V International Conference on Particle-based Methods – Fundamentals and Applications
PARTICLES 2017
P. Wriggers, M. Bischoff, E. Oñate, D.R.J. Owen, & T. Zohdi (Eds)

A LANGUAGE AND DEVELOPMENT ENVIRONMENT FOR PARALLEL PARTICLE METHODS

Sven Karol[1,4], Tobias Nett[1], Pietro Incardona[2,3], Nesrine Khouzami[1], Jeronimo Castrillon[1], Ivo F. Sbalzarini[2,3]

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

[2] Chair of Scientific Computing for Systems Biology, Faculty of Computer Science, TU Dresden, Dresden, Germany

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

[4] Baselabs GmbH, Ebertstr. 10, 09126 Chemnitz, Germany

Key words: Particles Method, discrete element method, Gray-Scott, Lennard-Jones, PPME, PPML, DSL

Abstract. We present the Parallel Particle-Mesh Environment (PPME), a domain-specific language (DSL) and development environment for numerical simulations using particles and hybrid particle-mesh methods. PPME is the successor of the Parallel Particle-Mesh Language (PPML), a Fortran-based DSL that provides high-level abstractions for the development of distributed-memory particle-mesh simulations. On top of PPML, PPME provides a complete development environment for particle-based simulations usin state-of-the-art language engineering and compiler construction techniques. Relying on a novel domain metamodel and formal type system for particle methods, it enables advanced static code correctness checks at the level of particle abstractions, complementing the low-level analysis of the compiler. Furthermore, PPME adopts Herbie for improving the accuracy of floating-point expressions and supports a convenient high-level mathematical notation for equations and differential operators. For demonstration purposes, we discuss an example from Discrete Element Methods (DEM) using the classic Silbert model to simulate granular flows.

1 INTRODUCTION

Developing applications for scientific high-performance computing (HPC) requires in-depth knowledge of the underlying, potentially heterogeneous hardware and programming

models, as well as the corresponding numerical simulation methods and parallel programming patterns. This leads to a low level of achievable abstraction, which is known to cause the "knowledge gap" in scientific programming [22]. To address this gap, scientific libraries and domain-specific languages (DSLs) have evolved into an important toolset in scientific HPC. In the domain of particle methods, this notably includes the Parallel Particle Mesh library (PPM) [23, 22, 4] and the Parallel Particle Mesh Language (PPML) [3, 5] as a library and a DSL for large-scale scientific HPC. The abstractions in PPM and PPML allow scientific programmers to write more concise and declarative code in comparison to hand-coded implementations. Essentially, it frees developers from the burden of writing boilerplate code that manages parallelism, synchronization, and data distribution. However, PPML has downsides, which we address in PPME [17]: The lightweight embedding of PPML into Fortran, based on language macros, and the lack of a fully integrated language model prevent advanced code analysis and complex compile-time computations. This makes debugging PPML programs hard and prohibits domain-specific static code optimization [11]. In contrast, PPME closely follows the paradigm of language-oriented programming [28], where extensible DSLs are created to describe and solve software problems instead of writing programs in general-purpose languages. This serves to increase maintainability and productivity through domain abstractions. PPME is integrated into the meta programming system (MPS) [8, 16], a model-driven language workbench [9]. We developed a language model that enables us to implement analysis and optimization algorithms that are well-suited for particle methods. The model is the basis of a formal type system for particle simulations, including optional verification of physical dimensions. This enables advanced domain-specific correctness checks at compile time, such as checking for dimensional correctness. PPME further supports numerical accuracy optimization capabilities of floating-point expressions by leveraging domain knowledge and adopting the Herbie accuracy checker [19]. Due to MPS' projectional editing capabilities, convenient high-level mathematical notation for equations and differential operators is supported. In this paper, we present PPME and show its use in Discrete Element Methods (DEM). In particular, we use PPME to implement a simulation of granular flows on distributed-memory parallel computers using a classical Herzian contact force model.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 presents the architecture of PPME and its integration with the PPM Library. Section 4 introduces our case study from the domain of discrete element methods while its implementation in PPME and results are discussed in Section 5. Section 6 concludes the paper and gives an outlook of future work.

