Towards a Next-Generation Parallel Particle-Mesh Language[*]

Sven Karol[1], Pietro Incardona[2,3], Yaser Afshar[2,3], Ivo F. Sbalzarini[2,3], and Jeronimo Castrillon[1]

1 **Chair for Compiler Construction, Center for Advancing Electronics Dresden, TU Dresden, Dresden, Germany**
`[sven.karol|jeronimo.castrillon]@tu-dresden.de`

2 **MOSAIC Group, Chair of Scientific Computing for Systems Biology, Faculty of Computer Science, TU Dresden, Dresden, Germany**

3 **Center for Systems Biology Dresden, Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany**
`[afshar|incardon|ivos]@mpi-cbg.de`

Abstract

We present our previous and current work on the parallel particle-mesh language PPML—a DSL for parallel numerical simulations using particle methods and hybrid particle-mesh methods in scientific computing.

1 Introduction

During the past years, domain-specific languages (DSLs) gained central importance in scientific high-performance computing (HPC). This is due to the trend towards HPC clusters with heterogeneous hardware—today, mainly using multi-core CPUs as well as streaming processors such as GPUs—in the future, using many-core CPUs, and potentially also reconfigurable processors or data-flow processing units. Writing programs for these machines is a challenging and time-consuming task for scientific programers, who do not only need to develop efficient parallel algorithms for the specific problem at hand, but also need to tune their implementations in order to take advantage of the cluster's hardware performance. This does not only require experience in parallel programming, e.g. using OpenMP, OpenCL, or MPI, but also in computer architectures and numerical simulation methods, leading to the so-called "knowledge gap" in program efficiency [12]. Besides, it renders the simulation codes hardly portable. DSLs can help two-fold: First, they allow scientific programmers to write programs using abstractions closer to the original mathematical representation, e.g., partial differential equations. Second, they transparently encapsulate hardware-specific knowledge.

In the proposed talk, we focus on the parallel particle-mesh language (PPML) [3]. This language provides a macro-based frontend to the underlying PPM library [13, 2] as a parallel run-time system. We analyze PPML's implementation as well as its advantages and disadvantages w.r.t. state-of-the-art DSL implementation techniques. Based on this analysis, we discuss our early efforts in realizing the next version of PPML (Next-PPML) in conjunction with a redesign of the PPM library in C++.

[*] This work is partially supported by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics Dresden".

2 Particle and Mesh Abstractions

In scientific computing, discrete models are naturally simulated using particles that directly represent the discrete entities of the model, such as atoms in a molecular-dynamics simulation or cars in a traffic simulation. These particles carry properties and interact with each other in order to determine the evolution of these properties and of their spatial location. But also continuous models, written as partial differential equations, can be simulated using particles. In this case, the particle interactions discretize differential operators, such as in the DC-PSE method [14]. This is often combined and complemented with mesh-based discretizations, such as the finite-difference method. Mesh and particle discretizations are equivalent in that they approximate the simulated system by a finite number of discrete degrees of freedom that are the particles or the mesh cells. When using particles together with meshes, it is sufficient to consider regular Cartesian (i.e., checkerboard) meshes, since all irregular and sub-grid phenomena are represented on the particles, which can arbitrarily distribute in the domain.

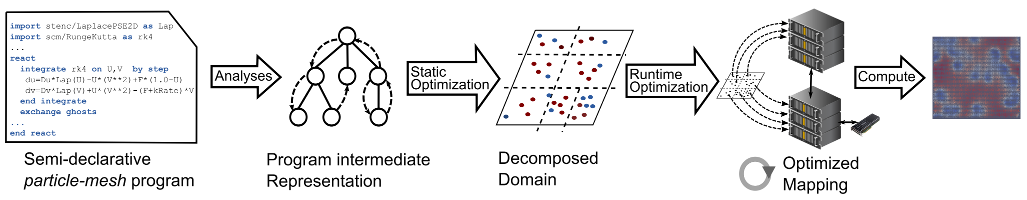
Particles and meshes hence define data abstractions. A particle is a point abstraction that associates a location in space with arbitrary properties, like color, age, or the value of a continuous field at that location. These properties, as well as the particle locations, are updated at discrete time steps over the simulation period by computing interactions with surrounding particles within a given cut-off radius. Meshes are topological abstractions with defined neighborhood relations between cells. The properties are stored either on the mesh nodes or the mesh cells. The PPM library supports both types of abstractions, and also provides conversion operators between them (i.e., particle-mesh interpolation).

