Programming Languages for Data-Intensive HPC Applications: a Systematic Mapping Study

Vasco Amaral[a], Beatriz Norberto[a], Miguel Goulão[a], Marco Aldinucci[b], Siegfried Benkner[c], Andrea Bracciali[d], Paulo Carreira[e], Edgars Celms[f], Luís Correia[g], Clemens Grelck[h], Helen Karatza[i], Christoph Kessler[j], Peter Kilpatrick[k], Hugo Martiniano[g], Ilias Mavridis[i], Sabri Pllana[l], Ana Respício[m], José Simão[n], Luís Veiga[e], Ari Visa[o]

[a]*NOVA LINCS, DI, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal*
[b]*University of Torino, Italy*
[c]*University of Vienna, Austria*
[d]*University of Stirling, UK*
[e]*INESC-ID, DEI, Instituto Superior Tecnico, Universidade de Lisboa, Portugal*
[f]*Institute of Mathematics and Computer Science, University of Latvia, Latvia*
[g]*BioISI, Faculdade de Ciências, Universidade de Lisboa, Portugal*
[h]*University of Amsterdam, Netherlands*
[i]*Aristotle University of Thessaloniki, Greece*
[j]*Linköping University, Sweden*
[k]*Queens University Belfast, U.K.*
[l]*Linnaeus University, Sweden*
[m]*LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal*
[n]*Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Portugal*
[o]*Tampere University, Finland*

Abstract

A major challenge in modelling and simulation is the need to combine expertise in both software technologies and a given scientific domain. When High-Performance Computing (HPC) is required to solve a scientific problem, software development becomes a problematic issue. Considering the complexity of the software for HPC, it is useful to identify programming languages that can be used to alleviate this issue.

Because the existing literature on the topic of HPC is very dispersed, we performed a Systematic Mapping Study (SMS) in the context of the European COST Action cHiPSet. This literature study maps characteristics of various programming languages for data-intensive HPC applications, including category, typical user profiles, effectiveness, and type of articles.

We organised the SMS in two phases. In the first phase, relevant articles are identified employing an automated keyword-based search in eight digital libraries. This lead to an initial sample of 420 papers, which was then narrowed down in a second phase by human inspection of article abstracts, titles and keywords to 152 relevant articles published in the period 2006–2018. The analysis of these articles enabled us to identify 26 programming languages referred to in 33 of relevant articles. We compared the outcome of the mapping study with results of our questionnaire-based survey that involved 57 HPC experts.

The mapping study and the survey revealed that the desired features of programming languages for data-intensive HPC applications are portability, performance and usability. Furthermore, we observed that the majority of the programming languages used in the context of data-intensive HPC applications are text-based general-purpose programming languages. Typically these have a steep learning curve, which makes them difficult to adopt. We believe that the outcome of this study will inspire future research and development in programming languages for data-intensive HPC applications.

Keywords:
High Performance Computing (HPC), Big Data, Data-Intensive Applications, Programming Languages, Domain-Specific Language (DSL), General-Purpose Language (GPL), Systematic Mapping Study (SMS)

Email addresses: vma@fct.unl.pt (Vasco Amaral), b.norberto@campus.fct.unl.pt (Beatriz Norberto), mgoul@fct.unl.pt (Miguel Goulão), marco.aldinucci@unito.it (Marco Aldinucci), siegfried.benkner@univie.ac.at (Siegfried Benkner), andrea.bracciali@stir.ac.uk (Andrea Bracciali), paulo.carreira@ist.utl.pt (Paulo Carreira), edgars.celms@lumii.lv (Edgars Celms), Luis.Correia@ciencias.ulisboa.pt (Luís Correia), c.grelck@uva.nl (Clemens Grelck), karatza@csd.auth.gr (Helen Karatza), christoph.kessler@liu.se (Christoph Kessler), p.kilpatrick@qub.ac.uk (Peter Kilpatrick), hfmartiniano@ciencias.ulisboa.pt (Hugo Martiniano), imavridis@csd.auth.gr (Ilias Mavridis), sabri.pllana@lnu.se (Sabri Pllana), alrespicio@fc.ul.pt (Ana Respício), jsimao@cc.isel.ipl.pt (José Simão), luis.veiga@inesc-id.pt (Luís Veiga), ari.visa@tut.fi (Ari Visa)

1. Introduction

Although being sometimes controversial to some practitioners because of not having a clear definition, Big Data is a term with increasing interest and widely used by both the industry and academia. Beyer and Laney [9] describe the Big Data using properties *Volume*, *Velocity*, and *Variety* (also known as the "3Vs") as follows:

"Big data is high-Volume, high-Velocity and/or high-Variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."

The "3Vs" description of Big Data has been further extended to "5Vs" [30], by adding a further "V" for *Veracity* that indicates that the quality and accuracy of the data may vary, and a "V" for data *Value*.

One of the major challenges of scientific computing in the context of Big Data is the need to combine software development technology for High Performance Computing (HPC) with the management and analysis of Big Data [11, 59, 50]. For instance, the Square Kilometre Array (SKA) [63] project is building a radio telescope with one square kilometre of collecting surface. SKA computing requirements are more than 100 petaflops, and the data traffic of SKA will exceed the data traffic of the whole Internet. Efficient processing of large amounts of data demands computational, communication and memory resources of large-scale HPC systems [1]. Modern HPC systems comprise a large amount of interconnected computing nodes, each having one or more multi-core or many-core processors. For instance, the *Summit* [64] supercomputer (Ranked 1st in the current TOP500 list [65]) has 4608 nodes, and each node comprises two IBM Power9 22-core processors and six Nvidia Volta GPUs.

While large-scale heterogeneous HPC systems provide high performance, there is a consensus that programming heterogeneous systems is not straightforward [32, 60]. Parallelization of sequential legacy code as well as writing parallel programs from scratch is not easy and the difficulty of programming multi-core systems is also known as *programmability wall* [57]. The multi-core shift in computer architecture has accelerated the research efforts in developing new programming frameworks for parallel computing, which should assist domain scientists, for instance, by generating and optimising low-level parallel code for coordination of computations across multiple cores and multiple computers [8, 58].

This study presents the results of a systematic mapping study carried out as part of the European COST Action cHiPSet [16] that addresses High-Performance Modelling and Simulation for Big Data Applications. The mapping study focuses on the main paradigms and properties of programming languages used in High Performance Computing for Big Data processing. Our initial literature search resulted in 420 articles, of which 152 articles were retained for final review after the evaluation of initial search results by domain experts. Results of our mapping study indicate, for instance, that the majority of the used HPC languages in the context of Big Data are text-based general-purpose programming languages and target the end-user community. To evaluate the outcome of the mapping study, we developed a questionnaire and collected the opinions of 57 HPC experts. A comparison of the mapping study outcome with opinions of HPC experts reveals that the key features of HPC programming languages for Big Data are portability, performance and usability. As key issues that need more attention in future research we identified the language learning curve and interoperability (for instance, interoperability of parallel runtime libraries). We consider that the outcome of this study may help in understanding the current limitations in HPC programming languages for Big Data, and it may also help the reader in the identification of issues that need to be addressed in future research.

The rest of the paper is organised as follows. We introduce the concepts of Systematic Mapping Studies and Systematic Literature Reviews and briefly discuss the related work, in Section 2. Section 3 describes the methodology of the Systematic Mapping Study (SMS). We present the obtained results in Section 4. Section 5 evaluates the SMS results via a survey involving HPC experts. Section 6 highlights our major observations and lists challenges and future research directions. The paper is concluded in Section 7 with a summary of the work.

2. Background and Related Work

In this section we describe the concepts of Systematic Mapping Studies and Systematic Literature Reviews and briefly discuss the related work with respect to parallel programming models and languages for data-intensive HPC applications.

Systematic Mapping Studies (SMS) [35, 55] take a *broad and shallow* approach to literature review and are typically used for structuring a research area. They are built on general questions to discover research trends. In this context, the quality assessment of primary studies is optional; for instance, primary studies without empirical evidence can be included.

On the contrary, Systematic Literature Reviews (SLR) take a *narrow and deep* approach to literature review. They are used for gathering and synthesising evidence on a well-defined area. They are built on focused questions to aggregate evidence on a very specific goal. Here, the quality assessment of primary studies is crucial; for instance, primary studies without empirical evidence should not be included.

