Data Decomposition for Code Parallelization in Practice: What Do the Experts Need?

Anne Meade*, Deva Kumar Deeptimahanti*, Michael Johnston†, Jim Buckley*, J.J. Collins*
* Lero - the Irish Software Engineering Research Centre, CSIS, University of Limerick, Ireland
{anne.meade, deva.kumar.deeptimahanti, jim.buckley, j.j.collins}@ul.ie
† High Performance Computing Group, Dublin Software Laboratories, Ireland
michaelj@ie.ibm.com

***Abstract*—Parallelizing serial software systems in order to run in a High Performance Computing (HPC) environment presents many challenges to developers. In particular, the extant literature suggests the task of decomposing large-scale data applications is particularly complex and time-consuming. In order to take stock of the state of practice of data decomposition in HPC, we conducted a two-phased study. Firstly, using focus group methodology we conducted an exploratory study at a software laboratory with an established track record in HPC. Based on the findings of this first phase, we designed a survey to assess the state of practice among experts in this field around the world. Our study shows that approximately 75% of parallelized applications use some form of data decomposition. Furthermore, data decomposition was found to be the most challenging phase in the parallelization process, consuming approximately 40% of the total time. A key finding of our study is that experts do not use any of the available tools and formal representations, and in fact, are not aware of them. We discuss why existing tools have not been adopted in industry and based on our findings, provide a number of recommendations for future tool support.**

***Index Terms*—High Performance Computing, Industry survey, Empirical Study, Tool support**

I. Introduction

Computationally intense applications that continuously generate demand for execution rates have driven developers to utilize multicore environments to meet performance needs [1]. Evolving sequential applications to run in parallel environments has been documented in the literature as a highly challenging process [2], [3], [4]. When parallelizing serial programs organized around the manipulation of a large data structure, synchronization of the data structure incurs a communication cost that negatively impacts the performance of the program. Therefore, it is of critical importance to determine a decomposition of the data that maximizes computation while minimizing communication across the available computational infrastructure [5], [6]. Developers need extensive knowledge and expertise to manage the data layout, as well as the communication and synchronization requirements that are governed by the inherent dependencies in the code. These tasks have been characterized as extremely complex and error-prone [7]. Massingill et al. [8] emphasize the complexity involved, arguing that, "*A good data decomposition balances the competing forces of flexibility, efficiency, and simplicity*."

Given the extent of the discussions in the literature on this phenomenon, one might anticipate that extensive tool support would be available and utilized in the practice of data decomposition. However, there are very few studies in the field of HPC that empirically assess the state of practice (notable exceptions are the studies by Basili et al. [9] and Hochstein and Basili [10]), and even fewer studies focusing on data decomposition. Pancake [11] argues that a systematic approach is required for HPC tool development, and we subscribe 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. Hence, the goal of this paper is to shed light on some of the practices and issues experienced by HPC experts during data decomposition and explore if these do confirm the literature findings.

Our research was conducted as a two-phased field study as follows: Phase I consisted of an exploratory study (using focus groups and interviews) at the HPC group at IBM in Dublin (Ireland), which has an established track record in High Performance Computing research and expertise. The goal of this phase was to identify key issues and seed themes that would inform the design of our subsequent research. Using a qualitative approach in this phase allowed us to gain a deep understanding of some of the practices and issues of HPC experts. Phase II consisted of a survey to gather deep insights from a wider population of HPC experts worldwide.

The contribution of this paper is the provision of empirical evidence on the state of practice of HPC developers with respect to data decomposition and the associated challenges of data communication and synchronization. We do so by first presenting a brief overview of the existing tools that have been developed so far. A key finding of our study is that HPC experts do not use these tool, and in fact are not familiar with them. Based on the analysis of our findings, we discuss possible reasons for this, and offer a number of recommendations for better tool support.

This paper is structured as follows: Section II outlines the background and our motivation for conducting this study including an overview of the available tools. Section III presents the research design for both the exploratory and survey study while Section IV presents the results. The findings are discussed in Section V and threats to validity are discussed in Section VI. Section VII concludes this paper and presents an outlook to future work.

II. Background and Motivation

Identifying an appropriate decomposition to use in a parallelization project, evaluating dependencies and deciding what needs to be synchronized across processors is not an easy feat. Following on from these decisions, the parallel developer is faced with the task of programming the solution and subsequently assessing it for efficiency. Data decomposition in the context of HPC occurs when the same operation is applied to different data in a parallel program [12]. This type of decomposition is commonly applied to multidimensional grids when performing scientific simulations [13]. However, decomposing data so that they are distributed over a multidimensional grid is a highly challenging process [14].

