Accepted refereed manuscript of: Amaral V, Norberto B, Goulão M, Aldinucci M, Benkner S, Bracciali A, Carreira P, Celms E, Correia L, Grelck C, Karatza H, Kessler C, Kilpatrick P, Martiniano H & Mavridis I (2020) Programming Languages for Data-Intensive HPC Applications: a Systematic Mapping Study. Parallel Computing, 91, Art. No.: 102584. DOI: https://doi.org/10.1016/j.parco.2019.102584
© 2019, Elsevier. Licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International http://creativecommons.org/licenses/by-nc-nd/4.0/

Programming Languages for Data-Intensive HPC Applications: a Systematic Literature Review[1]

version 1.0

Vasco Amaral, vma@fct.unl.pt, Universidade Nova de Lisboa, Portugal;
Miguel Goulão, mgoul@fct.unl.pt, Universidade Nova de Lisboa, Portugal;
Beatriz Norberto, b.norberto@campus.fct.unl.pt, Universidade Nova de Lisboa, Portugal;
Marco Aldinucci, marco.aldinucci@unito.it, University of Torino, Italy;
Siegfried Benkner, siegfried.benkner@univie.ac.at, University of Vienna, Austria;
Andrea Bracciali, andrea.bracciali@stir.ac.uk, University of Stirling, UK;
Paulo Carreira, paulo.carreira@ist.utl.pt, Universidade de Lisboa, Portugal;
Edgars Celms, edgars.celms@lumii.lv, University of Latvia, Latvia;
Luís Correia, Luis.Correia@ciencias.ulisboa.pt, Universidade de Lisboa, Portugal;
Clemens Grelck, c.grelck@uva.nl, University of Amsterdam, Netherlands;
Helen Karatza, karatza@csd.auth.gr, Aristotle University of Thessaloniki, Greece;
Christoph Kessler, christoph.kessler@liu.se, Linköping University, Sweden;
Hugo Martiniano, hfmartiniano@ciencias.ulisboa.pt, Universidade de Lisboa, Portugal;
Ilias Mavridis, imavridis@csd.auth.gr, Aristotle University of Thessaloniki, Greece;
Sabri Pllana, sabri.pllana@lnu.se, Linnaeus University, Sweden;
Ana Respício, alrespicio@fc.ul.pt, Universidade de Lisboa, Portugal;
José Simão, jsimao@cc.isel.ipl.pt, Instituto Superior de Engenharia de Lisboa, Portugal;
Luís Veiga, luis.veiga@inesc-id.pt, Universidade de Lisboa, Portugal;
Ari Visa, ari.visa@tut.fi, Tampere University of Technology, Finland;

Abstract

We present the results of a systematic literature review that examines the main paradigms and properties of programming languages developed for and used in High Performance Computing for Big Data processing. The systematic literature review is based on a combination of automated keyword-based search in the Elsevier Science Direct database and further digital databases for articles published in international peer-reviewed journals and conferences, leading to an initial sample of 420 articles, which was then narrowed down in a second phase to 152 articles found relevant and published 2006-2018. The manual analysis of these articles allowed us to identify 26 languages used in 33 of these articles for HPC for Big Data processing. We analyzed the languages and their usage in these articles by 22 criteria and summarize the results in this article. We evaluate the outcomes of the literature review by comparing them with opinions of domain experts. Our results indicate that, for instance, 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.

Keywords: High Performance Computing, Modelling and Simulation, Big Data, Data Mining, Dynamic Systems, Data Intensive Computing, Programming Languages

[1] This article is based upon work from COST Action IC1406 High-Performance Modelling and Simulation for Big Data Applications (cHiPSet), supported by the European Cooperation in Science and Technology.

1 Introduction

Big Data has become one of the most frequently used buzzwords of our times. In industry and academia alike, the interest is dramatically increasing, even though the term Big Data is not always clear. Big Data has been defined as the "3Vs" model, an informal definition proposed by Beyer and Laney [81] that has been widely accepted by the community: "*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."* More recently, the "3Vs" model has been further extended by adding *Veracity* that indicates that the quality and accuracy of the data may vary.

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 [3,56]. For instance, the Square Kilometre Array (SKA)[2] 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. 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[3] supercomputer (Rank 1 in the current TOP500 list[4]) 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 [57,58]. 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*" [55]. The multi-core shift in computer architecture has accelerated the research efforts in developing new programming frameworks for parallel computing, which has produced a rich variety of new designs of languages and of libraries using established HPC languages, which should assist domain programmers from science and engineering, e.g. by reducing the complexity of parallel programming, providing more domain-specific programming constructs, generating and optimizing low-level parallel code for coordination of computations across multiple cores and multiple computers.

This study presents the results of a systematic literature review carried out as part of the European COST Action cHiPSet[5] that addresses High-Performance Modelling and Simulation for Big Data Applications. The literature review focuses on the main paradigms and properties of programming languages used in High Performance Computing for Big Data processing. Our initial literature search resulted with 420 articles; 152 articles are retained for final review after

[2] The Square Kilometre Array (SKA) project, Accessed August 7, 2018, www.skatelescope.org

[3] Summit: Oak Ridge National Laboratory's next High Performance Supercomputer. Accessed August 6, 2018, https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/

[4] TOP500 list, June 2018, https://www.top500.org/

[5] ICT COST Action IC1406, cHiPSet, Accessed August 8, 2018, http://chipset-cost.eu/

the evaluation of initial search results by domain experts. Results of our literature review 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 literature review, we developed a questionnaire and collected the opinions of domain experts. A comparison of literature review outcome with opinions of domain experts reveals that the key features of HPC programming languages for Big Data are portability, performance and the usability. As key issues that need more attention in future research are identified the language learning curve and interoperability. We consider that the outcome of this study may help in understanding the limitations of the state of the art in HPC programming languages for Big Data, and may help the reader in identification of programming language issues that need to be addressed in future.

The rest of the paper is organized as follows. Section 2 describes the methodology of the Systematic Literature Review (SLR). We present the obtained results in Section 3. Section 4 evaluates SLR results via a questionnaire that involves domain experts. Section 5 summarizes our major observations and lists challenges and future research directions. After discussing the related work in Section 6, the paper is concluded in Section 7.

2 The Review Process

The methodology used in this Systematic Literature Review (SLR) follows the methodology proposed in [5, 6], which articulates in six successive steps, which are detailed in Sections 2.1-2.6

1. *Research Question*, aiming at formulating the research questions the SLR should answer;
2. *Search Strategy*, aiming at detecting the largest number of primary studies related to the proposed research questions;
3. *Selection of Primary Studies*, aiming at sieving false positive by a human-driven abstract inspection;
4. *Quality Assessment*, aiming at validating of the review process;
5. *Data Extraction Process*, which aims to answer to each research question all selected studies;
6. *Synthesis of the Information.*

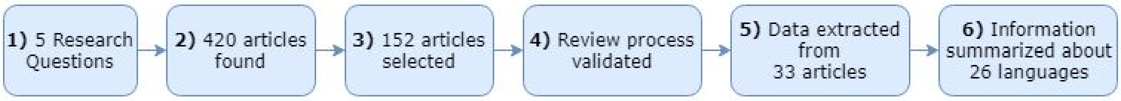
1) 5 Research Questions
2) 420 articles found
3) 152 articles selected
4) Review process validated
5) Data extracted from 33 articles
6) Information summarized about 26 languages

Figure 2.1: Methodology used in the SLR

2.1 The Research Questions

In order to frame the research questions, *PICOC criteria* [5] (Population, Intervention, Comparison, Outcomes, Context) were used and the question elements were defined as:

• *Population* - Composed by the primary studies found on Languages for High Performance Computing (HPC);

• *Intervention* (Software engineering methodology/tool/technology/procedure that addresses a specific issue) - This SLR investigates studies regarding languages for HPC, describing their details;

• *Comparison* (methodology/tool/technology/procedure with which the intervention is being compared) - Not applicable to this case;

• *Outcomes* (it should relate to factors of importance to practitioners) - The results should refer to technologies, methods and metrics that lead to an increase in the quality of the solution, ease of configuration, usability, productivity gains, such as an easy-to-use and easy-to-learn language, product performance gains, such as easy maintenance, solution scalability, and memory efficiency;

• *Context* - The participants involved in this study were researchers and specialists in this area.

The goal of this SLR is to *answer five research questions*, presented in Table 2.1.1, and for this purpose a number of sub-questions were formulated for each of them.

Table 2.1.1: Research Questions that were formulated

Question ID	Research Questions
RQ 1	**Which are the categories of languages in use?**
RQ 1.1	What are the current research trends in languages for HPC?
RQ 2	**What is the nature of the languages for HPC?**
RQ 2.1	What kind of language is it?
RQ 2.2	What is the execution model that is being used?
RQ 2.3	What are the key advantages of the language?
RQ 2.4	What is/are the application domain/s of the language?
RQ 2.5	What are the paradigms underlying the languages?
RQ 2.6	Which are the execution stack requirements (?-aaS) to support the artifacts created with those languages?
RQ 2.7	What is the existing tool support for the language?
RQ 2.8	What are the technologies used to create the language tool suite?
RQ 2.9	Does the language target specific hardware?
RQ 2.10	What is the purpose of the language?
RQ 2.11	What is the preferred language representation type?

RQ 3	**What are the typical user profiles for the languages?**
RQ 3.1	What are the roles of the users of this language?
RQ 3.2	What kind of technical knowledge is required?
RQ 4	**How effective are the languages?**
RQ 4.1	Is the success of the languages evaluated in the articles?
RQ 4.2	What is the impact on the productivity gains brought by the languages reported?
	What is the impact on the products' performance gains brought by the languages reported?
RQ 4.3	Is there an explicit comparison with competing approaches?
	Is the comparison quantitative, qualitative, or both?
	What are the comparison methodology and metrics used?
RQ 5	**What types of articles are published in the area of programming models for HPC?**
RQ 5.1	Does the article include COST cHiPSet's authors?
RQ 5.2	What are the institutions involved?
RQ 5.3	What is the name of the conference or journal?
RQ 5.4	Who is sponsoring the research?
RQ 5.5	What kind of research is being reported?

2.2 The Search Process

One of the main objectives to conduct a SLR is to *detect the largest number of primary studies related to the proposed research questions*.
Our research process is based on three main steps, identified in Figure 2.2.1.

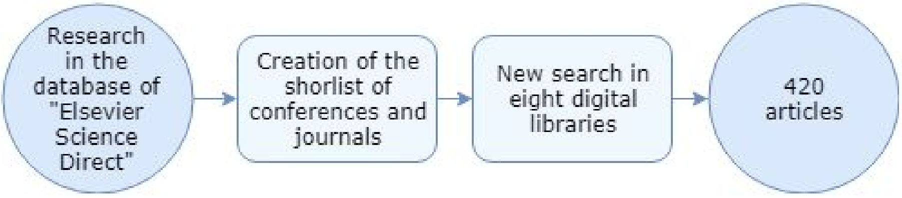
Research in the database of "Elsevier Science Direct"
Creation of the shorlist of conferences and journals
New search in eight digital libraries
420 articles

Figure 2.2.1: Stages of the Research Process

The cHiPSet ICT COST Action experts selected, by consensus, the Elsevier Science Direct database to use in the review. Initially, the following query was defined based on the chosen keywords:

"Big data" AND "Programming Model" AND "Programming Language" AND "High performance computing"

With the purpose of covering up the largest possible number of relevant studies, without discarding any, and considering that authors may use equivalent keywords, the initial query was reformulated into:

("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")
With this literature search we found *262 articles*.

The references found were then presented to the cHiPSet ICT COST Action group of experts to assess their completeness. From this analysis, it was found that the coverage of the literature offered in the Elsevier Science Direct database was insufficient for this domain, with a significant number of relevant publications that were not part of this selection because they were not contained in this digital library. For this reason, we considered a shortlist of conferences and journals that are relevant for the field in study, which is presented in Table 2.2.1.

Table 2.2.1: Conferences and journals considered in the study

Conferences	Journals
GTC / GPGPU conference	ACM Transactions on Parallel Computing
IEEE International Parallel and Distributed Processing Symposium	Concurrency and Computation Practice and Experience
International Conference on Parallel Processing	Future Generation Computer Systems
International Conference on Supercomputing	IEEE Computing in Science and Engineering
International European Conference on Parallel and Distributed Computing	Journal of Parallel and Distributed Computing
International 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	

A second search was done, both in the already verified digital library and in other existing databases, using queries similar to that presented above, but with the results filtered in the shortlist of conferences and journals in Table 2.2.1. Table 2.2.2 presents the list of digital libraries used and the respective number of articles obtained by the second search, after removal of duplicate documents. Except for Google Scholar database, where it was only possible to explore all combinations of the mentioned keywords, in all other databases complex queries were made.

Table 2.2.2: Results obtained with the second search

Source Name	Number of Publications obtained
academia.edu	1
ACM Digital Library	16
Compendex	6
Elsevier Science Direct	27
Google Scholar	32
IEEE Xplore	3
Research Gate	3
Springer Link	70
Total of articles found:	**158**

It is important to mention that *all searches were based on the title, abstract and keywords of articles published between January 2006 and March 2018*, a period of time that, in the opinion of experts, should cover the studies required for this SLR.

After removing any duplicate articles, we obtained a total of *420* (262 with the first search + 158 with the second one) *articles from 8 different digital databases* that were processed through the next phases of this SLR.

2.3 Selection of the Primary Studies

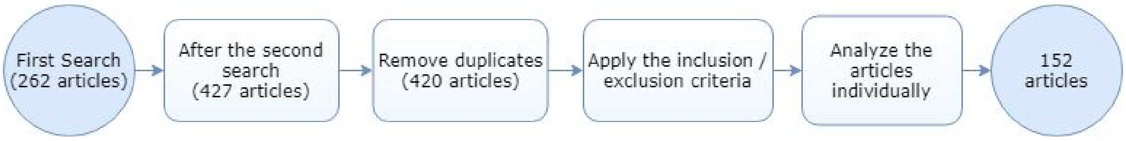
First Search (262 articles)
After the second search (427 articles)
Remove duplicates (420 articles)
Apply the inclusion / exclusion criteria
Analyze the articles individually
152 articles

Figure 2.3.1: Selection Process of the articles

As shown in Figure 2.3.1, we faced four stages throughout the selection process of the primary studies:

1. Select the related articles - a shortlist (Table 2.2.1) was created with conferences and journals considered relevant for the study. Next, the queries were executed in the online libraries listed in Section 2.2;

2. Remove duplicate documents found;

3. Apply the inclusion and exclusion criteria already defined, in order to verify whether the resulting documents are relevant to the review. The criteria adopted were those presented in Table 2.3.1;

4. Finally, a selection mechanism was used by analyzing the title, abstract and keywords of each of the remaining studies.

Table 2.3.1: Inclusion and Exclusion criteria of the articles

Inclusion Criteria	Exclusion Criteria
Study must have addressed HPC research	Irrelevant publication 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	Publication 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

After executing the described selection mechanism, 152 articles continued for the future phases.

2.4 Quality Assessment

To evaluate the quality of the present study, we used the criteria proposed by [5, 6] and examined the following four fundamental questions.

• *Are the review inclusion and exclusion criteria described and appropriate?* In this case, all the criteria (Table 2.3.1) considered appropriate are explicitly defined;

• *Does the literature search cover all relevant studies?* Initially, we only researched in journals that seemed more influential, but later, after a conversation with the experts, it was concluded that there was another type of literature that was not yet covered, but nonetheless considered important. Therefore, we created a shortlist (Table 2.2.1) with the conferences and journals considered relevant to the study to support the answers to the questions proposed. After that, researches of these conferences and journals were carried out in the following electronic platforms: academia.edu, ACM Digital Library, Compendex, Elsevier Science Direct, Google Scholar, IEEE Xplore, Research Gate and Springer Link;

• *Has the quality of the review been assessed?* The authors reviewed the studies resulting from the research done, noting whether they should be included or excluded from the review. Despite there is no absolute guarantee that important information is not being lost, as a safeguard mechanism, in one small sample of articles, one of the experts makes a second review of these documents, finding that there are no important data to be excluded;

• *Have primary studies been adequately referred to?* Throughout the SLR, all the studies published by the conferences and journals indicated by the specialists were identified. In addition to these studies, experts were asked to create a shortlist of articles that should be found.

2.5 Data Extraction Process

To preserve the consistency of the data extraction, a data collection template was created. At this time, when each study was analyzed, this form should be completed. All collected data was stored in a shared spreadsheet created for this purpose. A team of 17 researchers (being them the authors of this paper) participated to data extraction directly reading the primary studies selected during the previous stage. The initial data referred to the reviewer of the publication and to the publication itself. The remaining information concerned the research questions referred in Section 2.1. In this phase, a detailed analysis of the studies included in the previous stages was carried out, being extracted the information considered relevant by the respective reviewers of these studies.