2 RELATED WORK

In scientific computing, several DSL-like approaches have successfully been proposed in the past: Blitz++ [25] is a template-based library and DSL for generating stencils from high-level mathematical specifications of mesh-based computations. Freefem++ [10] is a software toolset and DSL for finite-element methods. This DSL allows users to

define analytic or finite-element functions using domain abstractions such as meshes and differential operators. Liszt [7] extends Scala with domain-specific statements for defining solvers for partial differential equations on unstructured meshes with support for several parallel programming models including message passing. The FEniCS project [13] created a finite element library, the unified form language (UFL) [1], and several optimizing compilers for generating code that can be used with the library. Firedrake [20] adds composing abstractions such as parallel loop operations.

Domain-specific optimizations carry great potential since scientific codes often induce specific boundaries on data access and numeric algorithms. This has been particularly studied for representation code of element tensors in the finite-element method [18]. The representation code is written in UFL variational forms. The proposed optimization strategies yield significant runtime speedups and leverage domain knowledge to automate nontrivial optimizations. Loop-level optimizations for finite-element solvers in the COmpiler For Fast Expression Evaluation (COFFEE) [15] are discussed in Ref. [14]. Therein, heuristics are used to predict operation counts at runtime, using well-known transformations such as code motion, expansion, and factorizations. Domain-specific optimizations are superior to general-purpose ones that are used by standard compilers to reduce the operation count in nested loops.

This includes checking and optimizing variables based on their physical dimensions. For this, an analysis technique based on unit annotations has been proposed [6] that does not require extending or changing the base language. Furthermore, unit annotations for linear-algebra and finite-element calculations are available [2], which is comparable to what we have realized for particle methods with dimensional annotations in PPME.

3 THE PARALLEL PARTICLE-MESH ENVIRONMENT (PPME)

PPME is part of the PPM stack and provides several abstractions and analysis algorithms for parallel particle-mesh methods. The PPM library supports simulations of both discrete and continuous models using either particles, meshes, or a combination thereof. In discrete models, particles directly interact by pairwise kernels. In continuous models, differential operators are discretized on particles using, e.g., the SPH or DC-PSE methods [24]. In DC-PSE, the discretized kernels are automatically computed at runtime. As PPM is a Fortran library, clients are required to write plain Fortran code in order to use the library. PPML partially frees the developer from this burden by providing a collection of macros and code transformations that can be used as high-level abstractions. However, using PPML can be problematic as it does not provide error checking and debugging capabilities so that errors are only detected by the compiler in the generated Fortran code and are not related to the PPML program. PPME addresses this problem by analyzing the user code and providing an extensible infrastructure for incorporating domain-specific optimizations.

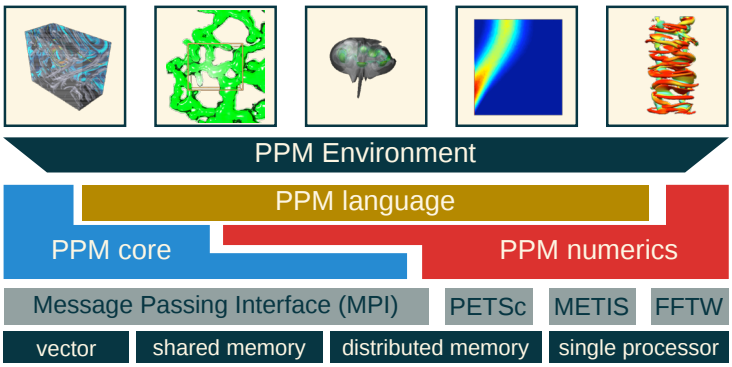


Figure 1: Layered architecture of the PPM/PPME stack.

3.1 Architecture of PPME

Figure 1 provides an overview of the current PPM/PPME stack. The architectural details of the computer hardware are located on the lowest layer and are accessed through common low-level libraries such as MPI for message passing, PETSc for direct and iterative solvers, METIS [12] for graph partitioning, and FFTW for Fourier transforms. In the layer above, the PPM library provides its major subcomponents: *PPM core* contains the distributed Fortran data structures and methods for describing particle simulations, while *PPM numerics* provides frequently used numerical algorithms such as multi-grid solvers, spectral solvers, boundary element methods, and fast multipole methods [21]. These are partly implemented using the objects provided by the PPM core, and partly based on wrapping external libraries. PPML sits on top of PPM. Based on its abstractions, mixed with plain Fortran for direct library access, PPML provides a high-level domain-specific language that is translated to plain Fortran code code that links against the PPM Library, as well as the underlying libraries [3, 5]. Offering a consistent DSL layer, PPME resides in the top-most layer and does not require any adaptation of the underlying framework. Thus, it uses PPML as an intermediate representation, preserving its concepts and abstractions so that the original tool chain remains intact. However, PPME allows scientists to bypass the details of the PPML programming languages and the problem associated with debugging or optimizing PPML programs. PPME is generic to all types of particle and particle-mesh methods, such that different "client applications", symbolized by the square boxes on top, can be implemented.

3.2 Internal Structure of PPME

Internally, PPME is organized in language packages as illustrated in Figure 2. These packages correspond to *solutions* in MPS, constituting the domain language model. We briefly describe the packages here. For a more detailed discussion on the model and type system we refer to Ref. [17].

The package `ppme.expressions` provides a domain-independent set of notations for

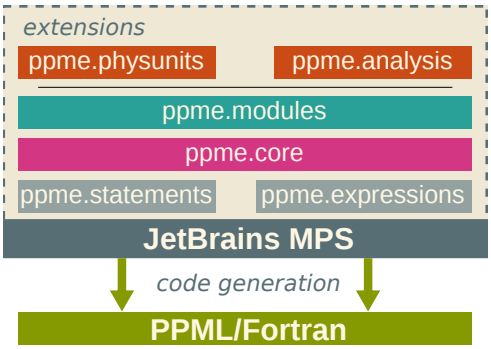


Figure 2: Internal structure of PPME and integration into the MPS language workbench.

mathematical and logical expressions, and literals for integer and floating-point numbers. Moreover, the base types available in PPME and parts of the type system (static analysis) are defined in this package. The package `ppme.statements` contains universal imperatives, such as expression statements, if-else clauses, and loops. The type system is enriched with variable support where necessary. The core package `ppme.core` contains elements that are tailored to particle methods, domain-specific types, expressions, and statements, e.g., a *timeloop* construct. The package `ppme.modules` provides the top-level structure for client programs written in PPME. Modules contain the simulation code and optional control parameters. A module translates to a PPML client that then translates to Fortran code.

A major concept of PPME, and of language-oriented programming in general, is to flexibly add language extensions as needed. So far, we have incorporated two optional extensions: The package `ppme.physunits` enables annotating further meta information to variables and constants. This includes specifications of physical units and dimension which are then accessible to the type system. The package `ppme.analysis` provides an exemplary binding of *Herbie* as an external analysis tool for improving floating-point expressions [19]. This integration is based on a general framework enabling the access of custom tools into the environment. For details on both extensions and application examples, we refer to Ref. [17].

3.3 Code Generation

We illustrate PPME by comparing the input and output of the PPME code generator in a simple example: a Gray-Scott reaction-diffusion simulation. The two versions of the code, PPML and PPME, are juxtaposed in Figure 3. For both, we show the part that integrates the governing equations discretized over the particle set c using the 4th-order Runge-Kutta method ("rk4"). In PPME, the equations are conveniently defined over particle attributes, here the two scalar fields U and V. Static analysis extracts the required information from the code, for example identifying two applications of differential operators, $\nabla^2 c \to U$ and $\nabla^2 c \to U$, and automatically deriving and adding local variables

PPME

```
deqn method "rk4" on c
∂c→U/∂t = constDu * ∇² c→U - c→U * c→V² + F * (1.0 - c→U)
∂c→V/∂t = constDv * ∇² c→V + c→U * c→V² + (F + kRate) * c→V
end deqn
```

PPML

```
rhs grayscott_rhs_0(U=>c, V)
  get_fields(dU, dV)

  dU = apply_op(L, U)
  dV = apply_op(L, V)

  foreach p in particles(c) with positions(x) sca_fields(U, V, dU, dV)
    dU_p = constDu * dU_p - U_p * (V_p**2) + F * (1.0 - U_p)
    dV_p = constDv * dV_p + U_p * (V_p**2) - (F + kRate) * V_p
  end foreach
end rhs
```

Figure 3: Equations of a Gray-Scott reaction-diffusion system in PPME (input) and PPML (output).

dU and dV for them. In the generated PPML code, this information is then contained explicitly, demonstrating some key benefits of a holistic code representation and analysis: since all required information is extracted by the PPME compiler, redundant statements encoding this information are explicitly avoided, which leads to less code, less compile-time errors, and improved readability. Also note PPME's support of basic mathematical notation, such as the Nabla operator ∇ and the partial derivative ∂. From the initial 4 lines of PPME code, 11 lines of PPML code, and more than 100 lines of Fortran code are generated, which corresponds to factor of 25 in code-size reduction (cf. [17]).

4 CASE STUDY: DISCRETE ELEMENT METHODS (DEM)

Discrete element methods are a fundamental tool for the study of granular materials. It has been shown that DEM methods allow determining material properties [32] and effective macroscopic dynamics for which closed-form theory lacks. Therefore, DEM simulations have become invaluable in the search for continuum theoretic descriptions of granular matter [36, 33, 34, 35]. In the processing industry, granular materials are center stage, and DEM simulations are widely used to engineer and optimize production and transport processes. DEM simulations, for example, have been used to understand packing in a cylindrical container [39] for better silo engineering, and to study concrete structures under load in order to predict failure points and weaknesses [29, 30, 31].

Granular materials can be modeled by interacting particles, where each particle is a physical granule or a volume of material [38]. The modeled interaction between the particles defines the microscopic behavior of the material.

A classical model for DEM simulations of spherical granular flows is the Silbert model [37]. It includes a Herzian contact force, as well as elastic deformation of the grains. Each particle is represented by the location of its center of mass $\vec{r}_i$ and is characterized by its radius R, mass m, and polar moment of inertia I. Whenever two particles i and j collide or are in contact with each other, the radial elastic contact deformation is given by:

$$\delta_{ij} = 2R - r_{ij}\,, \tag{1}$$

with $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$ the vector connecting the two particle centers and $r_{ij} = \|\vec{r}_{ij}\|_2$ its length. The normal (radial) and tangential components of the relative velocity at the point of contact are given by:

$$\vec{v}_{n_{ij}} = (\vec{v}_{ij} \cdot \vec{n}_{ij})\, \vec{n}_{ij}\,, \tag{2}$$

$$\vec{v}_{t_{ij}} = \vec{v}_{ij} - \vec{v}_{n_{ij}} - R\,(\vec{\omega}_i + \vec{\omega}_j) \times \vec{n}_{ij}\,, \tag{3}$$

where $\vec{n}_{ij} = \vec{r}_{ij}/r_{ij}$ is the unit normal vector connecting the two particle centers, $\vec{\omega}_i$ is the angular velocity of particle i, and $\vec{v}_{ij} = \vec{v}_i - \vec{v}_j$ the relative velocity of the two particles. The tangential elastic displacement $\vec{u}_{t_{ij}}$ is integrated over time for the duration of contact, using an explicit Euler time-stepping scheme:

$$\vec{u}_{t_{ij}} \leftarrow \vec{u}_{t_{ij}} + \vec{v}_{t_{ij}} \delta t \tag{4}$$

with time-step stize δt. This deformation is stored for each particle and for each contact point. For a new contact, the tangential elastic displacement is initialized to zero. Thus for each unique pair of interacting (colliding) particles, the normal and tangential contact forces become:

$$\vec{F}_{n_{ij}} = \sqrt{\frac{\delta_{ij}}{2R}}\ \left(k_n \delta_{ij} \vec{n}_{ij} - \gamma_n m_{\text{eff}} \vec{v}_{n_{ij}}\right)\,, \tag{5}$$

$$\vec{F}_{t_{ij}} = \sqrt{\frac{\delta_{ij}}{2R}}\ \left(-k_t \vec{u}_{t_{ij}} - \gamma_t m_{\text{eff}} \vec{v}_{t_{ij}}\right)\,, \tag{6}$$

where $k_{n,t}$ are the elastic constants in the normal and tangential direction, respectively, and $\gamma_{n,t}$ the corresponding viscoelastic constants. The effective collision mass is given by $m_{\text{eff}} = \frac{m_i m_j}{m_i + m_j}$. In order to enforce Coulomb's law $\|\vec{F}_{t_{ij}}\|_2 < \|\mu \vec{F}_{n_{ij}}\|_2$, the tangential force of each contact point is bounded by the normal force. This is achieved by scaling the tangential force with

$$\vec{F}_{t_{ij}} \leftarrow \vec{F}_{t_{ij}} \frac{\|\mu \vec{F}_{n_{ij}}\|_2}{\|\vec{F}_{t_{ij}}\|_2}. \tag{7}$$

This implies a truncation of the elastic displacement, since the Coulomb limit is reached when two spheres slip against each other without inducing additional deformations. Thus, the deformation is truncated as:

$$\vec{u}_{t_{ij}} = -\frac{1}{k_t}\left(\vec{F}_{t_{ij}} \sqrt{\frac{2R}{\delta_{ij}}} + \gamma_t m_{\text{eff}} \vec{v}_{t_{ij}}\right). \tag{8}$$

Considering that each particle i interacts with all particles j it is in contact with, the total resultant force on particle i is computed by summing the contributions of all contact pairs (i, j). Including also gravity, we obtain the total force on grain i:

$$\vec{F}_i^{\text{tot}} = m\vec{g} + \sum_j \left(\vec{F}_{n_{ij}} + \vec{F}_{t_{ij}}\right), \tag{9}$$

```
1
create topology topo with
  boundry condition:      "ppm_param_bcdef_periodic"
  decomposition:          <no decomposition>
  processor assignment:   <no processor_assignment>
  ghost size:             cutoff + skin

2
{! fields and properties
  property <real, ppm_dim, "velocity", <no prec>, true> v
  property <real, ppm_dim, "force", <no prec>, true> F
  property <real, ppm_dim, "omega", <no prec>, true> omega
  property <real, ppm_dim, "tau", <no prec>, true> tau
  property <real, max_contacts_def, "contacts_def", <no prec>, true> cpd
  property <real, max_contacts, "contacts_ids", <no prec>, true> cpi
  property <real, 1, "n_cp", <no prec>, true> ncp
  property <real, 1, "type", <no prec>, true> tt
  property <real, 1, "gid", <no prec>, true> gid
}

3
v_nij = (v_rel[1] * n_ij[1] + v_rel[2] * n_ij[2] + v_rel[3] * n_ij[3]) * n_ij;

4
v_tij = v_rel - v_nij - v_cross;

5
[F_nij = SQRT(delta_ij/2/R) * (k_n*delta_ij*n_ij(:) - gamma_t*m_eff*v_nij(:))]

6
F_tij = F_tij * (F_nij_sq / F_tij_sq);
```

Figure 4: Excerpts of the PPME code for parallel DEM simulations.

where $\vec{g}$ is the acceleration due to gravity. In the Silbert model, particles also have a rotational degree of freedom. Therefore, the total torque on particle i is calculated as:

$$\vec{T}_i^{\text{tot}} = -R \sum_j \vec{n}_{ij} \times \vec{F}_{t_{ij}} \,. \tag{10}$$

We integrate the equations of motion using the second-order accurate leap frog scheme

