Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

DORIAN KRAUSE AND KONSTANTIN FACKELDEY AND
ROLF KRAUSE

A parallel multiscale simulation toolbox for coupling molecular dynamics and finite elements

ZIB-Report 13-50 (September 2013)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

A parallel multiscale simulation toolbox for coupling molecular dynamics and finite elements

Dorian Krause*and Konstantin Fackeldey†and Rolf Krause‡

September 6, 2013

Abstract It is the ultimate goal of concurrent multiscale methods to provide computational tools that allow to simulation physical processes with the accuracy of micro-scale and the computational speed of macro-scale models. As a matter of fact, the efficient and scalable implementation of concurrent multiscale methods on clusters and supercomputers is a complicated endeavor. In this article we present the parallel multiscale simulation tool MACI which has been designed for efficient coupling between molecular dynamics and finite element codes. We propose a specification for a thin yet versatile interface for the coupling of molecular dynamics and finite element codes in a modular fashion. Further we discuss the parallelization strategy pursued in MACI, in particular, focusing on the parallel assembly of transfer operators and their efficient execution.

*Institute of Computational Science, Università della Svizzera Italiana, Lugano, Switzerland, dorian.krause@usi.ch

†Zuse Institut Berlin, Berlin, Germany, fackeldey@zib.de

‡Institute of Computational Science, Università della Svizzera Italiana, Lugano, Switzerland,rolf.krause@usi.ch

1 Introduction

The goal of sub-project C6 of the collaborative research center 611 "Singular Phenomena and Scaling in Mathematical Models" at the University of Bonn, Germany, was the development and implementation of novel computational techniques for the concurrent coupling of different physical models in the numerical simulation of solids. In particular, the project was concerned with multiscale coupling between atomistic and continuum models. Such concurrent multiscale approaches can be used, for example, in the numerical simulation of fracture processes. By using a molecular dynamics model to capture the complicated physical processes in the vicinity of the crack tip and a computationally faster but less accurate continuum model for the surrounding material, one can achieve good accuracy at a lower price compared to fully atomistic simulations.

The design of efficient computational tools for such multiscale simulations is itself a challenging task. This is even more so when building parallel simulation tools. In this article we describe the design of the versatile multiscale simulation toolbox MACI and discuss the novel parallelization approach used in MACI. We introduce a thin yet capable interface designed for efficient coupling between molecular dynamics (MD) and finite elements (FE) codes.

1.1 Related work

While the design of algorithms for concurrent multiscale coupling is an active field of research in the past years, relatively few work has been published about implementation and parallelization of these algorithms. Broughton et al. [8] report on a parallel multiscale simulation using the *concurrent coupling of length scales* method. This work is limited to one-dimensional domain decompositions for the molecular dynamics domain. Ma et al. [20] have implemented their MD/GIMP method in the SAMRAI framework. In comparison to most multiscale methods for the coupling of MD and finite elements their constraints are local. Xiao et al. [28] describe a parallel implementation of the Bridging Domain method in a grid environment. However, this work is restricted to one-dimensional simulations. Anciaux et al. [2] have implemented the Bridging domain method in the parallel LIBMULTISCALE. Their approach is closest to our work.

In this article we present a versatile interface for coupling MD and FE codes. The common component architecture (CCA) [3] aims to develop a component model for high performance scientific computing. So far we are not aware of any work using CCA for multiscale coupling between atomistic and continuum models.

It is one of the goals of the European MAPPER project [21] to develop software and services for distributed multiscale simulations. While our work focuses on tightly-coupled simulations on clusters and supercomputers, this work is aimed towards the utilization of distributed resources in the European e-Infrastructure.

1.2 Article contribution and outline

The outline of the article is as follows. In Section 2 we review the multiscale simulation method implemented in MACI, focusing on the computational aspects. In Section 3 we propose and discuss a thin interface allowing for the reusing coupling logic with different molecular dynamics and finite element codes. This work is not limited to the coupling algorithm presented in Section 2 but can be applied to a broad range

of multiscale coupling methods. In Section 4 we discuss the parallelization of MACI focusing on the description of the data and work distribution in the code. In comparison to our previous work [18] the focus of this section is the description of the high-level structure without a detailed discussion of the communication mechanisms employed.

2 Multiscale simulation method

In this section we shortly present atomistic (micro-scale) and continuum (macro-scale) models for the simulation of the behavior of a solid $\Omega \subset \mathbb{R}^3$. We then proceed to discuss an approach to concurrent coupling of these models using projection-based constraints.

2.1 Molecular dynamics

On an atomistic level we can model Ω as a discrete set of N atoms/particles $\mathbb{A} = \{\alpha\}$ with coordinates and momenta $(\vec{x}_\alpha, \vec{p}_\alpha) \in \mathbb{R}^6$. The motion of these particles is governed by the Hamiltonian equations

$$\begin{aligned} \dot{\vec{x}}_\alpha &= \frac{\partial \mathcal{H}}{\partial \vec{p}_\alpha} = \frac{1}{m_\alpha}\vec{p}_\alpha \\ \dot{\vec{p}}_\alpha &= -\frac{\partial \mathcal{H}}{\partial \vec{x}_\alpha} = -\nabla_{\vec{x}_\alpha} V + \vec{F}_\alpha^{\text{ext}} \end{aligned} \tag{1}$$

with the Hamiltonian $\mathcal{H} = K + V$. Here, K denotes the kinetic energy of the atomic system $K = \sum_\alpha \frac{1}{2m_\alpha}\vec{p}_\alpha^2$, V is the interaction potential $V = V(\vec{x}_1, \ldots, \vec{x}_N)$ and m_α the particle mass. In this article, we concentrate on short-ranged potential that allow for efficient (i.e., in linear time) computation of energy and forces using a linked cell method [16] or Verlet neighbor lists [1].

As a particle method, MD does not require discretization in space but only in time. Usually, lower order symplectic integrators (such as a second order Störmer-Verlet scheme) are used for their computational efficiency and good long-term stability properties.

2.2 Continuum mechanics and finite elements

In continuum mechanics, the macroscopic deformation of a body $\Omega \subset \mathbb{R}^3$ is described by a volume preserving mapping $\vec{\varphi} : [0,T] \times \Omega \to \mathbb{R}^3$, such that $\vec{\varphi}(\{t\} \times \Omega)$ equals the configuration of the body at time t. The deformation field $\vec{U} = \varphi - \vec{1}$ is the solution of the variational problem

$$\begin{aligned} \int_\Omega \rho \ddot{\vec{U}} \cdot \vec{V} \, \mathrm{d}\vec{x} = \int_\Omega \rho \vec{b} \cdot \vec{V} \, \mathrm{d}\vec{x} - \int_\Omega \underline{\vec{P}}(\vec{U}) : \vec{\nabla}\vec{V} \, \mathrm{d}\vec{x} + \int_{\Gamma_N} \vec{f} \cdot \vec{V} \, \mathrm{dS} \,, \\ \vec{U} = \vec{U}_D \text{ on } \Gamma_D \quad + \text{initial conditions for } \vec{U} \text{ and } \dot{\vec{U}} \,. \end{aligned} \tag{2}$$

Here, ρ denotes the density in the undeformed configuration, $\vec{b}$ and $\vec{f}$ are external body and surface forces (the latter one applied on $\Gamma_N \subset \partial\Omega$) and $\underline{\vec{P}}$ denotes the first Piola-Kirchhoff tensor. Dirichlet values $\vec{U}_D$ are applied on $\Gamma_D \subset \partial\Omega$. The test function $\vec{V}$ is an element of an appropriate subspace of $C^0\left([0,T]; H^1(\Omega)\right)$.