Related work was so far mainly focused on comparisons of few parallel programming languages [49], but there is a lack of comprehensive literature studies that address HPC programming languages in the context of Big Data. There are also some primary studies regarding tools, such as libraries [2, 24, 27, 29, 40, 62, 69], integrated in known languages used in this type of computation, or APIs and programming models [7, 20, 61, 68, 70, 72]. For instance, there are several well known mainstream libraries, such as the widespread and durable MPI. We discard those from our study as they mostly offer APIS for Programming at the transport layer embedded in General Purpose Languages (like C, C++, Python or others). The main reason is that, to program in those terms, it is an awkward way of thinking for numerical application developers or expert domain users. Thus, if the point is to make HPC more accessible and widely

adopted, we can consider those libraries to be at a wrong level of abstraction, as to increase productivity, it should be hidden the complexity of the communication from the domain scientist.

Diaz et al. [22] give a survey of parallel and distributed programming models for multi- and many-core computing, including PGAS, heterogeneous and hybrid programming models, though not covering the Big Data aspect yet. Dobre and Xhafa [23] review programming models for Big Data processing. Aldinucci et al. [3, 52] compared Big Data frameworks mapping their functional and parallelism aspects on a common lower-level programming model, i.e. the data-flow model. Different general-purpose parallel programming models for single-node multi-/many-core computing were compared quantitatively in terms of programmability and performance in a number of studies, such as, by Ali et al. [4] and by Memeti et al. [49].

3. The Review Process

The methodology used in this Systematic Mapping Study (SMS) follows the methodology proposed by Kitchenham and Charters [34, 35]. The review process has three main phases: *planning*, *conducting* and *reporting*. In the next subsections we outline how the first two phases are carried out for this SMS, followed by an explanation of procedure for data extraction and synthesis of information (input to the current publication, part of the reporting). Finally, we explain general quality concerns that had to take into account during the whole process.

3.1. Planning the SMS

Kitchenham and Charters propose 5 stages within planning: (i) identification of the need for a review, (ii) commissioning, (iii) specifying the research questions, (iv) developing a review protocol and (v) evaluating it.

3.1.1. Identification of the need for a review

The need for this study was identified within the scope of the cHiPSet COST Action [16]. There is a body of knowledge scattered in many publications, which makes it difficult for newcomers to HPC to have an entry point to this domain.

3.1.2. Commissioning the review

A working group focused on *parallel programming models* was assigned with the mission to conduct this systematic mapping study.

3.1.3. The Research Questions

We followed a systematic approach for the definition of our research questions, by adopting a subset of the *PICOC criteria* [56]: Population, Intervention, Comparison, Outcomes, Context. In particular, the *Comparison* criterion is not applicable to our study, as this criterion refers to the software engineering methodology, tool, technology, or procedure to which a given intervention is being compared [35]:

- *Population* is composed of the primary studies on Languages for High-Performance Computing (HPC);
- *Intervention* is the specification or adoption of languages for HPC;
- *Comparison* is not applicable in this framework;
- *Outcomes* is the overview of the language landscape for HPC;
- *Context* is a set of research papers produced by leading experts in HPC, as well as other stakeholders, such as domain scientists, who use HPC in their domains.

The goal of this SMS is to *answer five research questions*, further articulated in 22 sub-questions, which are presented in Table 1. The research questions have been discussed for over one year within the *parallel programming models* Working Group and in the plenary meetings of the CHiPSet COST Action.

3.1.4. Definition of the review protocol

A small task force within the working group developed an initial proposal for the review protocol. The protocol included the selection of information sources, the creation of a search string, the definition of explicit inclusion and exclusion criteria for the primary papers, and the development of a template to support data collection. All these articles were made available to the remaining members of the working group.

3.1.5. Evaluation of the review protocol

The members of the CHiPset working group in charge of this mapping study reviewed and proposed improvements to the initial proposal, leading to the protocol followed in this review, as described in the next section.

3.2. Conducting the SMS

The conduction of an SMS involves the following steps: (i) identification of research, (ii) selection of primary studies, (iii) data extraction and monitoring, and (iv) data synthesis. This process is outlined in Figure 1. Also, the process should also include quality assessment.

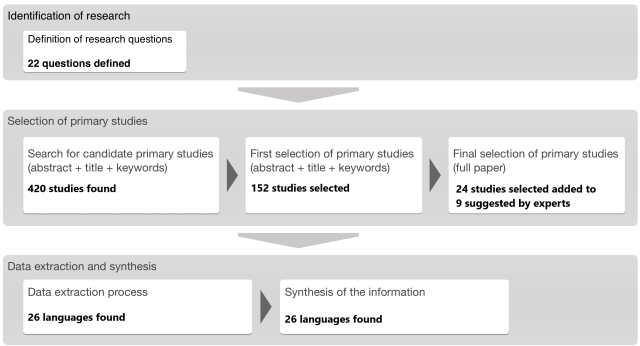

Figure 1: Primary studies selection and filtering

3.2.1. Identification of research

The main goal of an SMS is to *identify a large number of primary studies related to the proposed research questions*.

Table 1: Research Questions that were formulated

ID	Research Questions
RQ1	**What are the categories of languages in use?**
RQ1.1	What are the current research trends in languages for HPC?
RQ2	**What is the nature of the languages for HPC?**
RQ2.1	What kind of language is it?
RQ2.2	What is the execution model that is being used?
RQ2.3	What are the key advantages of the language?
RQ2.4	What are the application domains of the language?
RQ2.5	What are the paradigms underlying the languages?
RQ2.6	Which are the execution stack requirements (?-aaS) to support the artifacts created with those languages?
RQ2.7	What is the existing tool support for the language?
RQ2.8	What are the technologies used to create the language tool suite?
RQ2.9	Does the language target specific hardware?
RQ2.10	What is the purpose of the language?
RQ2.11	What is the preferred language representation type?
RQ3	**What are the typical user profiles for the languages?**
RQ3.1	What are the roles of the users of this language?
RQ3.2	What kind of technical knowledge is required?
RQ4	**How effective are the languages?**
RQ4.1	Is the success of the languages evaluated in the articles?
RQ4.2	– What are the productivity gains brought by the languages reported and what kind of measurement was used? – What are the products' performance gains brought by the languages reported and what kind of measurement was used?
RQ4.3	Is there an explicit comparison with competing approaches? – Is the comparison quantitative, qualitative or both? – What are the comparison methodology and metrics used?
RQ5	**What type of articles are published in the area of programming models for HPC?**
RQ5.1	What is the name of the conference or journal?
RQ5.2	Who is sponsoring the research?
RQ5.3	What kind of research is being reported?

3.2.2. Selection of primary studies

Search for candidate primary studies. We used the keywords in our research questions as a basis for designing our search query. We connected those terms with the AND operator:

```
''Big data'' AND ''Programming Model'' AND
''Programming Language'' AND
''High performance computing''
```

In order to increase the coverage of our study, we included closely related terms, selected by cHiPSet experts (herafter referred to as experts), for our index terms. These synonyms are included in our search string with the OR operator, leading to the final search query:

```
(''Big Data'' OR ''Data Intensive'' OR
''Stream Data'') AND
(''Programming Model'' OR ''Language Model'' OR
(''Modelling Language'') AND
(''Domain Specific Language'' OR
''General Purpose Language'' OR
''Programming Language'' OR
''Programming Framework'') AND
(''HPC'' OR ''High Performance Computing'' OR
''Grid Computing'' OR ''Supercomputing'' OR
''Parallel'' OR ''Concurrent'')
```

We selected 8 digital libraries for performing this SMS. These libraries were chosen to ensure coverage of the main venues where we expected to find candidate primary studies. We asked experts to identify those venues (Table 2). This shortlist of venues was used as a "gold standard" for our automated search.

All searches were based on the title, abstract and keywords of publications from January 2006 to March 2018, a period of time that was considered by the experts as adequate for this SMS. After removing duplicates, we obtained a total of 420 candidate primary studies (Table 3).

First selection of primary studies. We defined a set of *inclusion* and *exclusion* criteria (Table 4) to filter out candidate primary studies that would not contribute to answering our research questions. These criteria were set before the conduction of the automated search, while planning this SMS. They were applied by analysing the *title*, *abstract* and *keywords* of each of the 420 candidate primary studies. We took a conservative approach at this stage: when in doubt, we kept the candidate primary studies. It worth mentioning that all language application publications were excluded, as the focus of our study is on the design and development of languages and not about examples of their use. At the end of this step, we excluded 268 candidate studies, according to those criteria. This resulted in 152 candidate studies selected for the next steps.

