.3 Addressing the Derived Requirements

The design and implementation of a prototype tool was necessary in order to validate the requirements (R1-R6) derived from the empirical studies as well
6.3 Addressing the Derived Requirements

as the additional requirements (R7-R10) attained from the review of the state-of-the-art. Kruchten (1995) describes a scenario as an abstraction of the most important requirements. MPIGen is depicted in two scenarios, the first involves the modification of a serial codebase to incorporate MPI so as to execute in a multicore environment. The second is an evolutionary scenario, where the user can explore alternative settings such as changing the implemented decomposition strategy to assess scalability. Table 6.1 outlines the requirements which provide the motivation for the use cases in the MPIGen system. A description of each use-case is provided, along with a list of functional units detailing the interactions between the user and the system to achieve the goals of the use-case.

As R8-R10 are non-functional requirements, use cases and functional units are not applicable in their implementation. Requirement R8 states that use of a tool should require a minimal learning curve, for example, the requirement to learn a new language would set the learning curve to high. MPIGen is designed to have a low learning impact as its use does not involve the comprehension of a new language. The user is required to invoke functions which are implemented in a wrapper file. A demonstration/test file is provided to assist with this, providing sample invocation coding fragments (see Section 6.5 for an overview of the generated files). Requirement R9 states that the tool should not require an abstract machine or proprietary runtime library to execute. This requirement is met as MPIGen is not tied to a proprietary runtime environment as it can execute on any environment that supports MPI. Requirement R10 requires that the tool be extensible and not dependent on one programming environment or paradigm. This requirement is met as the tool can be extended to generate code based on an alternative language which supports the MPI specification. As the user inputs values using a front-end interface, their initial interaction with the system would remain unchanged.
## Table 6.1: MPIGen Use-Case Descriptions

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Requirement</th>
<th>Use-case description</th>
<th>Functional unit</th>
</tr>
</thead>
</table>
| R1   | MPI focused | The user inputs values describing the type of MPI communication they require in the parallelised code. The generated MPI is accessible to users in a familiar format, through generated .c and .h files. | • Set the MPI communication to be periodic or non-periodic, i.e. cyclic communication.  
• Set synchronisation properties by choosing blocking or non-blocking MPI communication.  
• If Cartesian decomposition, provide an option to let the MPI communicator reorder the ranks.  
• Provide an option to let the node distribution take default values where the MPI runtime determines the ideal allocation of nodes. |
| R2   | Automated grid indexing | The user inputs values according to the grid size and boundary constraints of the application. | • Set grid size on X and Y axes.  
• Set the boundary row and column count to a certain size. |
| R3   | Data structure & communication generation | The user inputs values according to the type of decomposition strategy they wish to implement. | • Create a decomposition strategy such as row, column, Cartesian.  
• Determine the node allocation using the manually entered values. |
|      |             | The user chooses data structure allocation | • Allocate a global sized grid to the root process. |
|      |             | The user generates the MPI-based code. | • Generate the MPI based code.  
• Allow the generated code to be downloaded onto client machine. |
| R4   | Ease of strategy conversion | The user re-enters input parameters choosing a different decomposition strategy and regenerates the code. The application code does not have to be altered. | • The functional units that meet R3 apply |

*Continued on next page*
### Key Design Considerations

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Use-case description</th>
<th>Functional unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>R5 Low application code impact</td>
<td>The user chooses options to generate template code.</td>
<td>• Generate custom matrix datatype to allow for creation of a template derived datatype.</td>
</tr>
<tr>
<td>R5</td>
<td>The user customises variable names.</td>
<td>• Change the names of the chosen MPI related variables such as rank, dims, root. • Change the names of the chosen index-related variables such as myLocalFirstRow.</td>
</tr>
<tr>
<td>R5</td>
<td>The user customises the file and function names.</td>
<td>• Prefix the generated function names with a chosen term. • Prefix the generated files with a chosen term.</td>
</tr>
<tr>
<td>R5</td>
<td>The user chooses to allow the function parameters to be shown or hidden.</td>
<td>• If show function parameter is set to true, do not encapsulate the core set of input values in the wrapper file and instead print in the demonstration file.</td>
</tr>
<tr>
<td>R6 Targeting structured grids</td>
<td>The user inputs values specific to the grid in the application being parallelised.</td>
<td>• Set dimensions of the grid to 1D or 2D. • Set the datatype as selected primitive (e.g. integer, double, float).</td>
</tr>
<tr>
<td>R7 Flexibility w.r.t. global boundary operations</td>
<td>The user sets constraints on the global boundary of the application</td>
<td>• Set the external boundary cell depth to a certain size.</td>
</tr>
</tbody>
</table>

6.4 Key Design Considerations

A core design decision regarded the complexity of the user-interface to the system. The original design of the tool required the user to enter input values as key-value
pairs in a properties file. A later version of the system, extended this to a web interface as shown in Figure 6.2. As the web interface contains validation logic, which inspects for numerical or alphanumerical values where necessary, this method is less error-prone for the user. Also, the risk that the user may delete a key-value pair and thus invalidate the system is removed.

Another design consideration involved the encapsulation of certain input settings which a user may potentially modify when executing the parallelised application using varying configurations. These configurations may be modified to reflect a different grid size, node allocation or periodicity (wrapping) setting. The system is designed so that the user can choose if such values are easily accessible and not encapsulated. This use-case choice is encompassed in the fourth use-case outlined to meet Requirement R5, listed in Table 6.1. If the value show function parameters (third input value illustrated in Figure 6.2) is set to true, a core set of input values are listed in the demonstration file and are not encapsulated in the generated code.\(^1\) The user is required to copy these into their application allowing for easy access if these are to be modified. If this value is set to false, this core set of input values are hidden from the user in the wrapper file and are less accessible.

Stencil operations were described in Section 2.5.2 and their exclusion as explicit input properties was a design-time decision. As this prototype is implemented to cater for all levels of parallel programmer and evaluated using a mix of levels (see Chapter 7), the inclusion of stencil properties such as star, cross etc. would create unnecessary complexity for those not familiar with computational-science concepts. In their absence, the interface requests cyclic communication properties such as left to right column, top to bottom row and corner to corner. MPIGen then uses these input values to create a stencil template in the generated code, which a user may use if so inclined.

---
\(^1\)These values include dimensions of grid, node allocation, cyclic settings on rows and columns, rank reorder variable
### 6.4 Key Design Considerations

**MPIGen tool**

MPI Grid-based Decomposition and Communication Code Generator

Please enter the values below in order to generate the MPI code

<table>
<thead>
<tr>
<th>Code Housekeeping Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File prefix</strong></td>
</tr>
<tr>
<td><strong>Function prefix</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Show function parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ true ☐ false</td>
</tr>
</tbody>
</table>

**Application Properties**

<table>
<thead>
<tr>
<th>Grid size on X axis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grid size on Y axis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grid data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ short ☐ int ☐ long ☐ float ☐ double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix storage layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ 1D ☐ 2D</td>
</tr>
</tbody>
</table>

**Decomposition Properties**

<table>
<thead>
<tr>
<th>Decomposition type</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ Cartesian ☐ Row ☐ Column</td>
</tr>
</tbody>
</table>

**Boundary Properties**

<table>
<thead>
<tr>
<th>Ghost Row count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ghost Column count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>External boundary Row count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>External boundary Column count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

**Communication Properties**

<table>
<thead>
<tr>
<th>Blocking communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ true ☐ false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cyclic communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ true ☐ false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Left to right column(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ true ☐ false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Top to bottom row(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ true ☐ false</td>
</tr>
</tbody>
</table>

**Other Properties**

<table>
<thead>
<tr>
<th>Generate a custom matrix datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ true ☐ false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allocate grid to root</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ true ☐ false</td>
</tr>
</tbody>
</table>

**Customization of Variables**

<table>
<thead>
<tr>
<th>Do you wish to customize variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ Yes ☐ No</td>
</tr>
</tbody>
</table>

**Code Generation**

Click to generate the code:

---

Created by: Anne Meade, Copyright © University of Limerick, Ireland, 2013. Email: anne.meade@ul.ie

This work was supported, in part, by Science Foundation Ireland grant 10CE1189 to Lero - the Irish Software Engineering Research Centre (www.lero.ie)

---

**Figure 6.2: MPIGen Graphical User Interface**

128
6.5 MPIGen Output

Five files are generated by the tool and a description of these files follows: (xxx represents the customisable file prefix and gen the customisable function prefix)

- **xxxlog.txt** contains logging information related to the execution of the tool and the setting of the properties in the input file.

- **xxxStructs.h** contains the structures necessary for implementing the MPI code such as xxxMPIData, a struct containing all the MPI related variables; xxxGridData, a struct containing all the grid related variables; xxxGridSpace, a struct containing all the memory structures.

- **xxxMPIWrappers.h** contains declarations of functions which are defined in xxxMPIWrappers.c. See Figure 6.3 for an extract of this file. Line 29 shows the declaration of the function to allocate the memory structures. Line 32 shows the declaration of the function which encapsulates the message passing communication which occurs between processors.

- **xxxMPIWrappers.c** This file contains the core MPI functionality of the code. All the functions necessary to decompose the data and parallelise the application are implemented here and can be invoked by the user in the serial application. See Figure 6.4 for an example snippet of this file. The gen_set_MPI_data function on line 2 does the following:
  - Implements the decomposition logic by invoking the function defined on line 25 gen_create_cart_topology. This function creates the Cartesian topology (line 27), determines the process coordinates in the Cartesian topology (line 30), and invokes the function to find neighbouring processes (line 32).
  - gen_find_my_neighbourcoords on line 43 determines the neighbouring nodes dependent on the chosen communication input parameters by returning the shifted source and destination ranks as seen on lines 45 and 46.
6.5 MPIGen Output

- Sets the indices so as to access the relevant data structures by invoking the function `gen_set_indices` defined on line 60.
- Creates appropriate datatypes according to the input parameters set by the user by invoking the function defined on line 85 `gen_create_mpi_datatypes`.

The user invokes the `gen_set_MPI_data` and all the listed functionality is encapsulated.

- `xxxAppFragments.c` This serves as a test or demonstration file and contains code fragments which provide suggestions to invoke the MPIGen wrapper methods. The user is free to copy and paste the fragments generated into the application being parallelised in order to invoke the functions required. Template/Skeleton methods are also provided to enable pre- and post-processing such as: populating the distributed grid structure, accessing the processed data, and returning data to the root process. These functions serve as suggestions and should be used accordingly. A description of the use of this file is included in Section 6.6.

```c
29 int gen_allocate_structure_all(struct myMPI_MPDData *mpi_data,
30    struct myMPI_GridData *grid_data, struct myMPI_2DGridSpace *grid_space);
31
32 int gen_boundaries_swap_2dAccess(struct myMPI_MPIData *mpi_data,
33    struct myMPI_GridData *grid_data, int **local_array);
```

**Figure 6.3**: MPI Wrappers Header File
Chapter 6. Design & Implementation

```c
int gen_set_MPI_data(struct myMPI_MPIData *mpi_data, struct myMPI_GridData *grid_data)
{
    MPI_Comm_size(MPI_COMM_WORLD, &mpi_data->size);
    //MPI_Comm_size determines the size of the group associated with a communicator
    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_data->rank);
    //MPI_Comm_rank determines the rank of the calling process in the communicator
    grid_data->ghost_rows= 1;  //Set in properties
    grid_data->ghost_cols= 1;  //Set in properties
    if((mpi_data->dims[0] == 0) && (mpi_data->dims[1] == 0)){
        //Default distribution will be used as dims[0][1] set to 0,0
        //MPI_Dims_create will fill in suitable values
        }else if((mpi_data->dims[0] == 0) || (mpi_data->dims[1] == 0)){
    if(mpi_data->dims[0] == 0){
        ...
    }
    
    int gen_create_cart_topology(struct myMPI_MPIData *mpi_data){
    ...
    MPI_Cart_create(MPI_COMM_WORLD, mpi_data->ndims, mpi_data->dims, mpi_data->periods,
    mpi_data->reorder, &mpi_data->comm_cart);
    MPI_Cart_coords(mpi_data->comm_cart, mpi_data->rank, mpi_data->ndims, mpi_data->coords);
    gen_find_my_neighbour coords(mpi_data); //This method locates neighbouring processors
    ...
    
    int gen_set_indices(struct myMPI_MPIData *mpi_data, struct myMPI_GridData *grid_data){
    ...
    grid_data->localInnerNumRows=(grid_data->NI/grid_data->dims[0]);
    grid_data->localInnerNumCols=(grid_data->NJ/grid_data->dims[1]);
    ...
    //Local column related variables
    grid_data->localFirstCol= 0;
    grid_data->localLastCol=(grid_data->localInnerNumCols*(grid_data->ghost_cols+2))-1;
    grid_data->innerLastCol=grid_data->localLastCol-grid_data->ghost_cols;
    ...
    //Local row related variables
    grid_data->localFirstRow= 0;
    grid_data->localLastRow=(grid_data->localInnerNumRows*(grid_data->ghost_rows+2))-1;
    grid_data->innerLastRow=grid_data->localLastRow-grid_data->ghost_rows;
    ...
    }
    
    int gen_create_mpi_datatypes(struct myMPI_MPIData *mpi_data, struct myMPI_GridData *grid_data){
    ...
    MPI_Type_vector(count_MPI_COL, blocklength_MPI_COL, stride_MPI_COL, MPI_INT, &mpi_data->MPI_COL);
    
    Figure 6.4: MPI Wrappers C File
```
MPIGen was successfully tested on three applications: The Game of life, the Advection Equation and JFLOW® Reduced Engine. The Game of life and the Advection Equation are test applications and have minimal Lines of Code (LoC). The Game of life is a cellular automaton and was considered relevant as a test case due to the common use of cellular automata in science to model a variety of natural phenomena. Examples in scientific research include modelling a spatial simulation of fire to predict its behaviour in the field of environmental science (Berjak and Hearne 2002), modelling brain cells to determine brain tumour growth.

### Table 6.2: MPIGen Test Applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Domain</th>
<th>Properties</th>
<th>LoC*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game of life</td>
<td>Open source</td>
<td>Cellular automaton</td>
<td>• C language</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2D Grid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 9-point stencil</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Ghost region depth: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Global boundary region depth: 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Cyclic communication</td>
<td></td>
</tr>
<tr>
<td>Advection Equation</td>
<td>ICHEC demonstration program</td>
<td>Computational Fluid Dynamics</td>
<td>• C language</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2D Grid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 5-point stencil</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Ghost region depth: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Global boundary region depth: 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Cyclic communication</td>
<td></td>
</tr>
<tr>
<td>JFLOW® Reduced Engine</td>
<td>JBA Consulting</td>
<td>Flood risk management</td>
<td>• C++ language</td>
<td>2506</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2D Grid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 5-point stencil</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Ghost region depth: 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Global boundary region depth: 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Non-cyclic communication</td>
<td></td>
</tr>
</tbody>
</table>

*In serial application*
in the field of neurosurgery (Kansal et al. 2000), and modelling the competitive
interaction of grass species in the field of ecology (Silvertown et al. 1992). The
Advection Equation application was devised by ICHEC for training purposes and
describes the motion of a conserved scalar as it is advected by a fixed velocity.
The JFLOW® Reduced Engine application is an industrial sized application with
over 2,500 LoC and includes the additional complexity of a halo region around
the external boundary of the global grid. This application was ported to BG/P in
the participant-observer study (see page 58).

