Companion data of a Systematic Mapping Study of Programming Languages for Data-Intensive HPC Applications

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

[a] *Universidade Nova de Lisboa, Portugal*
[b] *University of Torino, Italy*
[c] *University of Vienna, Austria*
[d] *University of Stirling, UK*
[e] *Universidade de Lisboa, Portugal*
[f] *University of Latvia, Latvia*
[g] *University of Amsterdam, Netherlands*
[h] *Aristotle University of Thessaloniki, Greece*
[i] *Linköping University, Sweden*
[j] *Queens University Belfast, U.K.*
[k] *Linnaeus University, Sweden*
[l] *Instituto Superior de Engenharia de Lisboa, Portugal*
[m] *Tampere University of Technology, Finland*

Abstract

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

We organised the SMS in two phases. In the first phase, relevant articles are identified employing an automated keyword-based search in eight digital libraries. This lead to an initial sample of 420 papers, which was then narrowed down in a second phase by human inspection of article abstracts, titles and keywords to 152 relevant articles published in the period 2006–2018. The analysis of these articles enabled us to identify 26 programming languages referred to in 33 of relevant articles. This document is the data companion for a paper published elsewhere and presents a detailed list of the selected papers. Besides, the document also presents the formof our questionnaire-based survey that involved 57 HPC experts.

Keywords:
High Performance Computing (HPC), Big Data, Data Intensive Applications, Programming Languages, Domain-Specific Languages

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

Appendix A. Languages used for Data-Intensive HPC Applications

Table A.2: CineGrid Description Language + Network Description Language [17]

		CineGrid Description Language + Network Description Language
RQ1	*Type*	Domain Specific Language
RQ2	*Kind*	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	Formalisation of the requirements of the problem; Formalisation of the solution; Data Interpretation
	Key advantages	Portability, easiness of configuration, visualisation of user-initiated query results
	Paradigms	Declarative (Data access service configuration and deployment structure graphs expressed in OWL/XML syntax)
	Concrete syntax	Textual
	Existing tool	Interpreters
	Technologies	XML based technology (Jess reasoner for querying OWL ontologies)
RQ3	*Users' role*	Developer
	Required knowledge	Tools (OWL/XML editor), Languages (SQWRL query language for OWL ontologies), Hardware/Systems (Data grids), Theoretical Background (XML database querying and reasoning)
RQ4	*Effectiveness*	*Success not evaluated*

Table A.3: Crucible [9]

		Crucible
RQ1	*Type*	Domain Specific Language
RQ2	*Kind*	Based on Java host language
	Application domain	Data analytic
	Key advantages	Portability, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Object-Oriented
	Concrete syntax	Textual
	Existing tool	Interpreters, Compilers, Tool suite
	Technologies	IBM Infosphere, Accumulo, HDFS
	Execution stack	OS (any), IO architecture (HDFS), Message Passing Middleware (IBM Infosphere)
	Execution model	Virtual Execution Environment (JVM), Distributed Middleware (IBM InfoSphere), Compiled code for CPU
RQ3	*Users' role*	End-user
	Required knowledge	Tools (XText), Languages (Java), Frameworks (IBM Infosphere), Hardware (CPU), Systems (Clusters), Theoretical Background (Communicating Sequential Processes)
RQ4	*Effectiveness*	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)

Table A.4: e-Science Central WFMS [6]

		e-Science Central WFMS
RQ1	*Type*	Domain Specific Language
RQ2	*Host language*	Workflow blocks can be written in Java, R, Octave and Javascript
	Application domain	Cloud-based data analysis
	Key advantages	Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)
	Concrete syntax	Diagrammatic
	Existing tool	Tool suite
	Technologies	They describe porting of a genomics data processing pipeline from a shell-script implementation on a HPC cluster, to e-Science Central based work-flow on Microsoft Azure cloud
RQ3	*Users' role*	End-user
	Required knowledge	Languages (workflow), Systems (Amazon AWS, Microsoft Azure)
RQ4	*Effectiveness*	Quantitative comparison performed, compared shell-script implementation on a HPC cluster with work-flow on Microsoft Azure cloud, 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)