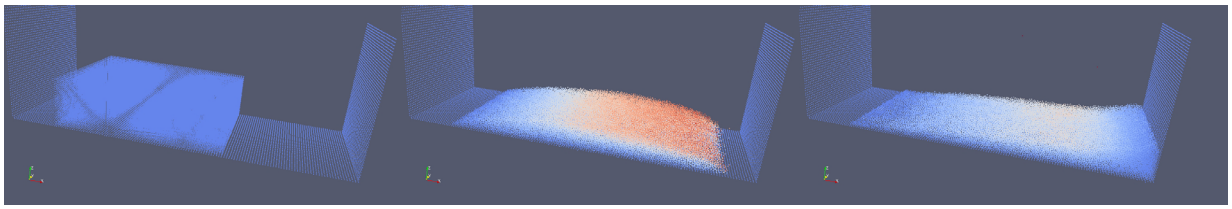
$$\vec{v}_i^{n+1} = \vec{v}_i^n + \frac{\delta t}{m} \vec{F}_i^{\text{tot}} \,, \qquad \vec{r}_i^{n+1} = \vec{r}_i^n + \delta t \vec{v}_i^{n+1} \tag{11}$$

$$\vec{\omega}_i^{n+1} = \vec{\omega}_i^n + \frac{\delta t}{I_i} \vec{T}_i^{\text{tot}} \,, \tag{12}$$

where $\vec{r}_i^n$, $\vec{v}_i^n$, $\vec{\omega}_i^n$ denote respectively the position, velocity, and angular velocity of particle i at time step n, and δt is the time-step size.

5 RESULTS

We describe how the above DEM model is implemented in PPME and present the results of a simulation using 82,300 particles. Figure 4 shows excerpts of the PPME code for the model described in Section 4. The code for setting up the parallel simulation is in section (1). It defines the external boundary conditions on the computational domain, decomposes the problem onto the available processors, and adds ghost (halo) layers around each processor. The particle properties and their datatypes are defined and initialized by the code section (2). Each particle property has a datatype, a dimension, a human-readable name, a numerical precision, a flag stating whether this property is required, and



(a) t = 0 s (b) t = 2.0 s (c) t = 4.0 s

Figure 5: Visualization of PPME simulation results of an avalanche down and inclined plane. The panels show the particles at the indicated times. Color from blue to red indicates the x-component of the velocity of the particles.

a variable name. The remaining code (3-6) shows the implementations of Equations 2, 3, 5 and 7, respectively. Equation 2 is an example of native PPME code, using particle properties. Equation 5 is an example of inline Fortran code, enabling access to functions not supported in PPME.

We illustrate the PPME implementation using a classical test case for granular flow simulations: an avalanche down an inclined plane. The same test case has also previously been implemented in PPM [26, 27], enabling direct comparison of the codes and results. We set up the simulation as described [27] and run it using 82,300 particles with $k_n = 7.849$, $k_t = 2.243$, $\gamma_n = 3.401$, $\mu = 1$, $R = 0.06$. All particle masses are set to $m = 0.001$ and the gravitational acceleration to $g = 9.81$. The size of the box-shaped domain is fixed to $8.4 \times 3.0 \times 3.18$. Initially, the particles are placed on a regular Cartesian grid inside a box of size $4.26 \times 3.06 \times 1.26$. The simulation box is inclined by 30 degrees in the xz plane by appropriately rotating the gravity vector. The y-direction of the domain is periodic. Figure 5 visualizes the simulation results at different time points, showing the avalanche down the inclined plane. Color indicates the x-component of the particle velocity from low (blue) to high (red). The fixed walls on the bottom, left, and right of the domain are modeled by immobile particles of the same kind. The visualizations are done using ParaView, directly reading the VTK files produced by a single PPME "print" statement, illustrating the parallel file I/O amenities of PPME.