In this article, we concern ourselves with first-order ($\mathbb{P}1$ or $\mathbb{Q}1$) finite elements for the spatial discretization of (2) resulting in a system of coupled partial differential equations

$$\ddot{\vec{U}}_A = M_A^{-1}\left(\vec{F}_A + \vec{F}_A^{\text{ext}}\right) \text{ for each mesh node } A\,,$$

which has the same structure as Equation (1). Hence, the same temporal discretization methods can be applied.

2.3 Coupling method

The goal of concurrent coupling schemes is to allow for interfacing highly accurate, but expensive, simulation techniques (such as MD) with less accurate, but faster, approximate schemes. For the latter we consider a continuum mechanics model discretized on a finite element mesh of a mesh size that is sufficiently larger than the average atomic distance. In the following we refer to this problem as *MD-FE coupling*.

The challenge in the design of concurrent coupling schemes is implementing appropriate transfer conditions between the *micro-* (MD) and *macro-* (FE) *scales*. Since each scale features a different resolution, not all modes (e.g., pressure waves of high wave number) can be resolved on all scales. The interface conditions need to account for this, in order not to create spurious effects (e.g., wave reflection) that spoil the solution accuracy.

In the following we review the coupling strategy using *projection-based constraints* described in [11, 12].

2.3.1 Coupling with overlap

We consider an overlapping decomposition of the simulation domain Ω into an MD domain Ω_{MD} and an FE domain Ω_{FE} with *handshake region* $\Omega_{\text{H}} = \Omega_{\text{MD}} \cap \Omega_{\text{FE}}$. In Ω_{H}, the micro- and macro-scale coexist. Inspired by the Bridging Domain method [29], volumetric constraints

$$\vec{0} \overset{!}{=} \vec{G}(\vec{u}, \vec{U}) = \vec{O}_1\vec{u} - \vec{O}_2\vec{U} \tag{3}$$

are used in [12] to couple the MD displacement field $\vec{u}$ and the FE displacement field $\vec{U}$. Here, the atomistic displacement field is given by $\vec{u}_\alpha(t) = \vec{x}_\alpha(t) - \vec{x}_\alpha(0)$.

In [12], the operators $\vec{O}_1$ and $\vec{O}_2$ are chosen to be equal to a projection Π from micro- to macro-scale and the identity $\vec{1}$, respectively. The projection Π allows for additively decomposing the micro-scale displacement field $\vec{u}$ into a macro-scale field $\overline{\vec{u}}$ and *high fluctuation* remainder $\vec{u}'$ (cf. [27]):

$$\vec{u} = \overline{\vec{u}} + \vec{u}' = \Pi\vec{u} + \left(\vec{1} - \Pi\right)\vec{u}\,.$$

Note that $\Pi\vec{u}' = \vec{0}$. Hence, the constraints $\vec{G}$ provide a pointwise coupling between $\vec{U}$ and $\overline{\vec{u}}$ while not affecting the high fluctuation field $\vec{u}'$ which is not representable on the macro-scale.

Inspired by non-conforming domain decomposition theory, in [10], an L^2 projection is proposed for micro-to-macro scale transfer. An embedding of the atomistic displacement space $\left(\mathbb{R}^3\right)^N$ into $L^2(\Omega)$ is constructed using scattered-data approximation methods. Hence, given a vector $(\vec{w}_\alpha)_{\alpha\in\mathbb{A}}$ a function $\vec{w}^\sharp$ is constructed such that $\vec{w}^\sharp(\vec{x}_\alpha(0)) \approx \vec{w}_\alpha$. One possible approach for constructing $\vec{w}^\sharp \in \mathbb{X} \subset L^2(\Omega_{\text{H}})$ is

to introduce a set partition of unity basis functions ψ_α (see, for example, [13]) with $\sum_{\alpha\in\mathbb{A}}\psi_\alpha = 1$ and define

$$\vec{w}^\natural = \sum_{\alpha\in\mathbb{A}} \vec{w}_\alpha\,\psi_\alpha\ .$$

Given the embedding of $\left(\mathbb{R}^3\right)^N$ into L^2 we can define the projection $\Pi : \left(\mathbb{R}^3\right)^N \to \mathbb{S}_\mathrm{H}$ by

$$\left(\Pi\vec{u},\vec{V}\right)_\rho = \left(\vec{u}^\natural,\vec{V}\right)_\rho \text{ for all } \vec{V}\in\mathbb{S}_\mathrm{H}\ .$$

Here, $\mathbb{S}_\mathrm{H}$ denotes the first-order finite element space on Ω_H (we assume that Ω_H can be written as the union of a set of elements in the tessellation of Ω_FE) and $(-,-)_\rho$ equals the L^2 scalar product weighted by the continuum mass density ρ.

The assembly of the L^2 projection Π requires the computation of (and quadrature on) the cuts between the elements in the tessellation of Ω_H and the support of the basis functions ψ_α. Even though, this computation needs to be performed only as part of the simulation setup (i.e., not during the time integration), the assembly can be costly. Alternatively, a least-squares projection

$$\Pi\vec{u} = \mathrm{argmin}_{\vec{V}\in\mathbb{S}_\mathrm{H}}\,\frac{1}{2}\sum_\alpha m_\alpha|\vec{u}_\alpha - \vec{V}\left(\vec{x}_\alpha(0)\right)|^2$$

has been discussed in [11].

Let us point out that in either case we can write $\Pi = \tilde{\tilde{M}}^{-1}\vec{T}$ with a mass matrix $\tilde{\tilde{M}}$ and a rectangular matrix $\vec{T}$ and hence we can equivalently use the constraints $\vec{G} = \vec{T}\vec{u} - \tilde{\tilde{M}}\vec{U}$.

The coupled equations of motion for the micro- and macro-scale are derived from a weighted Hamiltonian/Lagrangian (cf., [12, 29]) resulting in a system of algebraic differential equations. We use a RATTLE integration scheme, requiring two linear solves per time step.