2.6 Synthesis of the Information

During this stage, the previously extracted information was compiled 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, being given answers to the research questions and referring the found languages, as well as the individual data of each of them. This document has been checked by all reviewers of the articles included and all the information contained therein has been confirmed by them.

3 Discussion of the Results

Throughout this section, the answers to the proposed research questions will be discussed. We have separated the different categories of languages, which may be: *1)* a DSL (Domain Specific Language), which is a language adapted to a specific application domain that offers appropriate annotations and abstractions [8, 9, 10]; *2)* a GPL (General Purpose Language), which is a programming language designed to be used in writing Software in a wide variety of application domains [8]; *3)* a DSL embedded in another DSL; or *4)* a DSL embedded in a GPL.

In addition to the information gathered on the existing languages, several documents 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*. Appendix A presents a list of the languages identified describing 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.

RESEARCH QUESTION 1 - Which are the categories of languages in use?

According to the results presented in Figure 3.1, *54%* of the languages focused in the publications found are classified as being GPL *(14 languages)*, that is, a programming language designed to be used in Software writing in a wide variety of fields of application. It's possible to see that *31%* of these languages *(8 languages)* are DSL, being a language adapted to a specific application domain that offers appropriate annotations and abstractions. The remaining languages *(4 languages)* were considered DSL embedded in an GPL.

Though the study considered that a DSL embedded in another DSL would be found, our analysis did not find any.

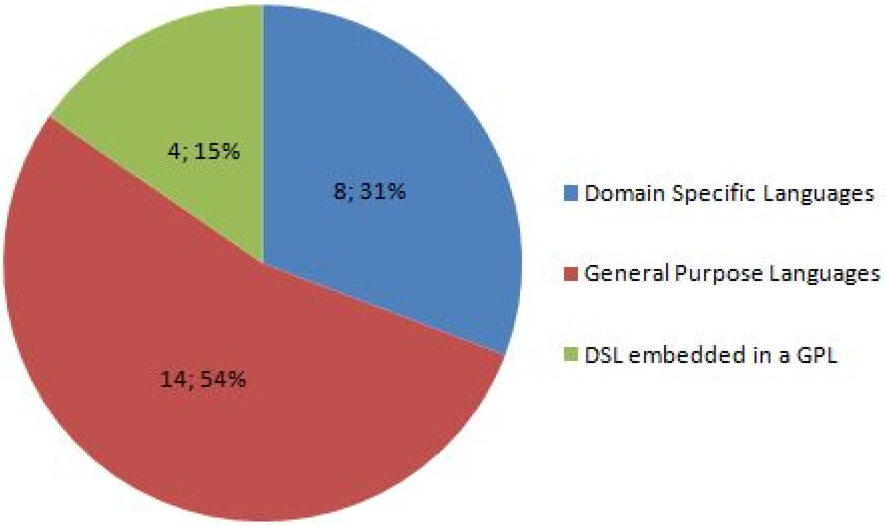

Figure 3.1: Which are the categories of languages in use? - RQ 1

RESEARCH QUESTION 2 - What is the nature of the languages for HPC?

The objective of this research question was to characterize the nature of languages for HPC. To accomplish this objective several sub research questions were identified and their results are here discussed.

The key advantage of the languages found 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 "Visualization of user-initiated query results", "Ease to express constraint problems" and "Enabling high-level parallel programming using skeletons" *(see Figure 3.2)***.**

Concerning tools supporting the languages, compilers are the most well represented support tool, followed by tool suite and interpreters *(see Figure 3.3)***.**

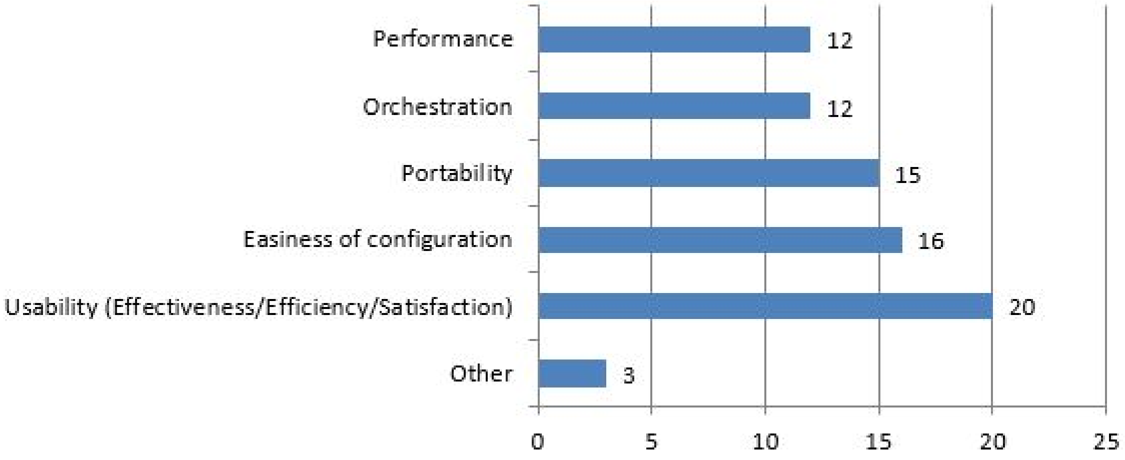

Figure 3.2: What are the key advantages of the language? - RQ 2.3

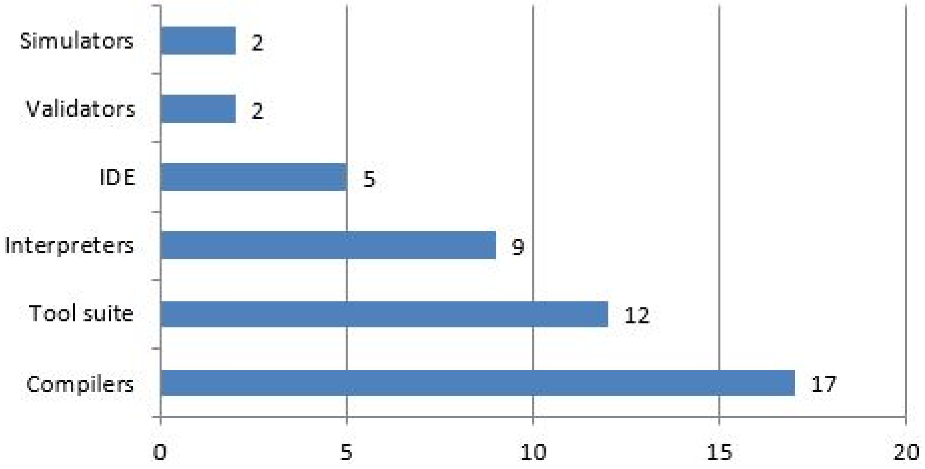

Figure 3.3: What is the existing tool support for the language? - RQ 2.7

Most of the surveyed languages do not target specific hardware *(85%)*. It’s known that *40%* of the languages found target GPUs or multi-core architectures.

The main purpose of the languages found is to "Implement the solution", followed by the "Formalization of the solution", "Formalization of the requirements of the problem" and "Data Interpretation" *(see Figure 3.4)*.

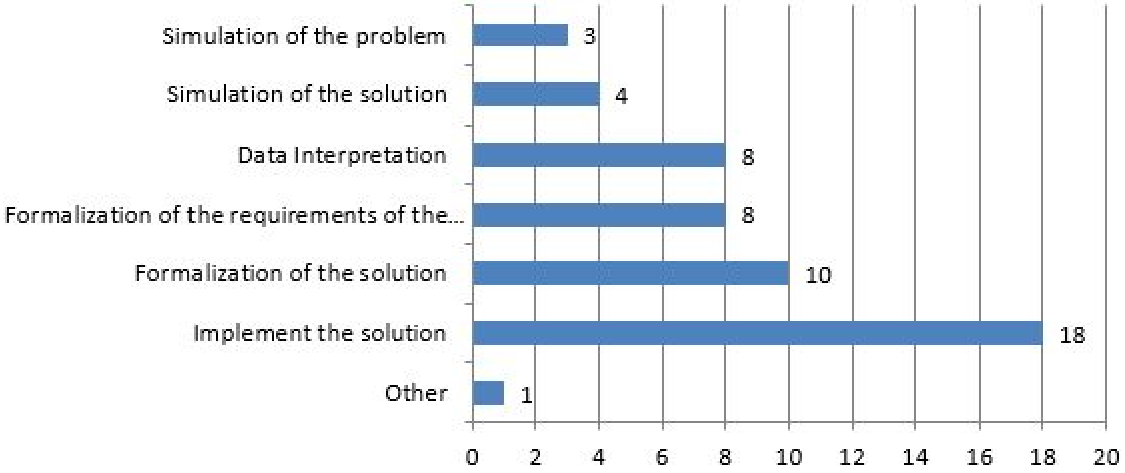

Figure 3.4: What is the purpose of the language? - RQ 2.10

Results for the language representation type revealed that there is a concrete syntax for all the languages found and the preferred representation type of *76%* of them is Textual *(see Figure 3.5)*.

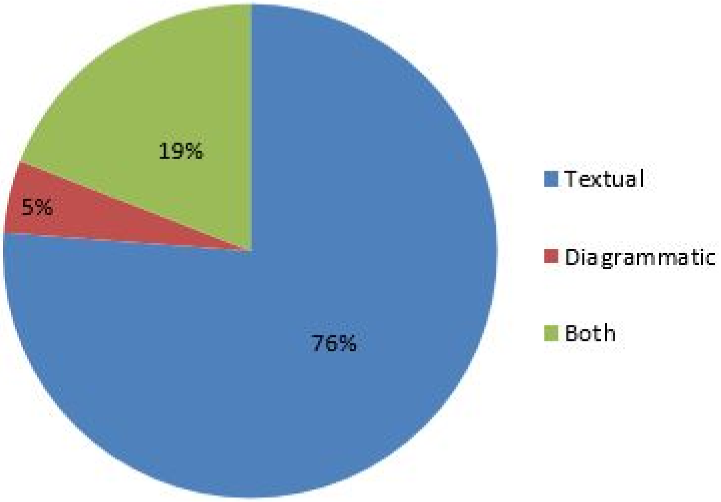

Figure 3.5: What is the preferred language representation type? - RQ 2.11

RESEARCH QUESTION 3 - What are the typical user profiles for the languages?

Figure 3.6 displays the distribution of the typical user profiles for the languages. Most of the identified languages are used by end-users, who utilize the language to solve problems. It is known that *16,5%* of the languages are used by developers, who utilize the language to create tools/setups/solutions for other users.

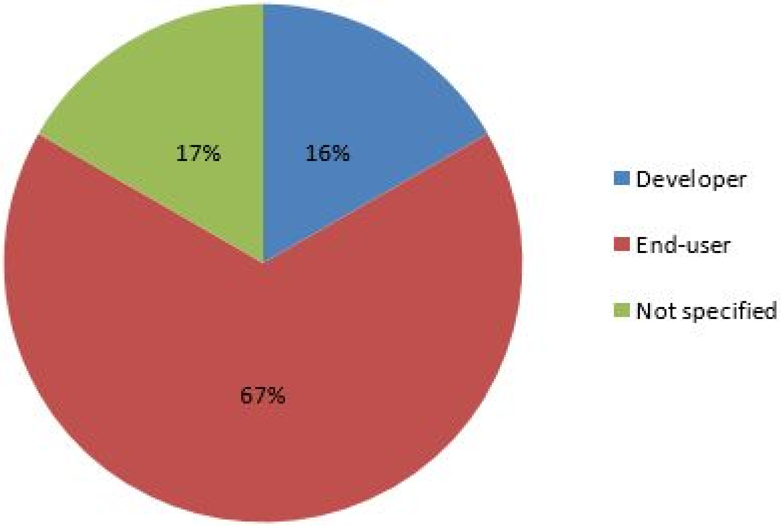

Figure 3.6: What are the roles of the users of this language? - RQ 3.1

RESEARCH QUESTION 4 - How effective are the languages?

Effectiveness is articulated in three aspects, addressed by RQ 4.1 (success), RQ 4.2 (productivity gain), and RQ 4.3 (advantage against competitive approaches). As shown in Figure 3.7, most of the articles reviewed are (somehow) evaluated the corresponding languages for success.

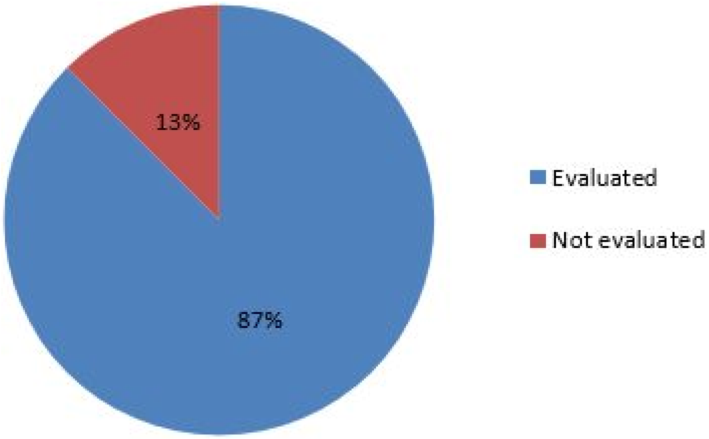

Figure 3.7: Is the success of the languages evaluated in the article? - RQ 4.1

Figure 3.8 displays a graphical representation for the impact on the productivity gains brought by the languages. With a *Quantitative* analysis, the productivity gain brought by the languages reported with the most impact was the easiness to use the language, followed by the learnability. With a *Qualitative* analysis, the productivity gain brought by the languages reported with the most impact was the easiness to use, followed by the lower cognitive overload and the learnability. Comparing these analysis, the productivity gains brought by the languages reported was mainly measured using qualitative methods.

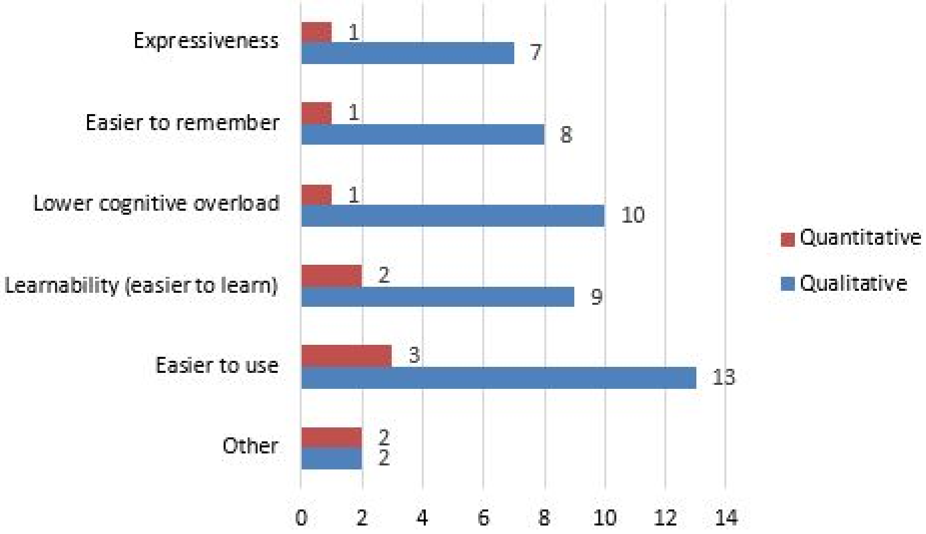

Figure 3.8: What is the impact on the productivity gains brought by the languages reported? - RQ 4.2

The chart in Figure 3.9 represents the impact on the products' performance gains brought by the languages reported. With a *Quantitative* analysis, the products' performance gains brought by the languages reported with the most impact were the computation efficiency and the scalability. With a *Qualitative* analysis, the products' performance gain brought by the languages reported with the most impact was evolvability/maintainability, followed by scalability. Unlike the productivity gains brought, comparing these analyses, the products' performance gains brought by the languages reported was mainly measured using *Quantitative* methods.

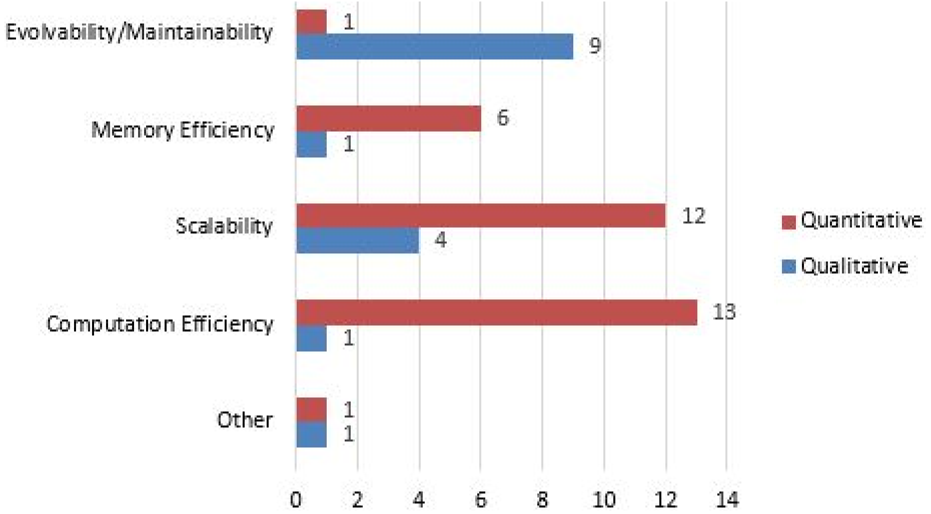

Figure 3.9: What is the impact on the products' performance gains brought by the languages reported? - RQ 4.2

According to the statistics, *64%* of the articles included an explicit comparison between the language reported and other competing approaches. Half of the articles included an explicit comparison of the language proposal with respect to distinct settings/context/configurations.