6 CONCLUSIONS

We presented PPME, an integrated development environment for parallel particle-mesh simulations. PPME is based on a complete language model for particle methods. Based on the core model, PPME provides an extensible static type system, including physical dimensions. For demonstration purposes, we exemplarily implemented a classical DEM model in PPME and used it to generate executable Fortran code that links against the PPM library and can be executed on distributed-memory computers using message passing. In the future, we will extend PPME to support additional abstractions from the particle-mesh domain. We will improve the internal architecture of PPME by adding

an additional target-language abstraction layer in order to be able to support output languages other than Fortran. This will provide a versatile and standard platform for parallel particle methods across a wide range of applications from discrete to continuum simulations.

ACKNOWLEDGEMENTS

This work was partly supported by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics Dresden".

REFERENCES

[1] Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Trans. Math. Softw.*, 40(2):9:1–9:37, 2014.

[2] Mark A. Austin. Matrix and finite element stack machines for structural engineering computations with units. *Adv. Eng. Softw.*, 37(8):544–559, 2006.

[3] Omar Awile. *A Domain-Specific Language and Scalable Middleware for Particle-Mesh Simulations on Heterogeneous Parallel Computers*. PhD thesis, Diss. ETH No. 20959, ETH Zürich, 2013.

[4] 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*, pages 1313–1316. AIP, 2010.

[5] Omar Awile, Milan Mitrović, Sylvain Reboux, and Ivo F. Sbalzarini. A domain-specific programming language for particle simulations on distributed-memory parallel computers. In *Proc. III Intl. Conf. Particle-based Methods (PARTICLES)*, Particles 2013, pages 436–447, Stuttgart, 2013.

[6] Phil Cook, Colin Fidge, and David Hemer. Well-Measuring Programs. In *Proc. of ASWEC'06*, volume 54, pages 253–261, Sydney, 2006. IEEE.

[7] Zachary DeVito, Niels Joubert, Francisco Palacios, Stephen Oakley, Montserrat Medina, Mike Barrientos, Erich Elsen, Frank Ham, Alex Aiken, Karthik Duraisamy, et al. Liszt: a domain specific language for building portable mesh-based PDE solvers. In *Proc. of SC '11*, page 9. ACM, 2011.

[8] Sergey Dmitriev. Language Oriented Programming: The Next Programming Paradigm. *JetBrains onBoard*, (November), 2004.

[9] Martin Fowler. Language workbenches: The killer-app for domain specific languages?, 2005.

[10] Frederic Hecht. New development in freefem++. *J. of Num. Math.*, 20(3-4):251–266, 2012.

[11] Sven Karol, Pietro Incardona, Yaser Afshar, Ivo F. Sbalzarini, and Jeronimo Castrillon. Towards a Next-Generation Parallel Particle-Mesh Language. In *Proc. of DSLDI'15*, pages 15–18, 2015.

[12] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. 48:96–129, 1998.

[13] Anders Logg, Kent-Andre Mardal, and Garth Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*. Number 84 in LNCSE. Springer, 1 edition, 2012.

[14] Fabio Luporini, David A. Ham, and Paul H. J. Kelly. An algorithm for the optimization of finite element integration loops. Technical Report, arXiv.org, 2016.

[15] Fabio Luporini, Ana Lucia Varbanescu, Florian Rathgeber, Gheorghe-Teodor Bercea, J. Ramanujam, David A. Ham, and Paul H. J. Kelly. Cross-loop optimization of arithmetic intensity for finite element local assembly. *ACM Trans. Archit. Code Optim.*, 11(4):57:1–57:25, 2015.

[16] MPS - 3.2 - Documentation. User's Guide, 2015.

[17] Tobias Nett, Sven Karol, Jeronimo Castrillon, and Ivo F Sbalzarini. A domain-specific language and editor for parallel particle methods. *arXiv preprint arXiv:1704.00032*, 2017.

[18] Kristian B. Ølgaard and Garth N. Wells. Optimizations for quadrature representations of finite element tensors through automated code generation. *ACM Trans. Math. Softw.*, 37(1):8:1–8:23, 2010.