2.3.2 Damping high fluctuation modes

The design of the projection-based constraints $\vec{G}$ ensures that the high fluctuation field $\vec{u}'$ is not affected by the constraints, irrespective of the resolution of the finite element mesh. To avoid spurious reflections at $\partial\Omega_\mathrm{MD}$, a modified perfectly matched boundary layer (PML) method is proposed in [12] which (approximately) removes the high fluctuation field and has only minor effect on the information transfer between micro- and macro-scale. To this end, an additional force term

$$\vec{f}_\alpha^\mathrm{PML} = -2d\left(\vec{x}_\alpha(0)\right)\left(\left(\vec{q}\vec{v}\right)_\alpha + d\left(\vec{x}_\alpha(0)\right)\left(\vec{q}\vec{u}\right)_\alpha\right)$$

is added to the MD forces $\vec{f}_\alpha$. Here, $d : \Omega_\mathrm{MD}\to[0,\infty)$ is a scalar function with support in $\overline{\Omega_\mathrm{H}}$ and $\vec{q} = \vec{1} - \vec{N}\Pi$, $\vec{N}$ being the interpolation operator from $\mathbb{S}_\mathrm{H}\to\left(\mathbb{R}^3\right)^N$.

2.3.3 Complete algorithm

In Algorithm 1 the seven most important steps in the RATTLE integration from time t to time $t+\tau$ are explained. As mentioned earlier, two linear systems need to be solved in each timestep to compute the Lagrange multipliers $\vec{\lambda}$ and $\vec{\mu}$. We refer to [12] for the definition of the symmetric positive definite matrix Λ.

Two simulation results for a wave propagation benchmark and mode-I fracture computation using this concurrent coupling technique are shown in Figures 1 and 2.

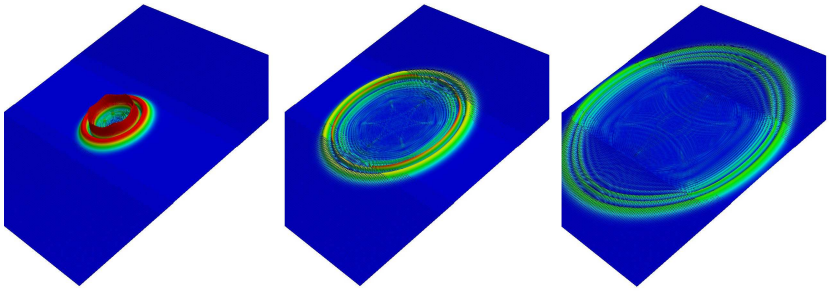
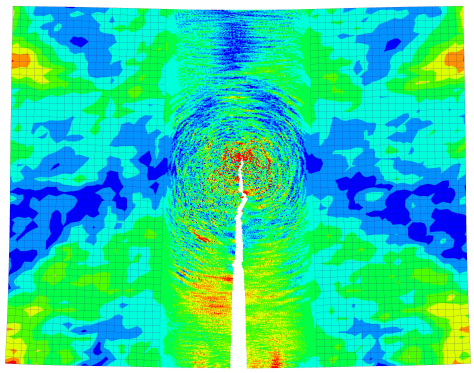
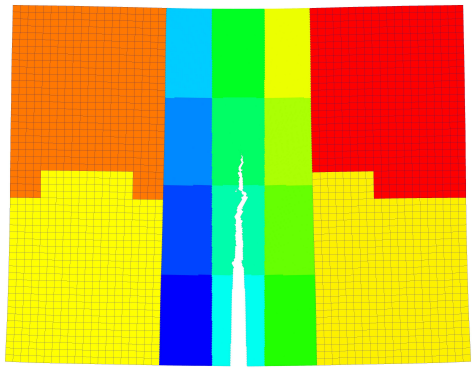


Figure 1: Results of a two-dimensional wave propagation benchmark at the beginning, middle and end of the simulation. A radial wave propagates from Ω_{MD} into Ω_{FE} on the lower and upper side of the MD domain. The elongation in z-direction equals the (scaled) magnitude of the displacement field.



(a) Velocity distribution. The velocities can be seen to fluctuate strongly in Ω_{MD} but to be smooth towards the boundary of the handshake region.



(b) Distribution of atoms and finite elements over $12 + 4$ processing elements.

Figure 2: Results of a mode-I fracture simulation using $2,496$ finite elements and $62,390$ atoms. Surface forces are applied on the left and right boundary of Ω_{FE}.

Algorithm 1 RATTLE time integration scheme

1. Apply standard "Verlet kicks" and "Verlet drifts" to the micro-and macro-scale displacements and velocities yielding trial values $\vec{u}^*$, $\vec{v}^*$, $\vec{U}^*$, $\vec{V}^*$:

$$\begin{bmatrix}\vec{v}^* \\ \vec{V}^*\end{bmatrix} = \begin{bmatrix}\vec{v}^n \\ \vec{V}^n\end{bmatrix} + \frac{\tau}{2}\begin{bmatrix}\vec{m}^{-1}\vec{f}^{n+1} \\ \vec{M}^{-1}\vec{F}^{n+1}\end{bmatrix}, \quad \begin{bmatrix}\vec{u}^* \\ \vec{U}^*\end{bmatrix} = \begin{bmatrix}\vec{u}^n \\ \vec{U}^n\end{bmatrix} + \tau\begin{bmatrix}\vec{v}^* \\ \vec{V}^*\end{bmatrix},$$

   where $\vec{f}^n$, $\vec{F}^n$ denote the forces computed in step 4 of the previous time step.
2. Evaluate the displacement residual $\vec{G}^* = \vec{T}\vec{u}^* - \tilde{\vec{M}}\vec{U}^*$ and solve $\vec{G}^* = \Lambda\vec{\lambda}$ for $\vec{\lambda}$.
3. Correct the trial values

$$\begin{bmatrix}\vec{v}^{n+\frac{1}{2}} \\ \vec{V}^{n+\frac{1}{2}}\end{bmatrix} = \begin{bmatrix}\vec{v}^* \\ \vec{V}^*\end{bmatrix} + \frac{1}{\tau}\begin{bmatrix}\vec{m}^{-1}\vec{T}^{\mathrm{T}}\vec{\lambda} \\ -\vec{M}^{-1}\tilde{\vec{M}}^{\mathrm{T}}\vec{\lambda}\end{bmatrix}, \quad \begin{bmatrix}\vec{u}^{n+1} \\ \vec{U}^{n+1}\end{bmatrix} = \begin{bmatrix}\vec{u}^* \\ \vec{U}^*\end{bmatrix} + \begin{bmatrix}\vec{m}^{-1}\vec{T}^{\mathrm{T}}\vec{\lambda} \\ -\vec{M}^{-1}\tilde{\vec{M}}^{\mathrm{T}}\vec{\lambda}\end{bmatrix}.$$

4. Evaluate forces $\vec{f}^{n+1}$, $\vec{F}^{n+1}$ according to the Hamiltonian equation (without constraints).
5. Add the damping term $\vec{f}^{\mathrm{PML}}$ to the MD force $\vec{f}^{n+1}$.
6. Compute trial velocity values

$$\begin{bmatrix}\vec{v}^* \\ \vec{V}^*\end{bmatrix} = \begin{bmatrix}\vec{v}^{n+\frac{1}{2}} \\ \vec{V}^{n+\frac{1}{2}}\end{bmatrix} + \frac{\tau}{2}\begin{bmatrix}\vec{m}^{-1}\vec{f}^{n+1} \\ \vec{M}^{-1}\vec{F}^{n+1}\end{bmatrix}.$$