Table A.5: Higher-order "chemical programming" language [13]

		Higher-order "chemical programming" language
RQ1	*Type*	Domain Specific Language
RQ2	*Application domain*	A rule-based coordination language for asynchronous, self-organizing parallel processing of scientific workflows
	Purpose	Formalisation of the solution, Implement the solution
	Key advantages	Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Declarative (rule-based asynchronous coordination), Hybrid (Atoms of the scripting language are usually written in some sequential HPC language like C)
	Concrete syntax	Textual
	Existing tool	Interpreters, Compilers
	Technologies	HOCL interpreter/JIT plus runtime support extensions for parallel / distributed processing, written in Java
	Execution stack	Message Passing Middle-ware (Java Message Service, ActiveMQ, DAIOS WS (WSDL, SOAP)), Java, HOCL Interpreter
	Execution model	Distributed middle-ware (Java Message Service, ActiveMQ, DAIOS WS (WSDL, SOAP)), Compiled code for CPU (using a JIT)
RQ3	*Users' role*	End-user
	Required knowledge	Languages (Java, "chemical programming" in HOCL), Theoretical Background (Rule-based programming, "chemical programming" for WS/work-flow coordination)
RQ4	*Effectiveness*	Success evaluated, Quantitative comparison performed, Experimental comparison with two traditional-style work-flow systems based on 3 HPC test problems, Metrics (Time), Productivity gains brought (Learnability, Lower cognitive overload, easier to remember, expressiveness (captures the concepts of the domain), easier to use - Qualitative), Products' performance gains brought (Computation efficiency – quantitative; Evolvability/Maintainability, Scalability – Qualitative)

Table A.6: Liszt [11]

		Liszt
RQ1	*Type*	Domain Specific Language
RQ2	*Nature*	A DSL, based on Scala, for solving partial differential equations (PDEs) on unstructured meshes
	Application domain	Constructing mesh-based partial differential equations solvers
	Purpose	Implement the solution
	Key advantages	Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Functional and Object-Oriented (The Liszt programming environment is based on Scala)
	Concrete syntax	Textual
	Existing tool	Compilers
	Execution model	The language target specific hardware and GPUs or multi-core architectures
RQ3	*Users' role*	Developer
	Required knowledge	Languages (Scala)
RQ4	*Effectiveness*	*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)

Table A.7: Mendeleev [8]

		Mendeleev
RQ1	*Type*	Domain Specific Language
RQ2	*Application domain*	Data analytics
	Key advantages	Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Declarative (Goal-based planning of analytic applications using an abstract model based on a semantically annotated type system)
	Concrete syntax	Textual
	Existing tool	Compilers, Tool suite
	Technologies	Compiler generators (IBM Infosphere Streams; Crucible), Goal-based planning of analytic applications with automatic code generation based on Crucible DSL
	Execution stack	IO architecture (HDFS and others), Message Passing Middleware (IBM Infosphere Streams)
	Execution model	Virtual Execution Environment (JVM), Distributed Middleware (IBM InfoSphere), Compiled code for CPU
RQ3	*Users' role*	End-user
	Required knowledge	Tools (Mendeleev DSL), Languages (RDF, IBM InfoSphere, Accumulo), Frameworks (Crucible, IBM Infosphere), Hardware (CPU), Systems (Clusters), Theoretical Background (RDF graphs)
RQ4	*Effectiveness*	*Success evaluated*

Table A.8: MiniZinc [5]