Final selection of primary studies. We refined the selection from the previous step by reading the full papers, rather than just the title, abstract, and keywords in order to reapply the inclusion and exclusion criteria, now to the full contents of the candidate primary studies. The goal was to filter out papers that, in practice, were beyond the scope of our research questions, even if their title and abstract suggested otherwise. Specifically, studies reporting libraries (rather than languages) were sieved out. In addition, in line with the guidelines [35] and common practice in performing the selection of primary studies [34], we also included a set of relevant papers that were jointly identified by experts. These were primary studies directly known by those experts that fell within the scope of this SMS. In this step 9 papers were added to the batch. These additions were mainly due to the usage of slightly different terminology, not captured by our automatic search query. The selection of the key terms in the automated search query has to balance coverage with the minimisation of false positives, to ensure the feasibility of the study in terms of scale (e.g. adding an extra key term can easily lead to hundreds of extra papers to analyse). Manually adding

Table 2: Conferences and journals considered in the study

Conferences (11)	Journals (8)
GTC / GPGPU conference	ACM Transactions on Parallel Computing
IEEE Intl. Parallel and Distributed Processing Symposium	Concurrency and Computation Practice and Experience
Intl. Conference on Parallel Processing	Future Generation Computer Systems
Intl. Conference on Supercomputing	IEEE Computing in Science and Engineering
Intl. European Conference on Parallel and Distributed Computing	Journal of Parallel and Distributed Computing
Intl. Supercomputing Conference	Journal of Supercomputing
Parallel Computing Conference	Parallel Computing
Principles and Practice of Parallel Programming	Scientific Programming
SIAM Conference on Parallel Processing for Scientific Computing	
Supercomputing Conference	

Table 3: Candidate primary studies per digital library

Source Name	Number of Studies
academia.edu	1
ACM Digital Library	16
Compendex	6
Elsevier Science Direct	289
Google Scholar	32
IEEE Xplore	3
ResearchGate	3
Springer Link	70
Total of articles found:	**420**

known papers to the batch is a pragmatic way of mitigating the risk of missing important references, while keeping the sample size manageable. Overall, 33 primary studies were selected for data extraction (please consult the companion site [5] for the complete list of resulting selected papers).

3.3. Data extraction and Synthesis

Data Extraction Process. To enhance data consistency, the data extraction process was guided by a template implemented as a shared spreadsheet. A team of 19 experts (the authors of this paper) participated in data extraction directly reading the primary studies selected during the previous stage and answering to the research questions listed in Table 1. Each study has been randomly assigned to one expert. We identified a total of 26 languages, including 8 Domain-Specific Languages, 14 general purpose languages, and 4 domain-specific languages embedded in general purpose languages.

Synthesis of the Information. During this stage, the previously extracted information was cross-reviewed by all experts with the aim of identifying possible clerical errors during previous steps and taking into account the research questions formulated (the studies that referred to the languages used for HPC). After this task, a shared document was created for entering answers to the research questions and referring to the found languages.

3.4. SMS Quality concerns

When conducting an SMS we need to address the following quality concerns:

- *Are the inclusion and exclusion criteria described and appropriate?* This aspect has been directly addressed during the *Selection of the Primary Study* stage (Sec. 3.2.2): all the criteria were explicitly defined (see Table 4) and the set of 152 studies at this step was filtered against inclusion and exclusion criteria.
- *Does the literature search cover all relevant studies?* We addressed this during the *Search of Primary Study* stage (Sec. 3.2.2) by selecting appropriate keywords with the help of the cHiPSet experts and then complementing the set of papers resulting from our automatic search and subsequent expert opinion-based filtering with a set of papers known by experts, to capture papers that were missed by the automatic search.
- *Was the information from the primary studies correctly extracted?* Each primary study was inspected by at least one of the authors of this SMS. Although there is no absolute guarantee that important information was not lost, as a safeguard mechanism, a sample of papers were inspected by a second expert. No information loss was detected.

4. Discussion of the Results

4.1. Programming languages for HPC

Throughout this section, the programming languages for data-intensive HPC applications are firstly categorised by *Research Question 1* (RQ1), then analysed through their main features with respect to *Research Questions 2 – 5* (RQ2 – RQ5).

In addition to the information gathered on the existing languages, several studies have been found regarding libraries and Application Programming Interfaces (API), which were not considered because they are integrated in the languages mentioned throughout this section. Respecting these conditions, at the end of this process, we identified *33 articles, to which corresponded 26 languages*.

In [5] (data set companion) it is presented a list of the identified languages and their characteristics. Due to the similarity of the answers given to the research questions, some languages were grouped, for instance, C and C++, or Python and R.

Table 4: Inclusion and Exclusion criteria of the articles

Inclusion Criteria	Exclusion Criteria
Study must have addressed HPC research	Irrelevant study that lay outside the core HPC research field
Peer reviewed study that had been published in journal, conference and workshop	– Non-peer reviewed study (abstract, tutorial, editorial, slides, talk, tool demonstration, poster, panel, keynote, technical report) – Peer-reviewed but not published in journal, conference, workshop (e.g., PhD thesis, book, patent)
Study must be written in English	Study not in English
Study must be accessible electronically	Electronically non-accessible study
Study is related with Computer science literature & Systems area	Article published before 2006

RQ1: Which are the categories of languages in use?. We have categorised the encountered programming languages in four classes:

1. *Domain Specific Language* (DSL), which is a language adapted to a specific application domain that offers appropriate annotations and abstractions [37, 38, 67];
2. *General Purpose Language* (GPL), which is a programming language designed to be used in writing software in a wide variety of application domains [38];
3. *DSL embedded in another DSL*;
4. *DSL embedded in a GPL.*

As shown in Figure 2, 54% of the languages have been classified as being *GPL (14 languages)*. Please note that 31% of these languages (8 languages) are DSL. The remaining 4 languages were considered DSL embedded in an GPL. Though the study considered that a DSL embedded in another DSL could be found, our analysis did not find any in the domain of data-intensive HPC applications.

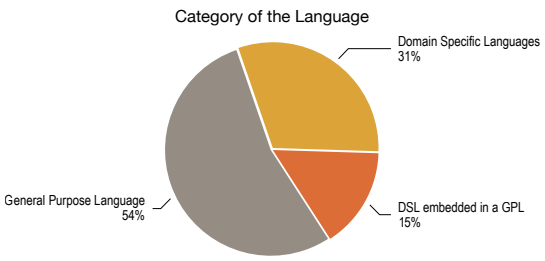

Figure 2: RQ1 – Which are the categories of languages in use?

RQ2: What is the nature of the languages for HPC?. The objective of this research question was to characterise the nature of the encountered languages for HPC. To accomplish this objective several sub research questions were identified and their results are here discussed.

The encountered languages use *Virtual Execution Environments* such as Java Virtual Machine (JVM) and *Distributed Middlewares* like Hadoop Distributed File System (HDFS) and IBM InfoSphere *(RQ 2.2)*.

The key advantage of the encountered languages is the *usability* of the language. The *ease of configuration*, *portability*, *orchestration* and *performance* of the language are the other advantages that are perceived as important. Other advantages referred to were *visualisation of user-initiated query results*, *ease to express constraint problems* and *enabling high-level parallel programming using skeletons*, see Figure 3.

The application domain of the encountered languages is the *parallel analysis of Big Data (RQ 2.4)*.

The main paradigm underlying the encountered languages is *Object-Oriented*, followed by *Functional (see Figure 4)*.

The encountered languages generally *support multiple OS*. Some of them require *Message Passing Middleware* such as Message Passing Interface (MPI) and *I/O architectures* such as HDFS. Some languages require *libraries* like Apache Hadoop to support the artifacts created *(RQ 2.6)*.

Concerning tools supporting the languages, *compilers* are the most well represented support tool, followed by *tool suite* and *interpreters (see Figure 5)*. IDE's, simulation and validation tools are comparatively more scarce.

There are some technologies used to create the languages tool suite. These are *compiler generators* such as IBM Infosphere Streams, some *XML based technologies* (e.g. in Java, they used a syntax to describe classes and methods), and some *frameworks* such as Knowledge Discovery Toolbox (KDT) *(RQ 2.8)*.

Most of the surveyed languages *do not target specific hardware* (*85%*). About *40%* of the encountered languages target *general-purpose multi-core architectures or GPUs (RQ 2.9)*.

The main purpose of the encountered languages is to *Implement the solution*, followed by the *Formalisation of the solution*, *Formalisation of the requirements of the problem* and *Data Interpretation (see Figure 6)*.

Results for the language representation type revealed that there is a concrete syntax for all the encountered languages and the preferred representation type of *76%* of them is *Textual (see Figure 7)*.

RQ3: What are the typical user profiles for the languages?. Figure 8 displays the distribution of the typical user profiles for the languages. Most of the identified languages are used by *end-users*, who utilise the language to solve problems. About *16,7%* of the languages are used by *developers*, who utilise the language to create tools/setups/solutions for other users.