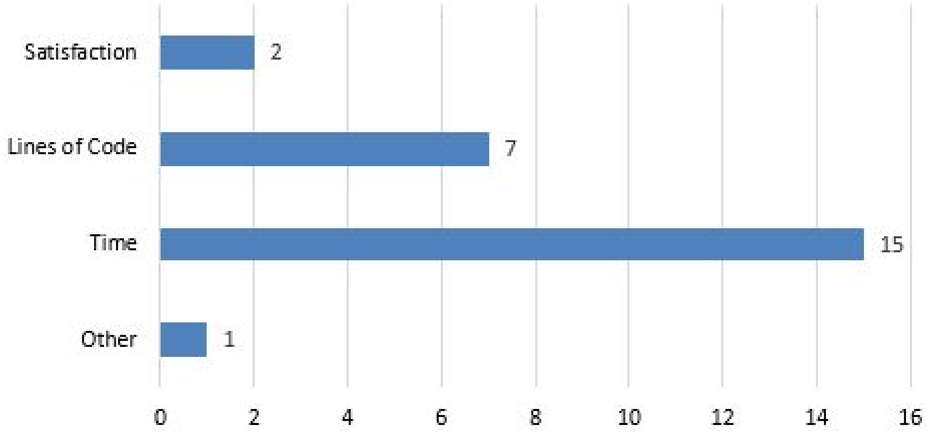

Figure 3.10: Number of articles using each metric - RQ 4.3: What are the metrics used?

All the metrics were measured using quantitative methods.

The most used metric was the computational time, followed by the lines of code and the satisfaction *(see Figure 3.10)***.**

RESEARCH QUESTION 5 - What types of articles are published in the area of programming models for HPC?

A large part of the articles that referred to languages for HPC do not include authors of the cHiPSet ICT COST Action but there are some exceptions like [39, 40, 43].

The scientific journal that published more articles was "Future Generation Computer Systems", followed by "Parallel Computing" and "Journal of Parallel and Distributed Computing", as illustrated by Figure 3.11. Most of these articles were sponsored by public or both public and private funds. Given that each of these can be classified as both a case study, as an experiment report and a comparative assessment, there was a greater occurrence of experience reports. Also case studies and comparative assessments were found in a similar number, as shown in Figure 3.12.

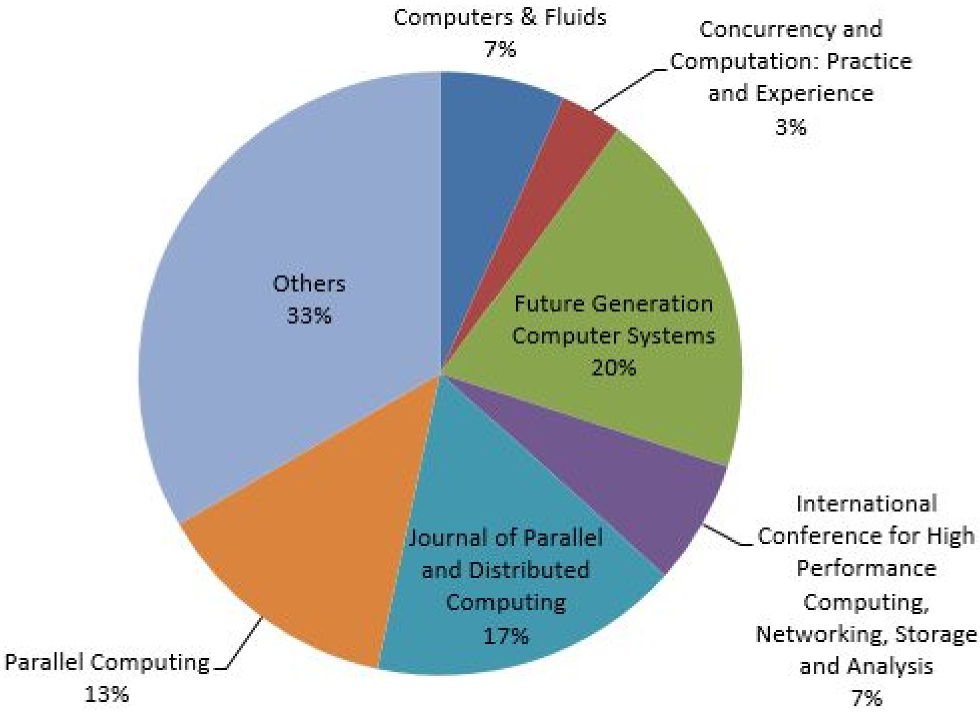

Figure 3.11: Which conferences and journals publish articles about languages for HPC? - RQ 5.3

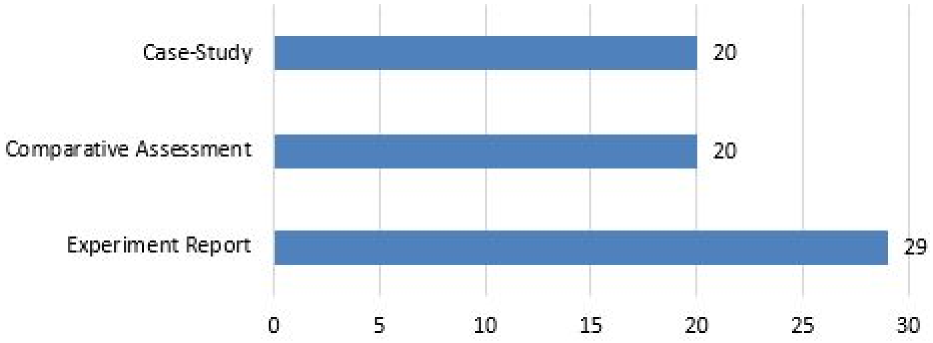

Figure 3.12: Number of articles reporting each type of research - RQ 5.5

According to the values presented in Figure 3.1, about half of the languages found are of type GPL, these being: Bobolang; C; C++; Erlang; FastFlow; Goal Language supported by RuGPlanner; Java; OpenCL; Python; R; Scout; Selective Embedded Just-In-Time Specialization; SkIE-CL; Swift. Several articles were found related to DSL, more specifically: CineGrid Description Language; CRUCIBLE; e-Science Central WFMS; Higher-order "chemical programming" language; Liszt; Mendeleev; MiniZinc; Network Description Language. Three of the articles found referred to languages considered DSL embedded in GPL: Pipeline Composition; Spark SQL; Spark Streaming; Weaver.

4 SLR Evaluation by Domain Experts

A questionnaire was prepared and used as a form of validation of the results found and confrontation with the opinion of what the domain experts expected to find, based on the research questions proposed for this SLR (Section 2.1).

This questionnaire is presented in the Appendix B and aims to find out: in what areas of engineering have the specialists worked; if their activity consists primarily in the development of new support tools or in the utilization of existing tools; which programming languages are used in this area; what makes them use these languages in relation to the others they know (in the context in question); what are the advantages of these languages; what existing support tools they know; for the domain where they are inserted, how effective are the languages used, that is, how successful they are in producing a desired result; what is the impact on the performance brought by the reported languages and their main limitations/difficulties of use.

4.1 Questionnaire Results

Taking into account the answers to the previous survey, it is possible to conclude that, *with a wide experience in HPC of the respondents* (claiming to work in the area for more than 10 years and considered with a high level of technical knowledge for the languages used *(see Figure 4.1.1)*):

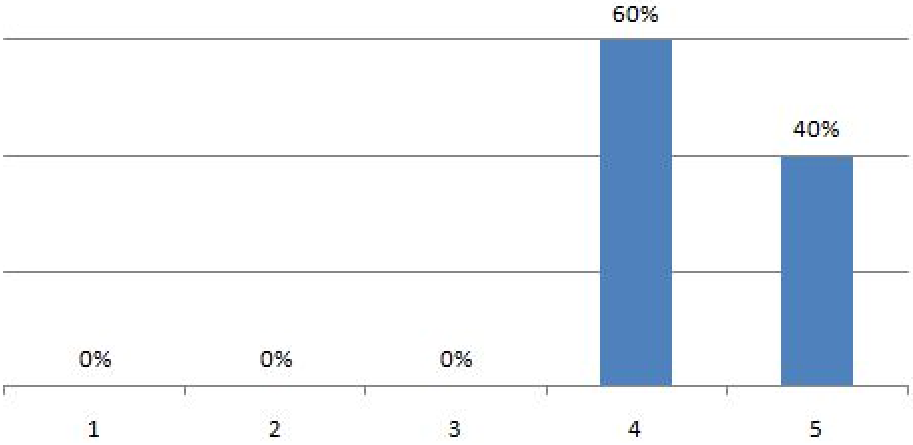

Figure 4.1.1: How do you rate your level of technical knowledge for languages used for HPC? - Question 14

• Their main activity consists on the *development of new support tools*, rather than the use of existing tools *(see Figure 4.1.2)*;

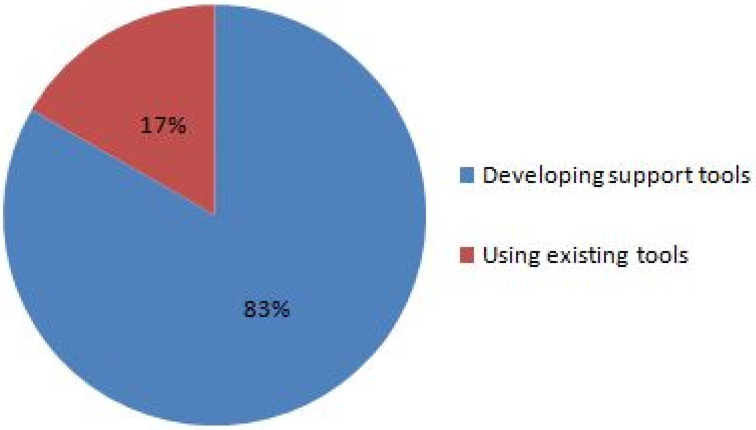

Figure 4.1.2: Does your High Performance Computing activity consist primarily of developing support tools or of using existing tools? - Question 4

• All of them use the *programming languages C, C++ and OpenCL,* and the following are also explored: *Java, Python and R*;

• *The usability and the nature of the problem in question* are the main reasons that make them use the above languages in relation to the others they know *(see Figure 4.1.3)*;

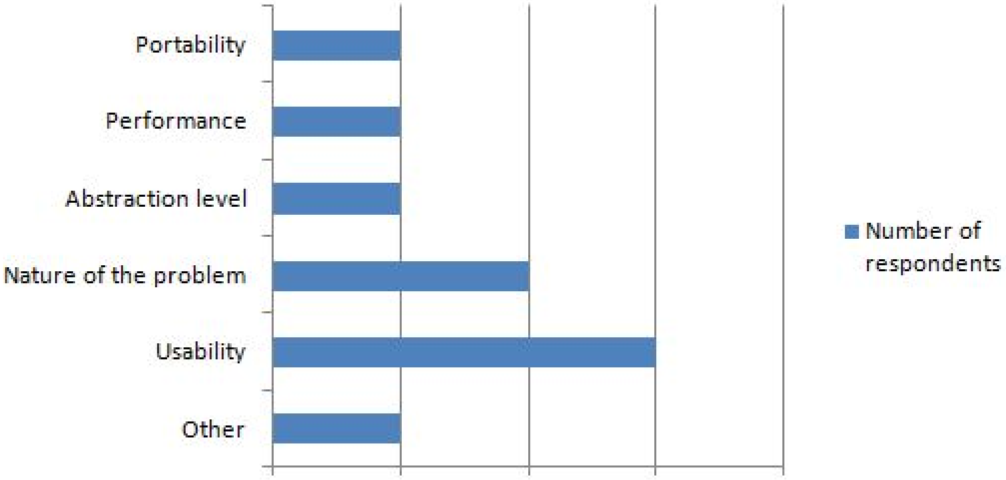

Figure 4.1.3: What made you use these languages in relation to the alternatives you know? - Question 6

• *The portability, the performance and the usability* of the referred languages are the main advantages pointed out *(see Figure 4.1.4)*;

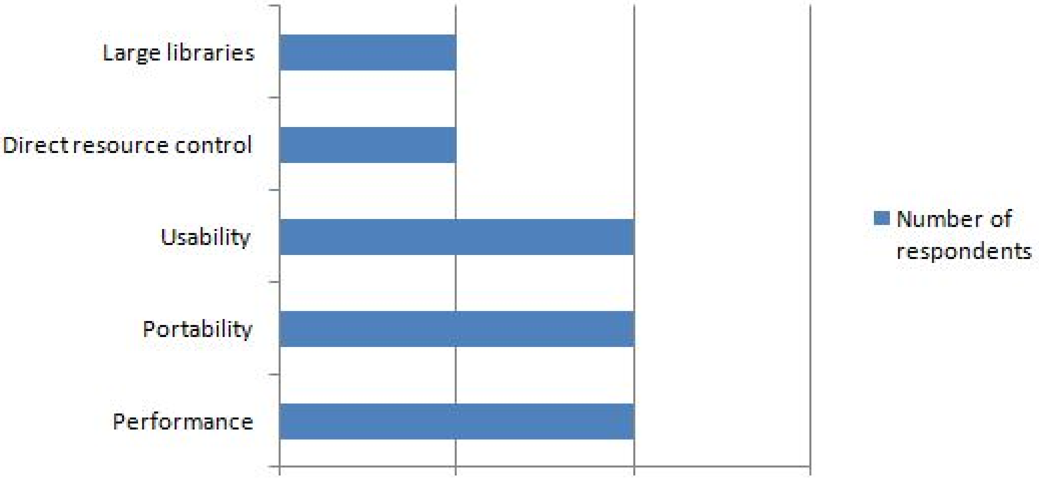

Figure 4.1.4: What are the key advantages of these languages? - Question 7

• The existing tool support for the languages used were rate *with a mean of 3* (from 1 to 5, where 1 indicates that the tool support is very poor and 5 that it is excellent, *see Figure 4.1.5*), and the existing support tools mentioned are: *VAMPIR, CUDA SDK, Performance API (PAPI) and Linux performance tools (see Figure 4.1.6)*;

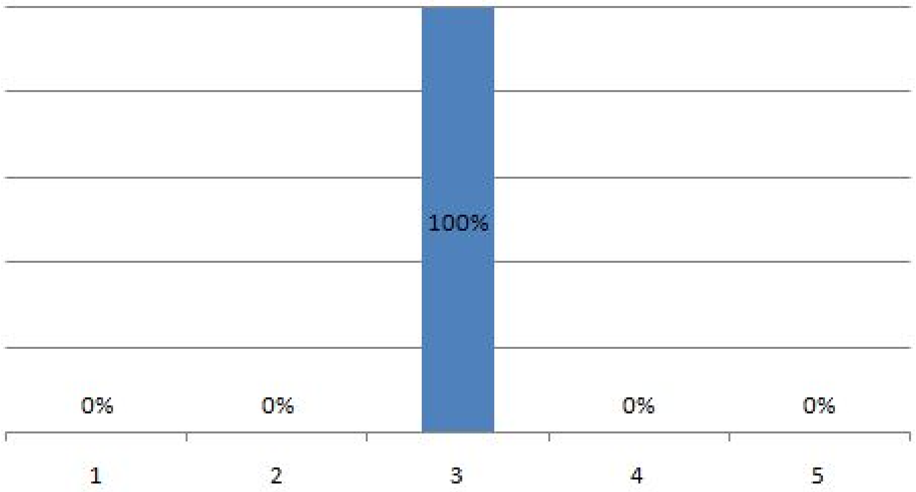

Figure 4.1.5: How do you rate the existing tool support for the languages you use for HPC? - Question 8

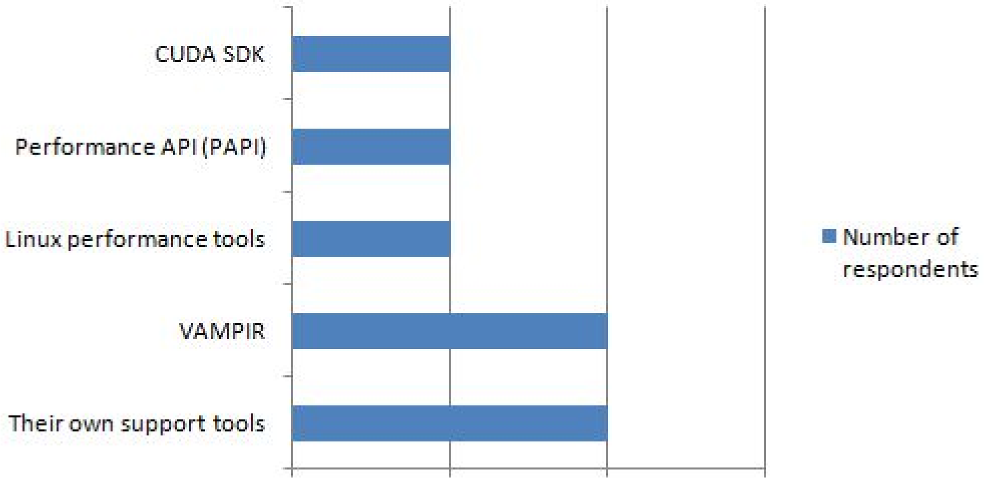

Figure 4.1.6: In relation to the previous question, what are the existing support tools you know? - Question 9

- The languages used were considered *effective with a mean of 3.6* (from 1 to 5, where 1 indicates that they are not effective and 5 that they are extremely effective, *see Figure 4.1.7*), and the fundamental language mechanisms that justify this decision are *the support for data parallelism and the direct control of resources, such as memory*;

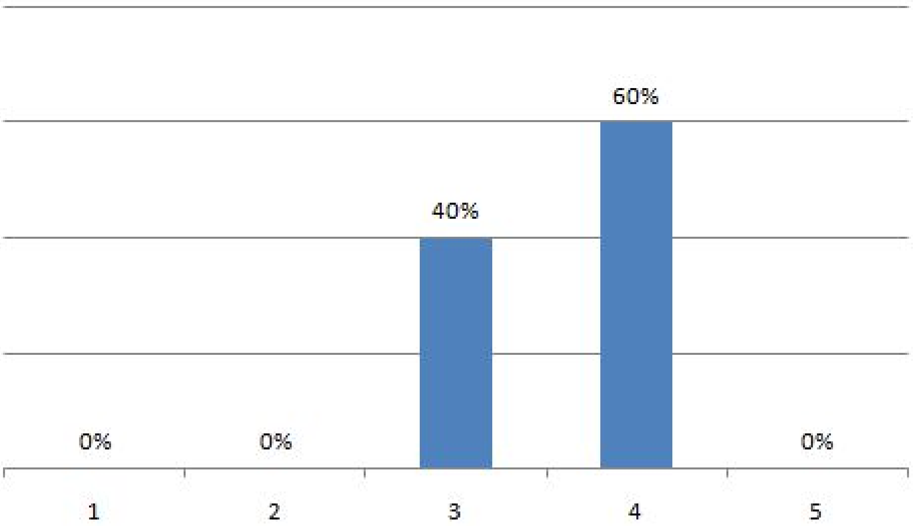

Figure 4.1.7: For the domain where you are, how do you rate the effectiveness of the languages you use? - Question 10

• Considering a diverse range of professionals, several obstacles have been identified but, according to the answers provided, *the interoperability and the learning curve* are the main difficulties felt *(see Figure 4.1.8)*.

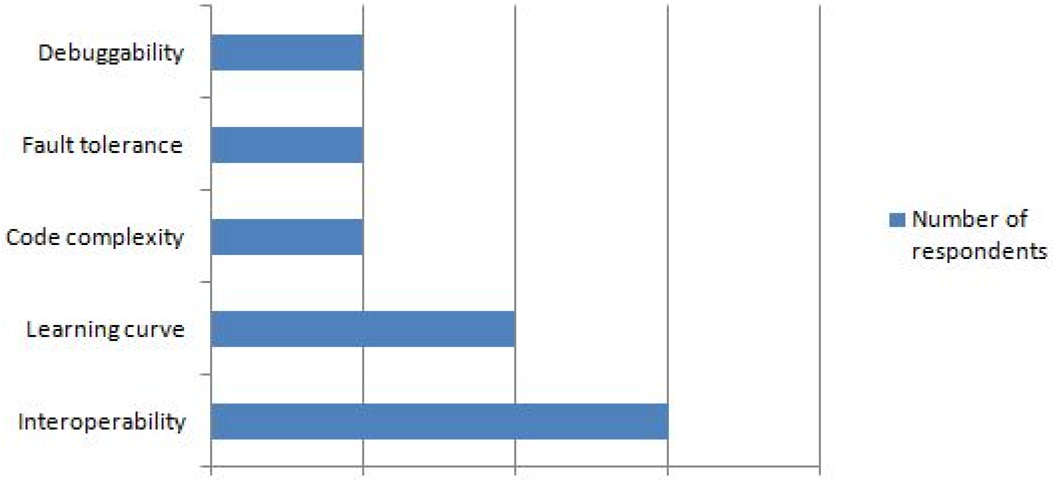

Figure 4.1.8: What are the limitations/difficulties of the languages you use? - Question 13

4.2 Comparison between the Questionnaire Results and the Information found

Comparing the results obtained by the analysis of the results of the questionnaire with the languages found through the research carried out, confronting the opinion of what was expected to be discovered, according to experts, it is known that there any many languages used in High Performance Computing, presented in Section 3, and for that reason a part of them were not known by the respondents. *The languages referenced in most of the articles, the most popular ones, are the languages used by these specialists: C, C++, OpenCL, Java, Python and R.*

The documents referred to a wider range of advantages than the experts, including the "Easiness of configuration" and the "Orchestration". However, *the main advantages pointed out by these people, like the portability, the performance and the usability of the languages were found in the documents*. Since we have known the advantages of the languages, according to the information found in the digital libraries, *the reasons, referred to by the experts, that make them use the above languages in relation to the others they know were the expected ones***.**

5 Observations, research challenges and future directions

In this section we summarize 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;
- 67% of the language users were 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 domain experts that responded to our questionnaire:

- key features of a HPC programming language for Big Data are performance, portability, and usability;
- usability of the language is decisive when selecting a language;
- existing tool support for HPC programming languages is average 3;
- popular tools include CUDA SDK; PAPI, Vampir, Linux performance tools;
- most (that is 83%) of the experts 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;
- future research should pay more attention to interoperability.

6 Related Work

The ultimate motivation of this manuscript is set a deep state-of-the-art on “Programming Languages for Data-Intensive HPC Applications” by systematically analysing the literature in the field reducing as much as possible bias due to authors direct experience during the analysis. There two well-known methods to target this aim are “Systematic Mapping Studies” (SMS) and “Systematic Literature Reviews” (SLR), which has been eventually adopted.

Systematic Mapping Studies [5, 168] support a *broad and shallow* approach to literature revision and are typically exploited for structuring a research area. They are built on general questions to discover research trends. In this, the quality assessment of primary studies is optional (e.g. primary studies without empirical evidence can be included). Examples of some SMS are [9, 169, 170, 171, 172].

On the contrary, Systematic Literature Reviews (SLR) support a *narrow and deep* approach to literature revision. They are used for gathering and synthesizing evidence on 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 (e.g. primary studies without empirical evidence should not be included).

In the first phase of the SLR, research was done to find out if there were studies combining the different languages used for HPC in the databases of several digital libraries, i.e. academia.edu, ACM Digital Library, Compendex, Elsevier Science Direct, Google Scholar, IEEE Xplore, Research Gate and Springer Link. We concluded that until now, there are only studies that refer to specific languages or compare few programming languages, but there is a lack of comprehensive literature studies of the kind of this paper that address HPC programming languages in the context of Big Data. There are also some primary studies regarding tools, such as libraries [29, 39, 44, 45, 46, 47, 48], integrated in known languages used in this type of computation, or APIs and programming models [32, 49, 50, 52, 53, 54].

7 Conclusions

We performed a systematic literature review to examine the main paradigms and properties of programming languages used in High Performance Computing for Big Data processing. Five main research questions drove our SLR. These were further decomposed into 22 sub-research questions. Automated search for articles including simultaneously at least one from four groups of keywords was undertaken in two stages. The first used the Elsevier Science Direct database. The second search used eight different digital databases, but restricted the journals/ conferences to a predefined shortlist. Only articles in the time period of January 2006 to March 2018 were considered. From a total of 420 articles found in the search, only 152 were considered relevant for our study. The analysis of these articles allowed us to identify 26 languages used in 33 articles for HPC for Big Data processing. We have provided a comprehensive classification of the languages encountered and their usage and evaluation by different criteria. We observed 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. Furthermore, results of the literature review are evaluated the by comparing them with opinions of domain experts. A comparison of literature review outcomes with opinions of domain experts revealed that the key features of HPC programming languages for Big Data are portability, performance and the usability.

APPENDIX A - Languages used for Data-Intensive HPC Applications

Table A.1: List of the Languages found

List of the Languages found
Domain Specific Languages
CineGrid Description Language + Network Description Language
CRUCIBLE
e-Science Central WFMS
Higher-order "chemical programming" language
Liszt
Mendeleev
MiniZinc
General Purpose Languages
Bobolang
C/C++
Erlang
FastFlow
Goal Language supported by RuGPlanner
Java
OpenCL
Python/R
Scout
Selective Embedded Just-In-Time Specialization
SkIE-CL
Swift
Domain Specific Languages embedded in General Purpose Languages
Pipeline Composition (PiCo)
Spark Streaming and Spark SQL
Weaver

A.1 Domain Specific Languages

A.1.1 CineGrid Description Language + Network Description Language [11]

Q2: NATURE OF THE LANGUAGE -> Ontology languages describing domain-specific services and network entities, for the domain of a non-public digital media data grid, in OWL (i.e., ultimately, XML)

Purpose of the language: Formalization of the requirements of the problem; Formalization of the solution; Data Interpretation

Key advantages: Portability, easiness of configuration, visualization of user-initiated query results

Paradigms underlying the language: Declarative (Data access service configuration and deployment structure graphs expressed in OWL/XML syntax)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Interpreters

Technologies used to create the language tool suite: XML based technology (Jess reasoner for querying OWL ontologies)

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: Developer

Technical knowledge required: Tools (OWL/XML editor), Languages (SQWRL query language for OWL ontologies), Hardware/Systems (Data grids), Theoretical Background (XML database querying and reasoning)

Q4: EFFECTIVENESS OF THE LANGUAGE -> Success not evaluated

A.1.2 CRUCIBLE [16]

Q2: NATURE OF THE LANGUAGE

Host language: Java

Application Domain: Data analytics

Purpose of the language: Implement the solution

Key advantages: Portability, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Object-Oriented

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Interpreters, Compilers, Tool suite

Technologies used to create the language tool suite: IBM Infosphere, Accumulo, HDFS

Execution stack requirements to support the artifacts created with those languages: OS (any), IO architecture (HDFS), Message Passing Middleware (IBM Infosphere)

Execution model that is being used: Virtual Execution Environment (JVM), Distributed Middleware (IBM InfoSphere), Compiled code for CPU

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user

Technical knowledge required: Tools (XText), Languages (Java), Frameworks (IBM Infosphere), Hardware (CPU), Systems (Clusters), Theoretical Background (Communicating Sequential Processes)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Explicit comparison with competing approaches, Quantitative comparison performed. Productivity gains brought by the languages reported* (Expressiveness and Easier to use - Qualitative), *Products' performance gains brought* (Evolvability/Maintainability - Qualitative)

A.1.3 e-Science Central WFMS [13]

Q2: NATURE OF THE LANGUAGE

Host languages: workflow blocks can be written in Java, R, Octave and Javascript

Application Domain: Cloud-based data analysis

Purpose of the language: Implement the solution

Key advantages: Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

There is a concrete syntax for the language and the preferred representation type is Diagrammatic

Existing tool support for the language: Tool suite

Technologies used to create the language tool suite: They describe porting of a genomics data processing pipeline from a shell-script implementation on a HPC cluster, to e-Science Central based workflow on Microsoft Azure cloud

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-users

Technical knowledge required: Languages (workflow), Systems (Amazon AWS, Microsoft Azure)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed,* Compared shell-script implementation on a HPC cluster with workflow on Microsoft Azure cloud, *Impact on the productivity gains brought* (Learnability, Lower cognitive overload, easier to remember, easier to use - Qualitative and e-Science Central enables users to design workflows for data analysis), *Products' performance gains brought* (Computation efficiency and Scalability - Quantitative; Evolvability/Maintainability - Qualitative)

A.1.4 Higher-order "chemical programming" language [12]

Q2: NATURE OF THE LANGUAGE

Application domain: a rule-based coordination language for asynchronous, self-organizing parallel processing of scientific workflows

Purpose of the language: Formalization of the solution, Implement the solution

Key advantages: Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Declarative (rule-based asynchronous coordination), Hybrid (Atoms of the scripting language are usually written in some sequential HPC language like C)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Interpreters, Compilers

Technologies used to create the language tool suite: HOCL interpreter/JIT plus runtime support extensions for parallel / distributed processing, written in Java

Execution stack requirements to support the artifacts created with those languages: Message Passing Middleware (Java Message Service, ActiveMQ, DAIOS WS (WSDL, SOAP)), Java, HOCL Interpreter

Execution model that is being used: Distributed middleware (Java Message Service, ActiveMQ, DAIOS WS (WSDL, SOAP)), Compiled code for CPU (using a JIT)

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-users

Technical knowledge required: Languages (Java, "chemical programming" in HOCL), Theoretical Background (Rule-based programming, "chemical programming" for WS/workflow coordination)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed,* Experimental comparison with two traditional-style workflow systems based on 3 HPC test problems, *Metrics* (Time), *Impact on the productivity gains brought* (Learnability, Lower cognitive overload, easier to remember, expressiveness, easier to use - Qualitative), *Products' performance gains brought* (Computation efficiency - quantitative; Evolvability/Maintainability, Scalability - Qualitative)

A.1.5 Liszt [14]

Q2: NATURE OF THE LANGUAGE -> A DSL, based on Scala, for solving partial differential equations (PDEs) on unstructured meshes

Application Domain: Constructing mesh-based partial differential equations solvers

Purpose of the language: Implement the solution

Key advantages: Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Functional and Object-Oriented (The Liszt programming environment is based on Scala)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Compilers

The language target specific hardware and GPUs or multi-core architectures

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Technical knowledge required: Languages (Scala)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed.* The authors ported four example applications to Liszt and ran these applications on three platforms: a GPU, an SMP, and a cluster. They evaluate the MPI-based runtime on both the cluster and the SMP since it can run on either platform. *Metrics* (Lines of Code, Time)**,** *Products' performance gains brought* (Computation efficiency and Scalability - Quantitative; Memory Efficiency - Qualitative)

A.1.6 Mendeleev [17]

Q2: NATURE OF THE LANGUAGE

Application Domain: Data analytics

Purpose of the language: Formalization of the requirements of the problem, Implement the solution

Key advantages: Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Declarative (Goal-based planning of analytic applications using an abstract model based on a semantically annotated type system)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Compilers, Tool suite

Technologies used to create the language tool suite: Compiler generators (IBM Infosphere Streams; CRUCIBLE), Goal-based planning of analytic applications with automatic code generation based on CRUCIBLE DSL

Execution stack requirements to support the artifacts created with those languages: IO architecture (HDFS and others), Message Passing Middleware (IBM Infosphere Streams)

Execution model that is being used: Virtual Execution Environment (JVM), Distributed Middleware (IBM InfoSphere), Compiled code for CPU

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user

Technical knowledge required: Tools (Mendeleev DSL), Languages (RDF, IBM InfoSphere, Accumulo), Frameworks (CRUCIBLE, IBM Infosphere), Hardware (CPU), Systems (Clusters), Theoretical Background (RDF graphs)

Q4: EFFECTIVENESS OF THE LANGUAGE -> Success evaluated

A.1.7 MiniZinc [15]

Q2: NATURE OF THE LANGUAGE

Application domain: Constraint modeling language

Purpose of the language: Formalization of the requirements of the problem, Formalization of the solution, Implement the solution

Key advantages: Usability (Effectiveness/Efficiency/Satisfaction), Easier to express constraint problems

Paradigms underlying the language: Hybrid (The constraints are expressed with logic operators)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Compilers, Tool suite, IDE

Technologies used to create the language tool suite: The compiler compiles MiniZinc to FlatZinc, a language that is understood by a wide range of solvers

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-users

Technical knowledge required: Theoretical Background (Constraint modelling)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Both Quantitative and Qualitative comparison performed,* The article compares base version of MiniZinc with one integrating the extensions, *Metrics* (Lines of Code, Time), *Impact on the productivity gains brought* (Expressiveness - Qualitative, Easier to use - Quantitative), *Products' performance gains brought* (Memory efficiency, Computation efficiency - Quantitative)

A.2 General Purpose Languages

A.2.1 Bobolang [34]

Q2: NATURE OF THE LANGUAGE -> Specification language for streaming applications

Application Domain: Design of streaming applications

Purpose of the language: Formalization of the solution, Data Interpretation

Key advantages: Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Declarative (it is a specification language dedicated to designing streaming applications)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing support for the language: Compilers

Technologies used to create the language tool suite: underlying system language (e.g. C++)

Execution model that is being used: Compiled code for CPU (from underlying system language)

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: Developer

Technical knowledge required: Theoretical Background (Domain of streaming applications)

Q4: EFFECTIVENESS OF THE LANGUAGE -> Success not evaluated

A.2.2 C/C++ [18, 22, 27, 29, 30, 31, 32]

Q2: NATURE OF THE LANGUAGE

Application Domain: CFD (any application that benefits from GPU), Heterogeneous Computing

Purpose of the language: Formalization of the requirements of the problem, Formalization of the solution, Simulation of the problem, Simulation of the solution, Implement the solution

Key advantages: Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Object-Oriented, Hybrid (supports heterogeneous environment and it can be event-driven)

There is a concrete syntax for the language and the preferred representation type can be both Textual and Diagrammatic

Existing tool support for the language: Interpreters, Compilers, Validators, Simulators, Tool suite, IDE

Technologies used to create the language tool suite: GenERTiCA source code generator

Execution stack requirements to support the artifacts created with those languages: multiple OS supported

Execution model being used: Virtual Execution Environment (self-managed), Distributed middleware (self-managed), Compiled code for CPU, Compiled code for GPU

The language target GPUs or multi-core architectures

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user and developer

Technical knowledge required: Languages (C/C++), Hardware (parallel & distributed systems; Grids; Clouds)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed,* Algorithms for task scheduling are evaluated, *Metrics* (Time), *Impact on the productivity gains brought* (Learnability - Quantitative and Lower cognitive overload, easier to remember, easier to use - Qualitative), *Products' performance gains brought* (Computation efficiency, Scalability - Quantitative and Evolvability/Maintainability, Scalability - Qualitative)

A.2.3 Erlang [35]

Q2: NATURE OF THE LANGUAGE

Application Domain: Computational and memory-intensive applications using a high number of cores (64). The use-case is urban traffic planning

Purpose of the language: Implement the solution, Data Interpretation

Key Advantages: Performance, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Functional

There is a concrete syntax for the language and the preferred representation type is Textual

Tool support for the language: Interpreters, Compilers, Tool suite, IDE

Execution stack requirements to support the artifacts created with those languages: Message Passing Middleware (Erlang uses a message passing system to communicate between agents), Libraries ("exometer" for global logging and "lcnt" to monitor lock contention)

Execution model that is being used: Virtual Execution Environment (Erlang includes a stack-based VM)

The language target GPUs or multi-core architectures

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Technical knowledge required: Languages (Erlang), Theoretical Background (Agent-oriented frameworks and Evolutionary systems)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Explicit comparison of the language proposal with respect to distinct settings/context/configurations, Quantitative comparison performed,* Scalability of the different techniques when increasing the number of cores, *Metrics* (Number of agent reproductions)

A.2.4 FastFlow [39, 40]

Q2: NATURE OF THE LANGUAGE

Host Language: C++

Application Domain: Streaming applications

Purpose of the language: Implement the solution

Key advantages: Performance, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Functional, Object-Oriented

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Compilers

The language target GPUs or multi-core architectures

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this languages: End-users

Technical knowledge required: Languages (C++), Hardware (CPU), Theoretical Background (Streaming Applications)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed,* The applicability of FastFlow has been illustrated by a number of studies in different application domains including image processing, file compression and stochastic simulation, *Metrics* (Time), *Product's performance gains brought* (Memory Efficiency, Computation Efficiency - Quantitative)

A.2.5 Goal Language supported by RuGPlanner [33]

Q2: NATURE OF THE LANGUAGE -> A declarative language for expressing extended goals, allows for continual plan revision to deal with sensing outputs, failures, long response times or time-outs, as well as the activities of external agents; Many elements of the language are inspired by XSRL (XML Service Request Language)

Purpose of the language: Formalization of the requirements of the problem, Formalization of the solution, Implement the solution, Data Interpretation

Key advantages: Performance, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Declarative (Provides the user with expressive constructs for stating complex goals, beyond the mere statement of properties that should hold in the final state), Functional (comprises a number of atomic service operations that can serve a variety of objectives with minimal request-specific configuration), Logic (it is based on translating the domain and the goal into a Constraint Satisfaction Problem)

There is a concrete syntax for the language and the preferred representation type is Textual

Technologies used to create the language tool suite: an extended language detached from the particularities and interdependencies of the available services

Execution model that is being used: Compiled code for CPU

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user

Technical knowledge required: Languages (Goal language)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed, Explicit comparison of the language proposal with respect to distinct settings/context/configurations,* Two test cases. They performed a number of tests regarding the scalability of the system with respect to a number of factors, *Metrics* (Lines of code, Satisfaction, Time), *Impact on the productivity gains brought* (Learnability, Lower cognitive overload, Easier to remember, Expressiveness, Easier to use - Qualitative), *Products' performance gains brought* (Computation efficiency, Scalability - Quantitative)

A.2.6 Java [18, 19, 20, 21, 22]

Q2: NATURE OF THE LANGUAGE

Application Domain: Grid w applications to Ray tracing and Sequencing; Machine Learning; Specify policies to transform divide and conquer sequential programs into parallel executions

Purpose of the language: Formalization of the requirements of the problem, Formalization of the solution, Simulation of the solution, Implement the solution, Data Interpretation

Key Advantages: Performance, Portability, Easiness of configuration, Orchestration and Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Object-Oriented, Hybrid (Language to schedule constraint solving)

There is a concrete syntax for the language and the preferred representation type is Textual

Tool support for the language: Interpreters, Compilers

Technologies used to create the language tool suite: XML based technology (A XML like syntax to describe classes and methods to be scheduled)

Execution stack requirements to support the artifacts created with those languages: VM Supervisor (JVM on grid), OS (any), IO architecture (Grid), Libraries (Apache Spark, 77 Weka 3.6.0, Hadoop 0.20)

Execution model that is being used: Virtual Execution Environment (Java Virtual Machine), Distributed middleware (Hadoop, Apache Spark), HPC Libraries (Apache Spark), Bytecode for virtual machine (JVM on Grid)

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user

Technical knowledge required: Languages (Java)

Q4: EFFECTIVENESS OF THE LANGUAGE -> ***S**uccess evaluated, Quantitative comparison performed, Metrics* (Lines of code, Time), *Impact on the productivity gains brought* (Easier to use, Compact representation), *Products' performance gains brought* (Computation efficiency, Scalability - quantitative)

A.2.7 OpenCL [27, 28]

Q2: NATURE OF THE LANGUAGE

Application Domain: CFD (any application that benefits from GPU), Big Data processing

Purpose of the language: Formalization of the requirements of the problem, Implement the solution

Key advantages: Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Object-Oriented

There is a concrete syntax for the language and the preferred representation type can be both Textual and Diagrammatic

Existing tool support for the language: Compilers, Tool suite

Technologies used to create the language tool suite: GenERTiCA source code generator

Execution stack requirements to support the artifacts created with those languages: multiple OS supported

Execution model being used: Distributed middleware, HPC Libraries, Bytecode for virtual machine, Compiled code for CPU, Compiled code for GPU

The language target specific hardware and GPUs or multi-core architectures

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user

Technical knowledge required: Tools (detailed knowledge required for using OpenCL for GPUs), Languages (OpenCL), Hardware (Clusters with GPUs)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed,* Algorithms for task scheduling are evaluated, *Metrics* (Time), *Impact on the productivity gains brought* (Learnability, lower cognitive overload, easier to remember, easier to use - Qualitative), *Products' performance gains brought* (Computation efficiency - Quantitative and Evolvability/Maintainability - Qualitative)

A.2.8 Python/R [18, 25, 26]

Q2: NATURE OF THE LANGUAGE

Application domain: High-level parallel programming language for scientific computing, distributed applications

Purpose of the language: Formalization of the requirements of the problem, Formalization of the solution, Simulation of the problem, Simulation of the solution, Implement the solution, Data Interpretation

Key advantages: Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Supports multiple programming paradigms (Object-Oriented, Imperative, Functional, ...)

There is a concrete syntax for the language and the preferred representation type is both Textual and Diagrammatic

Existing tool support for the language: Interpreters, Compilers, Validators, Simulators, Tool suite, IDE

Execution stack requirements to support the artifacts created with those languages: OS (Any), Message Passing Middleware (BSP model), Libraries for Python

Execution model that is being used: Virtual Execution Model (self-managed), Distributed Middleware (self-managed), Compiled code for CPU

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user and Developer

Technical knowledge required: Languages (Python/R), Hardware (parallel & distributed systems; Grids; Clouds)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Explicit comparison with competing approaches, Quantitative comparison performed, Metrics* (Time)*, Impact on the productivity gains brought* (Learnability - Easier to learn and Lower cognitive overload, easier to remember, easier to use - Qualitative), *Products' performance gains brought* (Computation efficiency, Scalability - Quantitative and Scalability - Qualitative)

A.2.9 Scout [36]

Q2: NATURE OF THE LANGUAGE

Purpose of the language: Formalization of the solution, Implement the solution, Data Interpretation, Compiler description

Key Advantages: Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Object-Oriented (the base language from which Scout extends is C*, which is object-oriented)

There is a concrete syntax for the language and the preferred representation type is Textual

Tool support for the language: Compilers

The language target specific hardware and GPUs or multi-core architectures

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Impact on the productivity gains brought* (Lower cognitive overload, Easier to use - Qualitative)

A.2.10 Selective Embedded Just-In-Time Specialization [38]

Q2: NATURE OF THE LANGUAGE

Host Language: Knowledge Discovery Toolbox (KDT)

Application Domain: Semantic Graphs

Purpose of the language: Graph Processing (Implement the solution)

Key advantages: Performance, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Functional, Object-Oriented

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Interpreters, Compilers, Tool suite

Technologies used to create the language tool suite: DSL frameworks (KDT), compBLAS library

Execution stack requirements to support the artifacts created with those languages: OS (any), Message Passing Middleware (MPI), Libraries (compBLAS)

Execution model that is being used: HPC Libraries (compBLAS), Compiled code for CPU

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this languages: End-users

Technical knowledge required: Languages (Python, C++), Libraries (KDT), Hardware (CPU), Systems (Clusters), Theoretical Background (Graph Algorithms)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, There is an explicit comparison with competing approaches, There is an explicit comparison of the language proposal with respect to distinct settings/context/configurations, Quantitative comparison performed,* Performance and coding complexity evaluation against direct usage of Python interface of KDT and direct usage of KDT backend (i.e. compBLAS) on standard graph algorithms and synthetic datasets (in-core), *Metrics* (Lines of Code, Satisfaction, Time), *Impact on the productivity gains brought* (Learnability, Lower cognitive overload, Easier to remember, Expressiveness, Easier to use - Qualitative), *Product's performance gains brought* (Memory Efficiency, Computation Efficiency, Scalability - Quantitative and Evolvability/Maintainability - Qualitative)

A.2.11 SkIE-CL [37]

Q2: NATURE OF THE LANGUAGE -> SkIE-CL, the programming language of the SkIE (SkIE stands for skeleton integrated environment) environment

Host language: C/C++, Fortran, Java

Application Domain: Data mining

Purpose of the language: Implement the solution

Key Advantages: Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction), Enables high-level parallel programming using skeletons

Paradigms underlying the language: Skeletons are used as basic constructs of coordination language (SkIE-CL)

There is a concrete syntax for the language and the preferred representation type is both Textual and Diagrammatic

Tool support for the language: Compilers, Tool suite and IDE

Execution stack requirements to support the artifacts created with those languages: OS (Multiple: Linux, ...), Message Passing Middleware (MPI)

Execution model that is being used: Compiled code for CPU

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-user

Technical knowledge required: Tools (Visual SkIE), Languages (SkIE-CL), Theoretical Background (Skeletons)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Explicit comparison with competing approaches, Explicit comparison of the language proposal with respect to distinct settings/context/configurations, Quantitative comparison performed,* The language is compared with MPI with respect to number of lines of code and development time, *Metrics* (Lines of Code, Time), *Impact on the productivity gains brought* (Learnability, Lower cognitive overload, Easier to use - Qualitative), *Products' performance gains brought* (Evolvability/Maintainability - Qualitative; Scalability - Quantitative)

A.2.12 Swift [23, 24]

Q2: NATURE OF THE LANGUAGE

Application Domain: Parallel workflow/Distributed parallel scripting

Purpose of the language: Implement the solution

Key advantages: Portability, easiness of configuration, orchestration, usability (effectiveness/efficiency/satisfaction)

Paradigms underlying the language: Functional (application components modelled as side-effect free functions)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Interpreters, tool suite

Execution stack requirements to support the artifacts created with those languages: OS (Linux), IO architecture (POSIX), Message Passing Middleware (Globus)

Execution model that is being used: Virtual Execution Environment (Cloud), Distributed Middleware (Globus Grid middleware)

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this languages: End-users

Technical knowledge required: Languages (Swift)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success evaluated, Quantitative comparison performed, Metrics* (Time, Utilization, Scalability), *Impact on the productivity gains brought* (Learnability, Lower cognitive overload, easier to remember, expressiveness, easier to use - Quantitative and Qualitative), *Products' performance gains brought* (Computation efficiency, evolvability/maintainability, scalability, resource utilization - Quantitative and Qualitative)

A.3 Domain Specific Languages embedded in General Purpose Languages

A.3.1 Pipeline Composition (PiCo) [43]

Q2: NATURE OF THE LANGUAGE

Host language: C++

Application Domain: Big Data Analytics

Purpose of the language: Formalization of the solution, Simulation of the solution, Implement the solution, Data Interpretation

Key advantages: Performance, Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Functional, Object-Oriented

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Compilers, Tool suite

Execution stack requirements to support the artifacts created with those languages: OS (PiCo application can be compiled to any target platform supporting a modern C++ compiler)

The language target GPUs or multi-core architectures

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-users

Technical knowledge required: Languages (C++), Frameworks (FastFlow), Theoretical Background (Batch and Streaming Applications)

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success of the language evaluated, Explicit comparison with competing approaches* (They have compared PiCo to two state-of-the-art frameworks: Spark and Flink) *and language proposal with respect to distinct settings/context/configurations, Quantitative comparison performed,* They have compared PiCo to two state-of-the-art frameworks (Spark and Flink) execution times in shared memory for both batch and stream applications, *Metrics* (Time), *Productivity gains brought by the languages* (Expressiveness, Easier to use - Qualitative)**,** *Products' performance gains brought* (Memory Efficiency, Computation efficiency, Scalability - Quantitative)

A.3.2 Spark Streaming and Spark SQL [41]

Q2: NATURE OF THE LANGUAGE

Host language: Spark applications can be written in Java, Scala, Python, R

Application Domain: Streaming analytics

Purpose of the language: Simulation of the problem, Implement the solution

Key advantages: Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Functional (Scala), Object-Oriented (Scala)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Compilers

Execution stack requirements to support the artifacts created with those languages: OS (Linux, MS Windows, macOS), IO architecture (Spark Core), Libraries (MLlib Machine Learning Library)

Execution model that is being used: Distributed Middleware (Hadoop Distributed File System (HDFS), OpenStack Swift,..)

The language target GPUs or multi-core architectures

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-users

Technical knowledge required: Frameworks (Apache Spark)

Q4: EFFECTIVENESS OF THE LANGUAGE -> Presented experimental results for three datasets, *Metrics* (Time), *Products' performance gains brought* (Computation efficiency, scalability - Quantitative)

A.3.3 Weaver [42]

Q2: NATURE OF THE LANGUAGE -> A DSL built on top of Python which allows researchers to construct scalable scientific data-processing workflows

Host language: Python

Application Domain: Scientific workflows

Purpose of the language: Formalization of the solution, Implement the solution

Key advantages: Performance, Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)

Paradigms underlying the language: Functional and Object-Oriented (built on top of Python)

There is a concrete syntax for the language and the preferred representation type is Textual

Existing tool support for the language: Compilers, Tool suite

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: End-users

Technical knowledge required: Python

Q4: EFFECTIVENESS OF THE LANGUAGE -> *Success of the language evaluated, Explicit comparison with competing approaches and language proposal with respect to distinct settings/context/configurations, Quantitative comparison performed,* They provided four applications constructed using Weaver and evaluated its effectiveness in the context of scripting scientific workflows for distributed systems, *Metrics* (Lines of Code, Time), *Productivity gains brought by the languages* (Learnability, Easier to use - Qualitative)*, Products' performance gains brought* (Computation efficiency, scalability - Quantitative and Evolvability/Maintainability - Qualitative)

APPENDIX B - Survey

This survey is being carried out within the scope of the dissertation "Big Data and High Performance Computing DSLs - A Systematic Literature Review", associated to the student Beatriz Norberto no. 42653, from Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa. For that, it is important to question people within that area.
The questionnaire is of short duration and all your answers are totally confidential.
Thank you for your attention.

1. Are you involved in the SLR?

O Yes, I am O No, I'm not

2. How long have you been working on High Performance Computing?

O Less than 2 years O Between 2 and 5 years

O Between 5 and 10 years O More than 10 years

3. In what areas of engineering have you worked? (e.g. Bioinformatics, telecommunications)

4. Does your High Performance Computing activity consist primarily of developing support tools or of using existing tools?

5. Which programming languages do you use for High Performance Computing?

6. What made you use these languages in relation to the alternatives you know? (this may include language properties and contextual factors, etc)

7. What are the key advantages of these languages?

8. How do you rate the existing tool support for the languages you use for HPC? (e.g. tool suite, IDE, simulators, etc.)

O Very Poor O Poor O Neutral O Good O Excellent

9. In relation to the previous question, what are the existing support tools you know?

10. For the domain where you are, how do you rate the effectiveness of the languages you use?

O Very Poor O Poor O Neutral O Good O Excellent

11. What are the fundamental language mechanisms that justify your previous answer?

12. What is the impact on the performance brought by the languages reported?

13. What are the limitations/difficulties of the languages you use?

14. How do you rate your level of technical knowledge for languages used for HPC?

O Very Poor O Poor O Neutral O Good O Excellent

APPENDIX C - Articles that are included in the final review

SLR Publication ID	Publication Reference
P003	[82]
P004	[83]
P006	[84]
P007	[85]
P008	[86]
P010	[87]
P018	[77]
P023	[49]
P030	[78]
P031	[88]
P046	[89]
P049	[90]
P051	[50]
P052	[37]
P055	[21]
P056	[20]
P060	[91]
P062	[92]
P064	[59]
P065	[23]
P066	[93]
P070	[94]
P073	[60]

P076	[79]
P077	[95]
P078	[96]
P079	[11]
P080	[12]
P082	[97]
P084	[80]
P085	[98]
P086	[44]
P090	[99]
P093	[25]
P095	[100]
P096	[101]
P097	[102]
P101	[103]
P102	[104]
P103	[105]
P104	[106]
P105	[107]
P106	[108]
P107	[109]
P109	[46]
P110	[110]
P111	[111]
P114	[112]
P115	[113]

P117	[114]
P118	[27]
P119	[30]
P120	[41]
P134	[31]
P135	[115]
P138	[15]
P139	[13]
P141	[26]
P142	[116]
P143	[117]
P145	[19]
P146	[61]
P147	[62]
P149	[118]
P150	[119]
P151	[22]
P153	[120]
P154	[121]
P155	[122]
P156	[36]
P157	[63]
P158	[123]
P159	[64]
P160	[124]
P161	[125]
P162	[33]

P163	[126]
P164	[127]
P165	[128]
P166	[129]
P167	[130]
P168	[131]
P169	[132]
P171	[133]
P172	[134]
P176	[135]
P177	[72]
P178	[136]
P179	[137]
P180	[138]
P181	[139]
P182	[140]
P184	[141]
P186	[65]
P188	[66]
P190	[67]
P192	[142]
P194	[68]
P195	[45]
P196	[143]
P199	[28]
P200	[144]
P203	[145]

P204	[146]
P209	[147]
P211	[18]
P230	[148]
P234	[149]
P235	[150]
P236	[151]
P237	[152]
P238	[69]
P243	[153]
P246	[52]
P247	[24]
P248	[154]
P250	[155]
P251	[156]
P257	[70]
P258	[53]
P261	[71]
P262	[35]
P274	[130]
P359	[32]
P360	[157]
P361	[16]
P364	[17]
P365	[38]
P367	[48]

P370	[54]
P371	[158]
P372	[34]
P374	[159]
P375	[160]
P376	[161]
P377	[162]
P378	[163]
P379	[164]
P380	[42]
P383	[14]
P387	[165]
P388	[166]
P412	[73]
P413	[40]
P414	[29]
P415	[167]
P416	[39]
P417	[74]
P418	[75]
P419	[76]
P420	[51]
P421	[43]

References

[1] K. Bakshi. “Considerations for Big Data: Architecture and Approach”. In: Aerospace Conference, 2012, IEEE, pp. 1–7. isbn: 978-1-4577-0557-1. doi: https://doi.org/10.1109/AERO.2012.6187357.

[2] A. De Mauro, M. Greco and M. Grimaldi. “What is big data? A consensual definition and a review of key research topics”. In: AIP conference proceedings. Vol. 1644. 1. AIP. 2015, pp. 97–104.
doi: https://doi.org/10.1063/1.4907823.

[3] S. Sagiroglu and D. Sinanc. “Big Data: A Review”. In: 2013 International Conference on Collaboration Technologies and Systems (CTS), IEEE, 2013, pp. 42–47. isbn: 978-1-4673-6404-1.
doi: https://doi.org/10.1109/CTS.2013.6567202.

[4] P. Pacheco. “An Introduction to Parallel Programming”. Morgan Kaufmann Publishers, 2011, pp. 1–14. isbn: 978-0-12-374260-5.

[5] B. Kitchenham and S. Charters. “Guidelines for performing Systematic Literature Reviews in Software Engineering”. Technical report, Ver. 2.3 EBSE Technical Report. EBSE. sn, 2007, pp. 1–57.

[6] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey and S. Linkman. “Systematic literature reviews in software engineering – A systematic literature review”. In: Information and Software Technology 51.1 (2009), 7—15. doi: https://doi.org/10.1016/j.infsof.2008.09.009.

[7] Z. Sharafi, Z. Soh and Y.-G. Guéhéneuc. “A systematic literature review on the usage of eye-tracking in software engineering”. In: Information and Software Technology 67 (2015), pp. 79–107.
doi: https://doi.org/10.1016/j.infsof.2015.06.008.

[8] T. Kosar, N. Oliveira, M. Mernik, M. J. V. Pereira, M. Crepinšek, D. da Cruz and P. R. Henriques. “Comparing General-Purpose and Domain-Specific Languages: An Empirical Study”. In: Computer Science and Information Systems 7.2 (2010), pp. 247–264. doi: https://doi.org/10.2298/CSIS1002247K.

[9] T. Kosar, S. Bohra and M. Mernik. “Domain-Specific Languages: A Systematic Mapping Study”. In: Information and Software Technology 71 (2016), pp. 77–91.
doi: https://doi.org/10.1016/j.infsof.2015.11.001.

[10] A. Van Deursen, P. Klint and J. Visser. “Domain-specific languages: An annotated bibliography”. In: ACM SIGPLAN Notices 35.6 (2000), pp. 26–36. doi: https://doi.org/10.1145/352029.352035.

[11] R. Koning, P. Grosso and C. de Laat. “Using ontologies for resource description in the CineGrid Exchange”. In: Future Generation Computer Systems 27.7 (2011), pp. 960–965.
doi: https://doi.org/10.1016/j.future.2010.11.027.

[12] H. Fernandez, C. Tedeschi and T. Priol. “Rule-driven service coordination middleware for scientific applications”. In: Future Generation Computer Systems 35 (2014), pp. 1–13.
doi: https://doi.org/10.1016/j.future.2013.12.023.

[13] J. Cala, E. Marei, Y. Xu, K. Takeda and P. Missier. “Scalable and efficient whole-exome data processing using workflows on the cloud”. In: Future Generation Computer Systems 65 (2016), pp. 153–168. doi: https://doi.org/10.1016/j.future.2016.01.001.

[14] 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. ACM. 2011, p. 9. doi: https://doi.org/10.1145/2063384.2063396.

[15] R. Caballero, P. J. Stuckey and A. Tenorio-Fornes. “Two type extensions for the constraint modeling language MiniZinc”. In: Science of Computer Programming 111 (2015), pp. 156–189. doi: https://doi.org/10.1016/j.scico.2015.04.007.

[16] P. Coetzee, M. Leeke and S Jarvis. “Towards unified secure on-and off-line analytics at scale”. In: Parallel Computing 40.10 (2014), pp. 738–753. doi: https://doi.org/10.1016/j.parco.2014.07.004.

[17] P Coetzee and S. Jarvis. “Goal-based composition of scalable hybrid analytics for heterogeneous architectures”. In: Journal of Parallel and Distributed Computing 108 (2017), pp. 59–73. doi: https://doi.org/10.1016/j.jpdc.2016.11.009.

[18] 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”. In: SoftwareX 3 (2015), pp. 32–36. doi: https://doi.org/10.1016/j.softx.2015.10.004.

[19] G. Caruana, M. Li and Y. Liu. “An ontology enhanced parallel SVM for scalable spam filter training”. In: Neurocomputing 108 (2013), pp. 45–57. doi: https://doi.org/10.1016/j.neucom.2012.12.001.

[20] C. Mateos, A. Zunino and M. Campo. “An approach for non-intrusively adding malleable fork/join parallelism into ordinary JavaBean compliant applications”. In: Computer Languages, Systems & Structures 36.3 (2010), pp. 288–315. doi: https://doi.org/10.1016/j.cl.2009.12.003.

[21] 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”. In: Advances in Engineering Software 42.4 (2011), pp. 172–186. doi: https://doi.org/10.1016/j.advengsoft.2011.02.003.

[22] A. Meade, D. K. Deeptimahanti, J. Buckley and J. Collins. “An empirical study of data decomposition for software parallelization”. In: Journal of Systems and Software 125 (2017), pp. 401–416. doi: https://doi.org/10.1016/j.jss.2016.02.002.

[23] 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”. In: Future Generation Computer Systems 54 (2016), pp. 206–218. doi: https://doi.org/10.1016/j.future.2015.03.017.

[24] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz and I. Foster. “Swift: A language for distributed parallel scripting”. In: Parallel Computing 37.9 (2011), pp. 633–652. doi: https://doi.org/10.1016/j.parco.2011.05.005.

[25] K. Hinsen, H. P. Langtangen, O. Skavhaug and Å. Ødegård. “Using B SP and Python to simplify parallel programming”. In: Future Generation Computer Systems 22.1-2 (2006), pp. 123–157. doi: https://doi.org/10.1016/j.future.2003.09.003.

[26] A. Luckow, M. Santcroos, A. Zebrowski and S. Jha. “Pilot-data: an abstraction for distributed data”. In: Journal of Parallel and Distributed Computing 79-80 (2015), pp. 16–30. doi: https://doi.org/10.1016/j.jpdc.2014.09.009.

[27] 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”. In: Control Engineering Practice 21.2 (2013), pp. 204–217. doi: https://doi.org/10.1016/j.conengprac.2012.10.001.

[28] 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”. In: Computers & Fluids 115 (2015), pp. 1–10. doi: https://doi.org/10.1016/j.compfluid.2015.03.021.

[29] 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. ACM. 2010, pp. 5–14. doi: https://doi.org/10.1145/1863482.1863487.

[30] T.-Y. Liang, H.-F. Li, Y.-J. Lin and B.-S. Chen. “A Distributed PTX Virtual Machine on Hybrid CPU/GPU Clusters”. In: Journal of Systems Architecture 62 (2016), pp. 63–77.
doi: https://doi.org/10.1016/j.sysarc.2015.10.003.

[31] C. Obrecht, F. Kuznik, B. Tourancheau and J.-J. Roux. “The TheLMA project: A thermal lattice Boltzmann solver for the GPU”. In: Computers & Fluids 54 (2012), pp. 118–126.
doi: https://doi.org/10.1016/j.compfluid.2011.10.011.

[32] 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. IEEE. 2015, pp. 1–12. doi: https://doi.org/10.1145/2807591.2807655.

[33] E. Kaldeli, A. Lazovik and M. Aiello. “Domain-independent planning for services in uncertain and dynamic environments”. In: Artificial Intelligence 236 (2016), pp. 30–64.
doi: https://doi.org/10.1016/j.artint.2016.03.002.

[34] Z. Falt, D. Bednárek, 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. ACM. 2014, pp. 311–314. doi: https://doi.org/10.1145/2600212.2600711.

[35] 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”. In: Journal of Computational Science 17 (2016), pp. 234–248. doi: https://doi.org/10.1016/j.jocs.2016.03.003.

[36] P. McCormick, J. Inman, J. Ahrens, J. Mohd-Yusof, G. Roth and S. Cummins. “Scout: a data-parallel programming language for graphics processors”. In: Parallel Computing 33.10-11 (2007), pp. 648–662. doi: https://doi.org/10.1016/j.parco.2007.09.001.

[37] M. Coppola and M. Vanneschi. “High-performance data mining with skeleton-based structured parallel programming”. In: Parallel Computing 28.5 (2002), pp. 793–813.
doi: https://doi.org/10.1016/S0167-8191(02)00095-9.

[38] 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”. In: Journal of Parallel and Distributed Computing 79 (2015), pp. 115–131. doi: https://doi.org/10.1016/j.jpdc.2014.08.010.

[39] M. Aldinucci, M. Danelutto, P. Kilpatrick and M. Torquati. “FastFlow: high-level and efficient streaming on multi-core,” in S. Pllana and F. Xhafa (eds.), “Programming Multi-core and Many-core Computing Systems” (2017), pp. 135–144. doi: https://doi.org/10.1002/9781119332015.ch13.

[40] G. Mencagli, M. Torquati, F. Lucattini, S. Cuomo and M. Aldinucci. “Harnessing sliding-window execution semantics for parallel stream processing”. In: Journal of Parallel and Distributed Computing 116 (2018), pp. 74–88. doi: https://doi.org/10.1016/j.jpdc.2017.10.021.

[41] G. Liu, W. Zhu, C. Saunders, F. Gao and Y. Yu. “Real-time complex event processing and analytics for smart grid”. In: Procedia Computer Science 61 (2015), pp. 113–119.
doi: https://doi.org/10.1016/j.procs.2015.09.169.

[42] P. Bui, L. Yu, A. Thrasher, R. Carmichael, I. Lanc, P. Donnelly and D. Thain. “Scripting distributed scientific workflows using Weaver”. Em: Concurrency and Computation: Practice and Experience 24.15 (2011), pp. 1685–1707. doi: https://doi.org/10.1002/cpe.1871.

[43] C. Misale, M. Drocco, G. Tremblay, A. R. Martinelli and M. Aldinucci. “PiCo: High performance data analytics pipelines in modern C++”. Em: Future Generation Computer Systems 87 (2018), pp. 392–403. doi: https://doi.org/10.1016/j.future.2018.05.030.

[44] M. B. Giles, G. R.Mudalige, B. Spencer, C. Bertolli and I. Reguly. “Designing OP2 for GPU architectures”. In: Journal of Parallel and Distributed Computing 3.11 (2013), pp. 1451–1460. doi: https://doi.org/10.1016/j.jpdc.2012.07.008.

[45] D. Hünich, A. Knüpfer and J. Gracia. “Providing Parallel Debugging for DASH Distributed Data Structures with GDB”. In: Procedia Computer Science 51 (2015), pp. 1383–1392. doi: https://doi.org/10.1016/j.procs.2015.05.345.

[46] 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”. In: Future Generation Computer Systems 23.4 (2007), pp. 547–557. doi: https://doi.org/10.1016/j.future.2006.10.003

[47] 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”. In: Gerhard R. Joubert, Hugh Leather, Mark Parsons, Frans Peters, Mark Sawyer (eds.): *Advances in Parallel Computing, Volume 27: Parallel Computing: On the Road to Exascale* (2016), pp. 135–144. doi: https://doi.org/10.3233/978-1-61499-621-7-135.

[48] M. Viñas, B. B. Fraguela, D. Andrade and R. Doallo. “High productivity multi-device exploitation with the Heterogeneous Programming Library”. In: Journal of Parallel and Distributed Computing 101 (2017), pp. 51–68. doi: https://doi.org/10.1016/j.jpdc.2016.11.001.

[49] M. B. Belgacem and B. Chopard. “MUSCLE-HPC: A new high performance API to couple multiscale parallel applications”. In: Future Generation Computer Systems 67 (2017), pp. 72–82. doi: https://doi.org/10.1016/j.future.2016.08.009.

[50] E. Dede, Z. Fadika, M. Govindaraju and L. Ramakrishnan. “Benchmarking MapReduce implementations under different application scenarios”. In: Future Generation Computer Systems 36 (2014), pp. 389–399. doi: https://doi.org/10.1016/j.future.2014.01.001.

[51] S. Memeti, L. Li, S. Pllana, J. Koƚodziej and C. Kessler. “Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption”. Em: Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing. ACM. 2017, pp. 1–6. doi: https://doi.org/10.1145/3110355.3110356.