		MiniZinc
RQ1	*Type*	Domain Specific Language
RQ2	*Application domain*	Constraint modelling language
	Purpose	Formalisation of the requirements of the problem, Formalisation of the solution, Implement the solution
	Key advantages	Usability (Effectiveness/Efficiency/ Satisfaction), Easier to express constraint problems
	Paradigms	Hybrid (The constraints are expressed with logic operators)
	Concrete syntax	Textual
	Existing tool	Compilers, Tool suite, IDE
	Technologies	The compiler compiles MiniZinc to FlatZinc, a language that is understood by a wide range of solvers
RQ3	*Users' role*	End-user
	Required knowledge	Theoretical Background (Constraint modelling)
RQ4	*Effectiveness*	*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), *Productivity gains brought* (Expressiveness - Qualitative, Easier to use - Quantitative), *Products' performance gains brought* (Memory efficiency, Computation efficiency - Quantitative)

Table A.9: Bobolang [5]

		Bobolang
RQ1	*Type*	General Purpose Languages
RQ2	*Nature*	Specification language for streaming applications
	Application domain	Constraint modelling language
	Purpose	Formalisation of the solution, Data Interpretation
	Key advantages	Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Declarative (it is a specification language dedicated to designing streaming applications)
	Concrete syntax	Textual
	Existing tool	Compilers
	Technologies	Underlying system language (e.g. C++)
	Execution model	Compiled code for CPU (from underlying system language)
RQ3	*Users' role*	Developer
	Required knowledge	Theoretical Background (Domain of streaming applications)
RQ4	*Effectiveness*	*Success not evaluated*

Table A.10: C/C++ [2, 3, 12, 18, 26, 29, 30]

		C/C++
RQ1	*Type*	General Purpose Languages
RQ2	*Nature*	Specification language for streaming applications
	Application domain	Scientific Computing, Heterogeneous Computing
	Purpose	Formalisation of the requirements of the problem, Formalisation 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	Object-Oriented, Hybrid (supports heterogeneous environment and it can be event-driven)
	Concrete syntax	Textual and Diagrammatic
	Existing tool	Interpreters, Compilers, Validators, Simulators, Tool suite, IDE
	Technologies	GenERTiCA source code generator
	Execution stack	Multiple OSes
	Execution model	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
RQ3	*Users' role*	End-user and Developer
	Required knowledge	Languages (C/C++), Hardware (parallel and distributed systems; Grids; Clouds)
RQ4	*Effectiveness*	*Success evaluated, Quantitative comparison performed*, Algorithms for task scheduling are evaluated, *Metrics* (Time), *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)

Table A.11: Erlang [31]

		Erlang
RQ1	*Type*	General Purpose Languages
RQ2	*Application domain*	Computational and memory-intensive applications using a high number of cores (64). The use-case is urban traffic planning
	Purpose	Implement the solution, Data Interpretation
	Key advantages	Performance, Usability (Effectiveness/Efficiency/ Satisfaction)
	Paradigms	Functional
	Concrete syntax	Textual
	Existing tool	Interpreters, Compilers, Tool suite, IDE
	Execution stack	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	Virtual Execution Environment (Erlang includes a stack-based VM), the language target GPUs or multi-core architectures
RQ3	*Required knowledge*	Languages (Erlang), Theoretical Background (Agent-oriented frameworks and Evolutionary systems)
RQ4	*Effectiveness*	*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)

Table A.12: FastFlow [1, 27]

	FastFlow	
RQ1	*Type*	General Purpose Languages
RQ2	*Host language*	C++
	Application domain	Streaming applications
	Purpose	Implement the solution
	Key advantages	Performance, Usability (Effectiveness/Efficiency/ Satisfaction)
	Paradigms	Functional, Object-Oriented
	Concrete syntax	Textual
	Existing tool	Compilers
	Execution model	The language target GPUs or multi-core architectures
RQ3	*Users' role*	End-user
	Required knowledge	Languages (C++), Hardware (CPU), Theoretical Background (Streaming Applications)
RQ4	*Effectiveness*	*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), *Products' performance gains brought* (Memory Efficiency, Computation Efficiency - Quantitative)

Table A.13: Goal Language supported by RuGPlanner [15]