A comparison of these three applications is provided in Table 6.2. The properties
listed are based on programming language, grid dimensions, communication
stencil, ghost region depth, global boundary region depth and cyclic/non-cyclic
attributes. The lines of code in each application (LoC) is listed as well as the
domain the codebase targets. The source of each application is also outlined. To
demonstrate the functionality of MPIGen, the open source Game of life application
is used and a description is provided in the following sub-section.

### 6.6.1 Game of Life

The Game of life application serves as the basis of the evaluation outlined in
Chapter 7 and so is described in detail in this section. The game was developed by
the British mathematician John Horton Conway in 1970 and does not depend on
players but instead requires an initial configuration of the cell values, which then
evolve in each generation or timestep. The game is based on a two dimensional
orthogonal grid of square cells which are randomly set to be either alive (1 value)
or dead (0 value). In order to evolve, each cell communicates with its eight
neighbours (north, south, east, west, northwest, northeast, southwest, southeast)
and based on the summation of the neighbouring values, the cell either lives or
dies in the next iteration. The stipulations are as follows:
- If a live cell has <2 neighbours alive, then the cell dies (as if caused by
  under-population).
- If a live cell has 2 or 3 neighbours alive, the cell lives to the next generation.
6.6 Test Cases

- If a live cell has >3 neighbours alive, the cell dies as if by overcrowding.
- If a dead cell has exactly 3 live neighbours then the cell becomes live as if by reproduction.

These rules are applied for each generation which is controlled in a timestep loop. Nine-point stencil communication is applied.

6.6.2 Applying MPIGen

In order to parallelise the Game of life application, the appropriate input values were entered into the MPIGen web interface. The Matrix storage layout for this application is 2D and the decomposition chosen is Cartesian. The depth of the ghost rows and columns is set to 1 and as the code is based on a 9-point stencil, communication is set to true for left-to-right columns, top-to-bottom rows and corner-to-corner. The code was subsequently generated and moved to the same file location as the serial application.

The generated AppFragments file provides suggestions for how to invoke the wrapper functions. Extracts from this file are shown in Figure 6.5. This will be used to describe the steps to parallelise the Game of life program as follows:

- The include statements (lines 3-4), function declarations (lines 7-11) and struct declarations (lines 14-18) are copied into the serial Game of life application.

- The MPI initialisation statement on line 22 is copied into the serial application as well as the gen_set_MPI_data function invocation on line 24. As shown in Section 6.5, the latter initialises the MPI environment and invokes functions which perform the data decomposition, determine neighbouring processes, set the appropriate indices and create relevant datatypes.

- The invocations on lines 25 and 26 are also copied into the serial application. gen_allocate_structure_all creates the necessary data structures.
As gen_fill_structure is a skeleton, this needs to be implemented. The user must populate the distributed structure with 1s and 0s to replicate the serial version.

- The function outlined on line 29 abstracts all the message passing required to populate ghost values and is copied into the serial application, while the gen_access_data function on line 31 is a skeleton and so the user must provide the core of its implementation. This skeleton is defined on line 57; a nested loop is provided (lines 64-65) as a means to access the data in each locally distributed chunk while a stencil based on the input from the user interface is included as a comment (lines 72-77). This is used in the Game of life to sum up the values of the eight surrounding cells.

- A reduction of the end result is provided as a skeleton function in the AppFragments and is used to abstract the MPI reduce function. The MPI_Finalize command is also available for the benefit of the programmer.

The above steps outline how to parallelise the Game of life application from a serial version. MPIGen can be used to assist the user when assessing which decomposition strategy results in greater performance gains. The user can change the input parameters from a Cartesian decomposition to a row decomposition, regenerate the code, overwrite the previously generated code and execute the parallelised version. In such a case, the program is then parallelised using a different decomposition strategy without any code modifications as the function invocations remain the same. Other parameters can also be changed for performance purposes such as changing from blocking to non-blocking communication. Again no code modifications are necessary as the original function invocations remain static.

---

2This is not implemented in the generated Wrappers file, but a template is provided in the AppFragments file. The user is required to fill in the appropriate functionality.
6.6 Test Cases

//To use the wrapper functions, the following include statements will need to be added:
#include "mpi.h"
#include "myMPI_MPIWrappers.h"

//The following are function declarations
int gen_fill_structure(struct myMPI_MPIData *mpi_data, struct myMPI_GridData *grid_data,
        struct myMPI_2DGridSpace *grid_space);
int gen_access_data(struct myMPI_MPIData *mpi_data, struct myMPI_GridData *grid_data,
        struct myMPI_2DGridSpace *grid_space);
int gen_gather_result(struct myMPI_MPIData *mpi_data, struct myMPI_GridData *grid_data,
        struct myMPI_2DGridSpace *grid_space);

//The following are struct declarations
struct myMPI_MPIData mpi_data;
//This is a reference to the generated struct containing all necessary MPI variables
struct myMPI_GridData grid_data;
//This is a reference to the generated struct containing all necessary Grid variables
struct myMPI_2DGridSpace grid_space;
//This is a reference to the generated struct containing all necessary array structures

int main(int argc, char *argv[]){
    MPI_Init(&argc, &argv); //Initialize the MPI execution environment
    gen_set_MPI_data(&mpi_data, &grid_data);
    gen_allocate_structure_all(&mpi_data, &grid_data, &grid_space);
    gen_fill_structure(&mpi_data, &grid_data, &grid_space);
    //A skeleton for this function is below

    //**** START OF HYPOTHETICAL Timestep LOOP ****/
    gen_boundaries_swap_2dAccess(&mpi_data, &grid_data, grid_space.local_array);
    //This function exchanges data on the boundaries of the local-sized array passed in
    gen_access_data(&mpi_data, &grid_data, &grid_space);
    //A skeleton for this function is below
    //This function provides a template to access data across the neighbouring cells.
    ....
    ....

    double mpi_time = MPI_Wtime();

    //The following loop provides access to all local cells EXCLUDING ghost cells
    for(i=grid_data->ghost_rows; i<grid_data->innerLastRow; i++)
        for(j=grid_data->ghost_cols; j<grid_data->innerLastCol; j++)
            //The following indices provide access to cells left,right, top and bottom across dimensions
            for(i_left = i-1; i_left < grid_data->innerLastRow; i_left++)
                for(j_bot = j-1; j_bot < grid_data->innerLastCol; j_bot++)
                    //All surrounding cells can be accessed as follows:
                    *grid_space->local_array[i_left][j_bot];

    return 0;
}

Figure 6.5: Extracts from AppFragments file
6.6.3 Performance of Game of Life

The serial version of the Game of life was parallelised manually where MPI statements were explicitly inserted in the codebase directly without the aid of tool support. This manually parallelised version was tested for performance against the MPIGen parallelised version and both applications were implemented using a Cartesian topology with a grid size of $73,728 \times 73,728$ and timestep count of 1000. The parallelised applications were ported to ICHEC’s Fionn Supercomputer (Fionn-Supercomputer 2014) and timing metrics were captured across 96, 192 and 384 cores. Table 6.3 shows the mean times recorded across five executions per core count for both parallelised codebases. The performance difference is minimal (1.4 seconds on 384 cores) which indicates that the MPIGen generated code does not incur significant performance overhead in comparison to manually parallelised code.

<table>
<thead>
<tr>
<th># cores</th>
<th>Average time in seconds</th>
<th>Manual</th>
<th>MPIGen</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>646.6</td>
<td>647.4</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>380.6</td>
<td>381.0</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>384</td>
<td>247.5</td>
<td>248.9</td>
<td>1.4</td>
<td></td>
</tr>
</tbody>
</table>

6.7 Functional Limitations

The MPIGen tool is built as a prototype and does not support the following functionality:

- One sided MPI communication using `get` and `put` operations. Currently only two sided `send` and `receive` operations are supported.
- Multiple-dimensional grids. Currently only two-dimensional grid structures are supported.
6.8 Conclusion

- Irregularly structured grids. Currently only structured regular grids are supported.
- Irregular decompositions. Currently only regular decomposition strategies based on blocks (row, column, Cartesian) are supported.
- Complex grid types in the form of structs containing various datatypes. Currently only primitive types are supported.
- More complex communication stencils. Current support covers star and cross stencil-based communication.

Additional functionality will be addressed in a future research plan (see Section 8.4).

6.8 Conclusion

MPIGen is a prototype tool implemented to automatically generate the MPI calls required for performing data decomposition and the necessary communication when porting serial applications to high performance environments. The design of the tool is founded on requirements derived from both practitioner studies and the state-of-the-art (see Table 6.1). The functionality of the tool was tested using three applications, one of an industrial size. Chapter 7 describes an evaluation of MPIGen which was conducted to provide validation that the tool successfully meets all the derived requirements.
7

Evaluation of MPIGen

7.1 Introduction

Chapter 6 outlined the design and implementation of a prototype tool MPIGen, which was developed to support the requirements derived from the empirical studies outlined in Chapter 4, as well as the comparative analysis literature review in Chapter 5. This chapter describes the process used to evaluate this prototype tool and presents an analysis of the findings. The following research question, previously outlined in Chapter 3, is addressed in this chapter:

RQ5: Does a tool that fulfils the derived requirements, effectively support the task of data decomposition and communication?

The aim of the evaluation is twofold: firstly to assess the extent to which the derived requirements are supported and secondly, to determine whether the support offered is effective in the task of performing data decomposition and communication when parallelising applications. The evaluation study was conducted in two phases; Phase I used HPC practitioners as subjects and gathered data in an interview and questionnaire, evaluating the tool against the derived requirements and a list of usability criteria. Phase II used students as subjects in an experimental setup and evaluated the tool against the same usability criteria as in Phase I. A justification for the research methods used in these phases was provided in Chapter 3.
7.2 Evaluation Overview

This chapter is structured as follows: a high-level overview of the study including the evaluation structure is presented in Section 7.2 while Section 7.3 expands upon the study design for Phase I. An assessment of the support offered for the derived requirements is described in Section 7.4 and an evaluation of the usability of the tool from the perspective of experts is outlined in Section 7.5. Other insights gained from Phase I are presented in Section 7.6. The study design for Phase II is outlined in Section 7.7, while Section 7.8 presents the usability findings from the perspective of students. An aggregation of the usability findings from both phases is presented in tabular format in Section 7.9, followed by a discussion in Section 7.10 and a review of the threats to validity in Section 7.11.

7.2 Evaluation Overview

Kitchenham et al. (1997) defines levels of software validation when evaluating tools; one of which involves assessing if the tool has satisfied specific criteria which are considered to add gain, such as requirements. Data were collected from HPC practitioners in Phase I of this evaluation to assess whether the following derived requirements are sufficiently supported:

- **R1**: MPI focused.
- **R2**: Automated grid indexing.
- **R3**: Data structure and communication generation.
- **R4**: Ease of strategy conversion.
- **R5**: Low application code impact.
- **R8**: Minimal learning curve.

Four derived requirements (R6, R7, R9, R10) are not assessed in this evaluation as sufficient evidence has been provided to indicate that these are supported. Requirement R6 states the tool should initially target structured grids. MPIGen was successfully used to parallelise three applications based on structured grids (see Chapter 6). The JFLOW® application was one of the three applications and involved global boundary processing as well as local boundary processing. MPI-
Gen successfully generated parallelised code to handle these conditions and so Requirement R7 which requires flexibility with regards to global boundary operations was shown to be supported. Requirement R9 emphasises the need for non-proprietary runtime support. This requirement is met as the tool produces code which can execute in any environment supporting MPI as demonstrated in the test case scenarios in Chapter 6. Requirement R10 states the tool should have the ability to be extended to support multiple languages. As outlined in Section 6.3, there are no foreseeable obstacles to extend the tool to support other languages for which MPI implementations exist (e.g., Fortran) and so this requirement is also considered to be supported.

Nielsen (1993, p.25) describes the utility of a system as a question of ‘whether the system can do what is needed’, in other words whether the system satisfies the proposed requirements, while he defines usability as a question of ‘how well users can use that functionality’. As prior experience in the field of HPC was considered a prerequisite to evaluate the level of support offered to fulfil the requirements, the utility of the system is evaluated in Phase I using experts. Nanz et al. (2013b) claim it is important to conduct empirical studies involving human subjects when assessing the usability of programming abstractions. The usability of MPIGen was evaluated using both experts in Phase I, as well as students in Phase II.

The usability criteria used in the evaluation stem from Nielsen’s list of usability components as follows (Nielsen 1993, p. 26):

- **Learnability**: the system should be easy to learn, so that a user can commence work quickly.

- **Satisfaction**: the system should be pleasant to use, so that users are subjectively satisfied when using it.

- **Errors**: the system should have a low error rate, so that users make few errors when using it and can easily recover from the errors that they do make.
7.2 Evaluation Overview

- **Efficiency**: the system should result in a rise in productivity once the user has learned how to use the system.

Nielsen also includes a fifth component in this list – Memorability, stating a system should be easy to remember so that a user can use the system after a period of inactivity and remember how it works. Measuring this attribute is challenging as it requires a significant study timeframe, as well as participants who are willing to be involved over this duration (Hornbæk 2006). The limited availability of participants and the time constraints of this research made it infeasible to measure the memorability of the tool, and so the usability is evaluated based on four of

![Figure 7.1: Evaluation Structure](image_url)
Nielsen’s five usability components. Figure 7.1 illustrates the structure of this evaluation in the context of both study phases.

7.3 Study Design – Phase I

Five experts in the field of HPC were targeted for this evaluation. Three of these experts (who are referred to as P1-P3) are based in ICHEC in Dublin, Ireland. Section 4.2 provides details regarding the ICHEC organisation. The evaluation involving P1-P3 was conducted in a group setting, face-to-face with the three participants and is referred to as Session I.A. The fourth participant (P4) is based in the Southern California Earthquake Centre (SCEC) in the USA. The core business of SCEC is to develop a comprehensive understanding of earthquakes. To achieve this goal, software developers work in conjunction with seismologists to run simulations in high performance computing environments. This evaluation was conducted remotely using video conferencing software (Vidyo 2013) and will be referred to as Session I.B. The final participant (P5) is based in the Scientific Computing Department in Wayne State University (WSU) located in Michigan, USA. The Scientific Computing Department manages research related projects which execute on an in-house high performance Grid enabled computing system. This final evaluation was done face-to-face with the participant and is referred to as Session I.C. The following tasks were conducted as part of all three sessions:

Profile Questionnaire: a survey was distributed in order to gather profile information about the participants such as experience in HPC, number of applications parallelised and tools used to assist decomposition and communication when porting MPI based applications. This was filled out as an initial task, before the tool was demonstrated. In Session I.B, the participant chose to complete this questionnaire offline, before the evaluation session began.
7.3 Study Design – Phase I

**Serial GoL Demo:** as the demonstration was based on the Game of life code, an initial walkthrough of the serial code was presented.

