CrossMark

ARTICLE

PyCAC: The concurrent atomistic-continuum simulation environment

Shuozhi Xu[a)]
California NanoSystems Institute, University of California, Santa Barbara, Santa Barbara, California 93106-6105, USA

Thomas G. Payne
School of Materials Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0245, USA

Hao Chen
Department of Aerospace Engineering, Iowa State University, Ames, Iowa 50011, USA

Yongchao Liu
School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332, USA

Liming Xiong
Department of Aerospace Engineering, Iowa State University, Ames, Iowa 50011, USA

Youping Chen
Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, Florida 32611-6250, USA

David L. McDowell
School of Materials Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0245, USA; and GWW School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0405, USA

(Received 28 August 2017; accepted 2 January 2018)

We present a novel distributed-memory parallel implementation of the concurrent atomistic-continuum (CAC) method. Written mostly in Fortran 2008 and wrapped with a Python scripting interface, the CAC simulator in PyCAC runs in parallel using Message Passing Interface with a spatial decomposition algorithm. Built upon the underlying Fortran code, the Python interface provides a robust and versatile way for users to build system configurations, run CAC simulations, and analyze results. In this paper, following a brief introduction to the theoretical background of the CAC method, we discuss the serial algorithms of dynamic, quasistatic, and hybrid CAC, along with some programming techniques used in the code. We then illustrate the parallel algorithm, quantify the parallel scalability, and discuss some software specifications of PyCAC; more information can be found in the PyCAC user's manual that is hosted on www.pycac.org.

I. INTRODUCTION

Despite substantial insights provided by atomistic simulations over the past few decades into the basic mechanisms of metal plasticity, there is an important limitation to these techniques.[1,2] Specifically, full atomistic models are impractical in simulating actual experiments of plastic deformation of metallic materials owing to the fact that dislocation pile-ups have long range stress fields that extend well beyond what can be captured using classical molecular dynamics (MD) and molecular statics (MS).[3,4] This has motivated researchers to develop partitioned-domain (or domain decomposition) multiscale modeling approaches that retain atomistic resolution in regions where explicit descriptions of nanoscale structure and phenomena are essential, while employing continuum treatment elsewhere.[5,6]

One framework for such mixed continuum/atomistic modeling is the concurrent atomistic-continuum (CAC) method.[7] In a prototypical CAC simulation, a simulation cell is partitioned into a coarse-grained domain at continuum level and an atomistic domain,[8] employing a unified atomistic-continuum integral formulation of the governing equations with the underlying interatomic potential as the only constitutive relation in the system. The atomistic domain is updated in the same way as in an atomistic simulation while the coarse-grained domain contains elements that have discontinuities between them.[9] As such, CAC admits propagation of displacement discontinuities (dislocations and associated intrinsic stacking faults) through the lattice in both coarse-grained and atomistic domains.[10] Distinct from most partitioned-domain multiscale methods in the literature, the CAC method (i) describes certain lattice defects and their

Contributing Editor: Vikram Gavini
[a)]Address all correspondence to this author.
e-mail: shuozhixu@ucsb.edu
DOI: 10.1557/jmr.2018.8

 © Materials Research Society 2018

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

interactions using fully resolved atomistics; (ii) preserves the net Burgers vector and associated long range stress fields of curved, mixed character dislocations in a sufficiently large continuum domain in a fully 3D model[11]; (iii) employs the same governing equations and interatomic potentials in both atomistic and coarse-grained domains to avoid the usage of phenomenological parameters, essential remeshing operations, and ad hoc procedures for passing dislocation segments between the two domains.[12,13]

In recent years, the CAC approach has been used as an effective tool for coarse-grained modeling of various nano/microscale thermal and mechanical problems in a wide range of monatomic and polyatomic crystalline materials. Important static properties of pure edge, pure screw, and mixed-type dislocations in metals, including the generalized stacking fault energy surface, dislocation core structure/energy/stress fields, and Peierls stress, have been benchmarked in the coarse-grained domain in CAC against atomistic simulations, and the trade-off between accuracy and efficiency in the coarse-graining procedure has been quantified.[8,14] The success of these calculations suggests the viability of using the CAC method to tackle more complex dislocation-mediated metal plasticity problems. Particularly for pure metals, CAC has been adopted to simulate surface indentation, quasistatic,[8] subsonic,[15] and transonic[16] dislocation migration in a lattice, dislocations passing through the coarse-grained/atomistic domain interface,[8] screw dislocation cross-slip,[17] dislocation/void interactions,[18] dislocation/stacking fault interactions,[19] dislocations bowing out from obstacles,[20] dislocation multiplication from Frank–Read sources,[14] sequential slip transfer of dislocations across grain boundaries (GBs),[21,22] and brittle-to-ductile transition in dynamic fracture.[15] It is shown that CAC provides largely satisfactory predictive results at a fraction of the computational cost of the fully atomistic version of the same models. While the net Burgers vector of dislocations is preserved in the coarse-grained domain, we remark that there exist coarse-graining errors in the dislocation description.[8] However, (i) these errors are not essential if the lattice defects are rendered in atomistic resolution because the dislocation will have a correct core structure once it migrates into the atomistic domain in which critical dislocation/defect interactions take place[21,22] and (ii) even with fully coarse-grained models, CAC is useful for problems that do not require highly accurate full-field reproduction of atomistic simulations (e.g., large models).

In this article, we introduce PyCAC, a novel numerical implementation of the CAC approach. Written mostly in Fortran 2008, PyCAC is equipped with a Python scripting interface to create a more efficient, user-friendly, and extensible CAC simulation environment, which consists of a CAC simulator and a data analyzer. In the remainder of this paper, we start with the theoretical background of CAC in Sec. II. Next, we discuss the serial algorithms of dynamic, quasistatic, and hybrid CAC, along with some programming techniques in the PyCAC code in Sec. III. Then, we present the parallelization steps in the CAC simulator in Sec. IV and the PyCAC software in Sec. V. In the end, a summary and discussions of future research directions are provided in Sec. VI.

II. THEORETICAL BACKGROUND

The theoretical foundation of the CAC method is the atomistic field theory (AFT),[23,24] which is an extension of the Irving–Kirkwood's nonequilibrium statistical mechanical formulation of "*the hydrodynamics equations for a single component, single phase system*"[25] to a two-level structural description of crystalline materials. It employs the two-level structural description of all crystals in solid state physics, i.e., the well known equation of "crystal structure = lattice + basis".[26] As a result of the bottom-up atomistic formulation, all the essential atomistic information of the material, including the crystal structure and the interaction between atoms, is built in the formulation. The result is a CAC representation of balance laws for both atomistic and continuum coarse-grained domains in the following form[23,24]:

$$\frac{\mathrm{d}\rho^{\alpha}}{\mathrm{d}t} + \rho^{\alpha}\left(\nabla_{\mathbf{x}} \cdot \mathbf{v} + \nabla_{\mathbf{y}^{\alpha}} \cdot \Delta\mathbf{v}^{\alpha}\right) = 0 \quad , \tag{1}$$

$$\rho^{\alpha}\frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{v} + \Delta\mathbf{v}^{\alpha}\right) = \nabla_{\mathbf{x}} \cdot \mathbf{t}^{\alpha} + \nabla_{\mathbf{y}^{\alpha}} \cdot \tau^{\alpha} + \mathbf{f}^{\alpha}_{\mathrm{ext}} \quad , \tag{2}$$

$$\rho^{\alpha}\frac{\mathrm{d}e^{\alpha}}{\mathrm{d}t} = \nabla_{\mathbf{x}} \cdot \mathbf{q}^{\alpha} + \nabla_{\mathbf{y}^{\alpha}} \cdot \mathbf{j}^{\alpha} + \mathbf{t}^{\alpha} : \nabla_{\mathbf{x}}\left(\mathbf{v} + \Delta\mathbf{v}^{\alpha}\right) + \tau^{\alpha} : \nabla_{\mathbf{y}^{\alpha}}\left(\mathbf{v} + \Delta\mathbf{v}^{\alpha}\right) \quad , \tag{3}$$

where $\mathbf{x}$ is the physical space coordinate; $\mathbf{y}^{\alpha}$ ($\alpha = 1, 2, \ldots, N_{\mathrm{a}}$ with N_{a} being the total number of atoms in a unit cell) are the internal variables describing the position of atom α relative to the mass center of the lattice cell located at $\mathbf{x}$; ρ^{α}, $\rho^{\alpha}(\mathbf{v} + \Delta\mathbf{v}^{\alpha})$, and $\rho^{\alpha}e^{\alpha}$ are the local densities of mass, linear momentum, and total energy, respectively, where $\mathbf{v} + \Delta\mathbf{v}^{\alpha}$ is the atomic-level velocity and $\mathbf{v}$ is the velocity field; $\mathbf{f}^{\alpha}_{\mathrm{ext}}$ is the external force field; $\mathbf{t}^{\alpha}$ and $\mathbf{q}^{\alpha}$ are the stress and heat flux tensors due to the homogeneous deformation of lattice, respectively; τ^{α} and $\mathbf{j}^{\alpha}$ are the stress and heat flux tensors due to the reorganizations of atoms within the lattice cells, respectively; and the colon : denotes the scalar product of two second rank tensors $\mathbf{A}$ and $\mathbf{B}$, i.e., $\mathbf{A}:\mathbf{B} = A_{ij}B_{ij}$.

For conservative systems, i.e., a system in the absence of an internal source that generates or dissipates energy, the energy equation [Eq. (3)] is equivalent to the linear momentum equation [Eq. (2)]. As a result, only the first two governing equations [Eqs. (1) and (2)] are explicitly implemented in the CAC simulator. Employing the classical

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

definition of kinetic temperature, which is proportional to the kinetic part of the atomistic stress, the linear momentum equations can be expressed in a form that involves $\mathbf{f}^{\alpha}_{\text{ext}}$, internal force density $\mathbf{f}^{\alpha}_{\text{int}}$, and temperature T,[27,28]

$$\rho^{\alpha}\ddot{\mathbf{u}}^{\alpha}(\mathbf{x}) + \frac{\gamma^{\alpha}k_{\text{B}}}{\Delta V}\nabla_{\mathbf{x}}T = \mathbf{f}^{\alpha}_{\text{int}}(\mathbf{x}) + \mathbf{f}^{\alpha}_{\text{ext}}(\mathbf{x}), \quad \alpha = 1, 2, \ldots, N_{\text{a}} \quad , \tag{4}$$

where $\mathbf{u}^{\alpha}$ is the displacement of the αth atom at point $\mathbf{x}$, the superposed dots denote the material time derivative, ΔV is the volume of the finite-sized material particle (the primitive unit cell for crystalline materials) at $\mathbf{x}$, k_{B} is the Boltzmann constant, $\gamma^{\alpha} = m^{\alpha}/\sum_{\alpha=1}^{N_{\text{a}}} m^{\alpha}$, T is the absolute temperature (in K), and $\mathbf{f}^{\alpha}_{\text{int}}(\mathbf{x})(= \nabla_{\mathbf{x}} \cdot \mathbf{t}^{\alpha})$ is a nonlinear nonlocal function of relative atomic displacements. For systems with a constant temperature field or a constant temperature gradient, the temperature term in Eq. (4), which can be denoted as $\mathbf{f}^{\alpha}_{T}$, has the effect of a surface traction on the boundary or a body force in the interior of the material.[28] In the CAC simulator, the term $\mathbf{f}^{\alpha}_{T}$ has not yet been implemented because, as will be discussed in Sec. V.A, the current version of PyCAC can only simulate materials in a constant temperature field, in which case the effect of $\mathbf{f}^{\alpha}_{T}$ on mechanical properties is small. We remark that there is ongoing work in interpreting $\mathbf{f}^{\alpha}_{T}$ and in comparing different descriptions of temperature in the coarse-grained domain[29,30]; new understanding obtained will be implemented in future versions of PyCAC.