[19] Pavel Panchekha, Alex Sachez-Stern, James R. Wilcox, and Zachary Tatlock. Automatically Improving Accuracy for Floating Point Expressions. In *PLDI'15*, volume 50, pages 1–11. ACM, jun 2015.

[20] Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew TT McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul HJ Kelly. Firedrake: automating the finite element method by composing abstractions. *arXiv preprint arXiv:1501.01809*, 2015.

[21] I. F. Sbalzarini, J. H. Walther, B. Polasek, P. Chatelain, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos. A software framework for the portable parallelization of particle-mesh simulations. *Lect. Notes Comput. Sc.*, 4128:730–739, 2006.

[22] Ivo F. Sbalzarini. Abstractions and middleware for petascale computing and beyond. *Intl. J. Distr. Systems & Technol.*, 1(2):40–56, 2010.

[23] Ivo F. Sbalzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos. PPM - A highly efficient parallel particle-mesh library for the simulation of continuum systems. *Journal of Computational Physics*, 215(2):566–588, 2006.

[24] Birte Schrader, Sylvain Reboux, and Ivo F. Sbalzarini. Discretization correction of general integral PSE operators in particle methods. *J. Comput. Phys.*, 229:4159–4182, 2010.

[25] Todd L. Veldhuizen. Blitz++: The library that thinks it is a compiler. In *Advances in Software Tools for Scientific Computing*, number 10 in LNCSE, pages 57–87. Springer, 2000.

[26] Jens H. Walther and Ivo F. Sbalzarini. Large-scale parallel discrete element simulations of granular flow. In *Proceedings of the International Conference on Discrete Element Modelling (DEM07)*, Brisbane, Australia, 2007.

[27] Jens H. Walther and Ivo F. Sbalzarini. Large-scale parallel discrete element simulations of granular flow. *Engineering Computations*, 26(6):688–697, 2009.

[28] Martin P. Ward. Language Oriented Programming. *Software Concepts and Tools*, 15(4):147–161, 1994.

[29] F. Camborde and C. Mariotti and F.V. Donzé *Numerical study of rock and concrete behaviour by discrete element modelling*. Computers and Geotechnics, Vol. 27., (2000).

[30] Sébastien Hentz, Laurent Daudeville, and Frédéric V. Donzé *Numerical study of rock and concrete behaviour by discrete element modelling*. Computers and Geotechnics, 130(6):709–719, (2004).

[31] W. J. Shiu, F. V. Donzé, and L. Daudeville *Compaction process in concrete during missile impact: a dem analysis*. Computers and Concrete, 5(4):329–342, (2008).

[32] Hunt, M.L. *Discrete element simulations for graular material flows: effective thermal conductivity and self-diffusivity*. International Journal of Heat and Mass Transfer, Vol. 40 No. 13, pp. 3059-68, (1997).

[33] Daerr, A. and Douady, S. *Two types of avalanche behaviour in granular media*. Nature, Vol. 399, pp. 241-3, (1999).

[34] Douady, S., Andreotti, B., Daerr, A. and Cladé *From a grain to avalanches: on the physics of granular surface flows*. Comptes Rendus Physique, Vol. 3, pp. 177-87, (2002).

[35] Dutt, M., Hancock, B., Bentham, C. and Elliot, J. *An implementation of granular dynamics for simulating frictional elastic particles based on the DL_POLY code*. Computer Physics Communications, Vol. 166, pp. 26-44, (2005).

[36] de Gennes, P.G. *Granular matter: a tentative view*. Reviews of Modern Physics, Vol. 71 No. 2, pp. 374-82, (1999).

[37] Silbert, L.E., Grest, G.S. and Plimpton, S.J. *Boundary effects and self-organization in dense granular flows*. Physics of Fluids, Vol. 14 No. 8, pp. 2637-46, (2002).

[38] Stefan Luding *Introduction to discrete element methods* . European Journal of Environmental and Civil Engineering (2008).

[39] Landry, J.W., Grest, G.S., Silbert, L.E. and Plimpton, S.J. *Confined granular packings structure, stress, and forces*. Physical Review E, Vol. 67, p. 041303, (2003).