**MPIGen Demo & Code Step Through:** the serial Game of life application was parallelised by the researcher using the MPIGen tool in a live demonstration in order to showcase the tool’s functionality. As this parallelisation does not demonstrate the entire functionality of MPIGen (e.g. custom matrix datatypes, external boundaries etc. are not required), the tool was shown from two perspectives — firstly highlighting the functionality required to parallelise the Game of life, and secondly, highlighting the functionality not necessary to parallelise the Game of life code but available in the tool. Table 7.1 outlines the input options in the MPIGen tool and their applicability to the Game of life application. The demonstration of MPIGen presented the following:

- Input parameters (specific to Game of life and otherwise).
- Customisation options.
- Step through of generated code (specific to Game of life and otherwise).
- Parallelisation of Game of life using appFragments file and template functions.
- Changing of an implemented decomposition strategy.

The participants were free to interrupt the demonstration and ask questions/seek clarification/make observations. Time was allowed for browsing the code generated by MPIGen. As Session I.B was conducted remotely where the researcher and participant were in different physical locations, the researcher’s computer screen was shared with the participant still enabling a live demonstration of the tool.

**Evaluation Questionnaire:** a detailed survey was distributed in order to solicit feedback regarding the level of support offered by MPIGen for the derived requirements listed in Section 7.2, as well as the support provided for usability (see Appendix E). Participants in Session I.A filled in the questionnaire during the allocated evaluation timeslot, while participants in Sessions I.B and I.C conduc-
Table 7.1: MPIGen Input Parameters applicable to Game of Life

<table>
<thead>
<tr>
<th>Input Parameter</th>
<th>Applicable to Game of Life</th>
</tr>
</thead>
<tbody>
<tr>
<td>File prefix</td>
<td>✓</td>
</tr>
<tr>
<td>Function prefix</td>
<td>✓</td>
</tr>
<tr>
<td>Show function parameters</td>
<td>✗</td>
</tr>
<tr>
<td>Grid size on X axis</td>
<td>✓</td>
</tr>
<tr>
<td>Grid size on Y axis</td>
<td>✓</td>
</tr>
<tr>
<td>Grid data type</td>
<td>✓</td>
</tr>
<tr>
<td>Matrix storage layout</td>
<td>✓</td>
</tr>
<tr>
<td>Decomposition type</td>
<td>✓</td>
</tr>
<tr>
<td>Ghost row count</td>
<td>✓</td>
</tr>
<tr>
<td>Ghost column count</td>
<td>✓</td>
</tr>
<tr>
<td>External boundary row count</td>
<td>✗</td>
</tr>
<tr>
<td>External boundary column count</td>
<td>✗</td>
</tr>
<tr>
<td>Blocking communication</td>
<td>✓</td>
</tr>
<tr>
<td>Cyclic communication</td>
<td>✓</td>
</tr>
<tr>
<td>Reorder ranks</td>
<td>✗</td>
</tr>
<tr>
<td>Generate a custom matrix datatype</td>
<td>✗</td>
</tr>
<tr>
<td>Allocate grid to root</td>
<td>✗</td>
</tr>
<tr>
<td>Customisation of variables</td>
<td>✗</td>
</tr>
</tbody>
</table>

ted this offline (outside the evaluation timeslot). The researcher had the option to email both participants once the questionnaire was received, and query any answers that needed clarification.

Focus Group/Interview: a discussion was then conducted which focused on the participants’ general thoughts on MPIGen usage and the tool’s potential target audience. This was conducted as a focus group in Session I.A and semi-structured interviews in Sessions I.B and I.C. The participants were encouraged to offer
7.3 Study Design – Phase I

insights/views not covered in the questionnaires and in the case of Session I.A, clarification was sought by the researcher regarding the returned questionnaires. This clarification helped the researcher to ensure all answers were legible and to ask for further elaboration where necessary. For example, in answer to the question asking whether the input values sufficiently captured the parallel concerns of the programmer, a participant answered ‘Yes for pretty general cases’. The clarification sought in this example, was to explain what constituted a ‘pretty general’ case.

Table 7.2: Timing Record

<table>
<thead>
<tr>
<th>Task</th>
<th>Duration in Minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I.A</td>
</tr>
<tr>
<td>Profile Questionnaire</td>
<td>10</td>
</tr>
<tr>
<td>Serial GoL Demo</td>
<td>10</td>
</tr>
<tr>
<td>MPIGen Demo &amp; Code Walkthrough</td>
<td>70</td>
</tr>
<tr>
<td>Evaluation Questionnaire</td>
<td>35</td>
</tr>
<tr>
<td>Focus group/Interview</td>
<td>25</td>
</tr>
<tr>
<td><strong>Total time</strong></td>
<td>150</td>
</tr>
</tbody>
</table>

*Not inclusive of offline timings

Table 7.2 breaks down the timings for these three Sessions. In the case of Sessions I.B and I.C, the participants conducted certain tasks before and after the session rather than during and so the associated timings were not captured. Unrecorded timings are labeled as not applicable (n/a) in the table and the total calculated timings are not inclusive of these. Session I.A took 150 minutes to complete, while sessions I.B and I.C took 80 and 100 minutes, respectively, excluding certain offline tasks.

7.3.1 Participants

In the Oxford dictionary of English (Dictionary 2013), the noun *expert* is defined as:
A person who is very knowledgeable about or skilful in a particular area.

The participants involved in this study have ported at minimum three applications from serial to parallel and have an average experience of 5.6 years ranging from one to nine years in HPC. Simon (1996, p.91) claims that a person requires approximately a decade to reach top professional proficiency in any domain. This includes learning, as well as practising the domain and so educational experience is considered relevant. However, Brown (1968) claims that verification of expertise is much more complex than counting years of experience as status among peers, amount of accessible information and a priori judgement factors can all contribute. Table 7.3 outlines the participants’ profiles including the number of years’ experience in HPC and the number of applications parallelised. The diversity of professional proficiency of all five participants enhances the evaluation as it contributes to a broader range of perspectives. All participants consented to have their data used in this research and were fully aware that the sessions were being

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Org.</th>
<th>Id.</th>
<th>Title</th>
<th>HPC experience (years)</th>
<th>Number of ports completed</th>
<th>Average port duration (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>ICHEC</td>
<td>P1</td>
<td>Computational scientist</td>
<td>9</td>
<td>10+</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P2</td>
<td>Computational scientist</td>
<td>6</td>
<td>5</td>
<td>3.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P3</td>
<td>Computational scientist</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>LC</td>
<td>SCEC</td>
<td>P4</td>
<td>HPC software developer</td>
<td>9</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>LC</td>
<td>WSU</td>
<td>P5</td>
<td>HPC research scientist</td>
<td>3</td>
<td>10</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Table 7.3: Participants’ Profile
7.4 Evaluation of the Requirements – Experts

recorded. The participants were informed that all data would be anonymously used.¹

The profiling questionnaire queried the percentage of time typically taken to perform data decomposition in parallelisation projects. The answers averaged 29% of time with a standard deviation of 19%. The participants were also requested to estimate the percentage of time spent devising communication calls in MPI based projects and the average of their answers was 16% with a standard deviation of 8%. To correlate with the findings from the empirical studies described in Chapter 4, the participants were also asked whether tools were used to assist data decomposition and communication when parallelising MPI based projects; the answer was a unanimous no. The participants were then asked to elaborate on why they do not use tools and the following are some of their answers:

*I don’t know any tools for this, I write from scratch using MPI manuals* — P2

*To be useful to me, I think such tools would have to either provide a way for me to specify my data structures … ideally it would produce code, which I could then make small changes if necessary* — P4

7.4 Evaluation of the Requirements – Experts

This section describes findings from the experts in Phase I and establishes the extent to which MPIGen supports the derived requirements R1 - R5 and R8.

7.4.1 R1: MPI Focused

The expert participants were asked to rate the importance of retaining MPI as an output. As shown in Table 7.4, two participants listed this as very important while three listed this as important. Participant P1 elaborated on this importance in the

¹The Science and Engineering Research Ethics committee in the University of Limerick approved the application for this study and is included in Appendix D.
focus group session claiming the portability achieved in an MPI based solution is preferable:

*You want to do changes on the MPI level, so need to see the MPI. Very few people look for options that are not portable.*

Participant P4 also discussed the benefits of generating MPI code, adding that the maintainability of a codebase is a significant consequence:

*It’s clean code, so it’s easy for someone to come in and maintain. It’s better than custom written stuff as it’s easier to read and to follow.*

Participant P5 reiterated this maintainability sentiment when discussing the tool:

*Code reusability for maintenance is improved through this implementation.*

The fact that MPIGen generates and allows access to MPI based code was considered to be an important feature by all five experts. The participants stated the portability, maintainability and reusability of a parallelised code would increase if based on MPI. This indicates strong support for the MPI focused requirement.

<table>
<thead>
<tr>
<th>Evaluation property</th>
<th>Very important</th>
<th>Important</th>
<th>Moderately important</th>
<th>Of little importance</th>
<th>Not important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintaining MPI output</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 7.4.2 R2: Automated Grid Indexing

The participants were asked to rate the ease of index calculation when using MPIGen. Table 7.5 outlines their answers. Two participants rated the calculation of indices as *very easy*, two participants rated this as *somewhat easy* and one
7.4 Evaluation of the Requirements – Experts

participant (P2) provided a neutral answer, rating it as not easy, not difficult. In a follow-on comment, Participant P2 claimed that all possible communication patterns were not supported by the tool and so the calculation of indices in certain applications would still be manual. Participant P4 spoke of indexing in the focus group when talking of the Game of life application:

> You already know the Game of life yourself, and you want something more complex but don’t want to write all the indices, it [MPIGen] becomes handy for that.

This appreciation of the automated grid indexing indicates that MPIGen provides appropriate support for requirement R2.

<table>
<thead>
<tr>
<th>Evaluation property</th>
<th>Very easy</th>
<th>Somewhat easy</th>
<th>Not easy, not difficult</th>
<th>Somewhat difficult</th>
<th>Very difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of index calculation</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 7.4.3 R3: Data Structure & Communication Generation

Table 7.6 outlines the ratings proffered when asked how useful the tool is for performing boundary operations based on the Game of life demonstration. Three participants claimed it was very useful while two rated this as useful. P3 listed this automation of data structures as a key advantage of the tool:

> Abstracts data transfers away from the end user. Easier to do halo swaps.

Other participants reiterated this sentiment when speaking of advantages stating:

> You don’t have to spend time in the MPI function syntax and usage —P2
Chapter 7. Evaluation

*It automatically sets up decomposition data structures for you, which is usually tedious and easy to make a mistake — P4*

These quotes represent empirical evidence that the requirement of data structure and communication generation is realised and provides benefit.

**Table 7.6: Expert Boundary Operations Ratings**

<table>
<thead>
<tr>
<th>Evaluation property</th>
<th>Very useful</th>
<th>Useful</th>
<th>Moderately useful</th>
<th>Of little use</th>
<th>Not useful</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performing boundary ops</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 7.4.4 R4: Ease of Strategy Conversion

The participants were asked to characterise the effort required to convert from one type of decomposition to another, for example, changing from a row topology to a Cartesian topology. As can be seen in Table 7.7, all five participants rated this effort as very easy. Participant P2 expressed this feature of the tool as very useful:

> Performance analysis is important for us. To check for performance, the data decomposition is important. This tool is useful as we can change the topology and can then check the performance.

Participant P1 described the decomposition as *relatively easy to change* using the tool, stating that it is *usually not that trivial to do.* This feature is one of the

**Table 7.7: Expert Decomposition Conversion Ratings**

<table>
<thead>
<tr>
<th>Evaluation property</th>
<th>Very easy</th>
<th>Somewhat easy</th>
<th>Not easy, not difficult</th>
<th>Somewhat difficult</th>
<th>Very difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effort to change Decomposition</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
core contributions of MPIGen, as altering an already implemented decomposition strategy can be achieved without the need to modify the invocations in the parallelised code. The user need only change the input parameters and regenerate the wrapper code.

### 7.4.5 R5: Low Application Code Impact

In order to reduce application code impact, parallelisation should not involve invasive rewriting of the serial application and should allow for some customisation of the generated output. The participants were asked to rate the relevance of the customisation of variable names. As can be seen in Table 7.8, four participants rated it as very relevant to relevant. One rated it as of little relevance. When probed in the follow-on interview, the participant added that he would like to see an option on the web based interface, which would allow for automatic identification of the serial variables rather than having to manually input these. When discussing the functionality of the tool, the participants claimed the ability to perform customisation was very useful, participant P4 elaborated as follows:

> I like that I can customise the file function and variable names.

The structure provided by the AppFragments file was also commended, as a programmer can use this demonstrative file as a guideline and invoke the wrapper functions accordingly in their serial applications. Participant P4 described the structure provided by the appFragments file as follows:

> It provides a framework in which you just have to fill in the application-specific details, which I think is useful to everyone.

The customisation of the variables to reflect the serial application’s domain along with the structure provided by the appFragments file are aspects which contribute to a low application code impact when producing a parallel solution. Table 7.18 in Section 7.8.4 outlines the difference in lines of code typed in correct solutions
Table 7.8: Expert Customisation Ratings

<table>
<thead>
<tr>
<th>Evaluation property</th>
<th>Very relevant</th>
<th>Relevant</th>
<th>Moderately relevant</th>
<th>Of little relevance</th>
<th>Not relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customisation of variable names</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

submitted by the students using the tool and not using the tool in Phase II. These numbers add further support to the low code impact claim.

7.4.6 R8: Minimal Learning Curve

To address Requirement R8, the learnability of the tool was assessed, however, there is a lack of consensus in the literature as to how to measure and evaluate learnability (Grossman et al. 2009; Hornbæk 2006; Seffah and Metzker 2004). In his description of learnability, Nielsen (1993, p. 27) highlights the importance of a novice user’s initial experience when using a system and so the experts were asked to rate the learning curve if using the tool for the first time. Rafique et al. (2012) claim learnability should be assessed on a more detailed level and devised a list of characteristics constituting a product’s learnability including the following:

- **Interface Understandability**: the capability of the software product to enable the user to understand the overall interface in order to help the user to perform his tasks.

- **Operational Momentum**: the degree to which the software helps the user to guide on to the next stage.

These characteristics were also used as a measure of the support provided for Requirement R8. To assess Interface Understandability, the participants were asked to rate their experiences inputting values into the tool. As use of the tool involves the invocation of the functions in the generated code, the participants were also asked to rate the understandability of the generated code. To address
7.5 Evaluation of the Requirements – Experts

Operational Momentum, the participants were required to rate the usability of the tool on a Likert scale of very easy to very difficult.