	Goal Language supported by RuGPlanner	
RQ1	*Type*	General Purpose Languages
RQ2	*Nature*	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	Formalisation of the requirements of the problem, Formalisation of the solution, Implement the solution, Data Interpretation
	Key advantages	Performance, Orchestration, Usability (Effectiveness/ Efficiency/Satisfaction)
	Paradigms	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)
	Concrete syntax	Textual
	Technologies	An extended language detached from the particularities and inter-dependencies of the available services
	Execution model	Compiled code for CPU
RQ3	*Users' role*	End-user
	Required knowledge	Languages (Goal language)
RQ4	*Effectiveness*	*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), *Productivity gains brought* (Learnability, Lower cognitive overload, Easier to remember, Expressiveness, Easier to use - Qualitative), *Products' performance gains brought* (Computation efficiency, Scalability - Quantitative)

Table A.14: Java [2, 7, 23, 24, 26]

		Java
RQ1	*Type*	General Purpose Languages
RQ2	*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	Formalisation of the requirements of the problem, Formalisation 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	Object-Oriented, Hybrid (Language to schedule constraint solving)
	Concrete syntax	Textual
	Existing tool	Interpreters, Compilers
	Technologies	XML based technology (A XML like syntax to describe classes and methods to be scheduled)
	Execution stack	VM Supervisor (JVM on grid), OS (any), IO architecture (Grid), Libraries (Apache Spark, 77 Weka 3.6.0, Hadoop 0.20)
	Execution model	Virtual Execution Environment (Java Virtual Machine), Distributed middleware (Hadoop, Apache Spark), HPC Libraries (Apache Spark), Bytecode for virtual machine (JVM on Grid)
RQ3	*Users' role*	End-user
	Required knowledge	Languages (Java)
RQ4	*Effectiveness*	*Success evaluated, Quantitative comparison performed, Metrics* (Lines of code, Time), *Productivity gains brought* (Easier to use, Compact representation), *Products' performance gains brought* (Computation efficiency, Scalability - Quantitative)

Table A.15: OpenCL [3, 16]

		OpenCL
RQ1	*Type*	General Purpose Languages
RQ2	*Application domain*	CFD (any application that benefits from GPU), Big Data processing
	Purpose	Formalisation of the requirements of the problem, Implement the solution
	Key advantages	Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Object-Oriented
	Concrete syntax	Textual and Diagrammatic
	Existing tool	Compilers, Tool suite
	Technologies	GenERTiCA source code generator
	Execution stack	Multiple OSes supported
	Execution model	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
RQ3	*Users' role*	End-user
	Required knowledge	Tools (detailed knowledge required for using OpenCL for GPUs), Languages (OpenCL), Hardware (Clusters with GPUs)
RQ4	*Effectiveness*	*Success evaluated, Quantitative comparison performed,* Algorithms for task scheduling are evaluated, *Metrics* (Time), *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)

Table A.16: Python/R [2, 14, 20]

		Python/R
RQ1	*Type*	General Purpose Languages
RQ2	*Application domain*	High-level parallel programming language for scientific computing, distributed applications
	Purpose	Formalisation of the requirements of the problem, Formalisation 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	Supports multiple programming paradigms (Object-Oriented, Imperative, Functional,)
	Concrete syntax	Textual and Diagrammatic
	Existing tool	Interpreters, Compilers, Validators, Simulators, Tool suite, IDE
	Execution stack	OS (Any), Message Passing Middleware (BSP model), Libraries
	Execution model	Virtual Execution Model (self-managed), Distributed Middleware (self-managed), Compiled code for CPU
RQ3	*Users' role*	End-user and Developer
	Required knowledge	Languages (Python/R), Hardware (parallel and distributed systems; Grids; Clouds)
RQ4	*Effectiveness*	*Success evaluated, Explicit comparison with competing approaches, Quantitative comparison performed, Metrics* (Time), *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)