Mattson et al. [15] present an example to demonstrate the differences between decomposition strategies using a division of an $N \times N$ matrix into four pieces applying either a column or a block/Cartesian decomposition. In the case of a column decomposition, the data are divided into columns of size $N \times \frac{N}{4}$. The communication perimeter (addition of all four sides) for each distributed column is $2N + 2\frac{N}{4}$ (or $2\frac{1}{2}N$). In the Cartesian decomposition, the data are divided into blocks of size $\frac{N}{2} \times \frac{N}{2}$ and the perimeter for each distributed block is $4\frac{N}{2}$ (or $2N$). From this we can see that the amount of data to be exchanged would be less for the Cartesian decomposition ($2N<2\frac{1}{2}N$). However, in certain computations, operations are only performed on data in cells left and right of the data cell in question, and not above and below. In this scenario, the full perimeter is not involved in the data exchange and so a column decomposition would be more efficient.

Deciding 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. [15] 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.*"

Another issue to consider is the computational load balance. In the case of linear algebra, elements of a matrix are eliminated as the computation executes. A solution is to distribute more pieces of data than there are processes in a cyclic manner so as to access the data in a round-robin manner, allowing for better balancing. In the context of large-scale finite element simulations, the discretization of a physical domain can be dynamically adjusted by executing adaptive grid computations as is done in Adaptive Mesh Refinement (AMR) [16]. Load imbalance can result from such adjustments and so the grid or graph needs to be repartitioned during execution to maintain performance [17].

Cache performance and latency are also issues that need to be considered when decomposing an application. With regards the latter, a developer may need to take into account the number of messages rather than the size of the messages communicated [18].

Another consideration is the error-prone task of calculating array indices. When distributing arrays of data, each element in the global array is mapped to an element in the locally distributed array but must be identified by either its global or local index. Calculating this mapping adds further complexity.

A. Tool Support: State-of-the-Art

Prior to presenting design and results of our study, an overview of approaches that support data decomposition is provided to assist with an understanding of the dimensions of the problem, to better interpret specific responses from the survey, and to identify recommendations for improving tool support. We recognize that there is extensive research in the area of graph based decomposition techniques and libraries, such as ParMetis [19]. Findings of our survey (see Section IV-B1) show that the majority of respondents use *structured grids*, and so we will focus our research within those parameters. The remainder of this section presents a brief overview of tools and libraries designed to perform structured grid based data decomposition.

1) Promoter: (PROgramming MOdel To Enable Real world computing) [20] is a language model designed for data parallel applications to abstract data and communication structures in a parallel programming environment. The Promoter runtime system consists of the Promoter runtime library and the Promoter Abstract Machine (PAM). The PAM is responsible for communication and synchronization and is platform dependent. The Promoter language is designed as an extension to C++, and was the first model to introduce a data *topology*, a set of distributed address spaces that function as an index space. A regular topology includes vectors, matrices or n-dimensional arrays and an irregular topology can include masked arrays, unions of sub-topologies or sets of arbitrary points. Promoter's compiler accepts Promoter programs as inputs and generates C++ output. Promoter does not implement dynamic topologies which would involve run-time mapping. All data partitioning and distribution is done at the language level at compilation time in Promoter, disallowing redistribution of data or any adaptive capability. The lack of run-time support is a serious drawback of the tool as this is crucial for adaptive structures.

2) Janus: [21] is a C++ template library and aims to simplify the implementation of finite element and other grid based methods. Janus utilizes three concepts: *domain*, *relation*, and *property function*. A domain is a finite set represented as a sequence, for example, the distributed grid over the processors assigned. Each element of the domain can be represented by an index which has a unique position. A relation between two sets (domains) is the description of data dependencies between these sets and is represented using adjacency matrices. The indexing technique used allows communication relations independent of the mapping of the indices to the underlying hardware. The property function describes data associated with the domains or relations. Janus is implemented as a C++

template library. Advantages of the Janus library include a higher-level abstraction for MPI as well as the ability to be used for the implementation of adaptive mesh methods due to the flexibility of the two-phase domains and relations. A disadvantage of Janus is the fact that there is no overlapping of domains resulting in a lot of communication for certain applications.

3) TACO: [22] stemming from the terms *Topologies and Collections* is a distributed object platform for cluster architectures based on C++ templates. TACO, based on dynamic distributed data structures, uses C++ methods to access objects within its own local space and synchronous and asynchronous remote method invocations to access remote objects. TACO implements global object pointers and has been used as a Partitioned Global Address Space (PGAS) framework [23]. TACO is based on topologies inspired by those used in the Promoter model and is an extension of the authors' previous work involving Multiple Threads Template Library (MTTL). As TACO uses a global address space data distribution, it is therefore not useful in a purely distributed memory environment.