Table 7.9 illustrates the findings. Two participants rated the learning curve as very easy, two rated it as somewhat easy and one (P2) rated it as neutral. With regards to understandability of the generated code, one participant rated it as very easy to understand, two rated it as somewhat easy and two rated it as neutral. The participants who rated the understandability as not easy, not difficult outlined that knowledge of MPI was necessary and so impacted the understandability. With regards the usability, one participant rated it as very easy, three rated it as somewhat easy and one (P2) rated it as neutral. Table 7.9 shows that four participants rated their experience inputting values to the tool as very to somewhat clear, while one (P2) rated it as neutral. When asked to elaborate on the neutral ratings entered for all four questions, Participant P2 stated that knowledge of the serial code can impact the learnability of the tool as a user must be aware of the parallel intent of the serial application in order to understand how to use the tool.

The ratings are consistent across all four properties and indicate that the requirement of a minimal learning curve is supported.

Table 7.9: Expert Characteristics determining Learnability

<table>
<thead>
<tr>
<th>Evaluation property</th>
<th>Very easy/clear</th>
<th>Somewhat easy/clear</th>
<th>Not easy/clear, not difficult/complex</th>
<th>Somewhat difficult/complex</th>
<th>Very difficult/complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Curve</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Understandability</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Usability</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Experience inputting values</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
7.5 Evaluation of the Usability Components – Experts

This section evaluates the tool according to Nielsen’s usability components that were outlined in Section 7.2.²

7.5.1 Satisfaction

The participants were not asked to explicitly rate their satisfaction with the tool, but were instead asked to speak of their experiences using the tool in general in the follow-on interviews. The following quotes suggest the participants’ satisfaction with the tool:

*I like the idea of the tool* — P1

*I like the capability of the tool … having something like this would be a benefit* — P4

*I don’t see any disadvantages* — P2

*[The tool] maximises innovation opportunity* — P5

7.5.2 Errors

The participants were asked if using the tool would increase or decrease index coding errors, communication related errors and decomposition related errors.³ The participants had the choice of increase, decrease or not applicable (n/a) as their answer. Table 7.10 illustrates these answers. A down arrow (↓) represents a decrease and an up arrow (↑) represents an increase. As can be seen, all five

²Learnability was addressed in subsection 7.4.6 when discussing requirement R8 of minimal learning curve and so is not included in this section.

³As the participants did not use the tool in a parallelisation exercise, it was not possible to categorise the experienced errors, as is done in the student experiment in Section 7.8.3.
participants claimed the use of MPIGen would decrease communication and decomposition related errors. Four of the five participants reported that it would decrease index coding errors. This predicted reduction in errors suggests that the tool would result in less error-prone parallelisation efforts. Participant P5 emphasised this benefit when he stated that the minimisation of potential allocation errors was the tool’s ‘most prolific advantage’.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index coding</td>
<td>↓</td>
<td>↓</td>
<td>n/a</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>Communication</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>Decomposition</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
</tbody>
</table>

7.5.3 Efficiency

The participants were queried whether use of MPIGen would change their overall development time. Table 7.11 outlines their answers. Four participants claimed it would result in a somewhat quicker development time, while one participant reported this as a much quicker development time. When discussing the most beneficial functionality in MPIGen, Participant P1 spoke of the reduced overhead resulting from the tested skeletons (template functions in AppFragments):

*The tested skeletons of communication routines and data decomposition reduce the overhead that can arise from testing and debugging these parts of the code.*

Participant P5 spoke of the potential of MPIGen:

*MPIGen can be the start of a much larger development towards using MPI with greater efficiency.*
Participant P3 acknowledged that copying from the AppFragments file may delay the development time:

*Danger of missing some part of the code when copying from appFragments.*

Participant P2 stated that the generated data decomposition code would be efficient for solving problems based on partial differential equations, while Participant P1 described its use for testing an application:

*If I want to develop an application to test something like a specific algorithm, time to do the data distribution could be a lot higher, testing the indices and distribution, this [MPIGen] could be used as it’s easy to set up.*

These findings suggest that use of the tool would result in a rise in productivity.

<table>
<thead>
<tr>
<th>Evaluation property</th>
<th>Much quicker</th>
<th>Somewhat quicker</th>
<th>Not quicker, not slower</th>
<th>Somewhat slower</th>
<th>A lot slower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development time change</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 7.6 Other Insights from Phase I

The participants were asked whether the tool would be applicable for typical parallelisation projects in their organisations. The answers varied. Three participants answered *some projects* while P1 answered *none*. Participant P3 answered by stating that he *didn’t know* as he was recently recruited to that particular organisation and was not familiar with all their projects. In the semi-structured interview, Participant P1 was asked to elaborate on his answer. He responded by describing the projects he works on as being *very complex applications* which
typically have evolved over the last decade or more. Parallelising such projects is a matter of ‘patching things’ rather than implementing a solution from scratch. For this reason, MPIGen is not suitable. Participants P2, P4 and P5 outlined why MPIGen did not suit all applications stating a lot of the projects were hybrid codes and some were based on the Fortran language which MPIGen currently does not support. Participant P4 stated that some of their projects use data which is not currently supported such as complex structures.

When asked what level of programmer would use MPIGen, Participant P4 claimed that potential users should be novices or intermediates. He stated that if the tool supported three-dimensional grids and multiple values of the same type, it could be very useful for all levels of user. Participant P1 advised against targeting a HPC novice programmer, as the tool hides too much complexity and a novice would not learn the MPI specification. He stated the target user base should be those who have an intermediate level of MPI programming ability:

*If you’ve used MPI seriously a few times, then you know the libraries. The tool would suit someone in between beginner and expert, someone in the learning process but who knows a bit.*

Participant P5 agreed that it may be disadvantageous to use without any knowledge of MPI:

*May be easy to get carried away in using the tool without having hardcore knowledge of MPI . . . up to the user to be responsible to know how to navigate the underlying functionality.*

Participant P5 also saw some benefit for advanced programmers stating:

*If you know the structure of your problem and it can fit to the palette of the decomposition techniques you have, it’s a good tool for advanced users as well.*
7.7 Study Design – Phase II

Phase II was conducted as an experiment, which can be defined (Sjøberg et al. 2005) as that in which:

individuals or teams (the experimental units) conduct one or more software engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages, or tools (the treatments).

In Phase II, the researcher had control over the conditions in which students performed two different exercises in two sessions. One exercise involved the parallelisation of a test application using MPI and the other involved the parallelisation of the same test application, again using MPI, but with the added assistance of the MPIGen tool. These exercises are referred to as A1 and A2 respectively. The students were randomly divided into two groups where half the students were allocated A1 as the first exercise, while half were allocated A2. This ordering was then reversed for the second session so as to balance any learning advantage gained from doing one exercise before the other. The term manual is used to describe A1 as the students did not use parallelisation-related tool support and were required to write explicit MPI calls using a basic C text editor. Table 7.12 summarises these exercises.

Table 7.12: Student Parallelisation Exercises

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Manual parallelisation of the test application.</td>
</tr>
<tr>
<td>A2</td>
<td>Parallelisation of the test application using MPIGen prototype tool support.</td>
</tr>
</tbody>
</table>

A questionnaire was distributed to the students once both exercises were completed and data were gathered using a mixture of qualitative and quantitative techniques. The questionnaire used is included as Appendix F. The data from both the experiment and the questionnaire were analysed to evaluate the usability
components: Learnability, Satisfaction, Errors and Efficiency. To further assess the Errors component the following research hypothesis was defined:

- **H₁**: the number of correct solutions submitted when using the MPIGen tool (A2) is greater than the number of correct solutions submitted when not having used the MPIGen tool (A1).

A metric representing correctness was recorded in both exercises. Correctness is a dichotomous variable (1 or 0) measured in terms of whether or not the final parallelised program encompasses the required functionality and outputs a correct answer. To determine support for H₁, the following null hypothesis was tested:

- **H₀₁**: the number of correct solutions submitted when using the MPIGen tool (A2) is equal to the number of correct solutions submitted when not having used the MPIGen tool (A1).

To assess the Efficiency component and determine whether using MPIGen has a measurable effect on programmer productivity, the following two research hypotheses were defined:

- **H₂**: the time taken to perform the parallelisation exercise when using the MPIGen tool (A2) is less than the time taken to perform the parallelisation exercise when not having used the MPIGen tool (A1).

- **H₃**: the effort expended on performing the parallelisation exercise when using the MPIGen tool (A2) is less than the effort expended on performing the parallelisation exercise when not having used the MPIGen tool (A1).

To determine support for H₂ and H₃, metrics representing timing and effort were recorded and the following null hypotheses were tested:

- **H₀₂**: the time taken to perform the parallelisation exercise when using the MPIGen tool (A2) is equal to the time taken to perform the parallelisation exercise when not having used the MPIGen tool (A1).
• $H_{03}$: the effort expended on performing the parallelisation exercise when using the MPIGen tool (A2) is equal to the effort expended on performing the parallelisation exercise when not having used the MPIGen tool (A1)

Timing refers to the length of time the participants took to perform the parallelisation exercises. Only correct submissions are included in this calculation. Measuring programmer effort objectively is a difficult task. For this evaluation, it is measured as a combination of two simple metrics; the time taken to complete a correct solution and the amount of lines of code (LoC) present in a correct solution.

Höst et al. (2005) claim that the level of experience of participants in an experiment can be a key factor affecting the results. He proposes a classification of participants and differentiates between undergraduate students and graduate students (both with less than three months of recent\(^4\) industrial experience). As the students who participated in the evaluation were at both undergraduate and postgraduate levels, with varying industrial experience (the participants’ profiles are elaborated in subsection 7.7.1), a breakdown by the two levels was proposed for the captured metrics. The aim of the breakdown is to determine whether postgraduates are more efficient and correct when using MPIGen than undergraduates, as the latter are less experienced according to the classification proposed by Höst et al. (2005); the following research hypotheses were therefore defined:

• $H_4$: in the case of undergraduate students, the number of correct solutions submitted in A2 is greater than the number of correct solutions submitted in A1.

• $H_5$: in the case of postgraduate students, the number of correct solutions submitted in A2 is greater than the number of correct solutions submitted in A1.

\(^4\)Less than two years ago.
7.7 Study Design – Phase II

- **H₆**: in the case of undergraduate students, the time taken to produce a correct solution in A2 is less than the time taken to produce a correct solution in A1.

- **H₇**: in the case of postgraduate students, the time taken to produce a correct solution in A2 is less than the time taken to produce a correct solution in A1.

- **H₈**: in the case of undergraduate students, the LoC typed in a correct solution in A2 is less than the LoC typed in a correct solution in A1.

- **H₉**: in the case of postgraduate students, the LoC typed in a correct solution in A2 is less than the LoC typed in a correct solution in A1.

To determine support for H₄ - H₉, null hypotheses were defined and are outlined in Section 7.8.5. A hypothesis is rejected if the p-value is less than the significance level of 0.05 (\(\alpha = 0.05\)). The researcher used SPSS version 21 (SPSS 2014) for all the statistical tests.

### 7.7.1 Participants

For this study, students were targeted from both undergraduate and postgraduate courses in the Computer Science and Information Systems (CSIS) department in the University of Limerick, Ireland. The postgraduate students were comprised of a class studying for the degree of Masters of Science in Software Engineering taking the module *Parallelism and Concurrency in Software Development*. This class had 21 students of which all consented for their data to be used in this study. The second class comprised of undergraduate students in their final year taking the *Software Architecture* module as part of a Bachelor of Science degree in one of the following courses: Health Informatics, Computer Systems and Computer Games Development. In a class of 44 students, 43 agreed to participate in this study. In total, 64 students agreed for their data to be used.

As part of the profiling section in the questionnaire, the students were asked about their previous software engineering work experience; the findings are de-
depicted in Figure 7.2. Of the undergraduate participants, 81% had some experience in industry as did 76% of postgraduate students. Of the postgraduate cohort, 33% had more than one year’s experience full time (F.T.) while the undergraduates’ experience was heavily based on internships (65%).\(^5\) The students were asked to state their level of knowledge in the field of HPC in the questionnaire; the results of which are depicted in Figure 7.3. No student claimed to have HPC knowledge\(^5\)

\(^{5}\)The internships conducted at undergraduate level through the University of Limerick are 9 months in duration.
gained from industrial experience and 67% claimed to be beginners. The remaining 33% claimed to have some practical or theoretical experience, however, it is unknown whether this knowledge was solely the result of the course work.

The topic of high performance computing, specifically message passing in a distributed memory environment was introduced to the students in both course modules. The participants were shown the basic concepts of MPI and were exposed to the routines necessary to parallelise the Game of life. To prepare for A2, students were shown a demonstration of the MPIGen prototype tool and were allowed to read the tool’s documentation and browse the generated code. Both A1 and A2 were mandatory exercises to fulfil the requirements of the two modules.

### 7.7.2 Parallelisation Exercises

The study required students to write code to parallelise a cellular automaton called *the Game of life*. Section 6.6.1 provides a description of this application including the rules of the game. The students were provided with a serial implementation of the code (written in the C language) a week in advance of the exercise, and were then required to produce two parallelised versions that could run on 9 processors in the laboratory scenario. The specification for the exercises was broken into tasks (T1 to T7) as outlined in Table 7.14. These tasks did not differ for A1 and A2, but the students were given instructions regarding the different implementations required i.e manual versus tool-assisted. The students were required to note three timing metrics as they worked on both exercises. These were recorded as outlined in Table 7.13.

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time1</td>
<td>After the students had completed the Cartesian decomposition of the grid (T2)</td>
</tr>
<tr>
<td>Time2</td>
<td>After the communication within the time step loop (T5)</td>
</tr>
<tr>
<td>Total Time</td>
<td>On the submission of the exercise (T7)</td>
</tr>
</tbody>
</table>

*Table 7.13: Recorded Timings*
### Table 7.14: Parallelisation Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Allocate appropriate memory structures, Use 1 ghost column and 1 ghost row on both sides of the locally distributed data.</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>Create a Cartesian decomposition to distribute the data:</td>
<td>Time1</td>
</tr>
<tr>
<td></td>
<td>• Set node distribution to default.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Determine the ranks located north, south, east, west, northeast, northwest, southeast, southwest to each process.</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>Distribute the random states (1 or 0) in the same distribution as on the serial grid.</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>Create a column derived datatype for boundary exchanges from west to east and east to west.</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>For each generation (timestep):</td>
<td>Time2</td>
</tr>
<tr>
<td></td>
<td>• Populate ghost boundary areas.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Maintain cyclic communication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Ensure corner values are exchanged.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Calculate the sum of all eight neighbours and determine the living/dead state of the cell updating the temporary array.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Copy the temporary array back into the original array.</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>Perform a reduction using a sum operation where each rank sends its end sum back to the root process (rank==0).</td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>The root process must output the final result of all the summed values.</td>
<td>Total time</td>
</tr>
</tbody>
</table>

Each student was allocated a timeslot of three hours per exercise (A1 or A2) and completed both exercises on two different days. The allocated time totalled 384 hours in total for the 64 students, however, not all students utilised the full time allocated. Both exercises were graded as part fulfilment of course module\(^6\) and bonus marks were awarded for correct submissions parallelised in a quicker time frame. The bonus marks were used as an incentive for students to submit completed exercises as quickly as possible ensuring the recorded final timing.