Table A.17: Scout [25]

		Scout
RQ1	*Type*	General Purpose Languages
RQ2	*Purpose*	Formalisation of the solution, Implement the solution, Data Interpretation, Compiler description
	Key advantages	Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Object-Oriented (the base language from which Scout extends is C*, which is object-oriented)
	Concrete syntax	Textual
	Tool support	Compilers
	Execution model	The language target specific hardware and GPUs or multi-core architectures
RQ4	*Effectiveness*	*Success evaluated, Productivity gains brought* (Lower cognitive overload, Easier to use - Qualitative)

Table A.18: Selective Embedded Just-In-Time Specialization [21]

		Selective Embedded Just-In-Time Specialization
RQ1	*Type*	General Purpose Languages
RQ2	*Host language*	Knowledge Discovery Toolbox (KDT)
	Application domain	Semantic Graphs
	Purpose	Graph Processing (Implement the solution)
	Key advantages	Performance, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Functional, Object-Oriented
	Concrete syntax	Textual
	Existing tool	Interpreters, Compilers, Tool suite
	Technologies	DSL frameworks (KDT), compBLAS library
	Execution stack	OS (any), Message Passing Middleware (MPI), Libraries (compBLAS)
	Execution model	HPC Libraries (compBLAS), Compiled code for CPU
RQ3	*Users' role*	End-user
	Required knowledge	Languages (Python, C++), Libraries (KDT), Hardware (CPU), Systems (Clusters), Theoretical Background (Graph Algorithms)
RQ4	*Effectiveness*	*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), *Productivity gains brought* (Learnability, Lower cognitive overload, Easier to remember, Expressiveness, Easier to use - Qualitative), *Products' performance gains brought* (Memory Efficiency, Computation Efficiency, Scalability - Quantitative and Evolvability/Maintainability - Qualitative)

Table A.19: SkIE-CL [10]

		SkIE-CL
RQ1	*Type*	General Purpose Languages
RQ2	*Nature*	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	Implement the solution
	Key advantages	Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction), Enables high-level parallel programming using skeletons
	Paradigms	Skeletons are used as basic constructs of coordination language (SkIE-CL)
	Concrete syntax	Textual and Diagrammatic
	Tool support	Compilers, Tool suite, IDE
	Execution stack	OS (Multiple: Linux, ...), Message Passing Middleware (MPI)
	Execution model	Compiled code for CPU
RQ3	*Users' role*	End-user
	Required knowledge	Tools (Visual SkIE), Languages (SkIE-CL), Theoretical Background (Skeletons)
RQ4	*Effectiveness*	*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), *Productivity gains brought* (Learnability, Lower cognitive overload, Easier to use - Qualitative), *Products' performance gains brought* (Evolvability/Maintainability - Qualitative; Scalability - Quantitative)

Table A.20: Swift [22, 32]

		Swift
RQ1	*Type*	General Purpose Languages
RQ2	*Application domain*	Parallel Workflow/Distributed parallel scripting
	Purpose	Implement the solution
	Key advantages	Portability, easiness of configuration, orchestration, usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Functional (application components modelled as side-effect free functions)
	Concrete syntax	Textual
	Existing tool	Interpreters, Tool suite
	Execution stack	OS (Linux), IO architecture (POSIX), Message Passing Middleware (Globus)
	Execution model	Virtual Execution Environment (Cloud), Distributed Middleware (Globus Grid middleware)
RQ3	*Users' role*	End-user
	Required knowledge	Languages (Swift)
RQ4	*Effectiveness*	*Success evaluated, Quantitative comparison performed, Metrics* (Time, Utilization), *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)

Table A.21: Pipeline Composition (PiCo) [28]