4) Hi-PaL: [24] is part of a larger framework called FraSPA (A framework for Synthesizing Parallel Applications). This work involves the creation of a domain specific language (DSL) called Hi-PaL (High Level Parallelization Language) with the aim of reducing the effort involved in writing parallel applications by using a higher level of abstraction than MPI. The process of generating and inserting the required code for parallelization into the existing sequential application is automated using source-to-source transformation techniques. Hi-PaL offers a higher level of abstraction and eliminates the need to directly write lower level MPI code. This DSL also reduces the lines of code (LOC) a developer needs to write. The developer does not need to modify the sequential codebase, however, the user is still required to learn a new API. Furthermore, Hi-PaL does not support adaptively changing of distributions.

5) Blatt and Bastian's Approach: [25] was used to implement the Iterative Solver Template Library (ISTL) as part of the Distributed and Unified Numerics Environment (DUNE) [26]. This library uses generic template classes in C++ to describe the data decomposition and communication. In this approach, data is stored in existent sequential data structures where each entry corresponds to an entry in the virtual global view similar to the concept of domains and relations in Janus (see Section II-A2). The communication, which is created automatically, and the decomposition are thus abstracted from the user. This library module ensures functionality for ghost cell communication by overlapping the data structures. The advantage of using this framework is to provide a higher level of abstraction than that of MPI where communication is automated to an extent.

TABLE I
STUDY PARTICIPANTS FOR PHASE I.

ID	Current Title	Experience in Computer Science
P1	Sr software architect	21 years (5 years HPC)
P2	HPC specialist	4 years (3 years HPC)
P3	Software engineer	4 years (4 years HPC)

III. RESEARCH DESIGN

A. Research Goals

Our research goal was to gain a better understanding of the parallelization practices of HPC experts when targeting multicore processors, as well as the challenges encountered in this process. We also set out to identify the tools and notations that experts use to perform data decomposition, which is a critical step in the parallelization process, as outlined above.

In order to shed light on these issues, we designed a two-phased study. The goal of the first phase was to explore practices and challenges at one organization with an established track record in parallelization of software. Section III-B presents the study design of this first phase. The goal of the second phase was to gain insights from a larger sample of HPC experts worldwide, by means of a survey. Section III-C presents the design of this second phase of the study.

B. Phase I: Exploratory Study Design

An exploratory study was organized in IBM's HPC centre in Dublin, where a focus group session and interviews were conducted with the aim of synthesizing empirical data with the findings from the literature. The core business of IBM's HPC center lies in taking existing commercial and scientific applications, porting and tuning these so that they run efficiently on IBM® Blue Gene® supercomputers [1]. We conducted two rounds of interviews in IBM's HPC centre involving three experts on parallel development. The first session was an exploratory focus group; such a session can produce insightful and candid information [27]. All three participants were present for this focus group, which lasted approximately two hours. Table I lists title and background experience of these participants. Participants were numbered so as to protect their privacy (e.g., P1). The aim of this focus group was to discuss parallelization in general, incorporating the challenges encountered by the participants. It is worth noting that we did not place any emphasis on decomposition, rather, we decided to gain insight into the challenges the studied organization encountered, assessing if decomposition featured as an issue. Questions included *Could you give a brief synopsis of how you parallelize systems?*, and *What are the main aspects of the system that the programmer should focus on during the parallelization task?*

A second round of interviews was subsequently carried out where we further conducted two thirty-minute semi-structured

[1]IBM and Blue Gene are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide

interviews based on questions derived from the findings of the focus group. This allowed us to focus on the area of data decomposition (a reported challenge in the focus group) and ask more detailed questions about the issues encountered and potential solutions. All interviews were conducted by two authors of this paper. All sessions were digitally recorded with the participants' consent. The interviews were transcribed verbatim, in order to allow for in-depth analysis. The transcriptions were analyzed using qualitative data analysis methods [28]. All interview transcripts were thoroughly read and phrases of particular interest were coded with labels to reflect the topic of that phrase. All the researchers involved in the project group verified the data analysis findings. The use of qualitative methods provide rich and informative results when used in empirical studies in technical fields [28].

C. Phase II: Survey Design

Phase II of our study consisted of a survey targeting experts in the field. The survey was presented as 13 questions broken into three sections. The first section gathered information regarding the profile of the participant including their experience in a HPC context (3 questions). The second section related to general parallelization experience and collected data related to past projects parallelized (4 questions). The third focused on data decomposition specifically (6 questions). The survey was compiled using the online tool *SurveyMonkey* and distributed to a wide global audience.