\(^6\)To fulfil the course module, the students were awarded grades on a scale reflective of their advancement through the exercise requirements. However, for the purpose of this evaluation, only a binary value reflecting the correctness of the end result is relevant.
(Total time) was an accurate reflection of the time taken to produce a working parallelised version. The correctness of the solution was measured by comparing the final output against the expected output (same as answer from serial execution). The lines of code were also recorded for each working exercise i.e. those checked and recorded as a correct solution.

7.7.3 Ethical Considerations

The exercises were mandatory and assessed as part of the course modules. The students had a non-mandatory option to agree or disagree to use of the data collected for research purposes and signed a consent form where in agreement. It was made clear that consent regarding the use of their data would in no way affect the grading of the assessment. The Science and Engineering Research Ethics committee in the University of Limerick approved the application for this experiment as can be seen in Appendix D.

7.8 Evaluation of the Usability Components – Students

This section evaluates the tool according to Nielsen’s usability components (out-lined in Section 7.2) from the perspectives of the students.

7.8.1 Learnability

Justification was provided in Section 7.4.6 to assess learnability taking initial learning curve, understandability of generated code, usability of tool, and experience inputting parameters into account. The same approach is applied here. Figure 7.4 (a) illustrates the answers provided by the students when asked to rate the learning curve of the tool. Of the 64 students, 63% rated the learning curve as easy or somewhat easy, 23% rated it as neutral - not easy/not difficult and 14% rated it as somewhat difficult to very difficult. Data determining the reasoning behind
the chosen ratings were not collected. The students were also asked to rate their experience inputting parameters to the tool. Their answers are depicted in Figure 7.4 (b). The percentage of students who rated their experience inputting values as clear was 89%, 9% rated the experience as neutral and 2% found it somewhat complex. Figure 7.5 (a) depicts the usability ratings provided by the students; 77% rated the usability as very easy to somewhat easy, 16% took a neutral stance rating it as not easy/not difficult and 7% rated it as somewhat to very difficult. Figure 7.5 (b) illustrates that 59% of the students rated the understandability of
the generated code as very easy to easy, 27% rated this as neutral while 13% rated it as difficult to very difficult to understand.

The data show that the majority of students rate the tool as easy to use and rate the generated code as easy to understand. This combined with the findings illustrating the clarity involved when entering input parameters and the ease of an initial learning curve, provide evidence to support the learnability component.

### 7.8.2 Satisfaction

The students were asked if they would choose MPIGen tool if given the choice in their next parallelisation project. The data collected showed that 89% of students would use the tool when given the choice while 11% said they would not use the tool and would instead perform a manual parallelisation (see Figure 7.6). When asked to elaborate on why they would choose one over the other, the students who chose the manual approach said it was better for learning purposes. The following are two of these responses:

*Though frustrating, I prefer the manual from a learning point of view.*

*I prefer to be in control and to know all the steps I am taking, it helps to keep me in practice in case there is not an MPIGen tool available.*

![Figure 7.6: Student Satisfaction](image)

168
Additional comments in the survey supporting satisfaction include the ‘clear reusable functions’ and ‘modifiable code’ generated by MPIGen.

### 7.8.3 Errors

The Errors component of usability is assessed using the data recorded in the experiment which focused on correctness of the two submitted solutions, as well as the qualitative data gathered in the survey where students reported the errors they experienced in both exercises.

**Correctness Attribute**

A submission deemed incorrect is one that: did not compile or did compile but did not execute or did execute but produced an incorrect result. Of the 64 students who consented to participate in this study:

- 17 students (27%) did not submit a correct solution for either A1 or A2
- 25 students (39%) submitted correct solutions for both A1 and A2
- 22 students (34%) submitted a correct solution for only one of the experiments
  - 17 students (27%) submitted a correct solution for A2 but not A1
  - 5 students (8%) submitted a correct solution for A1 but not A2

![Figure 7.7: Participants’ Success Rate](image)
7.8 Evaluation of the Usability Components – Students

Figure 7.7 illustrates the entire student population who partook in the experiment and their success rate in submitting correct solutions. Figure 7.8 depicts the total percentage of students who submitted correct solutions for A1 (8% A1 only + 39% A1&A2) and the total percentage who submitted correct solutions for A2 (27% A2 only + 39% A1&A2).

![Correct Submissions](image)

**Figure 7.8: Correct Submissions**

Correctness Statistical Analysis

Dependent samples can occur when the same subject is measured on two different occasions (Hollander and Wolfe 1999, p. 468). The subject being measured in this case is the correctness of the submitted solutions and the two occasions involve using the MPIGen tool (A2) and not using MPIGen (A1). As each student conducted both A1 and A2, the two occasions are not independent and so the correctness in A1 is paired with the correctness in A2. The McNemar Chi-Square significance test is a test used on paired nominal data and so is appropriate to test the null hypothesis $H_{01}$. As this test is only valid for binary responses, correct solutions were recorded with a 1 value and incorrect solutions were recorded with a 0 value. In total, 64 cases were tested and the result shows a significant
difference in correctness between A1 and A2 ($p<0.05$, $p$-value listed on page 183). Based on this result, $H_{01}$ can be rejected. The correctness of the final result in the parallelised exercise is not equal when having performed the parallelisation using the MPIGen tool and when not having used the MPIGen tool.

**Implication of $H_{01}$**: students are more likely to get a correct solution using MPIGen (A2) than when not using MPIGen (A1).

**Errors Reported**

The students were asked in the questionnaire if they experienced errors when coding either exercise and if so, to elaborate on these. The term *errors* was left open to the students’ own interpretation. When programming A1, 94% of students recorded that they experienced errors in the code, while this was reduced to 47% of students when programming A2. A classification of the root causes of software bugs, devised by Li et al. (2006), is used to illustrate the prevalence of particular types of bugs in either exercise. This classification divides potential errors into three disjoint categories: *Memory*, *Semantic* and *Concurrency* bugs. Memory bugs are described as anything that is caused by improper handling of memory objects. Semantic bugs are those which encompass anything that is inconsistent with the programmers’ intention and Concurrency bugs are those specific to a multi-process environment such as data races or synchronisation errors (Lu et al. 2005). The errors reported by the students do not include any that can be mapped to the Concurrency category due to the fact that the students were required to implement blocking MPI calls and so did not have to worry about synchronisation issues.

Li et al. (2006) break down the Memory and Semantic categories into subcategories and the errors reported by the students are mapped to these. The relevant errors are listed in Table 7.15, including descriptions for each subcategory and snippets of empirical evidence gathered from the returned surveys to support
### 7.8 Evaluation of the Usability Components – Students

Table 7.15: Bug Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Sub-Category</th>
<th>Description</th>
<th>Empirical Evidence</th>
<th>Abbr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Memory Leak</td>
<td>Failures to release unused memory</td>
<td>Segmentation faults</td>
<td>MLK</td>
</tr>
<tr>
<td>Memory</td>
<td>Uninitialised Memory Read</td>
<td>Read memory data before it is initialised.</td>
<td>Dims initialisation</td>
<td>UMR</td>
</tr>
<tr>
<td>Memory</td>
<td>Dangling Pointer</td>
<td>Pointers still keep freed memory addresses.</td>
<td>Mainly to do with pointers in the wrong place</td>
<td>Dangling</td>
</tr>
<tr>
<td>Semantic</td>
<td>Overflow</td>
<td>Illegal access beyond the buffer boundary.</td>
<td>Memory address errors (accessing) in MPI function calls</td>
<td>Overflow</td>
</tr>
<tr>
<td>Semantic</td>
<td>Missing Cases</td>
<td>A case required in the functionality is not implemented.</td>
<td>No call to MPI_Comm_size</td>
<td>MissC</td>
</tr>
<tr>
<td>Semantic</td>
<td>Wrong Control Flow</td>
<td>The control flow is incorrectly implemented.</td>
<td>I had loops in the wrong places</td>
<td>CtrlFlow</td>
</tr>
<tr>
<td>Semantic</td>
<td>Processing</td>
<td>Processing such as evaluation of expressions is incorrect.</td>
<td>I had big trouble with the boundary exchanges dealing with the exchanges from north-west to south-west</td>
<td>Process</td>
</tr>
<tr>
<td>Semantic</td>
<td>Typo</td>
<td>Typographical mistakes</td>
<td>I missed a } in my code</td>
<td>Typo</td>
</tr>
<tr>
<td>Semantic</td>
<td>Other Wrong Functionality Implementation</td>
<td>Any other semantic bug that does not meet the design requirement.</td>
<td>I could not get the program to run as expected</td>
<td>FuncImpl</td>
</tr>
</tbody>
</table>

The mappings. This table does not provide a comprehensive list of Semantic and Memory subcategories as listed by Li et al. (2006), as those errors related to dereferencing of NULL pointers, memory freed twice, missing features and exception handling were not reported by the students and so are omitted from the table. The Process category is described as encompassing errors related to
incorrect expressions. As the researcher anticipated errors related to incorrect expressions when sending or receiving of messages, this category was expanded to also cover communication errors and those related to boundary exchanges that were not specifically described as memory related errors.

Figure 7.9 illustrates the reported incidences of bugs for both A1 and A2. In the manually parallelised exercise (A1), the bugs predominantly reported were those in the following categories: Process (13 incidents), Other Wrong Functionality (12 incidents), Uninitialised Memory Read (10 incidents), Memory Leaks (9 incidents) and Typos (9 incidents). As A1 required the students to manually code the MPI calls, errors related to these calls were the most frequently reported as seen by the large number of incidents in the Process category. A large proportion of errors were vague in their description and have been categorised in the Other
Wrong Functionality category. Errors reported as general segmentation faults were categorised as Memory Leaks. Such leaks and uninitialised memory reads proved problematic in A1. Students also reported errors related to overflowing buffer accesses (6 incidents), missing cases in functionality (5 incidents), dangling pointers (2 incidents), and wrong control flow (1 incident).

The predominant errors in the case of the tool assisted parallelisation (A2), were those categorised as Typos (10 incidents), Other Wrong Functionality (8 incidents) and Process (7 incidents). To code A2, the students had to copy and paste in coding fragments generated from the MPIGen tool. This was the main cause of the high level of syntactical errors. Other errors reported include those categorised as Overflow, Memory Leaks and Control Flow, however these were only reported in three cases or less. The students did not report any Dangling pointer, Missing Cases or Uninitialised Memory Read errors when coding A2.

These findings show that the MPIGen assisted exercise (A2) resulted in a lower error rate when compared to the manual exercise. The errors that were experienced when coding A2 were predominantly syntactical and easier to recover from than the errors experienced when performing A1, which were mainly process related. These findings are reiterated in students’ comments reporting the advantages of the tool:

*It [the generated code] is more reliable than anything I could write.*

*It reduces the occurrence of faults/errors.*

### 7.8.4 Efficiency

The Efficiency component is assessed by testing for timing and effort (LoC combined with timing) attributes. Statistical analyses were performed on the data gathered from the controlled experiment to test the null hypotheses related to these attributes as listed in Section 7.7.
Timing Attribute

Three timing metrics were gathered for both A1 and A2 based on correct submissions. As outlined in Table 7.13, the overall time for completion of the experiment was recorded by the researcher and is referred to as \textit{Total time}. \textit{Time1} refers to the time recorded by the students when declaring and initialising all the appropriate variables needed to perform the parallelisation. This timing also includes the time taken to write the code to perform the Cartesian decomposition. For example, in A1 this would involve appropriate calls to \texttt{MPI\_Cart\_create} and \texttt{MPI\_Dims\_create}. The students recorded this timing while conducting the exercise. \textit{Time2} refers to the time recorded to initialise the dataset and subsequently write the code to perform the boundary exchanges. For example, in A1 this would involve calculating the indices and writing all the MPI calls to send and receive data from neighbours. The students conducting the exercise again recorded this timing.

Timing Statistical Analysis

Table 7.16 lists the descriptive statistics gathered for the six timing metrics recorded. \textit{Num.} refers to the number of correct submissions as the timing is irrelevant for students who did not produce a correct solution. The \textit{Minimum}, \textit{Maximum},

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 Total Time</td>
<td>30</td>
<td>40</td>
<td>180</td>
<td>98.73</td>
<td>37.463</td>
</tr>
<tr>
<td>A1 Time1</td>
<td>30</td>
<td>5</td>
<td>63</td>
<td>30.47</td>
<td>14.364</td>
</tr>
<tr>
<td>A1 Time2</td>
<td>30</td>
<td>10</td>
<td>118</td>
<td>40.77</td>
<td>28.621</td>
</tr>
<tr>
<td>A2 Total Time</td>
<td>42</td>
<td>12</td>
<td>180</td>
<td>54.49</td>
<td>40.477</td>
</tr>
<tr>
<td>A2 Time1</td>
<td>42</td>
<td>1</td>
<td>60</td>
<td>16.10</td>
<td>12.955</td>
</tr>
<tr>
<td>A2 Time2</td>
<td>42</td>
<td>3</td>
<td>125</td>
<td>18.66</td>
<td>19.632</td>
</tr>
</tbody>
</table>
Mean and Standard Deviation are all measured in minutes. As the total time allocated was 180 minutes for each exercise, it is not surprising that the maximum for the total time in A1 and A2 is the full time slot.

In order to test for statistical significance, the dataset had to be analysed for normality. The Shapiro-Wilk test tests whether a random sample \(x_1, x_2, \ldots, x_n\) comes from a normally distributed population (Shapiro and Wilk 1965). Shapiro-Wilks’ tests \((p>0.5)\), along with visual inspection of histograms and box plots were used to determine normality. Figure 7.10 illustrates the distribution of timing metrics through box plot representations for the three timing metrics in both A1 and A2. The time on the horizontal axis is measured in minutes. Since the data was skewed (total time for A1 had a skewness of 0.771 (Standard Error (SE) = 0.464) and a kurtosis of -0.245 (SE=0.902) while total time for A2 had a skewness of 1.163 (SE=0.464) and a kurtosis of 2.023 (SE=0.902)), the researcher could not assume that the data were based on a normal distribution (Doane and Seward 2011).

![Figure 7.10: Timings per Exercise](image-url)
The Wilcoxon signed-rank test was chosen as it is a non-parametric test (applicable to data not based on a normal distribution) used when comparing related samples (Hollander and Wolfe 1999, p. 113). $H_{02}$ asserts that the distribution of the timing data is equal for A1 and A2. Statistical significance was found between the total times recorded for correct submissions of A1 and the total times recorded for correct submissions of A2 ($p<0.01^7$). Similarly using Wilcoxon signed-rank tests, statistical significance was found for the Time1 data distribution ($p<0.01^7$) and the Time2 data distribution ($p<0.01^7$) when comparing A1 and A2. $H_{02}$ can be rejected based on these timing results. The time taken to perform the parallelisation exercise is not equal when using the MPIGen tool and when not using the MPIGen tool.