7. Evaluate the velocity residual $\dot{\vec{G}}^* = \vec{T}\vec{v}^* - \tilde{\vec{M}}\vec{V}^*$ and solve $\dot{\vec{G}}^* = \Lambda\vec{\mu}$ for $\vec{\mu}$.
8. Correct the velocities

$$\begin{bmatrix}\vec{v}^{n+1} \\ \vec{V}^{n+1}\end{bmatrix} = \begin{bmatrix}\vec{v}^* \\ \vec{V}^*\end{bmatrix} + \begin{bmatrix}\vec{m}^{-1}\vec{T}^{\mathrm{T}}\vec{\mu} \\ -\vec{M}^{-1}\tilde{\vec{M}}^{\mathrm{T}}\vec{\mu}\end{bmatrix}.$$

3 Multiscale simulation toolbox

The development of capable, efficient and scalable molecular dynamics or finite element codes is a complicated and labor-intensive task. Unless the scope of the application is limited, it is therefore often infeasible to develop a multiscale simulation tool as a single monolithic code, that implements MD and FE functionality along with the coupling logic. Instead we focus on reusing existing, established molecular dynamics and finite element implementations, such as TREMOLO [16], LAMMPS [19, 23] and UG [5].

In this article we are concerned with the design and efficient implementation of concurrent coupling codes for MD-FE coupling that allow for reuse of the coupling logic with different implementations of the molecular dynamics and finite element functionality. In comparison to the on-going research on *common component architecture* [3], we are restricting ourselves to the scenario of concurrent MD-FE coupling and expose more details (e.g., about the data distribution) to the coupling code to simplify the development of efficient and scalable code. Additionally we impose some restrictions onto the MD and FE codes that we consider (see below). We have verified that our assumptions are fulfilled by the major molecular dynamics and finite element software packages that are discussed in the literature.

3.1 Interface Design

In this section we propose a simple yet versatile interface for coupling molecular dynamics and finite element simulations. Our work addresses modularity, performance and parallelization. We assume that the FE component is parallelized with a standard domain decomposition approach based on a partition of elements. We do not expose halo or ghost-cells through the presented interface. We moreover assume that the finite element mesh is statically balanced, i.e., that no dynamic load balancing (as used, for example, for adaptive mesh refinement) is performed. For the MD component we also assume a non-overlapping decomposition of the particles, i.e., each particle is stored on exactly one processing element. This assumptions is fulfilled by the majority of the molecular dynamics codes known to the authors, whether they use a domain decomposition (as do most codes such as TREMOLO, LAMMPS, NAMD [22] and DESMOND [7]) or particle-based decomposition (as, e.g., used by DDCMD [26]). Note that the (potentially) dynamic distribution of particles is deliberately exposed to the coupling code for parallel scalability considerations. For further discussions of the parallelization aspects we refer to Section 4.

Our approach is based on the following three pillars:

- Use of opaque handlers for particles, node and elements to hide the details of the data layouts used by the MD and FE components.
- Use of *access epochs* to hide differences in data representation and allow to couple codes working in separate address spaces.
- The use of *piggybacking* to manage metadata in a simple and effective manner.

In the following we elaborate on theses three aspects of the coupling interface.

3.1.1 Opaque Handlers

In general, the data layouts used by different MD or FE codes will dependent strongly on the choice of the algorithms, the scope as well as the intended use case for the application. For example, the data layout used by an MD code based on a linked-cell method will be very different from the data layout used in a code that utilizes Verlet neighbor lists. Similarly, the data layout in an FE code will be different depending on whether the code supports dynamic (e.g., adaptively refined) or only static meshes. To hide these differences, the proposed interface provides opaque handles for the local particles, nodes and elements on a processing element in the form of iterators **`ParticleHandle`**, **`NodeHandle`** and **`ElementHandle`**. These iterators implement increment, comparison and assignment operators.

Since the abstraction of the data layout necessarily incurs a performance penalty, these iterators are intended for use in gather/scatter operations that copy the component data from or to a buffer in a layout suited for the coupling code. Each iterator provides a **`GetLocalId()`** function that can be used to address a contiguous buffer. Moreover, to have a unique local identifier for all mesh nodes in $\overline{\Omega_H}$ we provide **`GetUserChosenId()`** and **`SetUserChosenId()`** functions for **`NodeHandle`** that allow to assign arbitrary (local or global) indices to the mesh nodes (for **`ParticleHandle`** this index can be stored as part of the **`PiggybackType`**, see below). Access to the particle data and dynamic variables (**`ParticleMass`**, **`ParticlePos`**, **`ParticleDispl`**, **`ParticleVel`**, **`ParticleForce`**, **`FeDispl`**, **`FeVel`**, **`FeLumpedMass`**) is possible through static functions taking a **`ParticleHandle`** or **`NodeHandle`** instance as an argument.

Note that these functions are application specific (in this case targeted to coupled simulations of solids) but the approach can be generalized easily. Since the data distribution of the MD component can change dynamically in a parallel simulation, the life time of a **`ParticleHandle`** should be limited to the scope of the coupling routine that created it.

To allow for parallelization of the coupling code we also expose node ownership information (for nodes shared by multiple processing elements) as well as provide **`ParallelSumup`**, **`ParallelMax`** and **`ParallelCopy`** routines to compute the sum (or max) of values stored at duplicated mesh nodes. These functions can be more efficiently implemented by taking advantage of the communication primitives of the FE component.

3.1.2 Access Epochs

MD and FE components do not always work with compatible data representations or with the same reference frame. For example, some MD codes rescale the simulation domain to the unit cube $[0,1]^3$. Hence, all coordinates, velocities and forces need to be scaled before being accessed by the coupling code. Similarly, any updated particle position needs to be rescaled. Moreover, updating the particle positions might require a subsequent exchange of particles that have crossed subdomain boundaries (if the MD component uses a domain decomposition approach).

To cope with these difficulties we propose the use of *access epochs*, which work similar to RMA epochs in the MPI standard [14]. The coupling interface provides subroutines **`AccessBegin(int)`**, **`AccessEnd()`** and **`CanAccess()`**. A call to **`AccessBegin`** starts an access epoch. The bit field passed to **`AccessBegin`** specifies which data fields can be accessed in read, write or read-write mode during the epoch. Access to any data field (via the functions **`ParticlePos`**, **`FeDispl`**, etc.) outside of an access epoch is illegal. An access epoch ends with a call to **`AccessEnd`**. The function **`CanAccess`** allows to check whether access is permitted (in particular for debugging). For example, in Algorithm 1, the third step would be wrapped by calls to **`AccessBegin(VEL_RD | VEL_WR | DISPL_RD | DISPL_WR)`** and **`AccessEnd()`** (note that "**`|`**" is a bit-wise or operation in C allowing to build bitfields from, e.g., **`enum`** variables). Providing detailed information about read and write accesses to the state variables allows the interface code to optimize the actions performed in **`AccessEnd()`**. While it is likely that in a parallel MD code, **`AccessEnd`** needs to trigger an exchange of particles between processing elements after step three, this usually is not required after step six, since in this step only velocities are modified. The calls to **`AccessBegin`** and **`AccessEnd`** are collective, i.e., all MD or FE processing elements need to call these functions in order to achieve progress. The rationale for this decision is that **`AccessEnd`** might require exchange of particles and hence (global) communication.