The survey was circulated through avenues such as the STEM-Trek webpage [29] and LinkedIn HPC forums. Purposive and snowball sampling [30] were both used to choose these populations. Purposive sampling occurs when the researcher deliberately picks the sample from people likely to produce the data. We targeted those working or researching in HPC environments such as SCEDC (Southern California Earthquake Data Centre), US-XSEDE (Extreme Science and Engineering Discovery Environment) and EU-PRACE (Partnership for Advanced Computing in Europe). Then we used a method of snowball sampling where people in these fields passed on the survey link to those who they expected to be also relevant to the research area. In all, 32 participants finished the survey. Of these, 26 had industrial experience. Six participants had academic experience only, but all had significant experience in parallelizing applications (10 applications on average). Therefore, we argue that the academic participants also have extensive knowledge of this domain, possibly similar to that of the industry participants. The summary of the participants' experience is outlined in Table II.

TABLE II
PARTICIPANT EXPERIENCE

	Average	Min	Max
Industrial experience (years)	3.16	0	16
Academic experience (years)	5.52	0	27
No. of applications parallelized	7.90	4	20
Typical duration of parallelization projects (months)	7.91	0.5	36

IV. FINDINGS

A. Exploratory Study Findings

Data decomposition was mentioned by the participants when asked to speak about the parallelization process and the challenges experienced in general. The division of the data when performing data decomposition was expressed as "*the most complicated part*" by participant P2, who continued: "*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 whether you need just global information or local data.*" Participants indicated various issues associated with finding an efficient decomposition, characterizing the process as very ad-hoc; participant P1 explained: "*What people do is try a block distribution, benchmark and see what you get, then try a cyclic distribution and see what you get, but it's hit and miss, more miss than hit.*" The participants described data decomposition as a critical part of the parallelization process, emphasizing the large performance gains resulting from an optimal solution. One participant (P1) ranked an efficient decomposition mapping as the highest return in performance gain when rating optimization techniques. This was considered more important than an efficient algorithm.

A lack of good tool support was considered by the participants as highly frustrating. Participant P3 elaborated on this as follows, when speaking of converting serial code to parallel: "*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.*" The participants outlined the benefits of abstractions in this area in order to hide the complexity of low level details from the developer. One participant described the large effort involved in manually calculating indices of data elements that must be synchronized in a parallel environment. We found wide agreement that the developer should be able to focus on the decomposition strategy itself at a high level of granularity.

Our analysis of the exploratory findings highlighted three requirements for light-weight tool support. All the solutions center on abstractions that would target the bottlenecks encountered in relation to data decomposition in the parallelization process. Table III summarizes these requirements (numbered R1 to R3) and provides exemplary snippets of empirical evidence to support them. The aim of R1 is to map the decomposition a developer intends to implement and to create the data structures to carry this out. R2 focuses on generating MPI in order to implement the communication calls based on the decomposition and R3 is centered on finding a solution to assess whether a chosen decomposition strategy is optimal.

TABLE III
REQUIREMENTS FOR TOOL-BASED SUPPORT DATA DECOMPOSITION IDENTIFIED IN THE EXPLORATORY STUDY.

ID	Description	Empirical Evidence
R1	Provide support for automated indexing, as well as a notation (e.g., a diagram) to facilitate experts to specify indices.	*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* —P3, Software Engineer
R2	Provide an easy interface to specify the decomposition strategy, and to offer a notation to outline the required communication. A code skeleton can automatically be generated from this.	*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* —P3, Software Engineer
R3	Provide a means for profiling, inspection, and visualizing alternative decomposition strategies. This facilitates a practical approach to evaluate its efficiency (e.g., in terms of performance).	*Maybe you're doing some reconfiguring because you need calculations that you'd prefer to be a different way, it takes some time* —P1, Senior Software Architect

B. Survey Findings

As the participant set was from a globally scattered population the first two sections of the survey gathered data to categorize the type of parallelization projects the participants were involved in, the technologies used, the challenges associated with phases in the parallelization process as a whole, and the grid structures used. These findings are elaborated in the following section, followed by the findings regarding decomposition.

1) General Parallelization Findings: Figure 1 illustrates the breakdown of the past largest five projects the participants worked on. As can be seen, pure MPI based projects totaled the highest number at 31.51%. A hybrid of MPI and other technologies (MPI, OpenMP+MPI, CUDA+MPI) totaled 52.05% of all listed projects. Pure OpenMP totaled 13.13% while a hybrid of OpenMP and other technologies (OpenMP, OpenMP+MPI, OpenMP+CUDA) amounted to 32.87%. Heterogeneous projects totaled 10.96%, however other options such as a hybrid featuring CUDA also featured. PGAS (Partitioned Global Address Space) projects amounted to less than 1%, however, the *Other* category in the survey question amounted to 19.18% and this featured options not listed such as CUDA+MPI+OpenMPI, MATLAB, and PGAS using Global Arrays toolkit.