**Implication of $H_{02}$:** students are more likely to take less time when performing A2 than when performing A1.

To estimate the performance gain, speedup was calculated based on the median timings captured as depicted in Figure 7.11. The median total time was 85 minutes for A1 and 50 minutes for A2. Based on median timings, the speedup achieved when using the MPIGen tool was 1.7 times quicker over manual coding. The median Time1 for A1 was 30 minutes while this was 10 minutes for A2. Speedup of 3 times was realised when initialising variables and coding the decomposition when using MPIGen over manually writing these calls. The median Time2 for A1 was 35 minutes and 15 minutes for A2. Speedup of 2.33 times was realised when coding the boundary exchanges using MPIGen over manually writing the *MPI_Send* and *MPI_Recv* calls. These timings are summarised in Table 7.17.

**Effort Attribute**

$H_{03}$ asserts that the effort expended on performing the parallelisation exercise is equal when using the MPIGen tool and when not using the MPIGen tool. As

---

$^7$ *p*-values listed on page 183.
was outlined in Section 7.7, effort expended is measured as a combination of timing and LoC metrics for correct solutions. Timing analysis was reported in the previous sub-section and this sub-section will report on the LoC analysis.

The lines of code were compared for A1 and A2 in the cohort who completed both experiments successfully. Lines of code does not take into account coding comments or whitespace. A tool called Code Analyzer (Code Analyzer 2013) was used to determine this metric. Table 7.18 outlines the descriptive statistics captured. The LoC are only compared in the case of students who submitted correct solutions for both A1 and A2; in total, 25 pairs of submissions are analysed.

![Figure 7.11: Median Timings](image)
Table 7.18: Descriptive LoC Statistics

<table>
<thead>
<tr>
<th>LoC Metric</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Standard Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>131</td>
<td>188</td>
<td>147.64</td>
<td>13.913</td>
</tr>
<tr>
<td>A2</td>
<td>93</td>
<td>124</td>
<td>104.76</td>
<td>7.429</td>
</tr>
<tr>
<td>Difference: A1 &amp; serial*</td>
<td>+52</td>
<td>+109</td>
<td>+68.64</td>
<td>25.76</td>
</tr>
<tr>
<td>Difference: A2 &amp; serial*</td>
<td>+14</td>
<td>+45</td>
<td>+25.76</td>
<td>7.43</td>
</tr>
</tbody>
</table>

*79 LoC calculated using Code Analyzer

The minimum, maximum, mean and standard deviation were captured for LoC in A1 and A2 as well as for the difference in LoC between A1 and the serial application and the difference between A2 and the serial application.

![Figure 7.12: Lines of Code for both A1 and A2](image)

Figure 7.12: Lines of Code for both A1 and A2
7.8 Evaluation of the Usability Components – Students

**Effort Statistical Analysis**

Box plot representations are sufficient to depict the distribution of the LoC in A1 and A2. As can be seen from the box plots in Figure 7.12, both distributions are not normal. The Wilcoxon signed-rank non-parametric test is used to test for statistical significance (Hollander and Wolfe 1999, p. 113) between the LoC in the A1 submissions and the LoC in the A2 submissions. Significant statistical difference was found between these two distributions ($p<0.01$, $p$-value listed on page 183). $H_{03}$ can be rejected based on both this test and the previously reported timing test. The effort expended on performing the parallelisation exercise is not equal when using the MPIGen tool and when not using the MPIGen tool.

**Implication of $H_{03}$:** students are more likely to expend less effort (less time and less LoC) when using the MPIGen tool than when not using the tool.

### 7.8.5 Analysis According to Student Cohorts

Undergraduate and postgraduate students were also tested as separate groups to determine if use of MPIGen resulted in altered results with regards to the correctness, timing and LoC metrics. These findings are summarised in Table 7.19 where all $p$-values are listed.

**Correctness:** the following null hypotheses were tested to determine the correctness across student cohorts:

- $H_{04}$: in the case of undergraduate students, the number of correct solutions submitted in A2 is equal to the number of correct solutions submitted in A1.

- $H_{05}$: in the case of postgraduate students, number of correct solutions submitted in A2 is equal to the number of correct solutions submitted in A1.

The McNemar Chi-Square was used to test $H_{04}$, see Section 7.8.3 for a justification of the choice of test. The result shows statistical significance ($p<0.01$) in
the undergraduate population when assessing the number of correct solutions submitted for A1 and A2, therefore $H_{04}$ can be rejected.

**Implication of $H_{04}$**: undergraduates are more likely to submit a correct solution when using the tool (A2) than when not using the tool (A1).

The McNemar Chi-Square test was again executed to test $H_{05}$ and the result shows no statistical significance in the postgraduate population ($p>0.05$). $H_{05}$ cannot be rejected.

**Implication of $H_{05}$**: postgraduate students are likely to submit the same number of correct solutions when using the tool (A2) and when not using the tool (A1).

**Timing**: the following null hypotheses were tested to determine the timing across student cohorts:

- $H_{06}$: in the case of undergraduate students, the time taken to produce a correct solution in A2 is equal to the time taken to produce a correct solution in A1.

- $H_{07}$: in the case of postgraduate students, the time taken to produce a correct solution in A2 is equal to the time taken to produce a correct solution in A1.

The Wilcoxon signed rank test was executed to test $H_{06}$, see Section 7.8.4 for a justification of the choice of test. The result shows statistical significance ($p<0.05$) in the undergraduate population and so $H_{06}$ can be rejected.

**Implication of $H_{06}$**: undergraduate students are more likely to perform the parallelisation exercise in a quicker time when using the tool (A2), than when not using the tool (A1).
The Wilcoxon signed rank test was again executed to test $H_{07}$. The result shows statistical significance ($p<0.01$) and so $H_{07}$ can be rejected.

**Implication of $H_{07}$**: postgraduate students are more likely to perform the parallelisation exercise in a quicker time when using the tool (A2), than when not using the tool (A1).

**Lines of Code**: the following null hypotheses were tested to determine the LoC across student cohorts:

- $H_{08}$: in the case of undergraduate students, the LoC typed in a correct solution in A2 is equal to the LoC typed in a correct solution in A1.
- $H_{09}$: in the case of postgraduate students, the LoC typed in a correct solution in A2 is equal to the LoC typed in correct solution in A1.

The Wilcoxon signed rank test was executed to test $H_{08}$, see Section 7.8.4 for details regarding the choice of test. The result shows statistical significance ($p<0.01$) and so $H_{08}$ can be rejected.

**Implication of $H_{08}$**: undergraduates are more likely to type less lines of code when performing the parallelisation exercise using the tool (A2) than when not using the tool (A1).

The Wilcoxon signed rank test was again executed to test $H_{09}$. The result shows statistical significance $p<0.01$ and so $H_{09}$ can be rejected.

**Implication of $H_{09}$**: postgraduates are more likely to type less lines of code when performing the parallelisation exercise using the tool (A2) than when not using the tool (A1).
7.8.6 Summary of Statistical Tests Performed in Phase II

Table 7.19 summarises the statistical tests conducted in Phase II and shows the p-value for each test with a shaded row highlighting the retained hypothesis.

Table 7.19: Hypotheses and Resulting p-values

<table>
<thead>
<tr>
<th>Hyp.</th>
<th>Test</th>
<th>P-value</th>
<th>Decision</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{01}$</td>
<td>McNemar Chisquare</td>
<td>0.017</td>
<td>Reject $H_{01}$</td>
<td>Students are more likely to get a correct solution using the tool (A2) than when not using the tool (A1).</td>
</tr>
<tr>
<td>$H_{02}$</td>
<td>Wilcoxon signed-rank</td>
<td>0.000*</td>
<td>Reject $H_{02}$</td>
<td>Students are more likely to take less time when performing the tool assisted exercise (A2) than when performing the manual exercise (A1).</td>
</tr>
<tr>
<td>$H_{03}$</td>
<td>Wilcoxon signed-rank</td>
<td>0.000**</td>
<td>Reject $H_{03}$</td>
<td>Students are more likely to expend less effort (less time and less LoC) when using the tool (A2) than when not using the tool (A1).</td>
</tr>
<tr>
<td>$H_{04}$</td>
<td>McNemar Chisquare</td>
<td>0.001</td>
<td>Reject $H_{04}$</td>
<td>Undergraduates are more likely to submit a correct solution when using the tool (A2) than when not using the tool (A1).</td>
</tr>
<tr>
<td>$H_{05}$</td>
<td>McNemar Chisquare</td>
<td>0.625</td>
<td>Retain $H_{05}$</td>
<td>Postgraduates are likely to submit the same number of correct solutions when using the tool (A2) and when not using the tool (A1).</td>
</tr>
<tr>
<td>$H_{06}$</td>
<td>Wilcoxon signed-rank</td>
<td>0.041***</td>
<td>Reject $H_{06}$</td>
<td>Undergraduates are more likely perform the parallelisation exercise in a quicker time using the tool (A2), than when not using the tool (A1).</td>
</tr>
<tr>
<td>$H_{07}$</td>
<td>Wilcoxon signed-rank</td>
<td>0.001***</td>
<td>Reject $H_{07}$</td>
<td>Postgraduates are more likely perform the parallelisation exercise in a quicker time when using the tool (A2), than when not using the tool (A1).</td>
</tr>
<tr>
<td>$H_{08}$</td>
<td>Wilcoxon signed-rank</td>
<td>0.002</td>
<td>Reject $H_{08}$</td>
<td>Undergraduates are more likely to type less lines of code when performing the parallelisation exercise using the tool (A2) than when not using the tool (A1).</td>
</tr>
<tr>
<td>$H_{09}$</td>
<td>Wilcoxon signed-rank</td>
<td>0.001</td>
<td>Reject $H_{09}$</td>
<td>Postgraduates are more likely to type less lines of code when performing the parallelisation exercise using the tool (A2) than when not using the tool (A1).</td>
</tr>
</tbody>
</table>

* All three timing metrics: Time1, Time2 and Total_time
** LoC and Total_time
*** Total_time
The first three null hypotheses were rejected as a higher proportion of students submitted correct solutions, conducted the coding exercise in a faster time frame and expended less effort when using the MPIGen tool. When the student group was broken down into undergraduates and postgraduates, significant statistical difference was found for the total time captured and lines of code typed for both student cohorts. Undergraduates were more likely to produce more correct solutions using the tool, however, this was not the case for postgraduates who showed no statistical significance for this variable.

### 7.9 Usability Findings Aggregated

Table 7.20 provides an aggregation of the findings assessing Nielsen’s usability components from Phases I and II of the evaluation. The findings were consistent across both phases of the evaluation. The learnability of the tool was rated as predominantly easy in the expert evaluation and this was reflected in the student ratings regarding usability, learning curve, understandability of generated code and clarity when entering input values. With regards to satisfaction, 89% of students claimed they would prefer to use the tool if given the opportunity. No quantitative data is listed for the experts yet the quotes provided in Section 7.5.1 suggest they also rate satisfaction highly. The experts claimed that using the tool would result in a reduction of errors and this finding is reflected in the student experiment, where it was shown that correctness of a solution is more likely to increase when using the tool. The experts unanimously claimed the tool would lead to a quicker development time, which again is reflected in the student experiment where it was shown that the timing is more likely to improve and the effort more likely to decrease, when using the tool.
Chapter 7. Evaluation

Table 7.20: Findings Summary

<table>
<thead>
<tr>
<th>Usability Components</th>
<th>Phase I: 5 experts (p=number of participants)</th>
<th>Phase II: 64 students (% of total students)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learnability</strong></td>
<td>4p: easy learning curve, 1p: neutral</td>
<td>63%: easy learning curve, 23%: neutral, 14%: difficult</td>
</tr>
<tr>
<td></td>
<td>4p: clear to input parameters, 1p: neutral</td>
<td>89%: clear to input parameters, 9%: neutral, 2%: complex</td>
</tr>
<tr>
<td></td>
<td>3p: easy to understand, 2p: neutral</td>
<td>59%: easy to understand, 27%: neutral, 13%: difficult</td>
</tr>
<tr>
<td></td>
<td>4p: easy to use, 1p: neutral</td>
<td>77%: easy to use, 27%: neutral, 13%: difficult</td>
</tr>
<tr>
<td><strong>Satisfaction</strong></td>
<td>[no quantitative data collected]</td>
<td>89%: prefer to use MPIGen, 11%: prefer manual</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>5p: decrease comm &amp; decomp errors,</td>
<td>H_{01} rejected: Correctness is improved in A2</td>
</tr>
<tr>
<td></td>
<td>4p: decrease index coding errors</td>
<td>47% recorded errors using MPIGen, 94%: recorded errors in manual</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td>5p: quicker development using MPIGen</td>
<td>H_{02} rejected: Timing is quicker in A2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H_{03} rejected: Effort is decreased in A2</td>
</tr>
</tbody>
</table>

7.10 Discussion

The findings in Phase I showed that both Participants P1 and P4 shared similar perspectives on many topics despite coming from two different organisations (ICHEC and SCEC respectively). Both have nine years’ experience in HPC (see Table 7.3), so perhaps this is a reason for their comparable opinions. Both participants set the target user base for the tool at an intermediate programmer level, stating that a novice would have too little knowledge of parallelisation and an expert would have too much. The expert participants unanimously claimed that the abstraction offered by MPIGen limits exposure to the low-level MPI operations and so negatively impacts novices’ learning opportunities. The students who participated in Phase II of the study attended four weeks of MPI classes before partaking in the experiment. They were considered novice parallel programmers.
prior to these classes, however, their high success rate parallelising the code using MPIGen, suggests an advancement in their expertise.

Participants P1 and P4 did have conflicting opinions however, when they were queried about the applicability of the tool for projects in their organisation. Participant P4 saw some use for the tool for applications he had previously worked on, Participant P1 did not. When quizzed on this in the focus group session, P1 stated that he typically works on projects which involve already parallelised codebases and as MPIGen targets the task of converting code from serial to parallel, it is not applicable. The experts reported that MPIGen failed in its need to support languages other than C and C++. As the current tool is a prototype, extending it to support Fortran is part of future work. This corresponds to Requirement R10, which states a tool should have the ability to be extended to support multiple languages.

7.11 Threats to Validity

Within Phase I, the main limitation is the size of the participant set. As Phase II of the evaluation was conducted in vitro, there are a number of validity concerns. These considerations will be addressed in the following subsections.

