Problem Solving Environment and Compiler Optimizations for High Performance Particle-Mesh Numerical Simulations

Nesrine Khouzami
TU Dresden
Chair for Compiler Construction
Dresden, Germany
nesrine.khouzami@tu-dresden.de

Jeronimo Castrillon
TU Dresden
Chair for Compiler Construction
Dresden, Germany
jeronimo.castrillon@tu-dresden.de

***Abstract*—We present OpenPME (Open Particle-Mesh Environment), a Problem Solving Environment (PSE) which provides a Domain Specific Language (DSL) built atop a domain model general enough to write numerical simulations in scientific computing using particle-mesh abstractions. This helps to close the productivity gap in HPC applications and effectively lowers the programming barrier to enable the smooth implementation of scalable simulations. We also introduce a model-based autotuning approach of discretization methods for OpenPME compiler. We evaluate the autotuner in two diffusion test cases and the results show that we consistently find configurations that outperform those found by state-of-the-art general-purpose autotuners.**

***Index Terms*—Numerical simulations, Particle-mesh methods, Domain specific compilers, Performance models, Autotuning**

I. Introduction

Computer simulations become today an essential tool to predict the behavior of real-world and physical systems too complex to estimate with analytical solutions. These simulations are usually large-scale programs performed in High-performance computing (HPC) systems. However, the complexity of HPC environments and their programming models increasingly limit the implementation efficiency for computational scientists. Mitigating the entry barrier to HPC field to make the most of its resources has become a main research focus. Problem Solving Environments (PSE) [1], consisted of a Domain Specific Language (DSL) and an Integrated Development Environments (IDE), are typical examples that help to alleviate the programming challenges by providing higher-level domain-specific abstractions close to scientists jargon.

In the field of numerical simulations, particle methods are universal enough to enable the simulation of different types of models ranging from discrete, where particles represent entities in the model, to continuous where a fluid is discretized into particles, either stochastically or deterministically. Many libraries and DSLs offering simulations facilities are proposed in literature [2][3]. In this extended abstract, we briefly present the Open Particle-Mesh Environment (OpenPME) [4], a DSL and its IDE for particle, mesh, and hybrid particle-mesh simulations on parallel HPC systems. OpenPME is based on the Open Framework for Particles and Meshes (OpenFPM) [5], an open-source C++ template library for implementing scalable parallel particle-mesh simulations on multi-CPU and multi-GPU computer hardware. OpenFPM implements specific data structures to provide different layers of abstractions guaranteeing transparent memory-layouting and run-time dynamic load-balancing.

To improve the overall performance of the simulations, we leverage the high level abstractions present at OpenPME DSL to propose a compile-time autotuning optimization approach. Continuous fields in time and space are represented in the DSL through Partial Differential Equations (PDEs) which can be discretized and numerically solved by numerical methods expressed at the language level such as Smoothed Particle Hydrodynamics (SPH), Particle Strength Exchange (PSE), etc. These discretization methods are configured by parameters like the resolution time and time step size that can be autotuned to collectively reach the target accuracy and improve the runtime of the simulation. We describe briefly how our performance-model-based compiler autotuning approach works and we show how efficient it was comparing to generic autotuning algorithms.

II. OpenPME Problem Solving Environment

A. Motivation

OpenFPM relies heavily on C++ templates to achieve flexibility and high performance. This renders its use sometimes complex for novice programmers who had to deal with cryptic error messages emitted by the C++ compiler. Fig. 1 shows a typical example of an error-prone OpenFPM code to simulate the 3D Navier-Stokes equation of fluid mechanics. Such code makes it beyond the compiler abilities to detect semantic errors caused by frequent misplacing of operators in the formula.