The participants were asked to list the grid structures encountered when parallelizing message passing applications. Fifty-five percent of the participants claimed structured Cartesian grids constituted at least 50% of all message passing applications. Thirty-nine percent of the participants put this figure at over 70% of all their message passing applications. Thirty-five percent of the participants reported using irregular grids in a maximum of 30% of their projects, while 65% of the participants did not use irregular grids at all. Of the 45% of participants who claim to work with adaptive grids, 32% of reported working with these in a maximum of 30% of of applications. Fifty-five percent of participants did not work with adaptive grids, but listed a number of other structures, such as Yin-Yang, butterfly (two circles connecting in a common point), and a spectral (wave-space). The data gathered from this survey show MPI based projects with data centered on structured Cartesian grids are the most common parallelization scenarios.

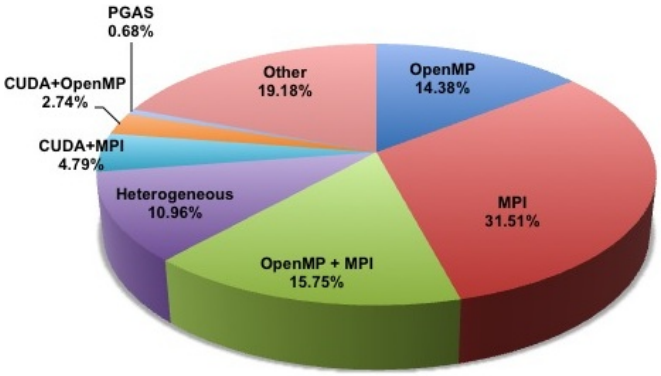

Fig. 1. Breakdown of project type

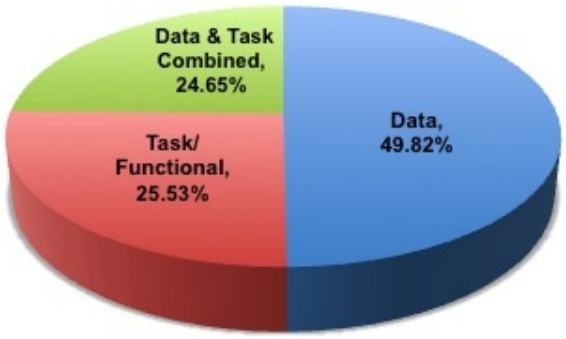

Fig. 2. Breakdown of decomposition type

2) Decomposition Findings: The breakdown of decomposition type across projects was questioned and the findings show that approximately 50% of decomposition is based on distributing the data (see Figure 2). Task decomposition and a hybrid of task and data decomposition make up approximately 50% of projects. In total, data decomposition (be it hybrid or pure) amounts to approximately 75% of decomposition type across projects thus confirming the importance of data decomposition as found in Phase I of our study.

We can divide the parallelization process into a number of separate phases: Profiling (pre-parallelization), Task Decomposition, Data Decomposition, Communication Design, and Performance Characterization (post-parallelization). The participants were asked to rate the difficulty of these phases in the parallelization process, using a five-level Likert scale, from very easy to very difficult. Figure 3 presents the results. Data decomposition was rated as most difficult. This is consistent with the findings in Phase I of our study. Performance characterization, communication and task decomposition were next on the difficulty rating. Profiling was considered to be less problematic with a neutral score.

On average, programmers reported spending 40.48% of time and effort performing data decomposition. When asked if and what tools or media were used to assist data decomposition, 70% 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. One participant commented: "*Usually custom tools; matlab scripts to plan out layouts; simple tests (not quite unit tests) to make sure indexing is working.*" Following this question, the participants were asked if 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;

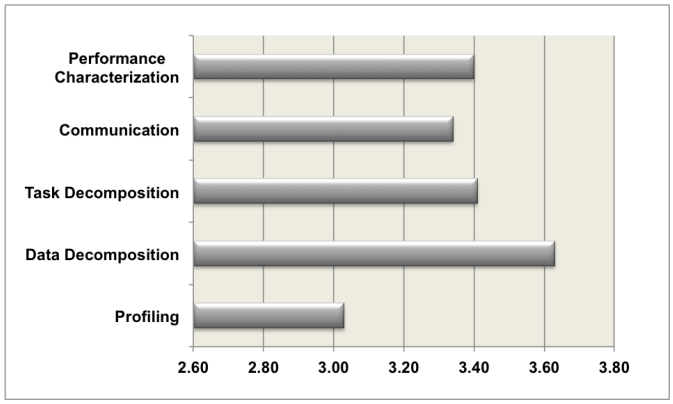

Fig. 3. Average difficulty rating of parallelization phases (1: Very easy, 2: Easy, 3: Neutral, 4: Difficult, 5: Very difficult)