7.11.1 External Validity

External validity is concerned with the extent to which it is possible to generalise the findings to other settings. In other words, are the findings of this study specific to the studied setting or are they more general? The number of participants in Phase I was limited to five, which is partly due to the limited number of organisations that are involved in porting applications to HPC environments to which the researcher had access. However, as the experience of all five participants was varied, the feedback provided rich insights from different perspectives. Also, as this evaluation was based on two phases, the validity as a whole is increased.
The Game of life application which was the focus of the demonstration in Phase I and the experiment in Phase II can be accused of being a toy problem and not realistic of an application taking months to parallelise. Although the Game of life was used to demonstrate use of the tool, the Phase I sessions did not solely involve this demonstration. The tool’s extensive functionality, beyond that which is adequate for the Game of life, was also shown to the participants. The experts had time to input parameters and generate sample code targeted at more complicated applications. This was done in an attempt to get expert feedback regarding the tool’s entire functionality. A noted limitation to the approach taken is that the experts reported on their feedback in one evaluation session rather than after prolonged use of the tool. Also, their usability feedback was subjective in nature as the participants provided opinion based ratings rather than participated in experiments to test the usability.

Using students as subjects in experiments can lead to a lack of realism (Sjøberg et al. 2002) and may hinder the results of an experiment being transferred from the research community to industry. As students were the subjects in Phase II of the evaluation, a number of measures were taken in an attempt to assess the external validity of the evaluation as a whole. To prepare the students sufficiently, they were required to participate in four weeks of MPI course work (lectures and laboratory practical sessions) before the experiments were conducted. Carver et al. (2010) claim that by integrating a study within the students’ coursework, it can help ensure the subjects have sufficient knowledge about the topic under study making them closer to professionals. The four weeks of coursework in this area helped to instil a basic level of HPC knowledge in the students. As part of the questionnaire distributed in Phase II, the students listed their previous work experience. 80% of the students who partook in the experiment had worked for some period of time in industry, be it as an intern, part time or full time employee (see Figure 7.2). However, even with this sprinkling of industrial experience and with the preceding coursework, the validity of the insights obtained from these students for industrial practice still remains questionable.
7.12 Threats to Validity

Other acknowledged limitations to phase II include the fact that the students were required to record the internal timings (Time1 and Time2) as it was not feasible for the researcher to capture these due to the large number of students participating in the study. Also, the correctness metric was recorded as a 1 or 0 variable representing working or not working. A more in depth study of the students’ progress through the exercise may add greater insights to the study.

7.11.2 Learning effect

The learning effect between two experiments such as A1 and A2, can greatly interfere with the results. For this reason, in Phase II of the evaluation, the participants in each session were broken into two groups where half the students conducted A1, while half conducted A2 in parallel. For the second session, this ordering was reversed. This was done to minimise learning effects as this can greatly affect the results of such empirical studies (Luff 2009).

7.11.3 Reliability

Reliability is concerned with the extent to which data and analysis are dependent on specific researchers (Wohlin et al. 2012, p. 69). A tactic that supports the reliability of our study is triangulation (Runeson et al. 2012, p. 72), namely across data sources as was the case in Phase II (the experiment and the survey). To improve reliability, a peer researcher who has five years of experience conducting empirical studies, was present for the Session I.B evaluation in Phase I as well as for the student based experiments. This researcher also assisted in the review of the submitted solutions in Phase II and provided consultation on the analysis of responses to the questionnaire in Phase I.
Chapter 7. Evaluation

7.12 Conclusion

This chapter presented the studies conducted to evaluate the MPIGen tool. Two phases were carried out in this process; Phase I involved the demonstration of the tool to experts followed by a survey and interviews; Phase II involved students using the tool in a controlled experiment.

Feedback from five HPC practitioners in Phase I, indicated that the evaluated requirements were supported. The MPI focus of the tool (R1) was rated as important, contributing to the portability and maintainability of the generated code. The automated support for indexing (R2) was rated as easy, reducing the complexity involved in devising communication calls. The experts claimed the effort involved in performing boundary operations would be reduced due to the data structure and communication generation (R3). The effort expended to change decomposition strategy (R4) was unanimously described as very easy. The tool’s ability to provide customisation of variable names was described as relevant providing support for low application code impact (R5). The experts rated the learnability of the tool as easy (R8) and claimed the task of entering input parameters was clear. Requirements not included in the evaluation (R6, R7, R9 and R10) were previously shown to be supported in Chapter 6.

Findings from the evaluation in both phases also affirm Nielsen’s usability components as the tool was shown to be easy to learn, lead to a high satisfaction level, result in a lower error rate (compared to not using the tool) and yield higher levels of efficiency.
8.1 Introduction

Computationally intensive applications that continuously generate demand for faster execution rates, have driven developers to utilise multicore environments to meet performance needs. In order to convert an application to run on a high performance architecture, the program must firstly be decomposed into subcomponents. This decomposition presents many challenges. The process of performing data decomposition and the subsequent communication has been the focus of this thesis. This chapter summarises the research presented and outlines suggestions for future work.

The chapter begins by revisiting the research objective and research questions in Section 8.2. This is followed by a summary of the key contributions of this thesis in Section 8.3. This chapter concludes with a number of suggestions for future work in Section 8.4.
8.2 Review of Research Objectives

The main objective of this research was defined in Chapter 1 as follows:

To investigate the specification and implementation of data decomposition as an issue in parallel programming and to address problems programmers face in these tasks.

This objective was broken down into two sub objectives:

1. To evaluate data decomposition and communication as a key challenge in the parallelisation process
2. To derive an empirically based approach to support effective data decomposition and communication.

Five research questions were derived to meet these objectives and discussions were presented on the techniques used and data gathered to address each research question. The first sub objective was fulfilled through Research Questions 1 and 2 while Research Questions 3, 4, and 5 focussed on fulfilling the second sub objective.

8.2.1 Research Question 1

To investigate the prevalence of the challenge of performing data decomposition in practice, exploratory interviews were conducted with parallel programmers from two different organisations. This led to the definition of the first research question as follows:

RQ1: Is data decomposition and communication a real and prevalent challenge that practitioners face?

The analysis of the data gathered from these interviews revealed that performing data decomposition consumed considerable time and effort, particularly when
deciphering message passing operations. To view the parallelisation process from a practitioner’s perspective, the researcher conducted a participant-observer study in IBM’s HPC group and ported a flood-risk management application to a HPC architecture. This was a mentor-driven exercise as expert parallel programmers provided guidance. The task of parallelising the codebase cognisant of the chosen data decomposition strategy proved to be extremely tedious and a significant bottleneck in the porting process, thus substantiated the findings from the exploratory study. The details and findings regarding these studies were reported in Chapter 4.

8.2.2 Research Question 2

Data decomposition emerged as a significant bottleneck, proving to be a real and prevalent challenge. In order to assess the available options to support this task, the following research question was defined:

RQ2: What do practitioners use to support their efforts when performing data decomposition and communication?

A focus group study was chosen as the research method to further explore the confirmed challenge of data decomposition. Experts from a HPC group participated in a session where they spoke of the challenges experienced when decomposing applications and distributing the data. The participants described the ad-hoc methods used to devise an optimal decomposition strategy and expressed their frustration surrounding the manual and error-prone nature of writing communication calls using MPI. The participants expressed the need for three potential tooling solutions, all offering abstractions for performing data decomposition and writing the message-passing calls. The aim of the first proposed solution is to map the decomposition a developer intends to implement and to create the data structures to carry this out. The second proposed solution focuses on generating MPI in order to implement the communication calls based on the decomposition and the third solution aims to assist a user when deciphering an
optimal decomposition strategy. These three proposed solutions are described in detail in Chapter 4.

A survey was subsequently distributed to participants on a global scale to determine whether the results of the focus group study could be generalised to a wider population. Similar challenges were reported and are documented in detail in Section 4.5.2. A key finding confirmed data decomposition as a difficult and time consuming activity, however, the respondents did not use or identify tool support to assist this problem. The surveyed practitioners employed manual approaches such as pen and paper to determine communication calls and the associated data indices.

8.2.3 Research Question 3

Unawareness of tool support was apparent in the findings from the empirical studies conducted when addressing Research Questions 1 and 2. This prompted the definition of the following research question, which focuses on determining tooling requirements for support in this area:

RQ3: What tooling requirements are necessary to support practitioners when performing data decomposition and communication?

Through analysis of the data gathered in the aforementioned empirical studies, a list of requirements was derived for tool support to address the challenge of performing data decomposition and communication when parallelising grid based applications.

8.2.4 Research Question 4

Based on an understanding of the requirements derived from practitioners’ experiences and discussions, the researcher proceeded to investigate the state-of-the-art. Hence, this resulted in the following research questions:
RQ4: What tooling support exists to assist the task of data decomposition and communication?

RQ4.1: Does the currently offered tooling support meet the derived requirements outlined in RQ3?

To investigate the state-of-the-art, a literature review was conducted and a list of tooling options were returned. The identified tools were grouped into five approaches. These included Pattern-based, Generic templates, DSLs (both external and internal), PSEs and Other. Four additional requirements emerged from the literature review.

8.2.5 Research Question 5

A tool named MPIGen was designed and implemented based on the requirements derived to address Research Question 3 and the additional requirements from the state-of-the-art review. This prompted the fifth research question:

RQ5: Does a tool that fulfills the derived requirements, effectively support the task of data decomposition and communication?

To address Research Question 5, a two-phased evaluation study was conducted. In the first phase, MPIGen was demonstrated to experts from three different organisations and a number of interviews were subsequently conducted. The gathered data were then analysed to assess the support offered for the derived requirements and to assess if Nielsen’s usability components were satisfied (Nielsen 1993). The second phase involved students (both undergraduates and postgraduates) and was conducted as a controlled experiment where students parallelised an application twice, with and without MPIGen support. A survey was also distributed to students to assess Nielsen’s usability components.
8.3 Contributions

This section presents the three key contributions of this thesis. The contributions comprise of C1) a list of empirically derived requirements, C2) a characterisation of existing tool support, and C3) a novel tool to support parallel programmers in the task of data decomposition. Each of these contributions is discussed in more detail in the following subsections.

8.3.1 C1: Empirically Derived Requirements

The empirical studies in this research program investigated and characterised the problem of performing data decomposition and communication when parallelising applications. So far, there have been few empirical studies in the field of High Performance Computing focussing on these issues. The state-of-practice was also explored and the use of tools to assist this task was probed. A list of empirically derived requirements was subsequently derived to facilitate data decomposition and communication when parallelising applications. This includes support for MPI based output, automated generation of indices, generation of communication routines and data structures, ease of decomposition strategy conversion, low application code impact, structured grid focus, flexibility when performing global boundary operations, minimal learning curve, non-proprietary runtime and language extensibility.

8.3.2 C2: A Characterisation of Existing Tool Support

The literature review outlined in Chapter 5 identified ten tools developed using different approaches to support data decomposition and communication. The resulting classification can provide the basis of a more detailed taxonomy of tools. A comparative analysis was conducted using the set of derived requirements as comparison parameters and provides insights into the extent to which the current state of the art fulfils practitioners’ requirements. A key finding of the
comparative analysis is that no reviewed tool satisfies the full set of empirically derived requirements.

### 8.3.3 C3: MPIGen Prototype Tool

MPIGen was developed as a prototype tool to support data decomposition and communication by satisfying practitioners’ requirements. The conducted evaluations described in Chapter 7 provide evidence that the current tool adds benefit when parallelising two-dimensional grid structures for message-passing environments. The surveyed experts indicated that support was sufficiently provided to satisfy the derived requirements. The experts unanimously claimed the MPI-focus of the tool and the ease of decomposition strategy conversion was important. The student evaluation showed an increase in the percentage of correct solutions submitted and a decrease in programmer time and effort. The level of abstraction provided by the tool reduces the programmer-effort involved in determining indices and performing boundary exchanges. These findings add to the body of knowledge regarding effective support for data decomposition.

### 8.3.4 Implications for Research

Previous empirical studies in this area have explored the process of parallelisation on a general level (Basili et al. 2008; Hochstein and Basili 2008) and so a focused study on the challenges related to decomposing a program and the related communication exchanges provides previously unrecorded insights. Mattson and Wrinn (2008) argue that researchers need to work on important industry-relevant problems, not fashionable ones. The investigation and characterisation of the task of data decomposition and the challenges associated, outlined in Chapter 4, emphasise the importance of this problem, and provide motivation for the research community to focus their attention on these issues. The list of derived requirements provides useful input for other researchers to develop a research agenda that targets the precise needs of practitioners. The comparative analysis
8.4 Future Work

based on the derived requirements, provides insights to researchers regarding the features and limitations of existing solutions and allows for discussion regarding the potential use of these tools in industry. Researchers can also use the evaluation findings to help direct investigation into further tooling support in this area.

8.3.5 Implications for Practice

This research revealed that practitioners were unaware of existent tool support and so the review of the state-of-the-art can inform practitioners of the potential for tool support in this area. As the tools are grouped under five different approaches, practitioners can focus on the findings that are most relevant to the approaches used in their projects. The list of derived requirements can provide practitioners with a feature list for determining the suitability of the reviewed tool support. Practitioners who are satisfied that MPIGen meets the needs of their organisation, and satisfies their requirements, are free to adopt the tool and produce parallelised applications in a decreased timeframe.

8.4 Future Work

Two main avenues for future work are identified based on the findings presented in this thesis. These generate new research questions and can be described as follows:

- **Exploration of the lack of tool support adoption**: a finding in this research unveiled unawareness on the part of practitioners with regards to available tool support to assist in the task of data decomposition and communication. This research identified a number of tools available in the review of the state-of-the-art, thus providing evidence that tool support does exist. A direction for a future study will involve an investigation into why tools developed as part of research initiatives, fail to advance to a level
Chapter 8. Conclusion

of adoption by practitioners in industry. The following research question is worth investigating:

1. Why do tools in the state-of-the-art not gain traction in industry?

- The expert parallel programmer’s view on using tools: a finding in the evaluation of MPIGen was that the tool was suited to practitioners having intermediate HPC experience. The experiment conducted in Phase II of the evaluation study revealed the tool hindered the MPI learning process of novice parallel programmers. The scope of this research program did not allow for in-depth investigation into the needs of experts in comparison to the needs of intermediates and novices and their attitudes towards tool support in general. A direction for future work is to address the following research questions:

1. How are the needs of expert parallel programmers different to the needs of novice and intermediate parallel programmers?

2. What training is needed to prepare novice parallel programmers for work in HPC environments?

Future plans also involve the further development and maturation of the MPIGen tool beyond a prototype phase to target the currently unsupported functionality (Section 6.7). This includes support for more complicated applications based on grids of varying dimensions. Extending the tool to support Fortran is also part of this future plan, as this would satisfy a wider user base and more application domains where Fortran codebases are still widely in use.

Mattson and Wrinn (2008) outline a call to action for support for problems in the field of parallelisation stating real workloads by real programmers are necessary when developing tools. MPIGen was tested using an industrial sized application, however, its evaluation involved an experiment based on students (programmers with limited experience) parallelising the Game of life application (a toy problem).
Future work will involve a rigorous evaluation of the next release of the tool, targeting expert parallel programmers using industrial sized applications.