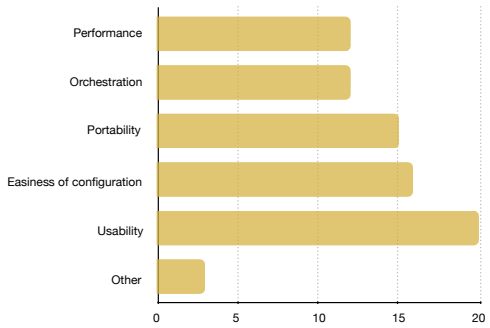

Figure 3: RQ2.3 – What are the key advantages of the language?

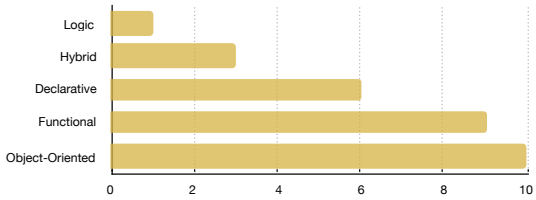

Figure 4: RQ2.5 – What are the paradigms underlying the languages?

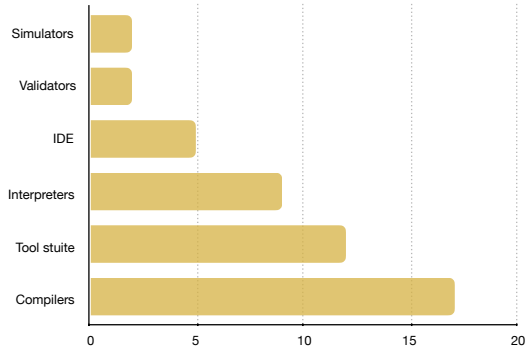

Figure 5: RQ2.7 – What is the existing tool support for the language?

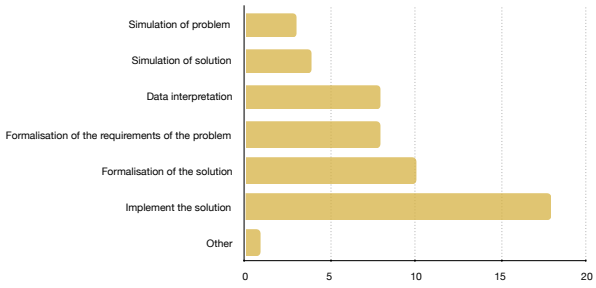

Figure 6: RQ2.10 – What is the purpose of the language?

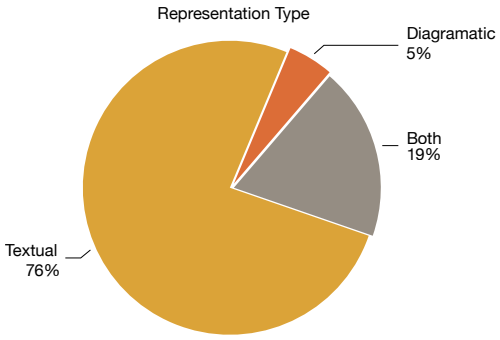

Figure 7: RQ2.11 – What is the preferred language representation type?

The technical knowledge required to use these languages is mainly based on the *languages themselves or those where they are embedded*, some *hardware knowledge*, *clusters* and *theoretical background related to the application domains* of the language in question. As an example, to use FastFlow we should be accustomed to C++ language, have some knowledge about CPU and theoretical background about Streaming Applications *(RQ 3.2)*.

RQ4: How effective are the languages?. Effectiveness is articulated in three aspects, addressed by RQ4.1 (success), RQ4.2 (productivity gain), and RQ4.3 (advantage against competitive approaches). Around 88% of the articles reviewed in this study *evaluated the corresponding languages for success*.

Figure 9 depicts productivity gains brought by the languages studied in this paper based on *qualitative* and *quantitative* analysis. Based on *quantitative* analysis, the most referred productivity gain brought by the reported languages was the *easiness to use the language*, followed by the *learnability*. With respect to *qualitative* analysis, the most referred productivity gain brought by the reported languages was the *easiness to use*, followed by the *lower cognitive overload* and the *learnability*. We may observe that productivity gains brought by the reported languages are mainly estimated using *qualitative methods*.

The chart in Figure 10 depicts the products' performance gains brought by the studied languages in this paper. Based on *quantitative* analysis, the most referred products' performance gains brought by the reported languages were the *computation efficiency* and the *scalability*. With respect to *qualitative* analysis, the most referred products' performance gain brought by the reported languages was *evolvability / maintainability*, followed by *scalability*. Unlike the productivity gains, the products performance gains brought by the reported languages are mainly estimated using *quantitative methods*.

According to the statistics, *64%* of the articles included an explicit comparison between the reported language and other

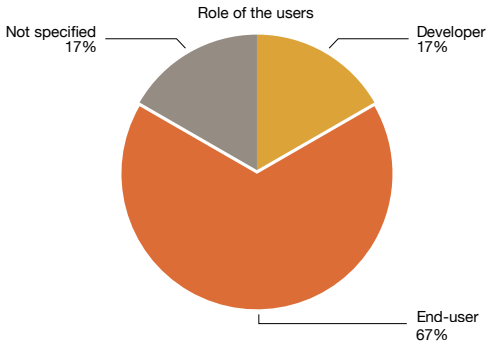

Figure 8: RQ3.1 – What are the roles of the users of this language?

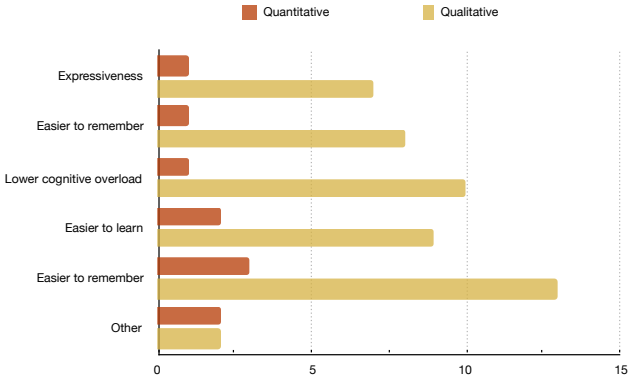

Figure 9: RQ4.2 – What are the productivity gains brought by the languages reported and what kind of measurement was used?

competing approaches. About half of the articles included an explicit comparison of the proposed language with respect to distinct settings/ context/ configurations *(RQ 4.3)*.

All metrics were obtained using *quantitative methods*. The most frequently used metric was *the computational time*, followed by *the lines of code and the satisfaction (see Figure 11)*.

RQ5: What types of articles are published in the area of programming models for HPC?. The scientific journals that published most of articles included in this study are *"Future Generation Computer Systems"*, *"Parallel Computing"*, and *"Journal of Parallel and Distributed Computing"*, as depicted in Figure 12 *(RQ 5.1)*.

Most of these articles were sponsored by *public or both public and private funds (RQ 5.2)*. Each of these articles can be classified as a case study, as an experiment report, or a comparative assessment; we observed that there was a *greater occurrence of experience reports*; *case studies and comparative assessments were found in a similar number*, as shown in Figure 13.

5. SMS Validation by HPC Experts

To validate the SMS results that are presented in Section 4, we conducted a survey (in November 2018) with 28 HPC experts involved in the cHiPSet COST action using a questionnaire (see [5]) to which we added, in October 2019, 29 HPC

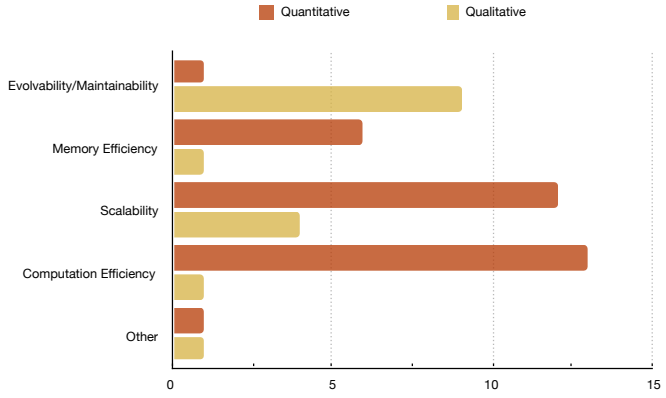

Figure 10: RQ4.2 – What are the products' performance gains brought by the languages reported and what kind of measurement was used?

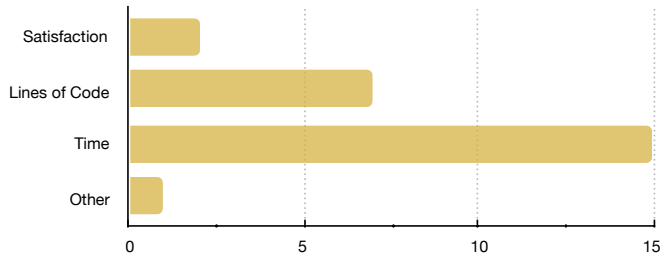

Figure 11: RQ4.3 – Number of articles using each metric: What are the metrics used?