3 Current Status of PPM(L)

Currently, PPM is implemented in object-oriented Fortran2003 [13, 2] and PPML is a macro system embedded in Fortran2003 [3]. PPML and PPM support transparently distributed mesh and particle abstractions, as well as parallel operations over them. This also includes properties and iterators. Different domain-decomposition algorithms allow for the automatic distribution of data over the nodes of an HPC cluster. Assigning data and work to processing elements is automatically done by a graph partitioning algorithm, and communication between processing elements in transparently handled by PPM "mappings". The mathematical equations of the model to be simulated are written in LaTeX-like math notation with additional support for differential operators and dedicated integration loops [3].

Syntactically, PPML is an extension of Fortran2003 providing the aforementioned abstractions as domain-specific language concepts. Technically, the language is implemented as a source-to-source transformation relying on a mixture of macro preprocessing steps where macro calls are interspersed with standard Fortran2003 code. Besides C-style preprocessor directives, PPML also supports non-local macros. These are implemented in Ruby using eRuby as a macro language and the ANTLR parser generator for recognizing macro output locations, such as integration loops. Hence, PPML is partially realized using an island grammar [11].

In this preliminary form, PPML has already nicely demonstrated the benefits of embedded DSLs for scientific HPC. It has reduced both the size and the development time of scientific simulation codes by orders of magnitude [3]. It hides much of the parallelization intricacies (PPML automatically generates MPI) from scientific programmers without preventing them from using all features of the underlying programming language. The latter is essential since a DSL may not cover all potential corner cases, and may not always deliver top performance. However, the current light-weight implementation of PPML has severe disadvantages when



```
import stenc/LaplacePSE2D as Lap
import scm/RungeKutta as rk4
...
react
  integrate rk4 on U,V  by step
   du=Du*Lap(U)-U*(V**2)+F*(1.0-U)
   dv=Dv*Lap(V)+U*(V**2)-(F+kRate)*V
  end integrate
  exchange ghosts
...
end react
```

Figure 1 Compiler and grammarware-based language processing chain of Next-PPML.

it comes to code analysis algorithms targeting the whole program and domain-specific optimizations based thereon. Moreover, PPML programs are difficult to debug due to a lack of semantic error messages. We hence present our intended improvements addressing these issues in Next-PPML.

4 Approach to Next-PPML

Next-PPML is a language extension using grammarware and compilerware. This allows us to analyze larger portions of the program code. Examples such as the universal form language (UFL) [1] for finite-element meshes, the Liszt language for mesh-based solvers [7], and the Blitz++ [15] stencil template library have shown that domain-specific analyses and built-in abstractions are beneficial for scientific computing DSLs. Hence, similar concepts will be considered in the Next-PPML language.

Figure 1 conceptually illustrates the planned tool chain. First, the embedded DSL program is parsed to an AST-based intermediate representation. This representation already contains control-flow edges. After computing domain-specific static optimizations on this intermediate representation, including optimizations to the communication pattern of the parallel program, the Next-PPML compiler generates an executable (or source code) which is then used to run the simulation on a parallel HPC cluster. During the simulation run, the application continuously self-optimizes, e.g., for dynamic load balancing. While static optimizations are handled by the DSL compiler, dynamic runtime optimization are handled by the PPM library, which may rely on information provided by the DSL program.

Ideally, the new language uses a declarative approach that bases on an existing programming-language grammar and extends it with new productions. Some well-known candidates for this are Stratego/XT [4], TXL [6], JastAdd [8] or EMFText [9]. However, the target language is C++11 which has no simple declarative specification. Hence, it is difficult to estimate if the above-mentioned tools would scale, and implementing a C++ frontend is a huge project on its own. Therefore, we prefer Clang [10] as an implementation framework, which already provides built-in analyses that can be adopted and extended.

5 Conclusions

Hybrid particle-mesh simulations are the only scientific computing framework that is able to simulate models of all four kingdoms: continuous/deterministic, continuous/stochastic, discrete/deterministic, and discrete/stochastic. This versatility makes the hybrid particle-mesh paradigm a prime target for a generic parallel HPC DSL for scientific computing. Prior work has shown the power of parallelization middleware libraries like PPM, and embedded DSLs like PPML. Over the past 10 years, they have reduced code development times for parallel scientific simulations from years to hours, and enabled unprecedented scalability on large HPC machines [5]. The envisioned Next-PPML will address current shortcomings in code generation, static and dynamic optimization, and semantic error checking and reporting. It is co-developed with the next generation of the PPM library in C++ using a semi-declarative language design.

References

[1] Martin Sandve Alnæs et al. “Unified Form Language: A domain-specific language for weak formulations of partial differential equations.” In: *CoRR* abs/1211.4047 (2012).

[2] Omar Awile, Ömer Demirel, and Ivo F. Sbalzarini. “Toward an Object-Oriented Core of the PPM Library.” In: *Proc. ICNAAM, Numerical Analysis and Applied Mathematics, International Conference.* AIP, 2010, pp. 1313–1316.

[3] Omar Awile et al. “A domainspecific programming language for particle simulations on distributed-memory parallel computers.” In: *Proceedings of the 3rd International Conference on Particle-based Methods.* 2013.

[4] Martin Bravenboer et al. “Stratego/XT 0.17. A Language and Toolset for Program Transformation.” In: *Science of Computer Programming* 72.1-2 (2008): *Second Issue of Experimental Software and Toolkits (EST)*, pp. 52–70. ISSN: 0167-6423.

[5] Philippe Chatelain et al. “Billion Vortex Particle Direct Numerical Simulations of Aircraft Wakes.” In: *Comput. Method. Appl. Mech. Engrg.* 197 (2008), pp. 1296–1304.

[6] James R. Cordy. “The TXL Source Transformation Language.” In: *Science of Computer Programming* 61.3 (2006): *Special Issue on The Fourth Workshop on Language Descriptions, Tools, and Applications (LDTA ’04)*, pp. 190–210. ISSN: 0167-6423.

[7] Zachary DeVito et al. “Liszt: a domain specific language for building portable mesh-based PDE solvers.” In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis.* ACM. 2011, p. 9.

[8] Torbjörn Ekman and Görel Hedin. “The JastAdd System—Modular Extensible Compiler Construction.” In: *Science of Computer Programming* 69.1-3 (2007): *Special Issue on Experimental Software and Toolkits*, pp. 14–26. ISSN: 0167-6423.

[9] Florian Heidenreich et al. “Model-Based Language Engineering with EMFText.” In: *GTTSE IV.* Vol. 7680. LNCS. Springer, 2013, pp. 322–345. ISBN: 978-3-642-35991-0.

[10] Chris Lattner. “LLVM and Clang: Next generation compiler technology.” The BSD Conference. 2008.

[11] Leon Moonen. “Generating Robust Parsers Using Island Grammars.” In: *Proceedings of the Eighth Working Conference on Reverse Engineering 2001.* Los Alamitos, CA, USA: IEEE Computer Society, 2001, pp. 13–22. ISBN: 0-7695-1303-4.

[12] I. F. Sbalzarini. “Abstractions and middleware for petascale computing and beyond.” In: *Intl. J. Distr. Systems & Technol.* 1(2) (2010), pp. 40–56.

[13] I.F. Sbalzarini et al. “PPM – A highly efficient parallel particle–mesh library for the simulation of continuum systems.” In: *Journal of Computational Physics* 215.2 (2006), pp. 566–588. ISSN: 00219991.

[14] Birte Schrader, Sylvain Reboux, and Ivo F. Sbalzarini. “Discretization Correction of General Integral PSE Operators in Particle Methods.” In: *J. Comput. Phys.* 229 (2010), pp. 4159–4182.

[15] Todd L. Veldhuizen. “Blitz++: The Library that Thinks it is a Compiler.” In: *Advances in Software Tools for Scientific Computing.* Ed. by Hans Petter Langtangen, Are Magnus Bruaset, and Ewald Quak. Lecture Notes in Computational Science and Engineering 10. Berlin/Heidelberg: Springer, 2000, pp. 57–87. ISBN: 978-3-642-57172-5.