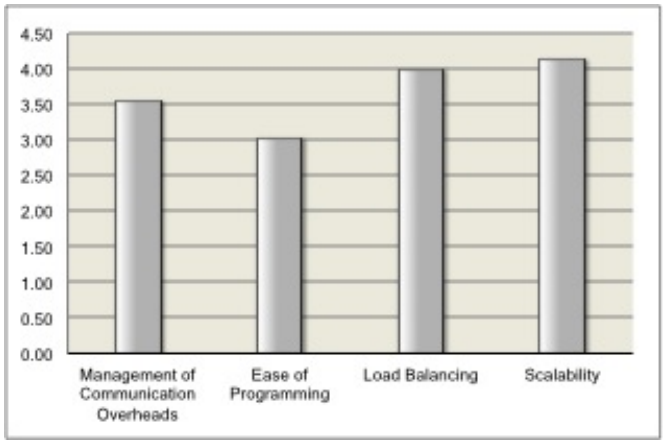

Fig. 4. Considerations when performing data decomposition (1: not at all important —5: extremely important)

- Representations as provided by customers (scientists), e.g., matrix, grids;
- Color-coding sections of a grid based on process number.

A lack of tool support was a key concern identified in Phase I; we asked the participants if they were aware of any tools or media available to facilitate data decomposition. Seventy-six percent of respondents 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.*" Two participants who answered *Yes* listed ParMetis, a graph partitioning tool (see Section II-A), as the only tool they were aware of.

To better understand the requirements of any potential tool support, we asked participants what considerations they deemed important during data decomposition. Figure 4 lists the responses. Scalability and load balancing were rated as the most important. This relates to the comment from P1 in the exploratory study where finding an *optimal* decomposition can lead to high performance gains. A load balanced and scalable strategy could lead to a more efficient solution yet the techniques currently used by experts to assess the varying strategies were described as "*hit and miss*." Management of communication overheads and ease of programming were both described as somewhat important considerations.

One participant also emphasized (as an additional comment) that the accuracy, or correctness, of the final (parallelized) result was important. Another participant remarked that no single consideration is most important for a good data decomposition, reinforcing the sentiment expressed by Massingill et al. [8] (see Section I) who argued it is a balance of competing forces: "*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.*"

V. Discussion

Both the exploratory study and the survey confirm the findings from the literature that the challenge of data decom-

position is very real for many HPC experts. As more than 40% of the parallelization process is consumed by this phase, it is remarkable that available tools have gained little traction in industry.

A review of the state-of-the-art literature provides some evidence of tools and libraries to perform grid based decomposition (see Section II), yet none of these were mentioned by the participants in either phase of our study. The participants did not use any formal representations nor were they aware of tooling support. The grid-based tools outlined in Section II-A all offer a type of framework/language/language-extension to abstract data decomposition in distributed scientific applications. We derived recommendations for future tool support which we will discuss in the next subsection as well as offering explanations as to why the available tools are not adopted in industry.

A. Recommendations for Future Tool Support

Based on our study, we derived a number of recommendations for future tool support to assist HPC experts in the challenging task of data decomposition. We briefly outline these below.