Besides, OpenFPM relies on users to manually place `ghost_get` operation which communicates needed data between processors in a distributed environment. These calls are often forgotten or misplaced which effect the simulation results as well as the whole performance.

```
g_dwp.template get<rhs>(key)[z]=
  fac1*(g_vort.template get<vorticity>(key.move(x,1))[z]+
    g_dwp.template get<rhs>(key)[y]=
      fac1*(g_vort.template get<vorticity>(key.move(x,1))[y]+
        g_dwp.template get<rhs>(key)[x]=
          fac1*(g_vort.template get<vorticity>(key.move(x,1))[x]+
          g_vort.template get<vorticity>(key.move(x,-1))[x])+
          fac2*(g_vort.template get<vorticity>(key.move(y,1))[x]+
          g_vort.template get<vorticity>(key.move(y,-1))[x])+
          fac3*(g_vort.template get<vorticity>(key.move(z,1))[x]+
          g_vort.template get<vorticity>(key.move(z,-1))[x])-
          2.0f*(fac1+fac2+fac3)*
          g_vort.template get<vorticity>(key)[x]+
          fac4*g_vort.template get<vorticity>(key)[x]*
          (g_vel.template get<velocity>(key.move(x,1))[x]-
          g_vel.template get<velocity>(key.move(x,-1))[x])+
          fac5*g_vort.template get<vorticity>(key)[y]*
          (g_vel.template get<velocity>(key.move(y,1))[x]-
          g_vel.template get<velocity>(key.move(y,-1))[x])+
          fac6*g_vort.template get<vorticity>(key)[z]*
          (g_vel.template get<velocity>(key.move(z,1))[x]-
          g_vel.template get<velocity>(key.move(z,-1))[x]);
```

Fig. 1. OpenFPM C++ code snippet to calculate the xyz components of the three-dimensional Navier-Stokes equation.

B. OpenPME Design and Implementation

OpenPME has as goal to provide domain-specific abstractions closer to computational scientists language so that they can write efficient simulation code with less likelihood to deal with hard to debug error messages. As shown in Fig. 2, OpenPME serves as an intermediate layer between user's simulation applications and OpenFPM library.

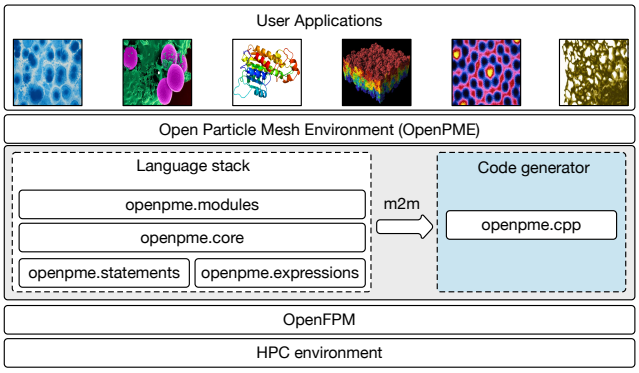

Fig. 2. OpenPME represents an intermediate layer between HPC application domains interfacing OpenFPM for HPC architecture environments.

OpenPME DSL internal modular architecture is designed based on two metamodels representing particle-mesh simulations and the C++ code using OpenFPM. Model-to-model transformations are performed to identify syntactic elements and semantic relations to enable expression rewriting and automatic insertion of communication statements `ghost_get`.

OpenPME is evaluated using three representative use-cases of particle-only (molecular dynamics of Lennard-Jones gas), mesh-ony (Gray-Scott reaction diffusion) and hybrid simulations (incompressible fluid dynamics by solving Navier-Stokdes equations). The results show that the code size written in OpenPME is reduced up to a factor of 7 comparing to C++ OpenFPM code. The generated code performs in general like the hand-written OpenFPM code.

III. Model-based Autotuning of Discretization Methods for OpenPME

We develop an autotuner [6] to be integrated in OpenPME DSL compiler for optimization of numerical discretization methods. Our autotuning approach is based on data-driven regression of performance models which are utilized at compile time to automatically determine the parameters of numerical simulations of continuous spatio-temporal models. The goal is to optimize the trade-off between simulation accuracy and runtime. Fig. 3 shows an excerpt of OpenPME code for the Gray-Scott simulation. The discretization parameters are not defined here and left for autotuning.

```
simulation
...
time loop
 start: 0  stop: 5000
 temporal method: explicit_euler
 spatial method: DC-PSE
  ∂u/∂t = Du * ∇²u - u * v² + F * (1 - u)
  ∂v/∂t = Dv * ∇²v + u * v² - v * (F + k)
```

Fig. 3. Excerpt from OpenPME code for the Gray-Scott simulation.

The autotuner works as follows: it starts by defining an infinite 4D continuous space (depending from 4 parameters of the simulation) which needs to be bounded and discretized. Then, configurations inside the search space are ranked according to an objective by fixing the error and finding the configuration with minimal runtime. The measurements to evaluate a given configuration are taken with regard to a reference best solution. Eventually, we apply a search algorithm that finds high-ranking configurations by measuring only a small subset of all configurations from the search space and avoid measuring slow configurations. This is achieved by leveraging predictive data-driven performance models of the numerical methods.

We conducted experiments on an HPC system of Intel Haswell 24-core nodes connected via an Infiniband network with 40 Gb/s bandwidth. The results show that our autotuner is able to find valid configurations in a very large search space which were orders of magnitude faster (up to 4.2x) than those found by general-purpose autotuners.

IV. Conclusion and Future Work

We presented the overall idea and motivation behind implementing OpenPME, a PSE for particle, mesh and hybrid particle-mesh simulations on parallel and high-performance computers. OpenPME programs are developed at a high-level of abstractions close to the underlying mathematical model. These programs are lowered to OpenFPM C++ through a sequence of compiler model-to-model transformations between two designed models with an automatic injection of communication and synchronization operations. The OpenPME compiler is extended by a performant model-based autotuner capable of determining the best configurations of numerical

discretization schemes when simulating PDEs within a short exploration time meeting the accuracy threshold.

In the future, we consider extending our particle-mesh metamodel by focusing on the communication side at the ghost area to reason about possible compiler optimizations that could be conducted through data-flow model analysis. Also, we plan to create an MLIR [7] dialect to leverage existing tool flows, like for instance towards FPGA acceleration [8].

REFERENCES

[1] E. Gallopoulos, E. Houstis, and J. R. Rice, "Computer as thinker/doer: problem-solving environments for computational science," *IEEE Comput. Sci. Engrg.*, vol. 1, no. 2, pp. 11–23, 1994.

[2] S. Karol, T. Nett, J. Castrillon, and I. F. Sbalzarini, "A domain-specific language and editor for parallel particle methods," *ACM Trans. Math. Softw.*, vol. 44, no. 3, pp. 34:1–34:32, 2018.

[3] O. Awile, M. Mitrovic, S. Reboux, and I. F. Sbalzarini, "A domain-specific programming language for particle simulations on distributed-memory parallel computers," in *Proc. III Intl. Conf. Particle-Based Methods (PARTICLES)*, Stuttgart, Germany, 2013.

[4] N. Khouzami, L. Schütze, P. Incardona, L. Kraatz, T. Subic, J. Castrillon, and I. F. Sbalzarini, "The openpme problem solving environment for numerical simulations," in *Computational Science – ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 614–627.

[5] P. Incardona, A. Leo, Y. Zaluzhnyi, R. Ramaswamy, and I. F. Sbalzarini, "OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers," *Comput. Phys. Commun.*, vol. 241, pp. 155–177, 2019.

[6] N. Khouzami, F. Michel, P. Incardona, J. Castrillon, and I. F. Sbalzarini, "Model-based autotuning of discretization methods in numerical simulations of partial differential equations," *Journal of Computational Science*, vol. 57, p. 101489, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877750321001563

[7] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14.

[8] S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillon, and C. Pilato, "Automatic creation of high-bandwidth memory architectures from domain-specific languages: The case of computational fluid dynamics," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, Sep. 2022. [Online]. Available: https://doi.org/10.1145/3563553