Table 5: List of the encountered languages in the Systematic Mapping Study

Domain Specific Languages (DSL)
CineGrid Description Language + Network D Language [36]
Crucible [18]
e-Science Central WFMS [14]
Higher-order "chemical programming" language [26]
Liszt [21]
Mendeleev [17]
MiniZinc [13]
General Purpose Languages (GPL)
Bobolang [25]
C/C++ [6, 10, 24, 39, 48, 54, 61]
Erlang [66]
FastFlow [2, 51]
Goal Language supported by RuGPlanner [31]
Java [6, 15, 45, 46, 48]
OpenCL [10, 33]
Python/R [6, 28, 42]
Scout [47]
Selective Embedded Just-In-Time Specialization [43]
SkIE-CL [19]
Swift [44, 71]
DSL embedded in GPL
Pipeline Composition (PiCo) [53]
Spark Streaming and Spark SQL [41]
Weaver [12]

experts which were not involved in that COST action. Participants were recruited through convenience sampling, and contacted directly by the authors of this paper. In total, we received

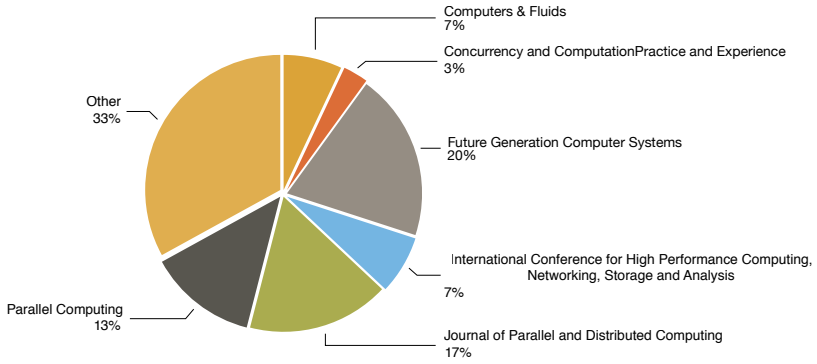

Figure 12: Which conferences and journals publish articles about languages for HPC? - RQ5.1

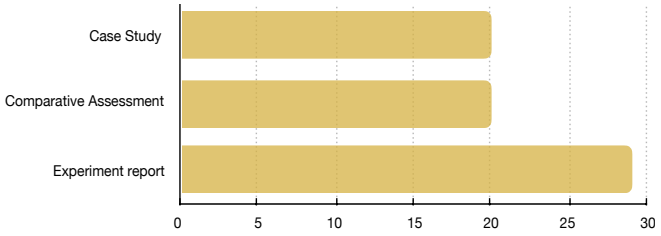

Figure 13: Number of articles reporting each type of research - RQ5.3

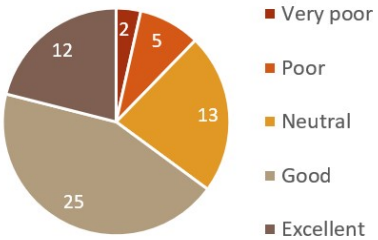

Figure 15: Expert sample: Self-estimation of technical knowledge in HPC programming languages/frameworks

57 filled survey forms.

The majority of respondents have more than 10 years experience in HPC (see Figure 14), and rate their own technical knowledge in HPC languages and tools as good or excellent (Figure 15). We have a fairly balanced sample of HPC framework and tool developers and primary HPC users (see Figure 16). Although most of the respondents work in computer science (including AI), several of them also work in particular application domains, such as Bioinformatics, albeit with a larger dispersion among those domains (see Fig. 17. 11 of the survey answers came from coauthors of this systematic mapping study, 17 from other cHiPSet members not involved in the SMS, and the remaining 29 from other respondents recruited through convenience sampling by this paper's co-authors. Hence, the expert sample may be somewhat biased towards computer science professors developing HPC programming frameworks.

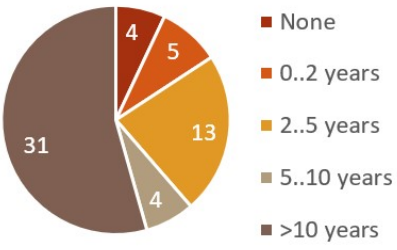

Figure 14: Expert sample: Level of experience of working in HPC

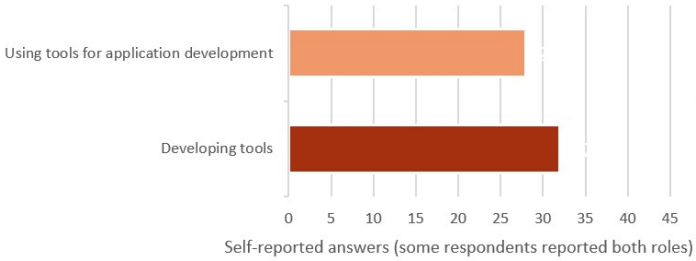

Figure 16: Expert sample: HPC Tool Developers vs. Users

Figure 18 shows the list of HPC programming languages, frameworks and tools that the respondents (a) use and (b) know. In the survey the questions had explicitly distinguished between "true" programming languages discussed in Section 4 and non-language programming frameworks (such as, libraries) and other programming tools. From the answers filled in on the survey form, it also became clear that a significant number of respondents do not see any particular difference between language and non-language programming frameworks or at least do not consider that as important in practice. For that reason, we show the answers for both (a) and (b) in the same diagram, using different colours. The three main blocks in Figure 18 coarsely structure more than 100 mentioned programming frameworks and

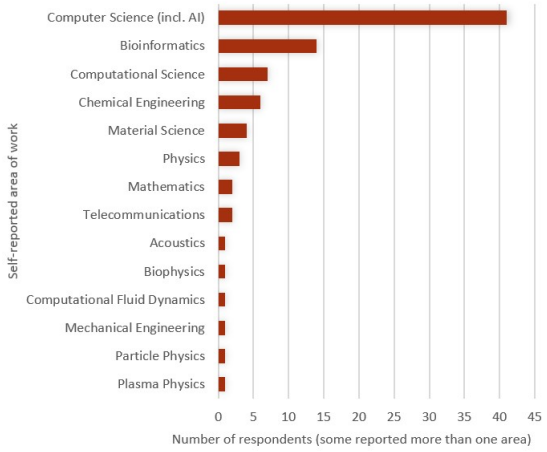

Figure 17: Expert sample: Work areas

tools into general-purpose frameworks, domain-specific (esp., machine learning) frameworks and tools.

Figure 18 shows high numbers of answers for a few (sequential) base-level programming languages of central importance in data-intensive HPC, including both the traditional HPC base languages C/C++ (highest score of all) and Fortran (less frequent) as well as the base languages Python, Java, Scala and (less frequently) R, which are the most common ones for use with current Big Data and machine learning programming frameworks. Among the truly parallel HPC programming frameworks, MPI dominates as expected together with OpenMP and (somewhat less frequent) pthreads, but also the GPU programming models CUDA, OpenCL and OpenACC score quite high numbers of answers. PGAS languages such as UPC, X10 and Chapel are known to a number of experts in the sample, but are hardly used by them. The most-used big data and machine learning programming frameworks among the answering experts are Hadoop / MapReduce, Spark, Tensorflow and Keras, though they receive less scores than the traditional parallel programming models. A number of skeleton/ pattern based parallel programming frameworks also occur in the list, with FastFlow and SkePU scoring the largest number of answers. Figure 18 also reveals a "long tail" of many programming frameworks and tools that were only named by one two persons.

Figure 19 charts the answers by the expert sample on the question about perceived advantages of the programming frameworks used. Most answers focused on performance/ efficiency, followed by programmability (easy coding), portability and availability of features. We also asked for honest answers about other factors that led to adoption of a certain programming framework. Although only few answered this question, it revealed factors such as popularity, mandatory use in a project or the use in teaching parallel programming, see Figure 20.

Comparing Figure 18 with the list of programming frameworks resulting from the systematic literature study in Table 5, we can see a number of similarities; for example, the expert sample confirms the high significance of C++ for the HPC domain. In contrast, apart from machine learning frameworks, other domain-specific languages tend to be more used in specific niches, as suggested by the lower frequency with which those languages are referred by our survey respondents. A possible explanation is that the expert sample answers focus on tools being used, i.e., mature technology that has been developed years (sometimes decades) ago, while the literature sample feeding the SMS results contained quite a number of papers about more exotic domain-specific programming frameworks that represent the current front of research in the design and development of programming frameworks rather than their use for solving actual HPC and big data application problems.