8.5 Concluding Remarks

To avail of the ever increasing performance gains offered by multicore environments, software developers must port serial applications to high performance architectures. The need for effective tooling support to assist parallel programmers is immediate and growing (Sutter and Larus 2005), yet the HPC community is slow to adopt new programming techniques or tools (Hochstein and Basili 2008). MPIGen was developed as an embodiment of empirically derived requirements. The aim of developing a tool in this manner is to assure the generally conservative scientific community (Dongarra et al. 2003) that the concepts on which MPIGen is founded, are relevant to practitioners. The contribution of this research serves as a drop in a large sea, however, a hope exists that a resultant ripple will inspire others to create solutions that stem from practitioners’ needs and build on currently adopted technologies such as MPI.
Bibliography


209


PITAC, United States President’s Information Technology Advisory (2005). Computational science.


Sangani, K (2007). 'Computing-Two good to be true-Multicore microprocessors may be great for doing lots of things at once, but what about doing one thing more quickly?' *Engineering & Technology* 2.1, pp. 40–43.


215


accessed November 2013.


A.1 Execution Results

Figure A.1 illustrates the timing metrics gathered from both the Cartesian and the row based implementations of JFLOW® Reduced Engine for a grid of size $16,388 \times 16,388$ on Blue Gene®/P. These metrics were gathered for executions on 256, 512 and 1024 cores. As can be seen, the row implementation outperformed the Cartesian implementation for all core counts.

Figure A.2 illustrates the results achieved when running the Cartesian implementation on two different grid sizes: $8196 \times 8196$ and $16,388 \times 16,388$ on Blue Gene®/P. The core counts used included 128, 256, 512 and 1024. Scaling is apparent as the core count increases for both grid sizes.
**Figure A.1:** Cartesian vs. Row Performance for grid size 16,388 x 16,388

**Figure A.2:** Cartesian Performance
B.1 General Parallelisation Process

1. What type of applications do you typically parallelise, are they:
   (a) Serial to parallel?
   (b) Parallel to more parallel?

2. Identify the core tasks that are required to parallelise an application.

3. What HPC technology (MPI/OpenMP/CUDA/etc.) do you predominantly use for parallelisation?

4. Select the top three most effort consuming activities when parallelising applications. For each, discuss the following:
   (a) What information do you need to carry out this activity
   (b) Where do you find this information?
   (c) Do you use tools to assist this activity?
      i. If yes, how much assistance do they provide? Do they fail at some point? If so, why?
      ii. If you do not use tools, why not?
(d) What solution can you suggest to decrease the effort expended on this activity?

5. When parallelising domain specific applications, do you need to have an understanding of the application domain before you parallelise? If yes,
   (a) Do you communicate with domain experts during the parallelisation process?
   (b) Please describe scenarios where there has been a need for this contact?

6. Do you look for patterns within the code you are parallelising?
7. Are there reasons applications would not be suited to a particular hardware? If so, please list.
8. What are the main issues associated with initiating new employees?
9. How important is documentation as part of the initiation process?
10. Is there a formal process for recording documentation?
11. What tool/visualisation tool would make parallelisation better (easier, faster, less error-prone)?

**B.2 Data Focused**

1. How much data is generated as part of the parallelisation process? Is it difficult to manage this data?
2. How do you identify the best dataset size for an application?
3. Does the size of the dataset have any influence on results/scaling?
4. Is it important to visualise the distributed data?

**B.3 Decomposition**

1. What issues are experienced when mapping from the problem domain to the HPC architecture?
2. What issues do you encounter when performing data decomposition specifically?
3. Can you describe scenarios where these issues were encountered?
4. Do you use any tooling support to assist these issues?
   (a) If yes, please expand and describe such tools
   (b) If no, are you aware of the existence of any tooling support?
      i. If yes, why are these tools not used?
      ii. If no, can you suggest any reasons for the lack of tooling support?
5. When designing the parallelisation strategy, do you spend much time planning the data decomposition?
6. What suggestions do you have for assistance when decomposing an application?
7. Do you find it problematic to determine indices when performing boundary exchanges?
8. Is there any current support for doing these exchanges?
9. Is there a way to see what’s happening to the data? (Any useful workbenches?)
10. What visualisation would be helpful?
11. How do you characterise an optimal (or near-optimal) decomposition?
12. Do profiling tools provide useful information for determining an optimal decomposition?
13. Is it common to have to change decomposition strategies for performance reasons?
14. Can you identify specific types of codebases that have problems related to data decomposition?
Survey Distributed to Parallel Programmers
1. **What is your experience in a Software Engineering/Computer Science field**

   - Industrial experience (number of years)
   
   - Academic experience (number of years)

2. **What area(s) in Software Engineering/Computer Science have you working knowledge?**
   - Testing/QA
   - Programming
   - Requirements
   - Design
   - System Administrator
   - Networking
   - Analyst
   - Database
   - Other (Please specify)

3. **What is your experience as a High Performance Computing (HPC) programmer?** Please state in number of years

   - Industrial experience (number of years)
   
   - Academic experience (number of years)

   - Approximate number of applications you have parallelised:

   - Average duration of Parallelisation projects:

---

### Section 2 – General parallelisation Project Information

4. **From the five largest parallelisation projects you have worked on, please state the HPC technology used per project**

<table>
<thead>
<tr>
<th>Project 1</th>
<th>Project 2</th>
<th>Project 3</th>
<th>Project 4</th>
<th>Project 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   - Shared Memory - OpenMP
   - Distributed Memory - MPI
   - Hybrid-OpenMP + MPI
   - Heterogeneous-CUDA/OpenCL/DirectCompute/ArBB
   - Hybrid-CUDA+MPI
   - Hybrid-CUDA+OpenMP
   - PGAS – UPC/Coarray/Fortran/Titanium
   - Other

   Please specify Other:
5. In distributed message passing applications, what grid structures do you deal with?

<table>
<thead>
<tr>
<th>Dimension (2D/3D)</th>
<th>Frequency of Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular/Cartesian</td>
<td></td>
</tr>
<tr>
<td>Adaptive</td>
<td></td>
</tr>
<tr>
<td>Irregular</td>
<td></td>
</tr>
<tr>
<td>Other (please specify)</td>
<td></td>
</tr>
</tbody>
</table>

6. In complicated (not embarrassingly parallel) message passing applications, how do you rate the following phases?

<table>
<thead>
<tr>
<th>Phases</th>
<th>Very Easy</th>
<th>Easy</th>
<th>Neutral</th>
<th>Difficult</th>
<th>Very Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Decomposition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Decomposition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Profiling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance Characterisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. From the applications you have worked on, what is the percentage breakdown of decomposition type?

<table>
<thead>
<tr>
<th>Type</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Task/Functional</td>
<td></td>
</tr>
<tr>
<td>Combination (Data &amp; Task/Functional)</td>
<td></td>
</tr>
<tr>
<td>Other (Please Specify)</td>
<td></td>
</tr>
</tbody>
</table>

Section 3 - Decomposition

8. For complicated (not embarrassingly parallel) message passing applications you have worked on, what is the average TIME spent on the data decomposition phase of the parallelisation process? (Please state this as a percentage of total time spent parallelising the application)

9. Do you use tools/media to assist with data decomposition? If yes, please outline the tool/medium used:

10. Do you use representations during data decomposition? If so, can you describe these:
11. Are you aware of other tools/media available to facilitate data decomposition? If yes, please list these and comment on their support for data decomposition

12. Please rate the following considerations when performing data decomposition

<table>
<thead>
<tr>
<th></th>
<th>Not at all important</th>
<th>Somewhat Important</th>
<th>Important</th>
<th>Very Important</th>
<th>Extremely Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of Programming</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management of Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overheads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other Considerations:

13. In a complicated (not embarrassingly parallel) message passing application, what is the average ERRORT spent on the data decomposition phase of the parallelisation process? (Please state this as a percentage of the total effort spent parallelising the application)
Ethical Approval for Evaluation Study
3rd October 2013

Mr. J.J. Collins  
Department of Computer Science & Information Systems  
University of Limerick

RE: 2013_09_04_S&E Identification of data oriented abstractions for Parallelization using MPIGen prototype tool – Student Study

Dear J.J.,

The Faculty of Science and Engineering Research Ethics Committee has approved the above application.

Yours sincerely

[Signature]
Dr. Thomas Waldmann  
Chair  
Science & Engineering Research Ethics Committee

cc Anne Meade
20th May 2013

Mr. J.J. Collins
Department of Computer Science & Information Systems
University of Limerick

RE: 2013_05_02_S&E Identification of Data Oriented Abstractions for Parallelization using MPIDeGen Prototype Tool

Dear J.J.,

The Faculty of Science and Engineering Research Ethics Committee has approved the above application.

Yours sincerely

[Signature]
Dr. Thomas Waldmann
Chair
Science & Engineering Research Ethics Committee

c.c. Anne Meade
Evaluation Survey:
Experts
Section 1 - Profiling & Background Questions

1. What is your experience as a High Performance Computing (HPC) programmer? Please state in number of years

2. How many serial applications have you converted to parallel using MPI?

3. What is the average duration of these projects? (Please state in number of months)

4. What is the percentage of time taken to perform data decomposition? (Please answer for MPI based projects only)

5. What is the percentage of time taken to devise the communication calls? (Please answer for MPI based projects only)

6. Do you use any tools to assist data decomposition and communication when parallelising MPI based applications?
   - Yes
   - No

7. If you answered Yes in 6, please list the tools used and the main features of each

8. If you answered No in 6, please elaborate on why you don’t use any tools and what features you would deem essential?

9. Do you use any other aids to assist data decomposition and communication when parallelising MPI based applications? If so, please elaborate on what these are
Section 2 - MPIGen Questions

1. What are the advantages of using the MPIGen tool?

2. What are the disadvantages of using MPIGen tool?

3. What of the current functionality in MPIGen is the most useful?

4. What of the current functionality in MPIGen is the least useful?

5. Would this tool be applicable for typical parallelisation projects in your organisation?
   Mark only one box.
   - All projects
   - Most projects
   - Some projects
   - Few projects
   - None
   Please give reasons for your answer

6. Rate the importance of maintaining MPI as the output of the tool?
   Mark only one box.
   - Very important
   - Important
   - Moderately important
   - Of little importance
   - Not important

7. Please rate the understandability of the code generated by MPIGen
   Mark only one box.
   - Very easy to understand
   - Easy to understand
   - Not easy/not difficult to understand
   - Difficult to understand
   - Very difficult to understand
8. Please rate the usability of the MPIGen tool

Mark only one box.

☐ Very easy
☐ Somewhat easy
☐ Not easy, not difficult
☐ Somewhat difficult
☐ Very difficult

9. What added functionality would you expect in MPIGen in order to adopt it as a tool in your work?

10. How do you rate the learning curve of MPIGen when using it for the first time?

Mark only one box.

☐ Very easy to learn
☐ Somewhat easy to learn
☐ Not easy, not difficult to learn
☐ Somewhat difficult to learn
☐ Very difficult to learn

11. Please rate your experience entering input values to the tool

Mark only one box.

☐ Very clear to input
☐ Somewhat clear to input
☐ Not clear/not complex to input
☐ Somewhat complex to input
☐ Very complex to input

12. How do you rate the tool with regards to performing boundary communication?

Mark only one box

☐ Very useful
☐ Useful
☐ Moderately useful
☐ Of little use
☐ Not useful

13. If using MPIGen, how would you characterise the effort required to convert from one type of decomposition to another e.g. changing from a row topology to Cartesian?

Mark only one box

☐ Very easy
☐ Somewhat easy
☐ Not easy, not difficult
☐ Somewhat difficult
☐ Very difficult

14. Do the input values sufficiently capture the parallel concerns of the programmer? Please elaborate on a yes/no answer


15. Please rate the ease of index calculation when performing communication using MPIGen? Mark only one box
   - Very easy
   - Somewhat easy
   - Not easy, not difficult
   - Somewhat difficult
   - Very difficult

16. Is the ability to customize the variable names a relevant feature? Mark only one box
   - Very relevant
   - Relevant
   - Moderately relevant
   - A little relevant
   - Not relevant

If very to moderately relevant, please give reasons for your answer

17. How would the overall development time change if using MPIGen to port an application from serial to parallel? Mark only one box
   - Much quicker
   - Somewhat quicker
   - Not quicker, not slower
   - Somewhat slower
   - A lot slower

18. Would using the tool change the frequency of the following errors typically produced when converting code from serial to parallel?

<table>
<thead>
<tr>
<th>Errors</th>
<th>Increase Errors</th>
<th>Decrease Errors</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index coding errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication related errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decomposition related errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other errors</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If Other errors selected, please elaborate on what these errors are
Evaluation Survey: Students
Student Survey

All data entered will be saved in a confidential and anonymous manner. If you wish to receive feedback of the results, please enter your email address at the end.

1. Please state any industrial experience you have in the field of computer science
   *Mark only one box.*
   - No experience
   - Experience as an intern (cooperative experience)
   - Experience as part time work
   - Full time experience < 1 year
   - Full time experience < 5 years
   - Full time experience < 10 years
   - Full time experience > 10 years

2. What is your experience of High Performance Computing?
   *Mark only one box.*
   - Beginner
   - Some theoretical knowledge
   - Some practical knowledge
   - Industrial experience working in HPC

3. What are the advantages of using the MPIGen tool?

   

4. Did you experience errors in your code when you were programming the manual exercise (P1)?
   If so please elaborate on what these were.

   

5. Did you experience errors in your code when you were programming the exercise using MPIGen (P2)? If so please elaborate on what these were.

   

5. Please rate the usability of MPIGen tool
   *Mark only one box.*
   - Very easy
   - Somewhat easy
   - Not easy, not difficult
   - Somewhat difficult
   - Very difficult
6. **How do you rate the learning curve of MPIGen when using for the first time?**
   
   *Mark only one box.*
   
   - [ ] Very easy to learn
   - [ ] Somewhat easy to learn
   - [ ] Not easy, not difficult to learn
   - [ ] Somewhat difficult to learn
   - [ ] Very difficult to learn

6. **Please rate the understandability of the code generated by MPIGen**
   
   *Mark only one box.*
   
   - [ ] Very easy to understand
   - [ ] Easy to understand
   - [ ] Not easy/not difficult to understand
   - [ ] Difficult to understand
   - [ ] Very difficult to understand

7. **Please rate your experience entering input values to the tool**
   
   *Mark only one box.*
   
   - [ ] Very clear to input
   - [ ] Somewhat clear to input
   - [ ] Not clear/not complex to input
   - [ ] Somewhat complex to input
   - [ ] Very complex to input

7. **What are the disadvantages of using the MPIGen tool?**

   
   

8. **Given the choice, which would you use if performing another parallelisation exercise?**
   
   *Mark only one box*
   
   - [ ] MPIGen tool
   - [ ] No tool, perform manual parallelisation

9. **Please give reasons for your choice above**

   
   

If you wish to receive feedback from the results of this survey, please enter your email address: