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

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;
Luis 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;
Ilia Mavridis, imavridis@csd.auth.gr, Aristotle University of Thessaloniki, Greece;
Sabri Pilana, sabri.pilana@lnu.se, Linnaeus University, Sweden;
Ana Respicio, 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 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

\(^4\) TOP500 list, June 2018, https://www.top500.org/
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**.

![Figure 2.1: Methodology used in the SLR](image)
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**

<table>
<thead>
<tr>
<th>Question ID</th>
<th>Research Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ 1</td>
<td>Which are the categories of languages in use?</td>
</tr>
<tr>
<td>RQ 1.1</td>
<td>What are the current research trends in languages for HPC?</td>
</tr>
<tr>
<td>RQ 2</td>
<td>What is the nature of the languages for HPC?</td>
</tr>
<tr>
<td>RQ 2.1</td>
<td>What kind of language is it?</td>
</tr>
<tr>
<td>RQ 2.2</td>
<td>What is the execution model that is being used?</td>
</tr>
<tr>
<td>RQ 2.3</td>
<td>What are the key advantages of the language?</td>
</tr>
<tr>
<td>RQ 2.4</td>
<td>What is/are the application domain/s of the language?</td>
</tr>
<tr>
<td>RQ 2.5</td>
<td>What are the paradigms underlying the languages?</td>
</tr>
<tr>
<td>RQ 2.6</td>
<td>Which are the execution stack requirements (?-aaS) to support the artifacts created with those languages?</td>
</tr>
<tr>
<td>RQ 2.7</td>
<td>What is the existing tool support for the language?</td>
</tr>
<tr>
<td>RQ 2.8</td>
<td>What are the technologies used to create the language tool suite?</td>
</tr>
<tr>
<td>RQ 2.9</td>
<td>Does the language target specific hardware?</td>
</tr>
<tr>
<td>RQ 2.10</td>
<td>What is the purpose of the language?</td>
</tr>
<tr>
<td>RQ 2.11</td>
<td>What is the preferred language representation type?</td>
</tr>
<tr>
<td>RQ 3</td>
<td>What are the typical user profiles for the languages?</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>RQ 3.1</td>
<td>What are the roles of the users of this language?</td>
</tr>
<tr>
<td>RQ 3.2</td>
<td>What kind of technical knowledge is required?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RQ 4</th>
<th>How effective are the languages?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ 4.1</td>
<td>Is the success of the languages evaluated in the articles?</td>
</tr>
<tr>
<td>RQ 4.2</td>
<td>What is the impact on the productivity gains brought by the languages reported?</td>
</tr>
<tr>
<td>RQ 4.3</td>
<td>What is the impact on the products' performance gains brought by the languages reported?</td>
</tr>
</tbody>
</table>

| RQ 4.3 | Is there an explicit comparison with competing approaches? |
| RQ 4.4 | Is the comparison quantitative, qualitative, or both? |
| RQ 4.5 | What are the comparison methodology and metrics used? |

<table>
<thead>
<tr>
<th>RQ 5</th>
<th>What types of articles are published in the area of programming models for HPC?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ 5.1</td>
<td>Does the article include COST cHiPSet's authors?</td>
</tr>
<tr>
<td>RQ 5.2</td>
<td>What are the institutions involved?</td>
</tr>
<tr>
<td>RQ 5.3</td>
<td>What is the name of the conference or journal?</td>
</tr>
<tr>
<td>RQ 5.4</td>
<td>Who is sponsoring the research?</td>
</tr>
<tr>
<td>RQ 5.5</td>
<td>What kind of research is being reported?</td>
</tr>
</tbody>
</table>

### 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.

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 chPiSet 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

<table>
<thead>
<tr>
<th>Conferences</th>
<th>Journals</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTC / GPGPU conference</td>
<td>ACM Transactions on Parallel Computing</td>
</tr>
<tr>
<td>IEEE International Parallel and Distributed</td>
<td>Concurrency and Computation Practice and</td>
</tr>
<tr>
<td>Processing Symposium</td>
<td>Experience</td>
</tr>
<tr>
<td>International Conference on Parallel Processing</td>
<td>Future Generation Computer Systems</td>
</tr>
<tr>
<td>International Conference on Supercomputing</td>
<td>IEEE Computing in Science and Engineering</td>
</tr>
<tr>
<td>International European Conference on Parallel</td>
<td>Journal of Parallel and Distributed Computing</td>
</tr>
<tr>
<td>and Distributed Computing</td>
<td></td>
</tr>
<tr>
<td>International Supercomputing Conference</td>
<td>Journal of Supercomputing</td>
</tr>
<tr>
<td>Parallel Computing Conference</td>
<td>Parallel Computing</td>
</tr>
<tr>
<td>Principles and Practice of Parallel Programming</td>
<td>Scientific Programming</td>
</tr>
<tr>
<td>SIAM Conference on Parallel Processing for</td>
<td></td>
</tr>
<tr>
<td>Scientific Computing</td>
<td></td>
</tr>
<tr>
<td>Supercomputing Conference</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Source Name</th>
<th>Number of Publications obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>academia.edu</td>
<td>1</td>
</tr>
<tr>
<td>ACM Digital Library</td>
<td>16</td>
</tr>
<tr>
<td>Compendex</td>
<td>6</td>
</tr>
<tr>
<td>Elsevier Science Direct</td>
<td>27</td>
</tr>
<tr>
<td>Google Scholar</td>
<td>32</td>
</tr>
<tr>
<td>IEEE Xplore</td>
<td>3</td>
</tr>
<tr>
<td>Research Gate</td>
<td>3</td>
</tr>
<tr>
<td>Springer Link</td>
<td>70</td>
</tr>
<tr>
<td><strong>Total of articles found:</strong></td>
<td><strong>158</strong></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Inclusion Criteria</th>
<th>Exclusion Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Study must have addressed HPC research</td>
<td>Irrelevant publication that lay outside the core HPC research field</td>
</tr>
<tr>
<td>Peer reviewed study that had been published in journal, conference and workshop</td>
<td>Non-peer reviewed study (abstract, tutorial, editorial, slides, talk, tool demonstration, poster, panel, keynote, technical report)</td>
</tr>
<tr>
<td>Study must be written in English</td>
<td>Publication not in English</td>
</tr>
<tr>
<td>Study must be accessible electronically</td>
<td>Electronically non-accessible study</td>
</tr>
<tr>
<td>Study is related with Computer science literature / Systems area</td>
<td>Article published before 2006</td>
</tr>
</tbody>
</table>

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 a 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](image)
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](image)

![Figure 3.3: What is the existing tool support for the language? - RQ 2.7](image)
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).

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).
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](image)

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](image)
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 analyses, 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](image1)

![Figure 3.12: Number of articles reporting each type of research - RQ 5.5](image2)
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)):
• Their main activity consists on the development of new support tools, rather than the use of existing tools (see Figure 4.1.2);

![Pie chart](image)

**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);

![Bar chart](image)

**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](image)

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](image)
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;
• 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](image)

### 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

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

Q3: TYPICAL USER PROFILES FOR THE LANGUAGE

Roles of the users of this language: 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)
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)

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

<table>
<thead>
<tr>
<th>SLR Publication ID</th>
<th>Publication Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>P003</td>
<td>[82]</td>
</tr>
<tr>
<td>P004</td>
<td>[83]</td>
</tr>
<tr>
<td>P006</td>
<td>[84]</td>
</tr>
<tr>
<td>P007</td>
<td>[85]</td>
</tr>
<tr>
<td>P008</td>
<td>[86]</td>
</tr>
<tr>
<td>P010</td>
<td>[87]</td>
</tr>
<tr>
<td>P018</td>
<td>[77]</td>
</tr>
<tr>
<td>P023</td>
<td>[49]</td>
</tr>
<tr>
<td>P030</td>
<td>[78]</td>
</tr>
<tr>
<td>P031</td>
<td>[88]</td>
</tr>
<tr>
<td>P046</td>
<td>[89]</td>
</tr>
<tr>
<td>P049</td>
<td>[90]</td>
</tr>
<tr>
<td>P051</td>
<td>[50]</td>
</tr>
<tr>
<td>P052</td>
<td>[37]</td>
</tr>
<tr>
<td>P055</td>
<td>[21]</td>
</tr>
<tr>
<td>P056</td>
<td>[20]</td>
</tr>
<tr>
<td>P060</td>
<td>[91]</td>
</tr>
<tr>
<td>P062</td>
<td>[92]</td>
</tr>
<tr>
<td>P064</td>
<td>[59]</td>
</tr>
<tr>
<td>P065</td>
<td>[23]</td>
</tr>
<tr>
<td>P066</td>
<td>[93]</td>
</tr>
<tr>
<td>P070</td>
<td>[94]</td>
</tr>
<tr>
<td>P073</td>
<td>[60]</td>
</tr>
<tr>
<td>Page</td>
<td>Number</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>P204</td>
<td>146</td>
</tr>
<tr>
<td>P209</td>
<td>147</td>
</tr>
<tr>
<td>P211</td>
<td>18</td>
</tr>
<tr>
<td>P230</td>
<td>148</td>
</tr>
<tr>
<td>P234</td>
<td>149</td>
</tr>
<tr>
<td>P235</td>
<td>150</td>
</tr>
<tr>
<td>P236</td>
<td>151</td>
</tr>
<tr>
<td>P237</td>
<td>152</td>
</tr>
<tr>
<td>P238</td>
<td>69</td>
</tr>
<tr>
<td>P243</td>
<td>153</td>
</tr>
<tr>
<td>P246</td>
<td>52</td>
</tr>
<tr>
<td>P247</td>
<td>24</td>
</tr>
<tr>
<td>P248</td>
<td>154</td>
</tr>
<tr>
<td>P250</td>
<td>155</td>
</tr>
<tr>
<td>P251</td>
<td>156</td>
</tr>
<tr>
<td>P257</td>
<td>70</td>
</tr>
<tr>
<td>P258</td>
<td>53</td>
</tr>
<tr>
<td>P261</td>
<td>71</td>
</tr>
<tr>
<td>P262</td>
<td>35</td>
</tr>
<tr>
<td>P274</td>
<td>130</td>
</tr>
<tr>
<td>P359</td>
<td>32</td>
</tr>
<tr>
<td>P360</td>
<td>157</td>
</tr>
<tr>
<td>P361</td>
<td>16</td>
</tr>
<tr>
<td>P364</td>
<td>17</td>
</tr>
<tr>
<td>P365</td>
<td>38</td>
</tr>
<tr>
<td>P367</td>
<td>48</td>
</tr>
<tr>
<td>Page</td>
<td>Number</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>P370</td>
<td>[54]</td>
</tr>
<tr>
<td>P371</td>
<td>[158]</td>
</tr>
<tr>
<td>P372</td>
<td>[34]</td>
</tr>
<tr>
<td>P374</td>
<td>[159]</td>
</tr>
<tr>
<td>P375</td>
<td>[160]</td>
</tr>
<tr>
<td>P376</td>
<td>[161]</td>
</tr>
<tr>
<td>P377</td>
<td>[162]</td>
</tr>
<tr>
<td>P378</td>
<td>[163]</td>
</tr>
<tr>
<td>P379</td>
<td>[164]</td>
</tr>
<tr>
<td>P380</td>
<td>[42]</td>
</tr>
<tr>
<td>P383</td>
<td>[14]</td>
</tr>
<tr>
<td>P387</td>
<td>[165]</td>
</tr>
<tr>
<td>P388</td>
<td>[166]</td>
</tr>
<tr>
<td>P412</td>
<td>[73]</td>
</tr>
<tr>
<td>P413</td>
<td>[40]</td>
</tr>
<tr>
<td>P414</td>
<td>[29]</td>
</tr>
<tr>
<td>P415</td>
<td>[167]</td>
</tr>
<tr>
<td>P416</td>
<td>[39]</td>
</tr>
<tr>
<td>P417</td>
<td>[74]</td>
</tr>
<tr>
<td>P418</td>
<td>[75]</td>
</tr>
<tr>
<td>P419</td>
<td>[76]</td>
</tr>
<tr>
<td>P420</td>
<td>[51]</td>
</tr>
<tr>
<td>P421</td>
<td>[43]</td>
</tr>
</tbody>
</table>
References


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


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

64


doi: https://www.jstor.org/stable/jeductechsoci.18.3.75.

doi: https://doi.org/10.1016/j.infsof.2010.05.011.