6. Observations, research challenges and future directions

In this section we summarise our major observations, research challenges and future directions in the domain of HPC programming languages for Big Data processing.

Major observations based on the reviewed literature:

- General-purpose programming languages are used most frequently (54% of observed cases);
- Majority (that is 76%) of the languages were text-based;
- Usability (Effectiveness, Efficiency, Satisfaction) is considered the key feature of the used language;
- Simulators, validators or IDEs are not often available;
- About 67% of the language users are end-users;
- 87% of the reviewed literature has provided a kind of language evaluation, with majority of the cases using computational time as metric;
- Majority of the reviewed literature reports experiments.

Major observations based on the opinions of HPC experts that responded to our questionnaire:

- Key features of a HPC programming language for data-intensive applications are performance / efficiency, programmability, and availability of tools / libraries;
- C/C++ and Python general-purpose programming languages are frequently used by HPC experts;
- Most frequently used parallel programming frameworks by HPC experts are MPI and OpenMP;
- While significant research effort has been invested in PGAS languages (such as, UPC, X10 and Chapel), they are not often used in practice;
- Programming models / languages for heterogeneous computing – CUDA, OpenCL and OpenACC – are popular among HPC experts;
- Apart from machine learning frameworks, other domain-specific languages tend to be more used in specific niches, as suggested by the lower frequency with which those languages are referred by our survey respondents;

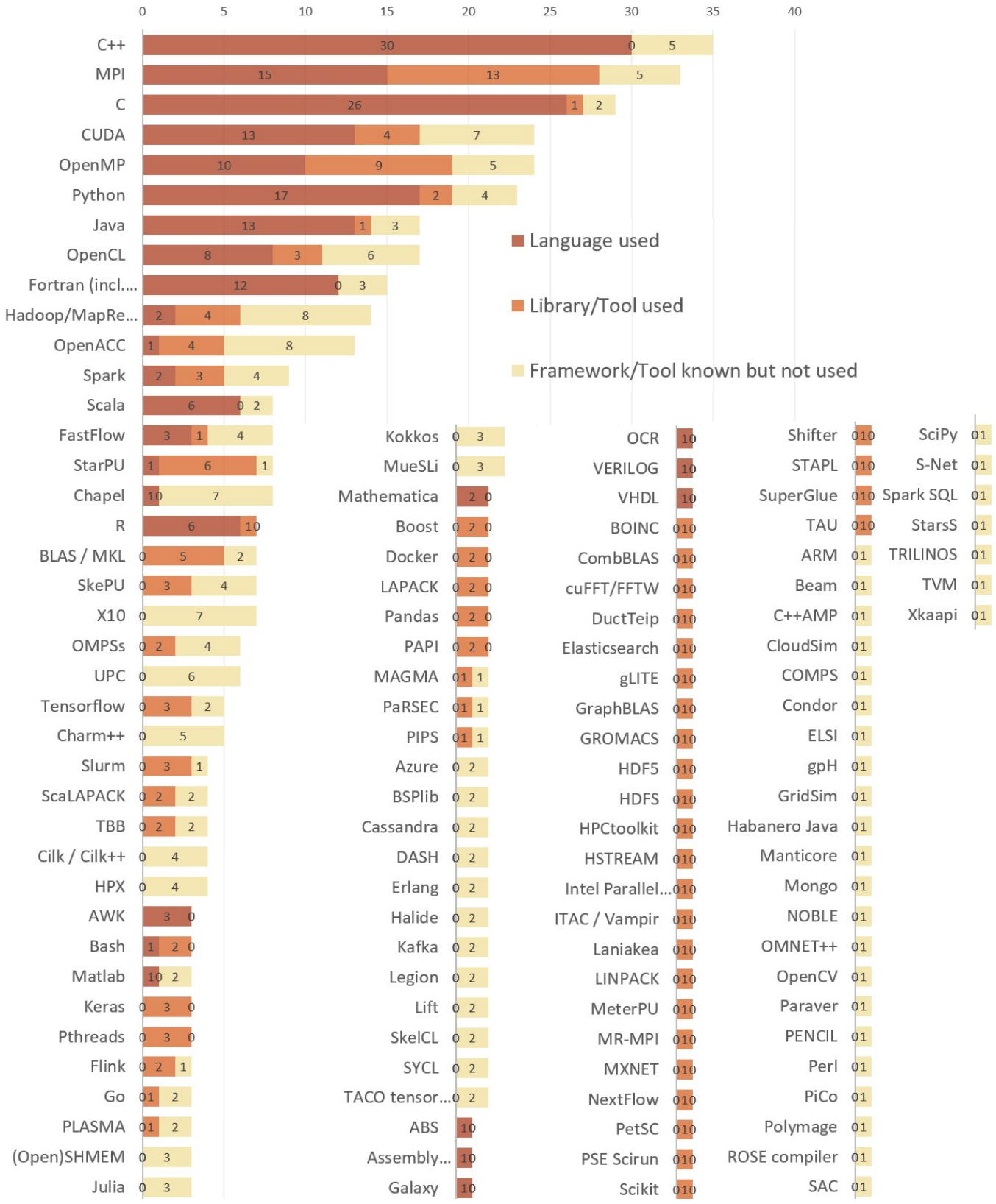

Figure 18: Expert sample: HPC programming languages, frameworks and tools used and known

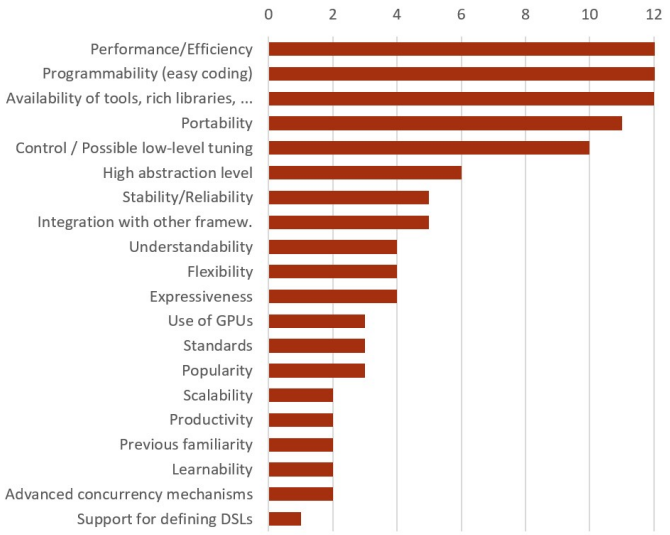

Figure 19: Expert sample: Mentioned advantages of HPC programming languages, frameworks and tools used

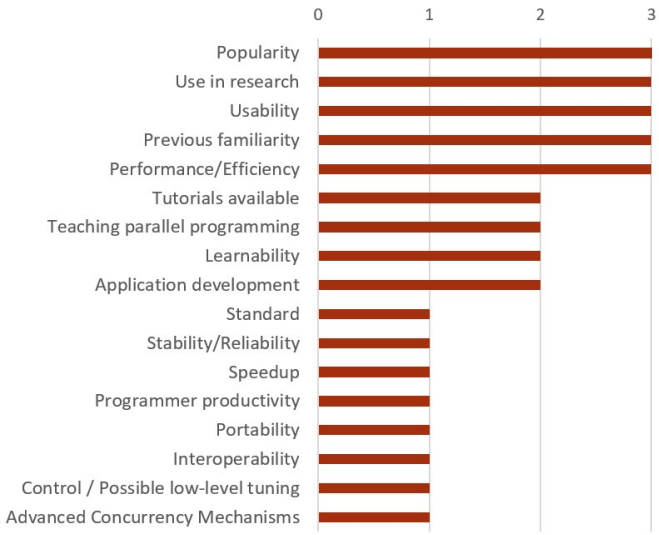

Figure 20: Expert sample: Additional reasons for using the HPC programming languages, frameworks and tools used

- Majority of the HPC experts that responded to our questionnaire develop support tools.

Major challenges and future research directions include:

- Learning curve of the HPC programming languages is a major challenge that needs to be addressed in future;
- Development of IDEs and supporting tools, e.g. simulators and validators, to fill in the current gap in HPC languages.

7. Summary

We performed a systematic mapping study to examine the main paradigms and properties of programming languages used in High-Performance Computing for data-intensive applications. We performed an automated search for articles in eight different digital databases, to select papers published from January 2006 to March 2018. We used a two-step filtering process to select the relevant primary studies. In addition, we also included 9 extra papers that were known among the experts to be relevant for our study. In the end, we conducted data extraction on 33 papers. We identified 26 languages for HPC for data-intensive applications. We provided a comprehensive classification of the languages encountered and their usage and evaluation by different criteria. The majority of the used HPC languages in the context of data-intensive applications are text-based general-purpose programming languages and target the end-user community. Those users, from different application domains, may lack a strong background in computing and HPC technologies. This creates several challenges: the expressiveness of the most commonly used languages, in terms of domain rules, is closer to the solution domain (programming technologies) than to the problem domain (the domain of application) - this mismatch introduces added accidental complexity; usability aspects such as learnability, or programmability, may benefit from the adoption of DSLs. As a complementary form of evaluation, we also performed a survey for collecting opinions from HPC experts on the languages they commonly use for HPC for data-intensive applications. Based on the mapping study outcomes and HPC experts opinions, we found that the key features of HPC programming languages for data-intensive applications are performance/efficiency, programmability, and availability of tools/libraries. This suggests that there is an opportunity for bridging the gap between HPC and its target application domains. One possible way of bridging this gap is to support a wider DSL adoption. DSLs can be as efficient as GPLs, and more accessible to domain experts (increased programmability), as long as robust tool/libraries support is made available to domain experts. However, this requires a stronger investment in developing languages that capture the target domain knowledge and established practices, while leveraging the HPC approaches.

As highlighted by the HPC expert survey (in line with common sense), the durable Message Passing Interface (MPI) standard, with send/receive, broadcast, reduction operators and global synchronisations (barriers), has remained the programming paradigm of choice for over twenty years to construct parallel applications composed of tens to hundreds of thousands of communicating processes. The reason dêtre of HPC is, by definition, high performance and a highly optimised MPI code exhibit unparalleled performances because each MPI application, despite its complexity, is conceived as a single program designed on developer-based precise knowledge of the whole code, the partitioning of data and the communication overheads. In the hierarchy of abstractions, this is only slightly above toggling absolute binary in the front panel of the machine. We believe that this approach is unable to effectively scale to support the main-

stream of software development where human productivity, total cost and time to a solution are equally, if not more, important aspects. However, to date, attempts to develop high-level programming abstractions, DSL and environments for HPC have mostly remained in the research labs, and they failed to scale to large scale scientific and industrial applications.

Acknowledgements

This work is a result of activities from COST Action IC1406 High-Performance Modelling and Simulation for Big Data Applications (cHiPSet), funded by the European Cooperation in Science and Technology.
We also aknowledge FCT, Portugal for grants: NOVA LINCS Research Laboratory Ref. UID/ CEC/ 04516/ 2019); INESC-ID Ref. UID/ CEC/ 50021/ 2019; BioISI Ref. UID/ MULTI/ 04046/ 2103;LASIGE Research Unit Ref. UID/ CEC/ 00408/ 2019.

References

[1] E. Abraham, C. Bekas, I. Brandic, S. Genaim, E. B. Johnsen, I. Kondov, S. Pllana, and A. Streit. Preparing HPC Applications for Exascale: Challenges and Recommendations. In *2015 18th International Conference on Network-Based Information Systems*, pages 401–406, Sep. 2015.

[2] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati. *Fastflow: High-Level and Efficient Streaming on Multicore*, chapter 13, pages 261–280. Wiley-Blackwell, 2017.

[3] M. Aldinucci, M. Drocco, C. Misale, and G. Tremblay. *Languages for Big Data analysis*. Springer International Publishing, Cham, 2019.

[4] A. Ali, U. Dastgeer, and C. Kessler. OpenCL on shared memory multicore CPUs. In E. Ayguade, B. Gaster, L. Howes, P. Stenström, and O. Unsal, editors, *Proc. 5th HiPEAC Workshop on Programmability Issues for Multicore Computers (MULTIPROG-2012)*, Jan. 2012. Available at: Linköping University Electronic Press, http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-93951.

[5] V. Amaral, B. Norberto, M. Goulão, M. Aldinucci, S. Benkner, A. Bracciali, P. Carreira, E. Celms, L. Correia, C. Grelck, H. Karatza, C. Kessler, P. Kilpatrick, H. Martiminiano, I. Mavridis, S. Pllana, A. Respício, J. Simão, and L. Veiga. Companion data of a Systematic Mapping Study of Programming Languages for Data-Intensive HPC Applications, May 2019. This work results from COST Action IC1406 High- Performance Modelling and Simulation for Big Data Applications (cHiPSet), funded by the European Cooperation in Science and Technology.

[6] R. M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes, and R. Sirvent. Comp superscalar, an interoperable programming framework. *SoftwareX*, 3:32–36, December 2015.

[7] M. B. Belgacem and B. Chopard. Muscle-hpc: A new high performance api to couple multiscale parallel applications. *Future Generation Computer Systems*, 67:72–82, February 2017.

[8] S. Benkner, S. Pllana, J. L. Traff, P. Tsigas, U. Dolinsky, C. Augonnet, B. Bachmayer, C. Kessler, D. Moloney, and V. Osipov. Peppher: Efficient and productive usage of hybrid computing systems. *IEEE Micro*, 31(5):28–41, 2011.

[9] M. A. Beyer and D. Laney. The importance of big data: A definition. *Stamford, CT: Gartner*, pages 2014–2018, June 2012.

[10] A. P. D. Binotto, M. A. Wehrmeister, A. Kuijper, and C. E. Pereira. Sm@rtconfig: A context-aware runtime and tuning system using an aspect-oriented approach for data intensive engineering applications. *Control Engineering Practice*, 21(2):204–217, February 2013.

[11] B. Brown, M. Chui, and J. Manyika. Are you ready for the era of 'big data'. *McKinsey Quarterly*, 4(1):24–35, 2011.

[12] P. Bui, L. Yu, A. Thrasher, R. Carmichael, I. Lanc, P. Donnelly, and D. Thain. Scripting distributed scientific workflows using weaver. *Concurrency and Computation: Practice and Experience*, 24(15):1685–1707, November 2011.

[13] R. Caballero, P. J. Stuckey, and A. Tenorio-Fornes. Two type extensions for the constraint modeling language minizinc. *Science of Computer Programming*, 111:156–189, 2015.

[14] J. Cala, E. Marei, Y. Xu, K. Takeda, and P. Missier. Scalable and efficient whole-exome data processing using workflows on the cloud. *Future Generation Computer Systems*, 65:153–168, December 2016.

[15] G. Caruana, M. Li, and Y. Liu. An ontology enhanced parallel svm for scalable spam filter training. *Neurocomputing*, 108:45–57, May 2013.

[16] ICT COST Action IC1406, High-Performance Modelling and Simulation for Big Data Applications (cHiPSet). `http://chipset-cost.eu/`, Apr. 2019 (last accessed).

[17] P. Coetzee and S. Jarvis. Goal-based composition of scalable hybrid analytics for heterogeneous architectures. *Journal of Parallel and Distributed Computing*, 108:59–73, October 2017.

[18] P. Coetzee, M. Leeke, and S. Jarvis. Towards unified secure on-and offline analytics at scale. *Parallel Computing*, 40(10):738–753, December 2014.

[19] M. Coppola and M. Vanneschi. High-performance data mining with skeleton-based structured parallel programming. *Parallel Computing*, 28(5):793–813, May 2002.

[20] E. Dede, Z. Fadika, M. Govindaraju, and L. Ramakrishnan. Benchmarking mapreduce implementations under different application scenarios. *Future Generation Computer Systems*, 36:389–399, July 2014.

[21] Z. DeVito, N. Joubert, F. Palacios, S. Oakley, M. Medina, M. Barrientos, E. Elsen, F. Ham, A. Aiken, K. Duraisamy, et al. Liszt: a domain specific language for building portable mesh-based pde solvers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 9. ACM, 2011.

[22] J. Diaz, C. Muñoz-Caro, and A. Ninõ. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Trans. on Par. and Distr. Syst.*, 23(8):1369–1386, Aug. 2012.

[23] C. Dobre and F. Xhafa. Parallel programming paradigms and frameworks in big data era. *International Journal of Parallel Programming*, 42(5):710–738, Oct 2014.

[24] J. Enmyren and C. W. Kessler. Skepu: a multi-backend skeleton programming library for multi-gpu systems. In *Proceedings of the fourth international workshop on High-level parallel programming and applications*, pages 5–14. ACM, September 2010.

[25] Z. Falt, D. Bednrek, M. Krulis, J. Yaghob, and F. Zavoral. Bobolang: A language for parallel streaming applications. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 311–314. ACM, 2014.

[26] H. Fernandez, C. Tedeschi, and T. Priol. Rule-driven service coordination middleware for scientific applications. *Future Generation Computer Systems*, 35:1–13, June 2014.