- **MPI Focused.** Our review of the state-of-the-art (Section II) showed that only one approach targets MPI. The results of our study clearly indicate that MPI is one of the most used technologies and, in fact, a de facto industry standard. Given that MPI is so prevalent throughout industry and HPC practitioners are slow to deviate from trusted technologies [9], there is a clear need for more and better tool support for this technology.
- **Low code impact.** The approaches reviewed in Section II all require a developer to learn a new language or programming interface in order to express the parallel intent of the code. As HPC applications have a long life span, developers may be reluctant to rewrite any of their code in an unfamiliar language [9]. 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 at adoption.
- **Ease of Strategy Selection.** The approaches outlined in the state-of-the-art review (Section II) do not provide support to the developer when deciding which decomposition to choose i.e. which strategy will result in better load balancing, scalability, and performance gain. Phase I of our study (see Section IV-A) 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) requires significant effort. R3, a derived requirement in Phase 1 of our research (Section IV-A), focused on this challenge reiterating the need for either an upfront strategy evaluation capability, or a facility to minimize developer effort when implementing alternative decomposition designs.
- **Automated Grid Indexing.** A derived requirement (R1) in Phase I of this study (see Section IV-A), involved the automation of indexing when calculating the communication calls associated with data decomposition. The survey findings (Section IV-B) confirmed that developers use manual drawings and their *imagination* to determine the necessary low level details. These results suggest that the effort involved in manually calculating indices and sketching grid diagrams is a core issue for programmers in this field.
- **Data Structure & Communication Generation.** A derived requirement (R2) from Phase I of our study (Section IV-A) 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 (Section IV-B), ensuring both accuracy and efficiency is crucial when writing MPI calls. This suggests the need to facilitate some degree of automation when generating the data structures and communication calls required.
- **Language Compatibility** The tools previously identified (Section II) were predominantly C++ based. In practice a large proportion of scientific applications are based on C or Fortran. A tooling option which would support these languages has greater potential to gain traction in industry.

VI. Threats to Validity

We are aware of a few limitations of this study. External validity is concerned with the extent to which findings can be generalized to other settings. As this study was based on the experiences of a significant number of experienced HPC programmers with extensive industry experience, we argue that our study presents a valuable contribution to the literature on the state of practice of HPC decomposition. Nevertheless, we also believe that further field studies are welcome to better understand how industry can be served by research.

To avoid bias, the list of topics appeared in a random order for each respondent so that one choice would not influence the response to another. We evaluated our survey before distribution to check that the questions were understandable, the questionnaire was trialled in a pilot study with a number of colleagues before it was distributed. The data collected in this pilot study helped us to assess the returned information [30]. The survey begins with profile questions as it is important to make sure that respondents have sufficient knowledge to answer the questions [31]. Questions on the expert survey included: *From the five largest parallelization projects you have worked on, please state the HPC technology used per project?* This improved the chances that only experts provided information. It was after this question that 13 of the 45 initial expert sample discontinued their questions, resulting in only 32 fully completed surveys.

To ensure we correctly captured and interpreted the findings, we sent an initial study report to the participants of our ex-

ploratory study (Phase I). This is a form of member checking, which is a tactic to assess a study's validity [32]. The participants confirmed the findings as accurate. The preliminary results of the survey were also distributed to the participants who had requested feedback (in the survey questionnaire). One of the participants expressed his approval of the findings and analysis as follows: "*I resonate with results that data decomposition is the most challenging. And it's interesting to see that people aren't using tools for that.*"

VII. Conclusions and Future Work

In this paper we presented a two-phase study to investigate the state of practice of data decomposition in High Performance Computing. Following an exploratory phase at IBM's HPC centre, where qualitative research methods were used to develop a deep understanding of the issues in practice, we conducted a survey targeting practitioners in the field. We gathered data from 32 HPC experts worldwide.

The findings reported in this paper shed light on practices and confirm challenges encountered by HPC experts in parallelization projects. Our findings show that data decomposition is one of the most challenging, and time- and effort-consuming phases during parallelization, yet practitioners are unaware of tools to support this phase and instead resort to manual efforts. So far, few empirical studies have been conducted on the actual needs of HPC experts when performing data decomposition. The study reported in this paper represents a first effort to close this gap in the literature. Our findings can be valuable for other researchers in this field, as it outlines a number of requirements for tool support for porting serial code bases to multicore platforms. We believe our empirical approach constitutes an important step in order to ensure relevance to practitioners in real-world HPC contexts.

The data collected in this study and a review of the state-of-the-art enabled a derivation of a list of recommendations for potential tool support to assist this task. We are currently conducting further research on this topic, and are developing an approach to address the issues outlined above. Future work will involve research to evaluate the functionality and usefulness of this tooling support in an industrial context.

Acknowledgment

We are grateful to the participants of the study for their time and enthusiasm. This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie)

References

[1] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's Journal*, vol. 30, no. 3, 2005.

[2] W.-m. Hwu, K. Keutzer, and T. Mattson, "The concurrency challenge," *Design & Test of Computers, IEEE*, vol. 25, no. 4, 2008.

[3] H. Vandierendonck and T. Mens, "Averting the Next Software Crisis," *Computer*, vol. 44, no. 4, 2011.

[4] A. Meade, J. Buckley, and J. J. Collins, "Challenges of evolving sequential to parallel code: an exploratory review," *Proceedings of the 12th International Workshop on Principles of Software Evolution*, 2011.

[5] D. Geer, "For programmers, multicore chips mean multiple challenges," *Computer*, Jan 2007.