Beyond a transparent handling of the differences in data representation between MD and FE components, this epoch-based design also permits for coupling codes storing data in a different address space than the one of the coupling code. For example we have successfully coupled MACI with a CUDA MD code. In this case, the coupling code ran on the CPU while the particle data resided in the graphics card memory. In **`AccessBegin`** and **`AccessEnd`** data is copied between CPU and GPU memory. The use of asynchronous copies is possible but would require modifications of the MD code to ensure that the MD code blocks for the completion of the host-to-device copy started in **`AccessEnd`** at the appropriate time.

3.1.3 Piggybacking of Metadata

For the efficient implementation of a concurrent coupling scheme such as Algorithm 1, a set of states need to be maintained for each particle. For example, each particle with $\vec{x}_\alpha(0) \in \overline{\Omega_\mathrm{H}}$ is assigned a local index and needs to store a weight $w \in (0,\infty)$ as well as the value $d\left(\vec{x}_\alpha(0)\right)$. Depending on the algorithm and use case, the amount of data and its structure can vary. In order not to impact the scalability of the coupling code this data should be migrated together with the particles. Hence, it appears impracticable to leave the management of it to the coupling code since particle migration is managed by the MD component. Here, we *piggyback* this data onto the particles and use the communication subroutines of the MD code to exchange it along with the other state of the particle (positions, velocities, etc.). This might require modification of the MD code, for example, to add a `PiggybackType` to the `Particle` structure and to ensure that the additional data is communicated correctly. We have done these modifications in a copy of the TREMOLO code in less than 50 lines of code (mainly to add serialization and de-serialization of `PiggybackType`). For other codes, such as LAMMPS even less modifications may be required since serialization and de-serialization routines can be easily added by defining a new `AtomVec` class.

Note that we do not provide a `PiggybackType` for the FE component because we restricted ourselves to statically balanced meshes. However, the same piggyback technique can used in dynamically balanced finite element simulations.

3.2 Description of the MACI code

We have implemented a new concurrent coupling code MACI (**M**ultiscale **a**tomistic **c**ontinuum **i**nterface) based on the interface defined in the previous section. In this section we shortly describe the architecture of MACI, as depicted in Figure 3.

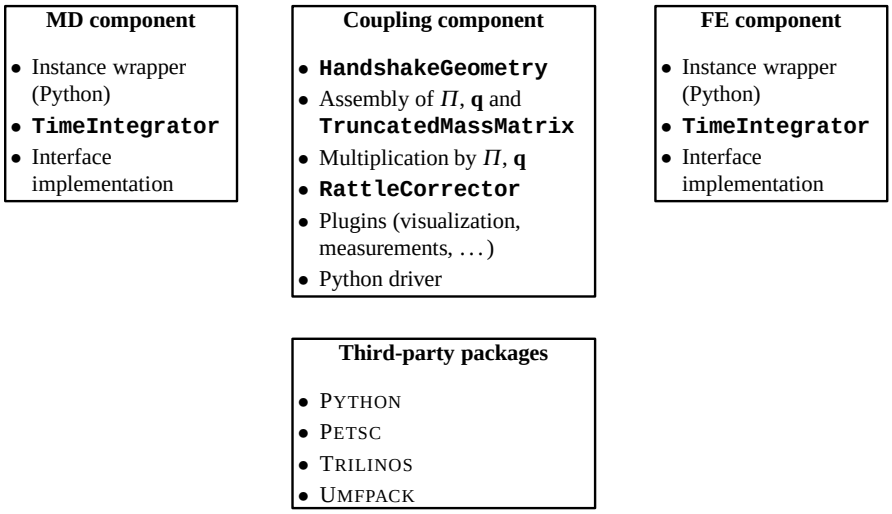


Figure 3: Overview of the architecture of the MACI multiscale simulation tool.

MACI is written in C/C++ for efficiency and portability. Since C, C++ and Fortran are the predominant programming language in high performance computing, this choice allows us to interface to most MD and FE codes without need for additional language translation (for example, via BABEL [25]). MACI is scriptable using the Python

programming language. The translation from C++ to Python is performed using the SWIG tool [6]. It is worth pointing out that while we believe that scripting capabilities are of great advantage for complicated scientific applications like MACI, the use of Python in this project had some inevitable impact on portability (for example, onto earlier Cray massively parallel systems) and complexity (in particular the handling of dynamically shared objects without circular references).

MACI consists of three major components (MD component, FE component, coupling component). The code is designed to run in a SPMD (single program, multiple data) fashion. Each processing element runs the coupling code along with either MD or FE code. Hence, MACI needs to run with at least two processing elements. Each component performs communication using different MPI communicators, effectively shielding the MD and FE code from mutual interference.

The coupling code implements functionality for managing the handshake geometry (allow, for example, to find all particles with $\vec{x}_\alpha(0) \in \overline{\Omega_{\mathrm{H}}}$), for the assembly of the transfer operator Π, the high fluctuation filter $\vec{q}$ and Λ; for computing the Lagrange multipliers λ, μ (cf. Algorithm 1) and the corresponding Lagrange accelerations as well as for the computation of $\vec{f}^{\mathrm{PML}}$. To solve the linear systems arising in the RATTLE integration scheme, MACI can use iterative solvers from the PETSC [4] and TRILINOS [17] packages as well as the direct solver packages UMFPACK [9] (if the handshake region $\overline{\Omega_{\mathrm{H}}}$ is not distributed over multiple FE processing elements).

4 Parallelization aspects

Molecular dynamics and finite element workloads each are well parallelizable and highly scalable implementations do exist. To allow for the treatment of interesting problem sizes using concurrent multiscale methods, their parallelization is of high interest. Unfortunately, the coupling of two scalable codes is *not* readily scalable. In fact, the parallelization of the coupled code introduces several challenges related to the data and work distribution and load balancing. In this section we describe how these challenges are approached in MACI.

4.1 Challenges

Finite elements codes are usually parallelized using an element-wise partitioning of the computational mesh (computed, for example, via graph partitioning algorithms). As mentioned earlier, we restrict ourselves to statically balanced meshes in which this *domain decomposition* is kept fixed over the course of the (time-dependent) simulation.