For monatomic crystals, $\mathbf{y}^{\alpha} = 0$ and $N_{\text{a}} = 1$; the governing equations in the physical space reduce to

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho\nabla_{\mathbf{x}} \cdot \mathbf{v} = 0 \quad , \tag{5}$$

$$\rho\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \nabla_{\mathbf{x}} \cdot \mathbf{t} + \mathbf{f}_{\text{ext}} \quad , \tag{6}$$

$$\rho\frac{\mathrm{d}e}{\mathrm{d}t} = \nabla_{\mathbf{x}} \cdot \mathbf{q} + \mathbf{t} : \nabla_{\mathbf{x}}\mathbf{v} \quad , \tag{7}$$

and, for conservative systems, in the absence of $\mathbf{f}^{\alpha}_{T}$, Eq. (4) becomes

$$\rho\ddot{\mathbf{u}} = \mathbf{f}_{\text{int}} + \mathbf{f}_{\text{ext}} \quad . \tag{8}$$

Discretizing Eq. (8) using the Galerkin finite element method yields

$$\int_{\Omega(\mathbf{x})} \Phi_{\xi}(\mathbf{x})(\rho\ddot{\mathbf{u}}(\mathbf{x}) - \mathbf{f}_{\text{int}}(\mathbf{x}) - \mathbf{f}_{\text{ext}}(\mathbf{x}))\mathrm{d}\mathbf{x} = 0 \quad , \tag{9}$$

where Ω is the simulation domain and $\mathbf{\Phi}_{\xi}$ is the finite element shape function. In the coarse-grained domain, the integral in Eq. (9) can be evaluated using numerical integration methods such as Gaussian quadrature. It is, however, difficult to employ a unified set of integration points within an element because the internal force density $\mathbf{f}_{\text{int}}$ can be a complicated and highly nonlinear function of $\mathbf{x}$, whose order and distribution are usually difficult to anticipate a priori.[8,11] To circumvent this problem, we divide each element into a number of nonoverlapping subregions, whose order is usually lower than that within the entire element and is thus more easily approximated. In the CAC simulator, within each subregion, the first order Gaussian quadrature with one integration point at the center of the subregion is adopted; then the integrals of all subregions within an element are summed, resulting in the force on node ξ as[8]

$$\mathcal{F}^{\xi} = \frac{\sum_{\mu}\omega_{\mu}\Phi_{\mu\xi}\mathbf{F}^{\mu}}{\sum_{\mu}\omega_{\mu}\Phi_{\mu\xi}} + \mathbf{F}^{\xi}_{\text{ext}} \quad , \tag{10}$$

where ω_{μ} is the weight of integration point μ, $\mathbf{\Phi}_{\mu\xi}$ is the shape function of node ξ at integration point μ, $\mathbf{F}^{\mu}$ is the interatomic potential-based atomic force on integration point μ, and $\mathbf{F}^{\xi}_{\text{ext}}$ is the external force applied on node ξ. Once the nodal positions are determined, the positions of atoms within an element are interpolated from those of the nodes using the shape function $\mathbf{\Phi}$; in other words, all elements are isoparametric.

In the atomistic domain, an atom can be viewed as a special finite element for which the shape function reduces to 1 at the atomic site, and the force on atom α is simply

$$\mathbf{F}^{\alpha} = -\nabla_{\alpha}E + \mathbf{F}^{\alpha}_{\text{ext}} \quad , \tag{11}$$

where E is the interatomic potential-based internal energy and $\mathbf{F}^{\alpha}_{\text{ext}}$ is the external force applied on atom α. The CAC method, as a coarse-grained atomistic methodology, reduces to standard MD or MS if only fully resolved atomistic domains are considered in the simulation, in which case Eq. (10) is no longer relevant and only Eq. (11) takes effect. In other words, the PyCAC code is capable of performing fully-resolved MD/MS simulations if only atoms are involved. In the remainder of this paper, both $\mathcal{F}$ and $\mathbf{F}$ are referred to in combination as $\mathbf{F}$.

III. SERIAL ALGORITHMS

While the CAC simulator runs in parallel, we focus on the serial algorithms of the CAC simulator and the data analyzer in this section to elucidate their differences from atomistic methods. The parallelization steps in the CAC simulator will be detailed in Sec. IV.

Similar to the MD and MS methods in atomistic simulations, users of PyCAC can choose among dynamic CAC, quasistatic CAC, and hybrid CAC, as illustrated in

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

the serial CAC simulation scheme (Fig. 1). In all CAC simulations, the nodes in the coarse-grained domain and the atoms in the atomistic domain interact with each other at each simulation step and are updated concurrently.

A. Dynamic CAC

In dynamic CAC, the forces **F** are used in the equation of motion

$$m\ddot{\mathbf{R}} = \mathbf{F} \quad , \tag{12}$$

or its modified version, where m is the normalized lumped or consistent mass in the coarse-grained domain or the atomic mass in the atomistic domain and **R** is the nodal/atomic position. Three options are provided: velocity Verlet (vv), quenched dynamics (qd), and Langevin dynamics (ld), as illustrated in Fig. S1 (Supplementary Material).

For both vv and qd options, Eq. (12) is solved with the velocity Verlet algorithm[31]; with qd, the nodal/atomic velocities $\dot{\mathbf{R}}$ are also adjusted by nodal/atomic forces **F** following the "quick-min" MD approach[32] to force the system energy toward a minimum at 0 K,[21,22] i.e.,

$$\dot{\mathbf{R}} = \begin{cases} \mathbf{0}, & \text{if } \dot{\mathbf{R}} \cdot \mathbf{F} < 0 \\ \frac{(\dot{\mathbf{R}} \cdot \mathbf{F})\mathbf{F}}{|\mathbf{F}|^2}, & \text{otherwise} \end{cases} \quad . \tag{13}$$

With the ld option, which is used to keep a constant finite system temperature,[15,17] Eq. (12) is extended by two terms, i.e.,

$$m\ddot{\mathbf{R}} = \mathbf{F} - \gamma m\dot{\mathbf{R}} + \Theta(t) \quad , \tag{14}$$

where γ is the damping coefficient with a unit of frequency and $\Theta(t)$ is the time-dependent Gaussian random variable with zero mean and variance of $\sqrt{2m\gamma k_B T/\Delta t}$, where k_B is the Boltzmann constant and Δt is the time step. Numerically, both Eqs. (12) and (14) are solved using the velocity Verlet formulation of the Brünger–Brooks–Karplus integrator.[33] In particular, when $T = 0$ K, Eq. (14) becomes damped MD, which is used for dynamic relaxation at near 0 K temperature.

In the early version of the AFT formulation,[23] the local densities were defined as ensemble averages following the Irving–Kirkwood formulations,[25] and hence, the governing equations were written in terms of ensemble-averaged local densities. In the later version of the AFT formulation,[24] the local densities are instantaneous quantities.[34] Consequently, the ensembles in AFT (and CAC) differ from other statistical mechanical formulations that follow the Gibbs' equilibrium statistical theory of ensembles. Popular equilibrium ensembles include (i) the microcanonical ensemble, which describes a system isolated from its surroundings and governed by Hamilton's equations of motion (NVE), (ii) the canonical ensemble, which describes a system in constant contact with a heat bath of constant temperature (NVT), and (iii) the isothermal–isobaric ensembles, which describe systems in contact with a thermostat at temperature T and a barostat at pressure P (NPT).[35] These ensembles, known as equilibrium ensembles and allowing a wide variety of thermodynamic and structural properties of systems to be computed, can be realized in dynamic CAC, in which a finite temperature can be achieved via lattice dynamic-based shape functions.[36] Alternatively, in the current code, a Langevin thermostat is realized using the ld option while a constant pressure/stress is maintained via a Berendsen barostat[37]; other thermostat and barostat may also be implemented.

Prior dynamic CAC applications using the vv option include phase transitions,[28] crack propagation and branching,[38,39] nucleation and propagation of dislocations,[7,16,40–42] formation of dislocation loops,[43] defect/interface interactions,[44,45] phonon/dislocation interactions,[46,47] and phonon/GB interactions[48,49]; prior dynamic CAC simulations using the ld option include dislocation/void interactions[18] ($T \approx 0$ K), dislocation migration[15] ($T \approx 0$ K), and screw dislocation cross-slip[17] ($T = 10$ K). These simulations have revealed the underlying mechanisms for a variety of experimentally

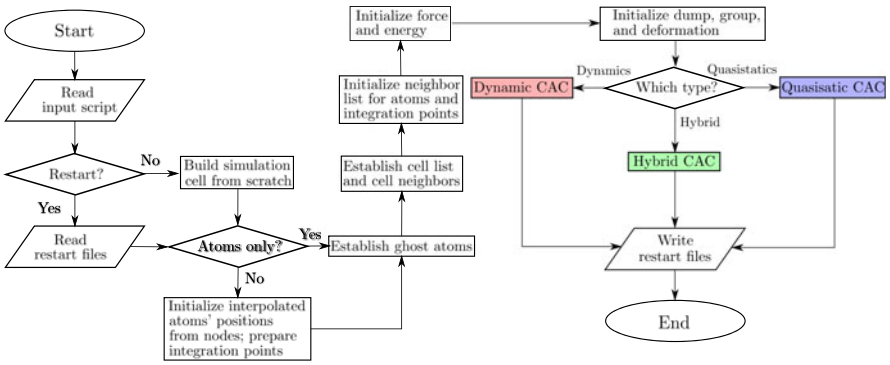


FIG. 1. Serial CAC simulation scheme.

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

observed phenomena, including, but not limited to, phonon focusing, phonon-induced dislocation drag, phonon scattering by interfaces and by other phonons, coexistence of the coherent and incoherent propagation of ultrafast heat pulses in polycrystalline materials, and Kapitza thermal boundary resistance.

B. Quasistatic CAC

In quasistatic CAC, which is analogous in character to MS, **F** is used to systematically adjust the nodal/atomic positions at each increment of system loading during the energy minimization.[8] Four energy minimization algorithms are introduced: conjugate gradient (cg), steepest descent (sd), quick-min (qm), and fast inertial relaxation engine (fire), as illustrated in Fig. S2 (Supplementary Material). The energy minimization procedure stops when either the number of iterations reaches a maximum value or the energy variation between successive iterations divided by the current energy is less than a tolerance. Here, the "energy" consists of the interatomic potential-based internal energy E and the traction boundary conditions-induced external energy.

Both cg and sd algorithms follow the standard procedure in MS, which contains an outer iteration to find the search direction and the inner iteration to determine the step size.[50] For the qm and fire algorithms, there is no inner iteration. Instead, dynamic-like runs are iteratively performed at each loading increment until the energy converges. The qm algorithm[32] is based on quenched dynamics, which is also used in dynamic CAC, except that in the latter case, only one quenched dynamics iteration is carried out at each simulation step. The fire algorithm is based on a local atomic structure optimization algorithm.[51,52]

A comparison of these four algorithms was presented by Sheppard et al.[32] In general, the cg algorithm is the most commonly used in atomistic simulations[53] and has been used in several quasistatic CAC simulations,[8,14,19,20] while the fire algorithm is considered the fastest in certain cases.[51]

C. Hybrid CAC

Besides the dynamic CAC and quasistatic CAC, the CAC simulator provides a third option: "hybrid CAC", which allows users to perform periodic energy minimization during a dynamic CAC simulation, as illustrated in Fig. 2. Users can choose among all three dynamic options and four energy minimization algorithms. Hybrid CAC has been used in Refs. 21 and 22 to simulate dislocation pile-ups across GBs: a series of dislocations migrate and interact with GBs during the quenched dynamic run, while the fully relaxed GB structures, important in predicting accurate dislocation/GB interaction mechanisms, are obtained via periodic energy minimization using the conjugate gradient algorithm. In practice, this enables the multiscale optimization for a sequence of constrained nonequilibrium states (defect configurations) in materials.[11] Accordingly, hybrid CAC yields results similar to quasistatic CAC at lower computational cost because the quenched dynamic run, which pertains to 0 K (or very nearly so), is more efficient than the conjugate gradient algorithm.[21]

D. Programming techniques

A framework for mixed atomistic/continuum modeling, the CAC algorithm adopts common finite element and atomistic modeling techniques. In the coarse-grained domain, the Garlekin method and Gaussian quadrature are used to solve Eq. (9).[8] To admit dislocation nucleation and migration, 3D rhombohedral elements with faces on {111} planes in a face-centered cubic (FCC) lattice or {110} planes in a body-centered cubic (BCC) lattice are used, as shown in Fig. 3. Within each element, lattice defects are not allowed and the displacement field has C^1 continuity.[8] Between elements, however, neither displacement continuity nor strain compatibility is required. In this way, lattice defects are accommodated by discontinuous displacements between elements, potentially including both sliding and separation.[11]

In the atomistic domain, Newton's third law is used to promote efficiency in calculating the force, pair potential, local electron density, and stress. In addition, due to the similarity between CAC and atomistic methods regarding the crystal structure and force/energy/stress calculations, the short-range neighbor search employs a combined cell list[54] and Verlet list[55] method. The neighbor lists, of the integration points in the coarse-grained domain and of the atoms in the atomistic domain, are updated on-the-fly when any node/atom is displaced by a distance that is larger than half a user-defined bin size. Displacement, traction, or mixed boundary conditions can be realized by assigning a displacement and/or force to the nodes/atoms within a short distance from the simulation cell boundaries. However, CAC with the present coarse-graining strategy differs from standard atomistic methods in five main aspects as follows:

(i) The rhombohedral element shape and the fact that the finite elements may take any crystallographic orientations with respect to a fixed coordinate system prevent one from constructing a parallelepipedonal coarse-grained domain. To facilitate application of the periodic boundary conditions (PBCs), one can fill in the otherwise jagged interstices at simulation cell boundaries with atoms, e.g., the dark red and dark green circles in Fig. 4 since in this example, PBCs are applied along the y direction.

(ii) The other issue in implementing PBCs is that an object crossing through one face of the simulation cell should enter the cell through the opposite face. In

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

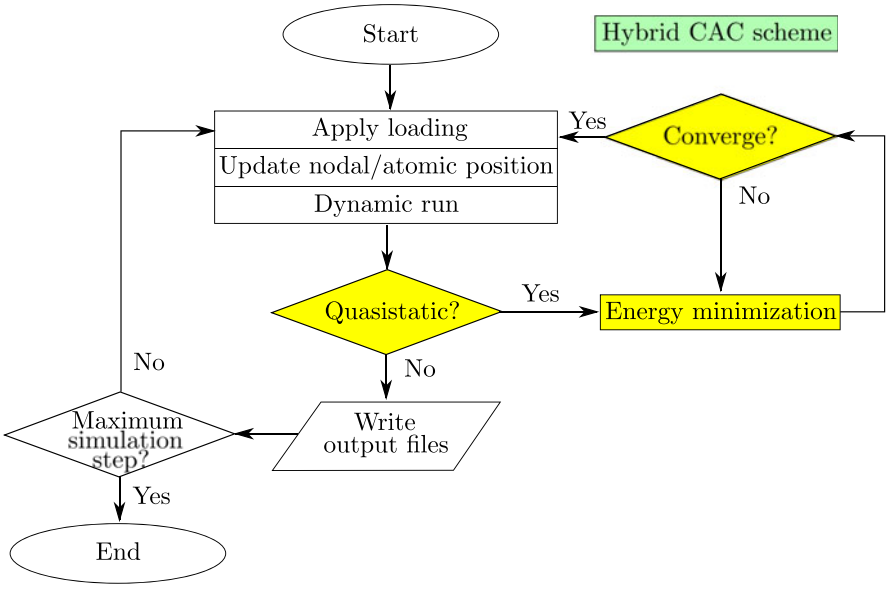


FIG. 2. Serial hybrid CAC simulation scheme. The quasistatic CAC procedures are highlighted in yellow, while the remaining procedures belong to the dynamic CAC simulation scheme. At the diamond box with "Quasistatic?", the code decides, based on some user-defined input parameters, whether it switches from dynamic run to quasistatic run.

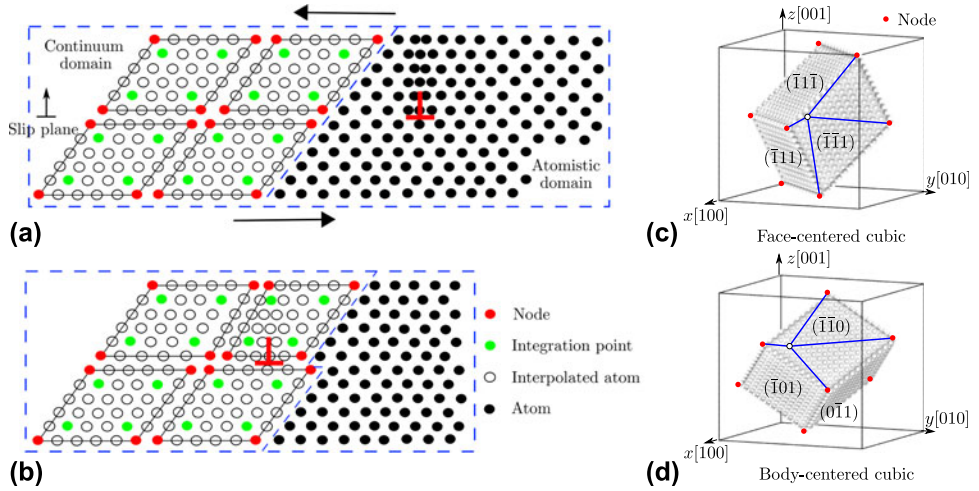


FIG. 3. (a, b) A 2D CAC simulation domain consisting of an atomistic domain (right) and a continuum domain (left).[8] In (a), an edge dislocation (red ⊥) is located in the atomistic domain. Upon applying a shear stress on the simulation cell, the dislocation migrates into the continuum domain in (b), where the Burgers vector spreads out between discontinuous finite elements. (c, d) In 3D, elements have faces on {111} planes and on {110} planes in an FCC and a BCC lattice, respectively. The positions of atoms (open circles) within each element are interpolated from the nodal positions (filled circles).

atomistic simulations, this is realized via displacement of certain atoms to bring them back inside the cell,[6] i.e.,

$$R^{\chi} = \begin{cases} R^{\chi} + L^{\chi}, & \text{if } R^{\chi} < R^{\chi}_{\text{lb}} \\ R^{\chi} - L^{\chi}, & \text{if } R^{\chi} > R^{\chi}_{\text{ub}} \end{cases}, \qquad (15)$$

where R^{χ} is the position of an atom along the χ direction, L^{χ}, R^{χ}_{lb}, and R^{χ}_{ub} are the length, lower boundary, and upper boundary of the simulation cell along the χ direction, respectively. In the coarse-grained domain, however, care must be taken when not all nodes of one element are displaced, i.e., an element is cut through by a periodic boundary.[11] In this case, the nodal positions should be reinstated to interpolate the positions of atoms within the element. Subsequently, the reinstated nodes and some of the interpolated atoms are displaced following Eq. (15). More details of this procedure can be found in Appendix C of Ref. 8.

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

(iii) Compared with the two-body Lennard-Jones (LJ) pair potential,[56] the many-body embedded-atom method (EAM) potential[57] adopts a more complicated formulation for the force **F**, i.e.,[57]

$$\mathbf{F}^k = \sum_{\substack{j \\ j \neq k}} \left\{ \frac{\partial \phi\left(R^{kj}\right)}{\partial R^{kj}} + \left[\frac{\partial \psi\left(\bar{\rho}^k\right)}{\partial \bar{\rho}^k} + \frac{\partial \psi\left(\bar{\rho}^j\right)}{\partial \bar{\rho}^j} \right] \frac{\partial \rho\left(R^{kj}\right)}{\partial R^{kj}} \right\} \frac{\mathbf{R}^{kj}}{R^{kj}} \quad , \tag{16}$$

where ϕ is the pair potential, ψ is the embedding potential, $\bar{\rho}$ is the host electron density, and $\mathbf{R}^{kj}$ is the vector from atom k to atom j with norm R^{kj}, i.e.,

$$\mathbf{R}^{kj} = \mathbf{R}^j - \mathbf{R}^k \quad , \tag{17}$$

$$\bar{\rho}^k = \sum_{\substack{j \\ j \neq k}} \rho^{kj}\left(R^{kj}\right) \quad , \tag{18}$$

where ρ is the local electron density contributed by atom j at site k.

In the atomistic domain, with the neighbor lists established for all atoms, the host electron densities $\bar{\rho}$ of all atoms are first calculated using Eq. (18), followed by the calculation of the atomic forces **F** of all atoms using Eq. (16), with the aid of Newton's third law in both calculations. In the coarse-grained domain, however, only the integration points have a neighbor list. On the other hand, in calculating the force on one integration point k using Eq. (16), one needs to know the host electron density $\bar{\rho}^j$ of its neighbors j, any of which, according to Eq. (18), requires a neighbor list for atom j, which may not be an integration point and does not naively have a neighbor list. Thus, the direct calculation of all $\bar{\rho}^j$ would require the establishment of neighbor lists for a lot of nonintegration-point atoms and would include a significant number of repeated computations because the atoms involved are located in close proximity. We found that this is computationally expensive, especially so in a large element with sparse integration points.[11] Therefore, an approximation is introduced that within one element, $\bar{\rho}$ of all interpolated atoms in one subregion is assumed equal to that of the integration point in the same subregion.[8] In this way, one needs only to calculate $\bar{\rho}$ of the integration points, increasing the efficiency substantially. It was found that this approximation only slightly affects the dislocation configuration while retaining the overall Burgers vector.[8] Note that this approximation does not apply to the simple LJ pair potential since it does not involve the electron density.

(iv) As mentioned in Sec. I, a CAC simulation cell generally consists of a coarse-grained domain and an atomistic domain, with the elements and atoms constructed following different crystallographic orientations in different grains. Elements of different sizes may exist in the coarse-grained domain. Within one grain, to distinguish between atoms and elements as well as between elements of different sizes, the term "subdomain" is introduced to refer to a region in a CAC simulation cell with only atoms or only elements of the same size, as illustrated in Fig. 4. Users can build single crystal, bicrystal, or polycrystal models with any number of subdomains and/or grains that are stacked along any Cartesian axis, e.g., the y axis in Fig. 4. The planar interface between grains and subdomains is not necessarily normal to the Cartesian axes but can have a user-defined orientation exhibited by the tilt angle θ in Fig. 4.

(v) The last issue lies in the visualization of the CAC results, which is supported by the data analyzer. Naturally, both the real atoms in the atomistic domain and the interpolated atoms in the coarse-grained domain can be visualized in the same way as in the atomistic

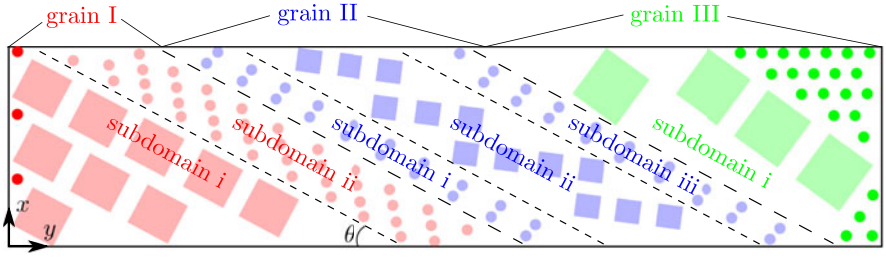


FIG. 4. A 2D schematic of the grains and subdomains in a CAC simulation cell. Squares are finite elements and circles are atoms. In grain I, there are two subdomains; subdomain i contains elements of the same size while subdomain ii is fully atomistically resolved. In grain II, there are three subdomains; subdomains i and iii have full atomistic resolution, while subdomain ii contains elements of the same size. In grain III, there is only one subdomain containing elements of the same size. The elements are rotated differently in three grains following their respective crystallographic orientations. Note that the dark red and dark green atoms at the leftmost and rightmost boundaries are added after all grains/subdomains are defined, to fill in the otherwise jagged interstices at the periodic boundaries because PBCs are applied along the y direction but not the x direction. Also note that the planar interface between grains and subdomains are not necessarily normal to the Cartesian axes but can have a user-defined orientation angle θ.

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

simulations using common atomistic configuration viewers such as OVITO,[58] AtomEye,[59] VMD,[60] and AtomViewer.[61,62] Unfortunately, these atomistic viewers do not naively support finite elements. Thus, we are in need of software that can concurrently render both elements and atoms as well as some nodal/atomic quantities such as energy, force, velocity (not for quasistatic CAC), and stress. For this purpose, the CAC simulator outputs vtk files for both elements and atoms, which can be visualized in the freely available software package ParaView,[63] such that CAC results are accessible to a larger community.

IV. PARALLELIZATION

A. Parallel algorithm

CAC simulations employ a distributed-memory parallel algorithm using the Message Passing Interface (MPI).[64] Among the three parallel algorithms commonly used in atomistic simulations—atom decomposition (AD), force decomposition (FD), and spatial decomposition (SD), SD yields the best scalability and the smallest communication overhead between processors and is therefore used in the CAC simulator.[11] At each simulation step, all atoms in the coarse-grained domain are interpolated from the nodal positions, enabling the employment of the same parallel atomistic algorithm used in the state-of-the-art atomistic modeling software LAMMPS.[65] In SD, each processor occupies a domain, with a certain number of local atoms. However, not all atoms that are within the interatomic potential cutoff distance of the local atoms are in the same processor domain. To address this problem, each processor domain is expanded outwards along the three Cartesian axes by a distance that equals the sum of the cutoff distance and a user-defined bin size; the atoms within the expanded regions are termed "ghost atoms". Then, real-time positions of the ghost atoms are exchanged between neighboring processors at each step, with the Newton's third law–induced numerical complication in the atomistic domain properly handled. The position of each node in the coarse-grained domain is also updated in this process because it is the center of mass of all atoms at that node, which is simply the atomic position at that node for monatomic crystals. Note that the ghost atoms are deleted and recreated every time the neighbor lists are updated. We emphasize that the ghost atoms, widely used in SD-based parallel atomistic simulations, are not related in any way to the "ghost force" at the continuum/atomistic domain interface in some multiscale modeling approaches, e.g., the local QC method[66]; there is no ghost force in both undeformed and affinely deformed perfect lattices in CAC because all the nodes/atoms are nonlocal. The parallel CAC simulation scheme is presented in Fig. 5.

Unlike AD and FD, the workload of each processor in SD, which is proportional to the number of interparticle interactions, is not guaranteed to be the same. On the one hand, in parallel computing, it is important to assign processors approximately the same workload which, in CAC simulations, is the calculation of force, energy, stress, position, velocity (not for quasistatic CAC), and electron density (only for the EAM potential). On the other hand, in CAC, the simulation cell has nonuniformly distributed integration points in the coarse-grained domain and atoms in the atomistic domain, such that the workload is poorly balanced if one assigns each processor an equally sized cubic domain as in full atomistics.[67] Thus, it is desirable to employ an algorithm to address the workload balance issue, which is not unique to CAC but is also encountered by other partitioned-domain multiscale modeling methods with highly heterogeneous models.[68,69]

In parallel CAC simulations involving both coarse-grained and atomistic domains, only the force/energy/host electron density (the last quantity pertains only to the EAM potential) of the integration points and atoms (referred to in combination as "evaluation points") are computed. Since the local density of interactions does not vary significantly within the simulation cell, the number of evaluation points is used as an approximation of the workload and each processor domain is assigned approximately the same number of evaluation points, which is re-evaluated at regular time intervals.[11] It follows that at periodic boundaries filled in with atoms or in the vicinity of lattice defects where full atomistics is used, the processors are assigned smaller domains that contain more atoms than other processors whose domains contain more elements.[8]