		Pipeline Composition (PiCo)
RQ1	*Type*	Domain Specific Languages embedded in General Purpose Languages
RQ2	*Host language*	C++
	Application domain	Big Data Analytics
	Purpose	Formalisation of the solution, Simulation of the solution, Implement the solution, Data Interpretation
	Key advantages	Performance, Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Functional, Object-Oriented
	Concrete syntax	Textual
	Existing tool	Compilers, Tool suite
	Execution stack	OS (PiCo application can be compiled to any target platform supporting a modern C++ compiler)
	Execution model	The language target GPUs or multi-core architectures
RQ3	*Users' role*	End-user
	Required knowledge	Languages (C++), Frameworks (FastFlow), Theoretical Background (Batch and Streaming Applications)
RQ4	*Effectiveness*	*Success 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* (Expressiveness, Easier to use - Qualitative), *Products' performance gains brought* (Memory Efficiency, Computation efficiency, Scalability - Quantitative)

Table A.22: Spark Streaming and Spark SQL [19]

		Spark Streaming and Spark SQL
RQ1	*Type*	Domain Specific Languages embedded in General Purpose Languages
RQ2	*Host language*	Spark applications can be written in Java, Scala, Python, R
	Application domain	Streaming analytics
	Purpose	Simulation of the problem, Implement the solution
	Key advantages	Performance, Portability, Easiness of configuration, Orchestration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Functional (Scala), Object-Oriented (Scala)
	Concrete syntax	Textual
	Existing tool	Compilers
	Execution stack	OS (Linux, MS Windows, macOS), IO architecture (Spark Core), Libraries (MLlib Machine Learning Library)
	Execution model	Distributed Middleware (Hadoop Distributed File System (HDFS), OpenStack Swift,..), the language target GPUs or multi-core architectures
RQ3	*Users' role*	End-user
	Required knowledge	Frameworks (Apache Spark)
RQ4	*Effectiveness*	Presented experimental results for three datasets, *Metrics* (Time), *Products' performance gains brought* (Computation efficiency, scalability - Quantitative)

Table A.23: Weaver [4]

		Weaver
RQ1	*Type*	Domain Specific Languages embedded in General Purpose Languages
RQ2	*Nature*	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	Formalisation of the solution, Implement the solution
	Key advantages	Performance, Portability, Easiness of configuration, Usability (Effectiveness/Efficiency/Satisfaction)
	Paradigms	Functional and Object-Oriented (built on top of Python)
	Concrete syntax	Textual
	Existing tool	Compilers, Tool suite
RQ3	*Users' role*	End-user
	Required knowledge	Languages (Python)
RQ4	*Effectiveness*	*Success 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* (Learnability, Easier to use - Qualitative), *Products' performance gains brought* (Computation efficiency, scalability - Quantitative and Evolvability/Maintainability - Qualitative)

Appendix B. Survey Form

Survey

This survey is carried out within the scope of the article in preparation "***Programming Languages for Data-Intensive HPC Applications: a Systematic Mapping Study***", initiated by **Vasco Amaral** (Univ. Nova de Lisboa) and co-authored by the 20 contributors to the SLR/SMS study during the last 4 years.

For complementation and validation of the literature review results, we would like to compare with the honest estimations of **experts** in data-intensive high-performance computing (that is, **you**). Please help us in collecting a sufficiently large and broad statistical basis for this validation by answering this survey form.

It only takes 2-3 minutes.

Please hand in the paper anonymously.
Many thanks in advance!

1. Were you involved in the SMS?
   O Yes   O No

2. How long have you been working in High-Performance Computing?
   O Not at all   O < 2 years   O 2 to 5 years   O 5 to 10 years   O > 10 years

3. In what areas of science or engineering have you worked?
   (e.g., computer science, bioinformatics, material science, telecommunications …)

   ____

4. Do your High-Performance Computing related activities consist primarily of
   O developing programming support tools, or   O using existing programming tools?

5. How do you rate your level of technical knowledge about languages/frameworks for HPC?
   O Very Poor   O Poor   O Neutral   O Good   O Excellent

6. Which programming languages do you *use* for High-Performance Computing?

   ____

7. What are, in your view, the key advantages of these languages (in relation to the alternatives you know)? (this may include language properties, performance, programmability, etc.)

   ____

   ____

8. What actually made you use these languages?   (if not already covered in 8.)

   ____

9. Which other programming frameworks (e.g., library-based) and tools do you *use* for HPC?

   ____

   ____

10. Which other HPC programming languages / frameworks / tools do you know about (but do not use)?

   ____

   ____

Questionnaire-Paper-CK.pdf

Appendix C. Articles selected in the Study

This section lists the 22 selected papers in the mapping study together with the extra 9 papers resulting from suggestion from experts.

[1] Aldinucci, M., Danelutto, M., Kilpatrick, P., Torquati, M., 2017. Fastflow: High-Level and Efficient Streaming on Multicore. Wiley-Blackwell. chapter 13. pp. 261–280. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119332015.ch13, doi:10.1002/9781119332015.ch13, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119332015.ch13.

[2] Badia, R.M., Conejero, J., Diaz, C., Ejarque, J., Lezzi, D., Lordan, F., Ramon-Cortes, C., Sirvent, R., 2015. Comp superscalar, an interoperable programming framework. SoftwareX 3, 32–36. doi:https://doi.org/10.1016/j.softx.2015.10.004.

[3] Binotto, A.P.D., Wehrmeister, M.A., Kuijper, A., Pereira, C.E., 2013. Sm@rtconfig: A context-aware runtime and tuning system using an aspect-oriented approach for data intensive engineering applications. Control Engineering Practice 21, 204–217. doi:https://doi.org/10.1016/j.conengprac.2012.10.001.

[4] Bui, P., Yu, L., Thrasher, A., Carmichael, R., Lanc, I., Donnelly, P., Thain, D., 2011. Scripting distributed scientific workflows using weaver. Concurrency and Computation: Practice and Experience 24, 1685–1707. doi:https://doi.org/10.1002/cpe.1871.

[5] Caballero, R., Stuckey, P.J., Tenorio-Fornes, A., 2015. Two type extensions for the constraint modeling language minizinc. Science of Computer Programming 111, 156–189. doi:https://doi.org/10.1016/j.scico.2015.04.007.

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

[7] Caruana, G., Li, M., Liu, Y., 2013. An ontology enhanced parallel svm for scalable spam filter training. Neurocomputing 108, 45–57. doi:https://doi.org/10.1016/j.neucom.2012.12.001.

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

[9] Coetzee, P., Leeke, M., Jarvis, S., 2014. Towards unified secure on-and off-line analytics at scale. Parallel Computing 40, 738–753. doi:https://doi.org/10.1016/j.parco.2014.07.004.

[10] Coppola, M., Vanneschi, M., 2002. High-performance data mining with skeleton-based structured parallel programming. Parallel Computing 28, 793–813. doi:https://doi.org/10.1016/S0167-8191(02)00095-9.

[11] DeVito, Z., Joubert, N., Palacios, F., Oakley, S., Medina, M., Barrientos, M., Elsen, E., Ham, F., Aiken, A., Duraisamy, K., et al., 2011. 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. p. 9. doi:https://doi.org/10.1145/2063384.2063396.

[12] Enmyren, J., Kessler, C.W., 2010. 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. pp. 5–14. doi:https://doi.org/10.1145/1863482.1863487.

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

[14] Hinsen, K., Langtangen, H.P., Skavhaug, O., Ødegård, Å., 2006. Using b sp and python to simplify parallel programming. Future Generation Computer Systems 22, 123–157. doi:https://doi.org/10.1016/j.future.2003.09.003.

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

[16] Kim, M., Lee, Y., Park, H.H., Hahn, S.J., Lee, C.G., 2015. Computational fluid dynamics simulation based on hadoop ecosystem and heterogeneous computing. Computers & Fluids 115, 1–10. doi:https://doi.org/10.1016/j.compfluid.2015.03.021.