In contrast to this fixed partition, molecular dynamics codes that support short-ranged interactions usually feature dynamically balanced load since the pair interaction lists (i.e., the set of tuples (α,β) of particles that interact with each others) depends on the current particle positions. Hence, to achieve maximum locality in the expensive force evaluation, particles are migrated between processing elements. One common scheme (found, e.g., in LAMMPS, TREMOLO, NAMD and DESMOND) is to statically decompose the simulation box $B = \bigcup_p B_p$ into subdomains B_p (one for each processing element) and to assign particles to processing element p if $\vec{x}_\alpha(t) \in B_p$. Hence, if a particle crosses a subdomain boundary it is assigned to a different processing element.

In the context of our concurrent coupling strategy the dynamic data distribution of particles is a challenge since our displacement-based constraints (3) are *non-local*. In

fact, we have $\vec{T}_{A\alpha} \neq 0$ if and only if $\mathrm{meas}(\mathrm{supp}\,\psi_\alpha \cap \mathrm{supp}\,\theta_A) > 0$, where θ_A is the nodal basis function with $\theta_A(\vec{x}_A) = 1$. Since $\mathrm{supp}\,\psi_\alpha$ is a polygon or sphere centered at the initial particle position $\vec{x}_\alpha(0)$ we can have $\vec{T}_{A\alpha} \neq 0$ even if $|\vec{x}_\alpha(t) - \vec{x}_A|$ is very large, cf. Figure 4. This implies that the *communication graph* (i.e., the graph with processing elements as nodes and edges between pairs of nodes that exchange messages) is dynamically changing. Thus a scalable implementation of the multiplication by the matrices $\vec{T}$ (the *scale transfer*) and $\vec{q}$ is much complicated compared to "classical" parallel sparse linear algebra, cf. [18].

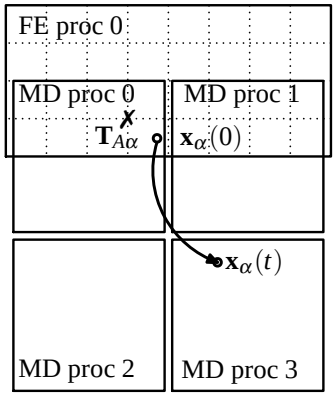
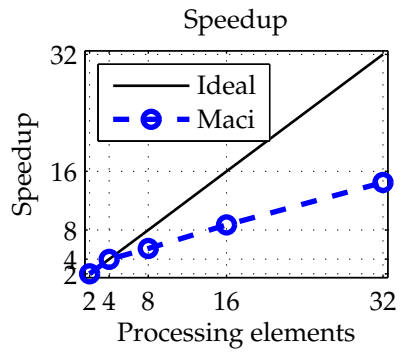


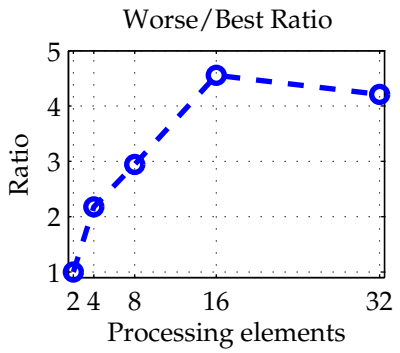
Figure 4: The challenge of dynamic particle distribution in parallel concurrent MD-FE coupling. Particle migration introduces new edges in the communication graph.

Additionally, parallel concurrent coupling introduces novel challenges for load balancing. Much research has been devoted to devising and implementing good load balancing schemes for MD and FE algorithms (and hence for Steps 1, 3, 4, 6 and 8 in Algorithm 1). However, Steps 2, 5 and 7 introduce additional load on a subset of processing elements. For example, the matrix Λ is of size $L \times L$ where L is the number of mesh nodes in $\overline{\Omega_{\mathrm{H}}}$. In practice L is much smaller than the total number of mesh nodes or particles and hence the (iterative) solver for $\Lambda\lambda = \vec{G}^*$ usually does not scale well enough to distribute this task over all processing elements. Instead only a subset (e.g., all the FE processing elements that own cells intersection the handshake region) will be responsible for solving the linear system. This introduces a strong load imbalance. Even worse the synchronous nature of the RATTLE integrator does not permit the other processing elements to overlap the wait time with other computations (since the Lagrange forces need to be available before the algorithm can proceed), resulting in unwanted idle time.

In this article we concentrate on the first challenge. At this point MACI does not provide functionality to optimize the load balancing. This is a strong limiting factor for the parallel efficiency (cf. Figure 5a). As can be seen in Figure 5b, for a fixed number of processing elements, the choice of the number of MD and the number of FE processing elements is crucial for the performance even on a moderate number of cores. Here, a priori load models need to be developed to assist users in finding an optimal configuration.



(a) Speedups (for the optimal choice of MD and FE processing element).



(b) Ratio between time per time step in the worst and best configuration.

Figure 5: Speedup plots from a two-dimensional fracture simulation with 142,628 atoms and 28,178 finite elements on up to 32 cores. The simulation was run on an 4x DDR Infiniband cluster with dual-socket quad-core Barcelona Opteron nodes.

4.2 Data distribution in the MACI code

MACI is written in an MPMD (multiple programm, multiple data) fashion (even though it is implemented as a single executable), i.e., processing elements that run the MD component (plus coupling code) take a (mostly) disjoint execution path compared to the execution path of the FE processing elements. Any exchange of data between MD and FE processing elements is done via message passing. To decrease communication cost it might be advantages to use threading and let one MD and one FE processing element share one address space. We refrained from this design in MACI since it complicates the coupling code (which in this case must be able to cope with one FE and one MD component) and requires good a priori knowledge about the communication graph including the communication volume per edge. Since the graph is dynamic, it usually is infeasible to do an optimal process mapping statically.

An example of the data distribution used by MACI is shown in Figure 2b. In this simulation, the MD domain is distributed over 12 processing elements. The FE mesh is distributed over 4 processing elements. All the datastructures (including the $\vec{T}$, $\vec{q}$ and Λ matrices) are distributed over the four FE processing elements and 8 MD processing that own mesh nodes with $\vec{x}_A \in \overline{\Omega_H}$ or owned (at $t = 0$) particles in the handshake region.

The matrices $\vec{T}$, $\vec{q}$ and Λ are distributed by row. For $\vec{T}$ and Λ the Ath row is stored on the processing element that owns the node A (note that the cellwise mesh decomposition results in the duplication of mesh nodes on several processing elements). Also for the matrix $\vec{q}$ we use a static distribution: The αth row is stored on the processing element p with $\vec{x}_\alpha(0) \in B_p$. This static decomposition of $\vec{q}$ implies that the matrix-vector multiplication $\vec{y} = \vec{q}\vec{x}$ requires two communication steps: one gather operation to collect $\vec{x}$ values on the processing elements storing rows of the matrix and a second scatter operation, after the local matrix vector multiplication, to send the entry $\vec{y}_\alpha$ to the current owner of particle α. On the other hand a dynamic distribution of $\vec{q}$ (where the αth row of $\vec{q}$ is stored on the processing element owning the particle α) would require MD processing elements to be informed about the particle distribution on other processing

elements. In particular if $\vec{q}_{\alpha\beta} \neq 0$ and $\beta \in B_q$, the processing element q would need to know on which processing element particle α is located. Maintaining such a mapping from particles to processing elements in a scalable manner is itself very complicated.

The sets $\left\{A \mid \vec{T}_{A\alpha} \neq 0\right\}$, $\left\{\beta \mid \vec{q}_{\beta\alpha} \neq 0\right\}$ are stored as part of the `PiggybackType` structure of the particle α and hence migrate with the particle. This allows MD processing elements to create lists of data item that have to be send to either FE processing elements (multiplication by $\vec{T}$) or handshake MD processing elements (multiplication by $\vec{q}$). Note however that this information is only available on the sender side. To the receiver side, the particle data distribution is unknown, cf. Subsection 4.4.

4.3 Parallel assembly

In order to avoid bottlenecks in the computational workflow it is crucial to parallelize the complete application, in particular, the assembly of the transfer operator $\vec{T}$ and of the interpolation matrix $\vec{N}$. In MACI, the matrices $\vec{q}$ and Λ are computed from $\vec{T}$, $\tilde{\vec{M}}$ and $\vec{N}$ using sparse matrix-matrix multiplications. Compared to direct assembly of the matrix entries (used initially in MACI), this approach is slower but the additional flexibility allows us to easily implement different transfer operators.

For a given mesh node A, the assembly of $\vec{T}$ and $\vec{N}$ requires the identification of all particles $\alpha \in \mathbb{A}$ such that either $\mathrm{meas}\,(\mathrm{supp}\,\psi_\alpha \cap \mathrm{supp}\,\theta_A) > 0$ or $\theta_A(\vec{x}_\alpha) \neq 0$. To find all α in (quasi-)optimal time we use (parallel) tree queries.

In a first step, a parallel quad- or octree is build with particles as leaves. The bounding boxes of intermediate nodes are chosen in a leaf-to-root pass such that the bounding box of a parent node contains the union of the bounding boxes of all children. On the leaf level, the bounding box is chosen to be the support of ψ_α. Note that, different from [24], we do not use the tree to construct $\omega_\alpha = \mathrm{supp}\,\psi_\alpha$, since we require an algorithm producing open covers without adding additional points to the set of particles. Instead we make use of the lattice structure of the MD system to choose the patch size $\vec{h}$ a priori in such a way that

$$\omega_\alpha = \prod_{i=1}^{3} \left((\vec{x}_\alpha)_i - \frac{1}{2}\vec{h}_i, (\vec{x}_\alpha)_i + \frac{1}{2}\vec{h}_i \right)$$

yields an open covering of Ω_{H}. When using Shepards method, the evaluation of the basis function ψ_α requires the detection of overlapping patches. This can be achieved by using again a tree. In MACI, the rectangular domain decomposition used by most MD applications is employed to compute a priori potential remote intersection partners (this is possible since we know the patch size $\vec{h}$). These lists are be exchanged and inserted into the local tree. Once this extended local tree is constructed, all intersection queries can be performed locally.

The assembly of the transfer operator $\vec{T}$ is performed on the MD processing elements as shown in Algorithm 2.

Note that basis functions ψ_α computed with Shepards method are only C^0. To achieve good accuracy in the computation of $\vec{T}$ via numerical quadrature, we need to compute a partition

$$\omega_\alpha = \bigcup_Q Q \quad \text{such that} \quad \psi_\alpha|_Q \in C^\infty \,. \tag{4}$$

This is possible (though not inexpensive) since our patches ω_α are axis parallel quadrilaterals or hexahedra.

Algorithm 2 Parallel assembly of $\vec{T}$.

1. Exchange tree root bounding boxes between MD and FE processing elements.
2. On FE processing elements: Build lists of elements E intersecting the bounding boxes of the MD processing elements.
3. Send element lists from FE processing elements to MD processing elements.
4. **for all** Elements E received (only on handshake MD processing elements) **do**
5. Query (locally) all particles α with $\mathrm{meas}(\mathrm{supp}\,\psi_\alpha \cap E) > 0$.
6. **for all** found α **do**
7. **for all** Q as in Equation (4) **do**
8. Compute intersection $Q \cap E$.
9. Perform quadrature on the intersection.
10. **end for**
11. **end for**
12. **end for**
13. Send computed values back to the FE processing element.
14. On FE processing elements: Merge the received lists and construct sparse matrix datastructure.

4.4 Runtime support

In this section we consider the implementation of Steps 2, 5 and 7 in Algorithm 1 in MACI. As noted in [18], the computation of the Lagrange forces $\vec{T}^{\mathrm{T}}\Lambda^{-1}\vec{G}^*$ and $\vec{T}^{\mathrm{T}}\Lambda^{-1}\dot{\vec{G}}^*$ can be handled in the same way as the multiplication by $\vec{q}$. These operations can be written as

$$\mathtt{Scat}_t \circ \mathtt{Op} \circ \mathtt{Gat}_t \ , \tag{5}$$

where $\mathtt{Scat}_t$ and $\mathtt{Gat}_t$ are time-dependent (since the particle distribution is changing over time) scatter and gather operations and where **Op** is some (black box) operation executed on a subset of processing elements (the *workers*). For example, in Step 2 of Algorithm 1, the black box operation is given by

$$\mathtt{Op}(\vec{z}) = \vec{T}^{\mathrm{T}}\Lambda^{-1}\left(\vec{T}\vec{z} - \tilde{\vec{M}}\vec{U}^*\right) \ .$$

Finite element processing elements owning handshake mesh nodes (or equivalently, a non-zero row of $\vec{T}$) are designated as workers. The computation of the Lagrange multipliers required for the correction of the FE displacement is a *side effect* of the execution of **Op** on the worker processing elements.

Note that the choice of the workers and the order of the input and output data to and from **Op** is not time-dependent. Hence, $\mathtt{Gat}_t$ and $\mathtt{Gat}_{t'}$ ($t \neq t'$) are required to order the input data for **Op** in the same way. Similarly, $\mathtt{Scat}_t$ receives the output of **Op** always in the same order.

The advantage of this approach is that the data distribution is *transparent* to the workers. In particular no adaptation of the worker data structures are required when the particle distribution changes (this is to compare with the approach in [2] where the worker datastructures are updated using an event-based notification system).

The implementation of the gather and scatter operations are based on the piggy-backed metadata. As noted in Subsection 4.2, the piggyback data allows MD processing elements to build send lists. However, workers/receivers do not know about

the particle distribution and therefore cannot post matching receives. To cope with this problem, in [18] the use of *one-sided communication* or *remote memory access* is proposed.