[27] M. B. Giles, G. R. Mudalige, B. Spencer, C. Bertolli, and I. Reguly. Designing op2 for gpu architectures. *Journal of Parallel and Distributed Computing*, 73(11):1451–1460, November 2013.

[28] K. Hinsen, H. P. Langtangen, O. Skavhaug, and Å. Ødegård. Using b sp and python to simplify parallel programming. *Future Generation Computer Systems*, 22(1-2):123–157, 2006.

[29] D. Hünich, A. Knüpfer, and J. Gracia. Providing parallel debugging for dash distributed data structures with gdb. *Procedia Computer Science*, 51:1383–1392, 2015.

[30] Ishwarappa and J. Anuradha. A brief introduction on big data 5vs characteristics and hadoop technology. *Procedia Computer Science*, 48:319 – 324, 2015. International Conference on Computer, Communication and Convergence (ICCC 2015).

[31] E. Kaldeli, A. Lazovik, and M. Aiello. Domain-independent planning for services in uncertain and dynamic environments. *Artificial Intelligence*, 236:30–64, July 2016.

[32] C. Kessler, U. Dastgeer, S. Thibault, R. Namyst, A. Richards, U. Dolinsky, S. Benkner, J. L. Träff, and S. Pllana. Programmability and performance portability aspects of heterogeneous multi-/manycore systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1403–1408. EDA Consortium, 2012.

[33] M. Kim, Y. Lee, H.-H. Park, S. J. Hahn, and C.-G. Lee. Computational fluid dynamics simulation based on hadoop ecosystem and heterogeneous computing. *Computers & Fluids*, 115:1–10, July 2015.

[34] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering a sys-

tematic literature review. *Information and Software Technology*, 51(1):7–15, 2009.

[35] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*, pages 1–57. sn, 2007.

[36] R. Koning, P. Grosso, and C. de Laat. Using ontologies for resource description in the cinegrid exchange. *Future Generation Computer Systems*, 27(7):960–965, July 2011.

[37] T. Kosar, S. Bohra, and M. Mernik. Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77–91, 2016.

[38] T. Kosar, N. Oliveira, M. Mernik, M. J. V. Pereira, M. repinÅek, D. da Cruz, and P. R. Henriques. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems*, 7(2):247–264, 2010.

[39] T.-Y. Liang, H.-F. Li, Y.-J. Lin, and B.-S. Chen. A distributed ptx virtual machine on hybrid cpu/gpu clusters. *Journal of Systems Architecture*, 62:63–77, January 2016.

[40] T.-Y. Liang, C.-Y. Wu, C.-K. Shieh, and J.-B. Chang. A grid-enabled software distributed shared memory system on a wide area network. *Future Generation Computer Systems*, 23(4):547–557, May 2007.

[41] G. Liu, W. Zhu, C. Saunders, F. Gao, and Y. Yu. Real-time complex event processing and analytics for smart grid. *Procedia Computer Science*, 61:113–119, 2015.

[42] A. Luckow, M. Santcroos, A. Zebrowski, and S. Jha. Pilot-data: an abstraction for distributed data. *Journal of Parallel and Distributed Computing*, 79-80:16–30, May 2015.

[43] A. Lugowski, S. Kamil, A. Buluç, S. Williams, E. Duriakova, L. Oliker, A. Fox, and J. R. Gilbert. Parallel processing of filtered queries in attributed semantic graphs. *Journal of Parallel and Distributed Computing*, 79:115–131, May 2015.

[44] K. Maheshwari, E.-S. Jung, J. Meng, V. Morozov, V. Vishwanath, and R. Kettimuthu. Workflow performance improvement using model-based scheduling over multiple clusters and clouds. *Future Generation Computer Systems*, 54:206–218, January 2016.

[45] C. Mateos, A. Zunino, and M. Campo. An approach for non-intrusively adding malleable fork/join parallelism into ordinary javabean compliant applications. *Computer Languages, Systems & Structures*, 36(3):288–315, 2010.

[46] C. Mateos, A. Zunino, M. Hirsch, M. Fernández, and M. Campo. A software tool for semi-automatic gridification of resource-intensive java bytecodes and its application to ray tracing and sequence alignment. *Advances in Engineering Software*, 42(4):172–186, 2011.

[47] P. McCormick, J. Inman, J. Ahrens, J. Mohd-Yusof, G. Roth, and S. Cummins. Scout: a data-parallel programming language for graphics processors. *Parallel Computing*, 33(10-11):648–662, November 2007.

[48] A. Meade, D. K. Deeptimahanti, J. Buckley, and J. Collins. An empirical study of data decomposition for software parallelization. *Journal of Systems and Software*, 125:401–416, March 2017.

[49] S. Memeti, L. Li, S. Pllana, J. Kołodziej, and C. Kessler. Benchmarking opencl, openacc, openmp, and cuda: programming productivity, performance, and energy consumption. In *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, pages 1–6. ACM, July 2017.

[50] S. Memeti and S. Pllana. Hstream: A directive-based language extension for heterogeneous stream computing. In *2018 IEEE International Conference on Computational Science and Engineering (CSE)*, pages 138–145, Oct 2018.

[51] G. Mencagli, M. Torquati, F. Lucattini, S. Cuomo, and M. Aldinucci. Harnessing sliding-window execution semantics for parallel stream processing. *Journal of Parallel and Distributed Computing*, October 2017.

[52] C. Misale, M. Drocco, M. Aldinucci, and G. Tremblay. A comparison of big data frameworks on a layered dataflow model. *Parallel Processing Letters*, 27(01):1740003, 2017.

[53] C. Misale, M. Drocco, G. Tremblay, A. R. Martinelli, and M. Aldinucci. Pico: High-performance data analytics pipelines in modern c++. *Future Generation Computer Systems*, 87:392–403, October 2018.

[54] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. The thelma project: A thermal lattice boltzmann solver for the gpu. *Computers & Fluids*, 54:118–126, January 2012.

[55] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. In *Evaluation and Assessment in Software Engineering*, volume 8, pages 68–77, 2008.

[56] M. Petticrew and H. Roberts. *Systematic reviews in the social sciences: A practical guide*. John Wiley & Sons, 2008.

[57] S. Pllana, S. Benkner, E. Mehofer, L. Natvig, and F. Xhafa. Towards an intelligent environment for programming multi-core computing systems. In *European Conference on Parallel Processing*, pages 141–151. Springer, 2008.

[58] S. Pllana and F. Xhafa. *Programming Multicore and Many-core Computing Systems*. John Wiley & Sons, Inc., Hoboken, New Jersey, USA, 1 edition, 2017.

[59] S. Sagiroglu and D. Sinanc. Big data: A review. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pages 42–47. IEEE, 2013.

[60] M. Sandrieser, S. Benkner, and S. Pllana. Using explicit platform descriptions to support programming of heterogeneous many-core systems. *Parallel Computing*, 38(1-2):52–65, 2012.

[61] D. Sengupta, S. L. Song, K. Agarwal, and K. Schwan. Graphreduce: processing large-scale graphs on accelerator-based systems. In *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*, pages 1–12. IEEE, 2015.

[62] O. Sjöström, S.-H. Ko, U. Dastgeer, L. Li, and C. Kessler. Portable parallelization of the edge cfd application for gpu-based systems using the skepu skeleton programming library. *ParCo-2015 conference*, 27:135–144, 2016.

[63] The square kilometre array (ska) project. `www.skatelescope.org`, Aug. 2018 (last accessed).

[64] Summit: Oak ridge national laboratory's next high performance supercomputer. `www.olcf.ornl.gov/olcf-resources/compute-systems/summit/`, Aug. 2016 (last accessed).

[65] Top500 list. `www.top500.org/`, June 2018 (last accessed).

[66] W. Turek, J. Stypka, D. Krzywicki, P. Anielski, K. Pietak, A. Byrski, and M. Kisiel-Dorohinicki. Highly scalable erlang framework for agent-based metaheuristic computing. *Journal of Computational Science*, 17:234–248, March 2016.

[67] A. Van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, 2000.

[68] M. Vanneschi. The programming model of assist, an environment for parallel and distributed portable applications. *Parallel computing*, 28(12):1709–1732, December 2002.

[69] M. Viñas, B. B. Fraguela, D. Andrade, and R. Doallo. High productivity multi-device exploitation with the heterogeneous programming library. *Journal of Parallel and Distributed Computing*, 101:51–68, March 2017.

[70] Z. Wang, Y. Liu, and P. Ma. A cuda-enabled parallel implementation of collaborative filtering. *Procedia Computer Science*, 30:66–74, 2014.

[71] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, September 2011.

[72] Y. Zhang and F. Mueller. Gstream: A general-purpose data streaming framework on gpu clusters. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 245–254. IEEE, 2011.