[52] M. Vanneschi. “The programming model of ASSIST, an environment for parallel and distributed portable applications”. In: Parallel computing 28.12 (2002), pp. 1709–1732. doi: https://doi.org/10.1016/S0167-8191(02)00188-6.

[53] Z. Wang, Y. Liu and P. Ma. “A CUDA-enabled parallel implementation of collaborative filtering”. In: Procedia Computer Science 30 (2014), pp. 66–74. doi: https://doi.org/10.1016/j.procs.2014.05.382.

[54] Y. Zhang and F. Mueller. “GStream: A general-purpose data streaming framework on GPU clusters”. In: International Conference on Parallel Processing (ICPP), 2011, pp. 245–254, IEEE. doi: https://doi.org/10.1109/ICPP.2011.22.

[55] S. Pllana, S. Benkner, E. Mehofer, L. Natvig, and F. Xhafa. “Towards an Intelligent Environment for Programming Multi-core Computing Systems”. In: César E. et al. (eds) Euro-Par 2008 Workshops - Parallel Processing. Euro-Par 2008. Lecture Notes in Computer Science, vol 5415. Springer, Berlin, Heidelberg. doi: https://doi.org/10.1007/978-3-642-00955-6_19

[56] B. Brown, M. Chui and J. Manyika. “Are you Ready for the era of ’Big Data’”. In: McKinsey Quarterly 4.1 (2011), pp. 24–35.

[57] 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”. Conference on Design Automation and Test in Europe (2012), Dresden, Germany, IEEE. doi: https://doi.org/10.1109/DATE.2012.6176582.

[58] M. Sandrieser, S. Benkner and S. Pllana. “Using explicit platform descriptions to support programming of heterogeneous many-core systems”, Parallel Computing, Volume 38, Issues 1-2, 2012, pages 52-65, ISSN 0167-8191. doi: https://doi.org/10.1016/j.parco.2011.10.008.

[59] Y. Ma, H. Wu, L. Wang, B. Huang, R. Ranjan, A. Zomaya and W. Jie. “Remote sensing big data computing: Challenges and opportunities”. In: Future Generation Computer Systems 51 (2015), pp. 47–60. doi: https://doi.org/10.1016/j.future.2014.10.029.

[60] A. Chianese and F. Piccialli. “A service oriented framework for analysing social network activities”. In: Procedia Computer Science 98 (2016), pp. 509–514. doi: https://doi.org/10.1016/j.procs.2016.09.087.

[61] S. Cavuoti, M. Garofalo, M. Brescia, M. Paolillo, A. Pescape, G. Longo and G. Ventre. “Astrophysical data mining with GPU. A case study: genetic classification of globular clusters”. In: New Astronomy 26 (2014), pp. 12–22. doi: https://doi.org/10.1016/j.newast.2013.04.004.

[62] U. Erra, S. Senatore, F. Minnella and G. Caggianese. “Approximate TF–IDF based on topic extraction from massive message stream using the GPU”. In: Information Sciences 292 (2015), pp. 143–161. doi: https://doi.org/10.1016/j.ins.2014.08.062.

[63] C. Mayr, U. Zdun and S. Dustdar. “View-based model-driven architecture for enhancing maintainability of data access services”. In: Data & Knowledge Engineering 70.9 (2011), pp. 794–819. doi: https://doi.org/10.1016/j.datak.2011.05.004.

[64] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang and J.-J. Li. “Distributed evolutionary algorithms and their models: A survey of the state-of-the-art”. In: Applied Soft Computing 34 (2015), pp. 286–300. doi: https://doi.org/10.1016/j.asoc.2015.04.061.

[65] N. K. Alham, M. Li, Y. Liu and S. Hammoud. “A MapReduce-based distributed SVM algorithm for automatic image annotation”. In: Computers & Mathematics with Applications 62.7 (2011), pp. 2801–2811. doi: https://doi.org/10.1016/j.camwa.2011.07.046.

[66] L. Hu and S. Nooshabadi. “G-SHOT: GPU accelerated 3D local descriptor for surface matching”. In: Journal of Visual Communication and Image Representation 30 (2015), pp. 343–349. doi: https://doi.org/10.1016/j.jvcir.2015.05.008.

[67] M. Aldinucci and M. Danelutto. “Securing skeletal systems with limited performance penalty: the Muskel experience”. In: Journal of Systems Architecture 54.9 (2008), pp. 868–876. doi: https://doi.org/10.1016/j.sysarc.2008.02.008.

[68] S. S. de Almeida, A. C. de Nazaré Júnior, A. de Albuquerque Araújo, G. Cámara-Chávez and D. Menotti. “Speeding up a video summarization approach using GPUs and multicore CPUs”. In: Procedia Computer Science 29 (2014), pp. 159–171. doi: https://doi.org/10.1016/j.procs.2014.05.015.

[69] C. Yan, T. Yue and G Zhao. “GPU Accelerated High Accuracy Surface Modelling”. In: Procedia Environmental Sciences 13 (2012), pp. 555–564. doi: https://doi.org/10.1016/j.proenv.2012.01.046.

[70] J. Wang, D. Crawl and I. Altintas. “A framework for distributed data-parallel execution in the Kepler scientific workflow system”. In: Procedia Computer Science 9 (2012), pp. 1620–1629. doi: https://doi.org/10.1016/j.procs.2012.04.178.

[71] M. Vaidya and S. Deshpande. “Critical Study of Performance Parameters on Distributed File Systems using MapReduce”. In: Procedia Computer Science 78 (2016), pp. 224–232. doi: https://doi.org/10.1016/j.procs.2016.02.037.

[72] T. Gautier and H.-R. Hamidi. “Re-scheduling invocations of services for RPC grids”. In: Computer Languages, Systems & Structures 33.3-4 (2007), pp. 168–178.
doi: https://doi.org/10.1016/j.cl.2006.07.006.

[73] C. Misale, M. Drocco, M. Aldinucci and G. Tremblay. “A comparison of big data frameworks on a layered dataflow model”. In: Parallel Processing Letters 27.01 (2017), p. 1740003.
doi: https://doi.org/10.1142/S0129626417400035.

[74] D. De Sensi, T. De Matteis, M. Torquati, G. Mencagli and M. Danelutto. “Bringing Parallel Patterns Out of the Corner: The P 3 ARSEC Benchmark Suite”. In: ACM Transactions on Architecture and Code Optimization (TACO) 14.1 (2017), p. 33. doi: https://dl.acm.org/citation.cfm?doid=3132710.

[75] A. Viebke, S. Memeti, S. Pllana and A. Abraham. “Chaos: a parallelization scheme for training convolutional neural networks on intel xeon phi”. In: The Journal of Supercomputing (2017), pp. 1–31. doi: https://doi.org/10.1007/s11227-017-1994-x.

[76] 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”. In: IEEE Micro 31.5 (2011), pp. 28–41. doi: http://doi.ieeecomputersociety.org/10.1109/MM.2011.670.

[77] H.-G. Lee, N. Park, H.-I. Jeong and J. Park. “Grid enabled MRP process improvement under distributed database environment”. In: Journal of Systems and Software 82.7 (2009), pp. 1087–1097.
doi: https://doi.org/10.1016/j.jss.2009.01.041.

[78] S. Rathi. “Optimizing Sorting Algorithms using Ubiquitous multi-core massively parallel GPGPU processors”. In: Procedia Computer Science 79 (2016), pp. 231–237.
doi: https://doi.org/10.1016/j.procs.2016.03.030.

[79] M. Krämer and I. Senner. “A modular software architecture for processing of big geospatial data in the cloud”. In: Computers & Graphics 49 (2015), pp. 69–81.
doi: https://doi.org/10.1016/j.cag.2015.02.005.

[80] W. Kolberg, P. D. B. Marcos, J. C. Anjos, A. K. Miyazaki, C. R. Geyer and L. B. Arantes. “Mrsg–a mapreduce simulator over simgrid”. In: Parallel Computing 39.4-5 (2013), pp. 233–244.
doi: https://doi.org/10.1016/j.parco.2013.02.001.

[81] M. A. Beyer and D. Laney. The importance of big data: A definition. Technical report, Stamford, CT: Gartner, June 2012.

[82] K. Soni, D. D. Chandar and J. Sitaraman. “Development of an overset grid computational fluid dynamics solver on graphical processing units”. In: Computers & Fluids 58 (2012), pp. 1–14.
doi: https://doi.org/10.1016/j.compfluid.2011.11.014.

[83] U. Sivarajah, M. M. Kamal, Z. Irani and V.Weerakkody. “Critical analysis of Big Data challenges and analytical methods”. In: Journal of Business Research 70 (2017), pp. 263–286.
doi: https://doi.org/10.1016/j.jbusres.2016.08.001.

[84] Y.-L. Su, P.-C. Chen, J.-B. Chang and C.-K. Shieh. “Variable-sized map and locality-aware reduce on public-resource grids”. In: Future Generation Computer Systems 27.6 (2011), pp. 843–849.
doi: https://doi.org/10.1016/j.future.2010.09.001.

[85] C.-H. Tang, M.-F. Tsai, S.-H. Chuang, J.-J. Cheng and W.-J. Wang. “Shortest linkage-based parallel hierarchical clustering on main-belt moving objects of the solar system”. In: Future Generation Computer Systems 34 (2014), pp. 26–46. doi: https://doi.org/10.1016/j.future.2013.12.029.

[86] A. Simonet, G. Fedak and M. Ripeanu. “Active Data: A programming model to manage data life cycle across heterogeneous systems and infrastructures”. In: Future Generation Computer Systems 53 (2015), pp. 25–42. doi: https://doi.org/10.1016/j.future.2015.05.015.

[87] J. Sui, C. Xu, S.-C. Cheung, W. Xi, Y. Jiang, C. Cao, X. Ma and J. Lu. “Hybrid CPU–GPU constraint checking: Towards efficient context consistency”. In: Information and Software Technology 74 (2016), pp. 230–242. doi: https://doi.org/10.1016/j.infsof.2015.10.003.

[88] L. M. Pinho, V. Nélis, P. M. Yomsi, E. Quiñones, M. Bertogna, P. Burgio, A. Marongiu, C. Scordino, P.

Gai, M. Ramponi et al. “P-SOCRATES: A parallel software framework for time-critical many-core systems”. In: Microprocessors and Microsystems 39.8 (2015), pp. 1190–1203.
doi: https://doi.org/10.1016/j.micpro.2015.06.004.

[89] W. Jing, D. Tong, Y.Wang, J.Wang, Y. Liu and P. Zhao. “MaMR: High-performance MapReduce programming model for material cloud applications”. In: Computer Physics Communications 211 (2017), pp. 79–87. doi: https://doi.org/10.1016/j.cpc.2016.07.015.

[90] E. Deelman, D. Gannon, M. Shields and I. Taylor. “Workflows and e-Science: An overview of workflow system features and capabilities”. In: Future generation computer systems 25.5 (2009), pp. 528–540. doi: https://doi.org/10.1016/j.future.2008.06.012.

[91] S. Crafa. “The role of concurrency in an evolutionary view of programming abstractions”. In: Journal of Logical and Algebraic Methods in Programming 84.6 (2015), pp. 732–741.
doi: https://doi.org/10.1016/j.jlamp.2015.07.006.

[92] Z. Ma, H Wang and S. Pu. “GPU computing of compressible flow problems by a meshless method with space-filling curves”. In: Journal of Computational Physics 263 (2014), pp. 113–135.
doi: https://doi.org/10.1016/j.jcp.2014.01.023.

[93] L. Han, C. S. Liew, J. Van Hemert and M. Atkinson. “A generic parallel processing model for facilitating data mining and integration”. In: Parallel Computing 37.3 (2011), pp. 157–171.
doi: https://doi.org/10.1016/j.parco.2011.02.0060.

[94] M. Hammoudeh and R. Newman. “Information extraction from sensor networks using the Watershed transform algorithm”. In: Information Fusion 22 (2015), pp. 39–49.
doi: https://doi.org/10.1016/j.inffus.2013.07.001.

[95] R. Giachetta. “A framework for processing large scale geospatial and remote sensing data in MapReduce environment”. In: Computers & graphics 49 (2015), pp. 37–46.
doi: https://doi.org/10.1016/j.cag.2015.03.003.

[96] G. Giuliani, N. Ray and A. Lehmann. “Grid-enabled Spatial Data Infrastructure for environmental sciences: Challenges and opportunities”. In: Future Generation Computer Systems 27.3 (2011), pp. 292–303. doi: https://doi.org/10.1016/j.future.2010.09.011.

[97] J. Kranjc, R. Orac, V. Podpecan, N. Lavrac and M. Robnik-Šikonja. “ClowdFlows: Online workflows for distributed big data mining”. In: Future Generation Computer Systems 68 (2017), pp. 38–58.
doi: https://doi.org/10.1016/j.future.2016.07.018.

[98] J. Kranjc, J. Smailovic, V. Podpecan, M. Grcar, M. Žnidaršic and N. Lavrac. “Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the ClowdFlows platform”. In: Information Processing & Management 51.2 (2015), pp. 187–203.
doi: https://doi.org/10.1016/j.ipm.2014.04.001.

[99] M. Fernández, J. C. Pichel, J. C. Cabaleiro and T. F. Pena. “Boosting performance of a statistical machine translation system using dynamic parallelism”. In: Journal of Computational Science 13 (2016), pp. 37–48. doi: https://doi.org/10.1016/j.jocs.2016.01.003.

[100] C.-H. Hsu. “Intelligent big data processing”. In: Future Generation Computer Systems 36 (2014), pp. 16–18. doi: https://doi.org/10.1016/j.future.2014.02.003.

[101] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny et al. “Pegasus, a workflow management system for science automation”. In: Future Generation Computer Systems 46 (2015), pp. 17–35. doi: https://doi.org/10.1016/j.future.2014.10.008.

[102] C.-H. Hsu, K. D. Slagter and Y.-C. Chung. “Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications”. In: Future Generation Computer Systems 53 (2015), pp. 43–54. doi: https://doi.org/10.1016/j.future.2015.04.006.

[103] A. Núñez, J. Fernández, R. Filgueira, F. García and J. Carretero. “SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications”. In: Simulation Modelling Practice and Theory 20.1 (2012), pp. 12–32.

doi: https://doi.org/10.1016/j.simpat.2011.08.009.

[104] V. Fernández, V. Méndez and T. F. Pena. “Federated Big Data for resource aggregation and load balancing with DIRAC”. In: Procedia Computer Science 51 (2015), pp. 2769–2773.
doi: https://doi.org/10.1016/j.procs.2015.05.430.

[105] N. Emad and S. Petiton. “Unite and conquer approach for high scale numerical computing”. In: Journal of computational science 14 (2016), pp. 5–14. doi: https://doi.org/10.1016/j.jocs.2016.01.007.

[106] D. Mourtzis, M. Doukas and D. Bernidaki. “Simulation in manufacturing: Review and challenges”. In: Procedia CIRP 25 (2014), pp. 213–229. doi: https://doi.org/10.1016/j.procir.2014.10.032.

[107] P. D. Diamantoulakis, V. M. Kapinas and G. K. Karagiannidis. “Big data analytics for dynamic energy management in smart grids”. In: Big Data Research 2.3 (2015), pp. 94–101.
doi: https://doi.org/10.1016/j.bdr.2015.03.003.

[108] B. P. Pickering, C. W. Jackson, T. R. W. Scogland, W.-C. Feng and C. J. Roy. “Directive-based GPU programming for computational fluid dynamics”. In: Computers & Fluids 114 (2015), pp. 242–253.
doi: https://doi.org/10.1016/j.compfluid.2015.03.008.

[109] A. Papagiannis and D. S. Nikolopoulos. “Hybrid address spaces: A methodology for implementing scalable high-level programming models on non-coherent many-core architectures”. In: Journal of Systems and Software 97 (2014), pp. 47–64. doi: https://doi.org/10.1016/j.jss.2014.06.058.

[110] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic. “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”. In: Future Generation computer systems 25.6 (2009), pp. 599–616. doi: https://doi.org/10.1016/j.future.2008.12.001.

[111] M. Bux and U. Leser. “Dynamiccloudsim: Simulating heterogeneity in computational clouds”. In: Future Generation Computer Systems 46 (2015), pp. 85–99.
doi: https://doi.org/10.1016/j.future.2014.09.007.

[112] M. Liroz-Gistau, R. Akbarinia, D. Agrawal and P. Valduriez. “FP-Hadoop: Efficient processing of skewed MapReduce jobs”. In: Information Systems 60 (2016), pp. 69–84.
doi: https://doi.org/10.1016/j.is.2016.03.008.

[113] G. A. Papakostas, K. I. Diamantaras and T. Papadimitriou. “Parallel pattern classification utilizing GPU-based kernelized Slackmin algorithm”. In: Journal of Parallel and Distributed Computing 99 (2017), pp. 90–99. doi: https://doi.org/10.1016/j.jpdc.2016.09.001.