In MACI we use the newly developed communication library MEXICO [18] to implement the Operation (5). The unique feature of MEXICO is that the library provides gather and scatter operations in the described asymmetric setup. All information required by MEXICO is provided by the source processing elements (processing elements that provide data to $\texttt{Gat}_t$) and target processing elements (processing elements that retrieve data from $\texttt{Scat}_t$). In MACI this information (the list of worker processing elements including local indices in the input and output buffers of **Op**) are stored in the piggyback data. MEXICO can use MPI RMA, MPI Point-to-point, MPI collectives, the GLOBAL ARRAYS library [15] or SHMEM for inter-process communication.

In Algorithm 3, the implementation of the second step in the RATTLE integration scheme (cf. Algorithm 1) is shown. The computational work is performed in Steps 5, 6 and 7. The input and output buffers for these operations are ordered according to an a priori (during the assembly phase) chosen ordering. Hence, the sparse-matrix storage scheme for, e.g., $\vec{T}$, can be kept unmodified over time.

Algorithm 3 Implementation of Step two in Algorithm 1.

1. Pack displacements into contiguous buffer.
2. On MD processing elements: Extract list of workers and corresponding local indices for each particle.
3. On FE processing elements: Compute $\tilde{\vec{M}}\vec{U}^*$.
4. Communicate MD displacements to worker (MEXICO).
5. On worker: Compute $\vec{G}^*$ using the input buffer containing MD displacements.
6. On worker: Solve $\Lambda\vec{\lambda} = \vec{G}^*$.
7. On worker: Compute $\vec{T}^{\mathrm{T}}\lambda$ and store the result in the output buffer.
8. Communicate Lagrange forces to MD processing elements (MEXICO).
9. Correct displacements and velocities.

Acknowledgement
This work was supported by the Deutsche Forschungsgemeinschaft through the SFB 611 “Singular Phenomena and Scaling in Mathematical Models”, by the “Swiss High Performance and Productivity Computing” (HP2C) initiative and by the DFG Research Center MATHEON.

References

[1] Allen MP, Tildesley DJ (1987) Computer Simulation of Liquids. Oxford Science Publications

[2] Anciaux G, Coulaud O, Roman J (2006) High Performance Multiscale Simulation or Crack Propagation. In: Proceedings of the 2006 International Conference Workshops on Parallel Processing, IEEE Computer Society, pp 473 – 480

[3] Armstrong R, Gannon D, Geist A, Keahey K, Kohn S, McInnes L, Parker S, Smolinski B (1999) Toward a common component architecture for high-performance scientific computing. In: Eighth International Symposium on High Performance Distributed Computing, 1999. Proceedings., pp 115 –124

[4] Balay S, Brown J, Buschelman K, Gropp W, Kaushik D, Knepley M, McInnes L, Smith B, Zhang H (?) Petsc web page. http://www.msc.anl.gov/petsc

[5] Bastian P, Birken K, Johannsen K, Lang S, Neuss N, Rentz-Reichert H, Wieners C (1997) UG - A Flexible Software Toolbox for Solving Partial Differential Equations. Comp Vis Science 1:27 – 40

[6] Beazley DM (1996) Swig: an easy to use tool for integrating scripting languages with c and c++. In: Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4, USENIX Association, Berkeley, CA, USA, TCLTK'96, pp 15 – 15

[7] Bowers KJ, Chow E, Xu H, Dror RO, Eastwood MP, et al (2006) Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC06), Tampa, Florida, pp 84 – 88

[8] Broughton JQ, Abraham FF, Bernstein N, Kaxiras E (1999) Concurrent coupling of length scales: Methodology and application. Phys Rev B 60

[9] Davis TA (2004) Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. ACM Trans Math Softw 30:196 – 199

[10] Fackeldey K, Krause R (2009) Multiscale coupling in function space – weak coupling between molecular dynamics and continuum mechanics. Int J Num Meth Engrg 79(12):1517 – 1535

[11] Fackeldey K, Krause D, Krause R (2010) Numerical validation of constraints based multiscale methods. In: Proc. Meshfree Methods for Partial Differential Equations V, Lecture Notes in Computational Science and Engineering, vol 79, pp 141 – 154

[12] Fackeldey K, Krause D, Krause R, Lenzen C (2011) Coupling molecular dynamics and continua with weak constraints. Multiscale Modeling & Simulation 9(4):1459 – 1494

[13] Fasshauer GE (2007) Meshfree Approximation Methods with Matlab. World Scientific

[14] Forum MPI (2009) MPI: A Message-Passing Interface Standard, Version 2.2

[15] Global_Array (?) http://www.emsl.pnl.gov/docs/global

[16] Griebel M, Knapek S, Zumbusch G (2007) Numerical Simulation in Molecular Dynamics. Springer-Verlag Berlin Heidelberg

[17] Heroux MA, Bartlett RA, Howle VE, Hoekstra RJ, et al (2005) An overview of the trilinos project. ACM Trans Math Softw 31(3):397–423

[18] Krause D, Krause R (2011) Parallel scale-transfer in multiscale md-fe coupling using remote memory access. In: IEEE 7th International Conference on E-Science, e-Science 2011, Workshop Proceedings, Stockholm, Sweden, December 5 – 8, 2011, pp 66 – 73

[19] Lammps (2012) http://lammps.sandia.gov

[20] Ma J, Lu H, Wang B, Hornung R, Wissink A, Komanduri R (2006) Multiscale Simulations Using Generalized Interpolation Material Point (GIMP) Method and Molecular Dynamics (MD). Computer Modeling in Engineering Sciences 14:101–118

[21] Mapper_Project (?) http://www.mapper-project.eu

[22] Phillips JC, Braun R, Wang W, Gumbart J, Tajkhorshid E, Villa E, Chipot C, Skeel RD, Kalé L, Schulten K (2005) Scalable molecular dynamics with NAMD. J Comput Chem 26(16):1781 – 1802

[23] Plimpton SJ (1995) Fast parallel algorithms for short-range molecular dynamics. J Comp Phys 117:1–19

[24] Schweitzer MA (2003) A Parallel Multilevel Partition of Unity Method for Elliptic Partial Differential Equations, Lecture Notes in Computational Science and Engineering, vol 29. Springer

[25] Smolinski BA, Kohn SR, Elliott N, Dykman N (1999) Language interoperability for high-performance parallel scientific components. In: Proceedings of the Third International Symposium on Computing in Object-Oriented Parallel Environments, Springer-Verlag, ISCOPE '99, pp 61 – 71

[26] Streitz FH, Glosli JN, Patel MV, Chan B, Yates RK, et al (2005) 100+ TFlop Solidification Simulations on BlueGene/L. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC05), Seattle, Washington

[27] Wagner GJ, Liu WK (2003) Coupling of atomistic and continuum simulations using a bridging scale decomposition. J Comp Phys 190:249 – 274

[28] Xiao S, Ni J, Wang S (2008) The Bridging Domain Multiscale Method and its High Performance Computing Implementation. J Comp Theo Nanoscience 5:1–10

[29] Xiao SP, Belytschko T (2004) A bridging domain method for coupling continua with molecular dynamics. Comp Meth Appl Engrg 193:1645 – 1669