[6] P. Lee and Z. M. Kedem, "Automatic data and computation decomposition on distributed memory parallel computers," *ACM Trans. Program. Lang. Syst.*, vol. 24, no. 1, 2002.

[7] F. Chan, J. Cao, and Y. Sun, "High-level abstractions for message-passing parallel programming," *Parallel Computing*, vol. 29, no. 11, 2003.

[8] B. Massingill, T. Mattson, and B. Sanders, "Reengineering for parallelism: an entry point into plpp for legacy applications," *Concurrency and Computation.: Practice and Experience*, vol. 19, no. 4, 2007.

[9] V. R. Basili, J. C. Carver, D. Cruzes, L. M. Hochstein, J. K. Hollingsworth, F. Shull, and M. V. Zelkowitz, "Understanding the high-performance-computing community: A software engineer's perspective," *IEEE software*, vol. 25, no. 4, 2008.

[10] L. Hochstein and V. R. Basili, "The ASC-Alliance Projects: A Case Study of Large-Scale Parallel Scientific Code Development," *Computer*, vol. 41, no. 3, 2008.

[11] C. M. Pancake and C. Cook, "What users need in parallel tool support: survey results and analysis," in *Scalable High-Performance Computing Conference, 1994., Proceedings of the*, 1994, pp. 40–47.

[12] M. McCool, "Scalable programming models for massively multicore processors," *Proceedings of the IEEE*, vol. 96, no. 5, may 2008.

[13] B. Hendrickson, "Graph partitioning and parallel solvers: Has the emperor no clothes?" in *Solving Irregularly Structured Problems in Parallel*. Berlin/Heidelberg: Springer-Verlag, 1998, pp. 218–225.

[14] E. Dovolnov, A. Kalinov, and S. Klimov, "Natural block data decomposition for heterogeneous clusters," in *Parallel and Distributed Processing Symposium*, 2003.

[15] T. Mattson, B. Sanders, and B. Massingill, *Patterns for parallel programming*, 1st ed. Addison-Wesley Professional, 2004.

[16] G. H. Weber, H. Childs, and J. S. Meredith, "Efficient Parallel Extraction of Crack-free Isosurfaces from Adaptive Mesh Refinement (AMR) Data," in *Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, Oct. 2012, lBNL-5799E.

[17] G. Karypis and V. Kumar, "A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm," *8th Siam Conference on Parallel Processing for Scientific Computing*, 1997.

[18] B. Hendrickson, "Graph partitioning," in *Encyclopedia of Parallel Computing*. Springer US, 2011, vol. 4, pp. 805– 808.

[19] G. Karypis and K. Schloegel, "P A R M E T I S: Parallel Graph Partitioning and Sparse Matrix Ordering Library," University of Minnesota," Manual v4.0, 2011.

[20] W. K. Giloi, M. Kessler, and A. Schramm, "Promoter: A high level object-parallel programming language," *Proceedings of International Conference on High Performance Computing*, 1995.

[21] J. Gerlach, "Generic programming of parallel applications with Janus," *Parallel Processing Letters*, vol. 12, no. 02, pp. 175–190, 2002.

[22] J. Nolte, Y. Ishikawa, and M. Sato, "Taco: prototyping high-level object-oriented programming constructs by means of template based programming techniques," *ACM Sigplan Notices*, vol. 36, no. 12, 2001.

[23] T. Prescher, R. Rotta, and J. Nolte, "Flexible sharing and replication mechanisms for hybrid memory architectures," in *Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium*, vol. 55, 2012.

[24] R. Arora, P. Bangalore, and M. Mernik, "Raising the level of abstraction for developing message passing applications," *The Journal of Supercomputing*, 2010.

[25] M. Blatt and P. Bastian, "C++ components describing parallel domain decomposition and communication," *International Journal of Parallel, Emergent and Distributed Systems*, 2009.

[26] DUNE, "Institute of Parallel and Distributed Systems, University of Stuttgart," http://www.dune-project.org accessed January 2013.

[27] F. Shull, J. Singer, and D. Sjøberg, *Guide to Advanced Empirical Software Engineering, 1st edition*. Springer-Verlag, 2010.

[28] C. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Trans. Softw. Eng*, vol. 25, no. 4, 1999.

[29] StemTrek Organization, "StemTrek," available from: http://www.stem-trek.org/ accessed April 2013.

[30] B. Oates, *Researching information systems and computing*. Sage Publications Limited, p.93-107, 2005.

[31] B. Kitchenham and S. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. Sjøberg, Eds. Springer London, 2008, ch. 3, pp. 63–92.

[32] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*. Hoboken New Jersey: Wiley & Sons Inc., 2012.