[114] L. Burgueño, M. Wimmer and A. Vallecillo. “A linda-based platform for the parallel execution of out-place model transformations”. In: Information and Software Technology 79 (2016), pp. 17–35.
doi: https://doi.org/10.1016/j.infsof.2016.06.001.

[115] P. van Oosterom, O. Martinez-Rubi, M. Ivanova, M. Horhammer, D. Geringer, S. Ravada, T. Tijssen, M. Kodde and R. Gonçalves. “Massive point cloud data management: Design, implementation and execution of a point cloud benchmark”. In: Computers & Graphics 49 (2015), pp. 92–125.
doi: https://doi.org/10.1016/j.cag.2015.01.007.

[116] A. Panyala, D. Chavarría-Miranda, J. B. Manzano, A. Tumeo and M. Halappanavar. “Exploring performance and energy tradeoffs for irregular applications: A case study on the Tilera many-core architecture”. In: Journal of Parallel and Distributed Computing 104 (2017), pp. 234–251.
doi: https://doi.org/10.1016/j.jpdc.2016.06.006.

[117] C. Obrecht, F. Kuznik, B. Tourancheau and J.-J. Roux. “A new approach to the lattice Boltzmann method for graphics processing units”. In: Computers & Mathematics with Applications 61.12 (2011), pp. 3628–3638. doi: https://doi.org/10.1016/j.camwa.2010.01.054.

[118] R. Guha, N. Bagherzadeh and P. Chou. “Resource management and task partitioning and scheduling on a run-time reconfigurable embedded system”. In: Computers & Electrical Engineering 35.2 (2009), pp. 258–285. doi: https://doi.org/10.1016/j.compeleceng.2008.06.008.

[119] D. Eränen, J. Oksanen, J.Westerholm and T. Sarjakoski. “A full graphics processing unit implementation of uncertainty-aware drainage basin delineation”. In: Computers & Geosciences 73

(2014), pp. 48–60. doi: https://doi.org/10.1016/j.cageo.2014.08.012.

[120] T. Gunarathne, B. Zhang, T.-L. Wu and J. Qiu. “Scalable parallel computing on clouds using Twister4Azure iterative MapReduce”. In: Future Generation Computer Systems 29.4 (2013), pp. 1035–1048. doi: https://doi.org/10.1016/j.future.2012.05.027.

[121] J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami. “Internet of Things (IoT): A vision, architectural elements, and future directions”. In: Future generation computer systems 29.7 (2013), pp. 1645–1660. doi: https://doi.org/10.1016/j.future.2013.01.010.

[122] É. Michon, J. Gossa, S. Genaud, L. Unbekandt and V. Kherbache. “Schlouder: A broker for IaaS clouds”. In: Future Generation Computer Systems 69 (2017), pp. 11–23.
doi: https://doi.org/10.1016/j.future.2016.09.010.

[123] A. Morajko, T. Margalef and E. Luque. “Design and implementation of a dynamic tuning environment”. In: Journal of Parallel and Distributed Computing 67.4 (2007), pp. 474–490.
doi: https://doi.org/10.1016/j.jpdc.2007.01.001.

[124] I. Grasso, M. Ritter, B. Cosenza, W. Benger, G. Hofstetter and T. Fahringer. “Point distribution tensor computation on heterogeneous systems”. In: Procedia Computer Science 51 (2015), pp. 160–169. doi: https://doi.org/10.1016/j.procs.2015.05.217.

[125] J. Espinosa-Oviedo, G. Vargas-Solar, V. Alexandrov and G. Castel. “Comparing Electoral Campaigns by Analysing Online Data”. In: Procedia Computer Science 80 (2016), pp. 1865–1874.
doi: https://doi.org/10.1016/j.procs.2016.05.480.

[126] B. Svensson et al. “Evolution in architectures and programming methodologies of coarse-grained reconfigurable computing”. In: Microprocessors and Microsystems 33.3 (2009), pp. 161–178.
doi: https://doi.org/10.1016/j.micpro.2008.10.003.

[127] V. Garousi, L. C. Briand and Y. Labiche. “Traffic-aware stress testing of distributed real-time systems based on UML models using genetic algorithms”. In: Journal of Systems and Software 81.2 (2008), pp. 161–185. doi: https://doi.org/10.1016/j.jss.2007.05.037.

[128] G. Folino, A. Forestiero, G. Papuzzo and G. Spezzano. “A grid portal for solving geoscience problems using distributed knowledge discovery services”. In: Future Generation Computer Systems 26.1 (2010), pp. 87–96. doi: https://doi.org/10.1016/j.future.2009.08.002.

[129] J. Secretan, M. Georgiopoulos, A. Koufakou and K. Cardona. “APHID: An architecture for private, high-performance integrated data mining”. In: Future Generation Computer Systems 26.7 (2010), pp. 891–904. doi: https://doi.org/10.1016/j.future.2010.02.017.

[130] T. Kajdanowicz, P. Kazienko and W. Indyk. “Parallel processing of large graphs”. In: Future Generation Computer Systems 32 (2014), pp. 324–337.
doi: https://doi.org/10.1016/j.future.2013.08.007.

[131] G. Fortino, D. Parisi, V. Pirrone and G. Di Fatta. “BodyCloud: A SaaS approach for community body sensor networks”. In: Future Generation Computer Systems 35 (2014), pp. 62–79.
doi: https://doi.org/10.1016/j.future.2013.12.015.

[132] R. Albodour, A. James and N. Yaacob. “QoS within business grid quality of service (BGQoS)”.
In: Future Generation Computer Systems 50 (2015), pp. 22–37.
doi: https://doi.org/10.1016/j.future.2014.10.027.

[133] M. R. Selim and M. Z. Rahman. “Carrying on the legacy of imperative languages in the future parallel computing era”. In: Parallel Computing 40.3-4 (2014), pp. 1–33.
doi: https://doi.org/10.1016/j.parco.2014.02.001.

[134] A. M. Aji, A. J. Peña, P. Balaji and W.-c. Feng. “MultiCL: Enabling automatic scheduling for task-parallel workloads in OpenCL”. In: Parallel Computing 58 (2016), pp. 37–55.
doi: https://doi.org/10.1016/j.parco.2016.05.006.

[135] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann et al. “T-CREST: Time predictable multi-core architecture for

embedded systems". In: Journal of Systems Architecture 61.9 (2015), pp. 449–471.
doi: https://doi.org/10.1016/j.sysarc.2015.04.002.

[136] C. Kaiser and A. Pozdnoukhov. "Enabling real-time city sensing with kernel stream oracles and MapReduce". In: Pervasive and Mobile Computing 9.5 (2013), pp. 708–721.
doi: https://doi.org/10.1016/j.pmcj.2012.11.003.

[137] M. Schneider, D. Fey, D. Kapusi and T. Machleidt. "Performance comparison of designated preprocessing white light interferometry algorithms on emerging multi-and many-core architectures". In: Procedia Computer Science 4 (2011), pp. 2037–2046.
doi: https://doi.org/10.1016/j.procs.2011.04.222.

[138] R. S. Kalawsky. "The Next Generation of Grand Challenges for Systems Engineering Research".
In: Procedia Computer Science 16 (2013), pp. 834–843.
doi: https://doi.org/10.1016/j.procs.2013.01.087.

[139] I. Satoh. "Pervasive Data Processing". In: Procedia Computer Science 63 (2015), pp. 16–23.
doi: https://doi.org/10.1016/j.procs.2015.08.307.

[140] D. Serrano, S. Bouchenak, Y. Kouki, F. A. de Oliveira Jr, T. Ledoux, J. Lejeune, J. Sopena, L. Arantes and P. Sens. "SLA guarantees for cloud services". In: Future Generation Computer Systems 54 (2016), pp. 233–246. doi: https://doi.org/10.1016/j.future.2015.03.018.

[141] K. He, S. X.-D. Tan, H. Zhao, X.-X. Liu, H. Wang and G. Shi. "Parallel GMRES solver for fast analysis of large linear dynamic systems on GPU platforms". In: INTEGRATION, the VLSI journal 52 (2016), pp. 10–22. doi: https://doi.org/10.1016/j.vlsi.2015.07.005.

[142] F. Huang, J. Tao, Y. Xiang, P. Liu, L. Dong and L. Wang. "Parallel compressive sampling matching pursuit algorithm for compressed sensing signal reconstruction with OpenCL". In: Journal of Systems Architecture 72 (2017), pp. 51–60. doi: https://doi.org/10.1016/j.sysarc.2016.07.002.

[143] R. Hanisch, G. Berriman, T. Lazio, S. E. Bunn, J Evans, T. McGlynn and R Plante. "The Virtual Astronomical Observatory: Re-engineering access to astronomical data". In: Astronomy and Computing 11 (2015), pp. 190–209. doi: https://doi.org/10.1016/j.ascom.2015.03.007.

[144] J. A. Ross, D. A. Richie, S. J. Park and D. R. Shires. "Parallel programming model for the Epiphany many-core coprocessor using threaded MPI". In: Microprocessors and Microsystems 43 (2016), pp. 95–103. doi: https://doi.org/10.1016/j.micpro.2016.02.006.

[145] J. C. Anjos, I. Carrera, W. Kolberg, A. L. Tibola, L. B. Arantes and C. R. Geyer. "MRA++: Scheduling and data placement on MapReduce for heterogeneous environments". In: Future Generation Computer Systems 42 (2015), pp. 22–35. doi: https://doi.org/10.1016/j.future.2014.09.001.

[146] D. S. Katz, A. Merzky, Z. Zhang and S. Jha. "Application skeletons: Construction and use in eScience". In: Future Generation Computer Systems 59 (2016), pp. 114–124.
doi: https://doi.org/10.1016/j.future.2015.10.001.

[147] D. Barseghian, I. Altintas, M. B. Jones, D. Crawl, N. Potter, J. Gallagher, P. Cornillon, M. Schildhauer, E. T. Borer, E.W. Seabloom et al. "Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis". In: Ecological Informatics 5.1 (2010), pp. 42–50.
doi: https://doi.org/10.1016/j.ecoinf.2009.08.008.

[148] C. Seceleanu, M. Johansson, J. Suryadevara, G. Sapienza, T. Seceleanu, S.-E. Ellevseth and P. Pettersson. "Analyzing a wind turbine system: From simulation to formal verification". In: Science of Computer Programming 133 (2017), pp. 216–242. doi: https://doi.org/10.1016/j.scico.2016.09.007.

[149] Q. Zagarese, G. Canfora, E. Zimeo, I. Alshabani, L. Pellegrino, A. Alshabani and F. Baude. "Improving data-intensive EDA performance with annotation driven laziness". In: Science of Computer Programming 97 (2015), pp. 266–279. doi: https://doi.org/10.1016/j.scico.2014.03.007.

[150] D. Zhang, P. Coddington and A. Wendelborn. "Web services workflow with result data forwarding as resources". In: Future Generation Computer Systems 27.6 (2011), pp. 694–702.
doi: https://doi.org/10.1016/j.future.2010.12.015.

[151] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy and R. Buyya. “Deadline driven provisioning of resources for scientific applications in hybrid clouds with Aneka”. In: Future Generation Computer Systems 28.1 (2012), pp. 58–65.doi: https://doi.org/10.1016/j.future.2011.05.008.

[152] P. Wang, J. Wang, Y. Chen and G. Ni. “Rapid processing of remote sensing images based on cloud computing”. In: Future Generation Computer Systems 29.8 (2013), pp. 1963–1968.
doi: https://doi.org/10.1016/j.future.2013.05.002.

[153] O. Yildiz, S. Ibrahim and G. Antoniu. “Enabling fast failure recovery in shared Hadoop clusters: towards failure-aware scheduling”. In: Future Generation Computer Systems 74 (2017), pp. 208–219.
doi: https://doi.org/10.1016/j.future.2016.02.015.

[154] G. Teodoro, T. Pan, T. Kurc, J. Kong, L. Cooper, S. Klasky and J. Saltz. “Region templates: Data representation and management for high-throughput image analysis”. In: Parallel computing 40.10 (2014), pp. 589–610. doi: https://doi.org/10.1016/j.parco.2014.09.003.

[155] Z. Wei and J. JaJa. “A fast algorithm for constructing inverted files on heterogeneous platforms”.
In: Journal of Parallel and Distributed Computing 72.5 (2012), pp. 728–738.
doi: https://doi.org/10.1016/j.jpdc.2012.02.005.

[156] W. Yan, U. Brahmakshatriya, Y. Xue, M. Gilder and B. Wise. “p-PIC: Parallel power iteration clustering for big data”. In: Journal of Parallel and Distributed computing 73.3 (2013), pp. 352–359.
doi: https://doi.org/10.1016/j.jpdc.2012.06.009.

[157] L. Salucci, D. Bonetta and W. Binder. “Lightweight Multi-language Bindings for Apache Spark”.
In: European Conference on Parallel Processing. Springer. 2016, pp. 281–292.
doi: https://doi.org/10.1007/978-3-319-43659-3_21.

[158] D.Wang, D. J. Foran, X. Qi and M. Parashar. “HetroCV: Auto-tuning Framework and Runtime for Image Processing and Computer Vision Applications on Heterogeneous Platform”. In: Parallel Processing Workshops (ICPPW), 2015 44th International Conference on. IEEE. 2015, pp. 119–128.
doi: https://doi.org/10.1109/ICPPW.2015.21.

[159] M.Wahib and N. Maruyama. “Automated GPU kernel transformations in large-scale production stencil applications”. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing. ACM.2015, pp. 259–270. doi: https://doi.org/10.1145/2749246.2749255.

[160] P. Hijma, R. Van Nieuwpoort, C. J. Jacobs and H. E. Bal. “Stepwise-refinement for performance: a methodology for many-core programming”. In: Concurrency and Computation: Practice and Experience 27.17 (2015), pp. 4515–4554. doi: https://doi.org/10.1002/cpe.3416.

[161] R. Behrends, K. Hammond, V. Janjic, A. Konovalov, S. Linton, H.-W. Loidl, P. Maier and P. Trinder. “HPC-GAP: engineering a 21st-century high-performance computer algebra system”. In: Concurrency and Computation: Practice and Experience 28.13 (2016), pp. 3606–3636.
doi: https://doi.org/10.1002/cpe.3746.

[162] N. Edmonds, J. Willcock and A. Lumsdaine. “Expressing graph algorithms using generalized active messages”. In: Proceedings of the 27th international ACM conference on International conference on supercomputing. ACM. 2013, pp. 283–292. doi: https://doi.org/10.1145/2464996.2465441.

[163] C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell and J. Shalf. “Software design space exploration for exascale combustion co-design”. In: International Supercomputing Conference. Springer. 2013,
pp. 196–212. doi: https://doi.org/10.1007/978-3-642-38750-0_15.

[164] L. Chen, X. Huo and G. Agrawal. “A pattern specification and optimizations framework for accelerating scientific computations on heterogeneous clusters”. In: Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International. IEEE. 2015, pp. 591–600.
doi: https://doi.org/10.1109/IPDPS.2015.13.

[165] H. Singh and S. Bawa. “HadoopWeb: MapReduce Platform for Big Data Analysis”. In: International Research Journal of Engineering and Technology (IRJET) 3.7 (2016), pp. 1355–1361.

[166] X. Li, M. Grossman and D. Kaeli. “Mahout on heterogeneous clusters using HadoopCL”.

In: Proceedings of the 2nd Workshop on Parallel Programming for Analytics Applications. ACM. 2015, pp. 9–16. doi: https://doi.org/10.1145/2726935.2726940.

[167] 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". In: ParCo-2015 conference 27 (2016), pp. 135–144. doi: https://doi.org/10.3233/978-1-61499-621-7-135.

[168] K. Petersen, R. Feldt, S. Mujtaba and M. Mattsson. "Systematic Mapping Studies in Software Engineering". In: Evaluation and Assessment in Software Engineering. Vol. 8. 2008, pp. 68–77.

[169] A. Abdelmaboud, D. N. Jawawi, I. Ghani, A. Elsafi and B. Kitchenham. "Quality of service approaches in cloud computing: A systematic mapping study". In: Journal of Systems and Software 101 (2015), pp. 159–179. doi: https://doi.org/10.1016/j.jss.2014.12.015.

[170] D. Dicheva, C. Dichev, G. Agre and G. Angelova. "Gamification in education: A systematic mapping study." In: Journal of Educational Technology & Society 18.3 (2015).
doi: https://www.jstor.org/stable/jeductechsoci.18.3.75.

[171] E. Engström and P. Runeson. "Software product line testing–a systematic mapping study".
In: Information and Software Technology 53.1 (2011), pp. 2–13.
doi: https://doi.org/10.1016/j.infsof.2010.05.011.

[172] A. Fernandez, E. Insfran and S. Abrahão. "Usability evaluation methods for the web: A systematic mapping study". In: Information and Software Technology 53.8 (2011), pp. 789–817.
doi: https://doi.org/10.1016/j.infsof.2011.02.007.