[17] Koning, R., Grosso, P., de Laat, C., 2011. Using ontologies for resource description in the cinegrid exchange. Future Generation Computer Systems 27, 960–965. doi:https://doi.org/10.1016/j.future.2010.11.027.

[18] Liang, T.Y., Li, H.F., Lin, Y.J., Chen, B.S., 2016. A distributed ptx virtual machine on hybrid cpu/gpu clusters. Journal of Systems Architecture 62, 63–77. doi:https://doi.org/10.1016/j.sysarc.2015.10.003.

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

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

[21] Lugowski, A., Kamil, S., Buluç, A., Williams, S., Duriakova, E., Oliker, L., Fox, A., Gilbert, J.R., 2015. Parallel processing of filtered queries in attributed semantic graphs. Journal of Parallel and Distributed Computing 79, 115–131. doi:https://doi.org/10.1016/j.jpdc.2014.08.010.

[22] Maheshwari, K., Jung, E.S., Meng, J., Morozov, V., Vishwanath, V., Kettimuthu, R., 2016. Workflow performance improvement using model-based scheduling over multiple clusters and clouds. Future Generation Computer Systems 54, 206–218. doi:https://doi.org/10.1016/j.future.2015.03.017.

[23] Mateos, C., Zunino, A., Campo, M., 2010. An approach for non-intrusively adding malleable fork/join parallelism into ordinary javabean compliant applications. Computer Languages, Systems & Structures 36, 288–315. doi:https://doi.org/10.1016/j.cl.2009.12.003.

[24] Mateos, C., Zunino, A., Hirsch, M., Fernández, M., Campo, M., 2011. A software tool for semi-automatic gridification of resource-intensive java bytecodes and its application to ray tracing and sequence alignment. Advances in Engineering Software 42, 172–186. doi:https://doi.org/10.1016/j.advengsoft.2011.02.003.

[25] McCormick, P., Inman, J., Ahrens, J., Mohd-Yusof, J., Roth, G., Cummins, S., 2007. Scout: a data-parallel programming language for graphics processors. Parallel Computing 33, 648–662. doi:https://doi.org/10.1016/j.parco.2007.09.001.

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

[27] Mencagli, G., Torquati, M., Lucattini, F., Cuomo, S., Aldinucci, M., 2017. Harnessing sliding-window execution semantics for parallel stream processing. Journal of Parallel and Distributed Computing doi:http://dx.doi.org/10.1016/j.jpdc.2017.10.021.

[28] Misale, C., Drocco, M., Tremblay, G., Martinelli, A.R., Aldinucci, M., 2018. Pico: High-performance data analytics pipelines in modern c++. Future Generation Computer Systems 87, 392–403. doi:https://doi.org/10.1016/j.future.2018.05.030.

[29] Obrecht, C., Kuznik, F., Tourancheau, B., Roux, J.J., 2012. The thelma project: A thermal lattice boltzmann solver for the gpu. Computers & Fluids 54, 118–126. doi:https://doi.org/10.1016/j.compfluid.2011.10.011.

[30] Sengupta, D., Song, S.L., Agarwal, K., Schwan, K., 2015. Graphreduce: processing large-scale graphs on accelerator-based systems, in: High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for, IEEE. pp. 1–12. doi:https://doi.org/10.1145/2807591.2807655.

[31] Turek, W., Stypka, J., Krzywicki, D., Anielski, P., Pietak, K., Byrski, A., Kisiel-Dorohinicki, M., 2016. Highly scalable erlang framework for agent-based metaheuristic computing. Journal of Computational Science 17, 234–248. doi:https://doi.org/10.1016/j.jocs.2016.03.003.

[32] Wilde, M., Hategan, M., Wozniak, J.M., Clifford, B., Katz, D.S., Foster, I., 2011. Swift: A language for distributed parallel scripting. Parallel Computing 37, 633–652. doi:https://doi.org/10.1016/j.parco.2011.05.005.