Another issue that does not exist in parallel atomistic simulations but requires special attention in parallel finite element implementations is that in the latter, some elements may be shared between neighboring processors.[11] In CAC, this issue originates from the difference in shape between the parallelepipedonal processor domain and the rhombohedral finite elements with arbitrary crystallographic orientations, the latter of which also results in the jagged simulation cell boundaries, as discussed in Sec. III.D. Instead of having all relevant processors calculate the same quantities within a shared element, in the CAC simulator, each relevant processor only calculates quantities of the integration points its domain contains; then these quantities are summed and sent to all relevant processors. This simple summation is feasible because of the trilinear shape function used in the finite elements.[8] Particular attention must be paid in the cases of (i) the EAM potential, where the host electron density approximation (Sec. III.D) should be correctly implemented because a subregion within an element may also be shared between processors and (ii) the Langevin

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

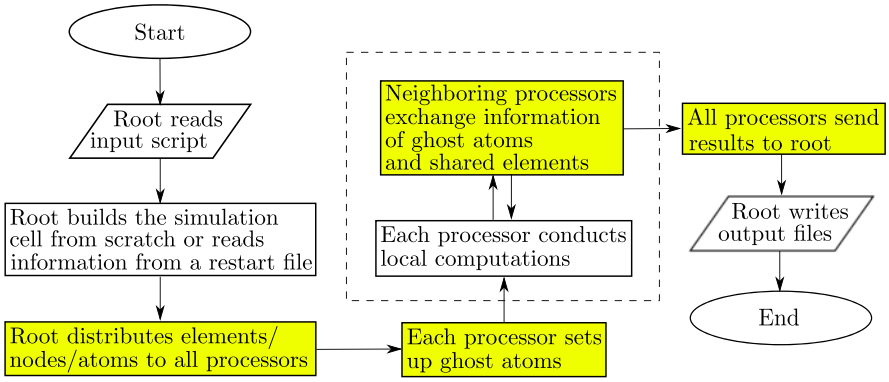


FIG. 5. Parallel CAC simulation scheme. Procedures that do not exist in the serial scheme (Fig. 1) are highlighted in yellow. Note that (i) in the serial scheme, the root processor does everything and (ii) the two procedures in the dashed box are conducted back and forth until the output begins.

dynamics, where different processors may have different $\Theta(t)$ [Eq. (14)] for the nodes of the same shared element because $\Theta(t)$ is a processor-dependent random variable; in practice, for a shared element, $\Theta(t)$ for the same node is averaged among all relevant processors. Distinct from atomistic simulations, each relevant processor needs to have the same copy of nodal positions of all its shared elements in its local memory to correctly interpolate its integration points' and atoms' neighbors. In practice, this leads to a lower parallel scalability and a poorer memory usage scaling than full atomistics.

We remark that in the CAC simulator, the input and output parts are not fully parallelized. Instead, following LAMMPS,[65] the input script is first read by the root processor, which then builds the simulation cell from scratch or reads necessary information from a restart file. The simulation cell information is stored in some global arrays accessible only to the root processor. It follows that the root processor distributes all elements/nodes/atoms to all processors (including the root itself), which store the relevant data in local arrays and conduct local computations. For the output, all processors (including the root processor) send their local results to some global arrays accessible only to the root processor, which then writes the information to the file system. All global arrays are eliminated as soon as they are used to minimize the memory usage.

B. Parallel scalability

The primary motivation for developing the CAC method, or any partitioned-domain multiscale modeling method, is to run simulations at a cost lower than that of the full atomistic method. In our early work, the coarse-graining efficiency—the serial runtime of a full coarse-grained CAC simulation divided by that of an equivalent full atomistic simulation—was found to be about 50 when all elements are of second nearest neighbor (2NN) type and each contains 9261 atoms.[8] It was also found that the coarse-graining efficiency for a quasistatic run is higher than that of a dynamic run.[8]

In this section, we conduct benchmark simulations to analyze the CAC simulator's parallel scalability. There are two basic ways to measure the parallel scalability of a given application: strong scaling and weak scaling.[70] As the number of processors increases, the problem size (e.g., number of elements/atoms in a simulation cell) remains the same in strong scaling but increases proportionally in weak scaling to keep a constant problem size per processor. Consequently, each processor communicates the same/smaller amount of data with its neighbors in weak/strong scaling, respectively. In this section, we investigate both strong (Fig. 6) and weak (Fig. 7) scaling of the CAC simulator by conducting fully coarse-grained simulations in Ni single crystals. To further analyze the parallel performance, the total runtime of each job is partitioned into four components: the time for interatomic interactions, the time for establishing/updating neighbors, the time for interprocessor communication, and the time for input/output. Each runtime component is calculated by averaging the corresponding component among all processors. The memory usage and processor workload are also analyzed. All simulations run on the Bridges cluster, with a peak speed of 1.35 petaflops per second, at National Science Foundation (NSF) Extreme Science and Engineering Discovery Environment (XSEDE).[71] In each simulation, up to 1024 processors are used on Hewlett Packard Enterprise Apollo 2000 servers using 2.30 GHz Intel Xeon E5-2695 v3 CPUs with 35 MB cache size and Intel Omni-Path Fabric interconnection. Each node has 2 CPUs (14 cores per CPU), 128 GB RAM, and 8 TB storage. We emphasize that the simulations conducted here are solely for the purpose of code performance analysis and are not intended to shed light on any realistic material response.

In strong scaling, four simulation cells, with the number of 2NN elements being 3430, 29,660, 102,690, and 246,520, respectively, are used. With a uniform element size of 2197 atoms and a lattice constant of

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

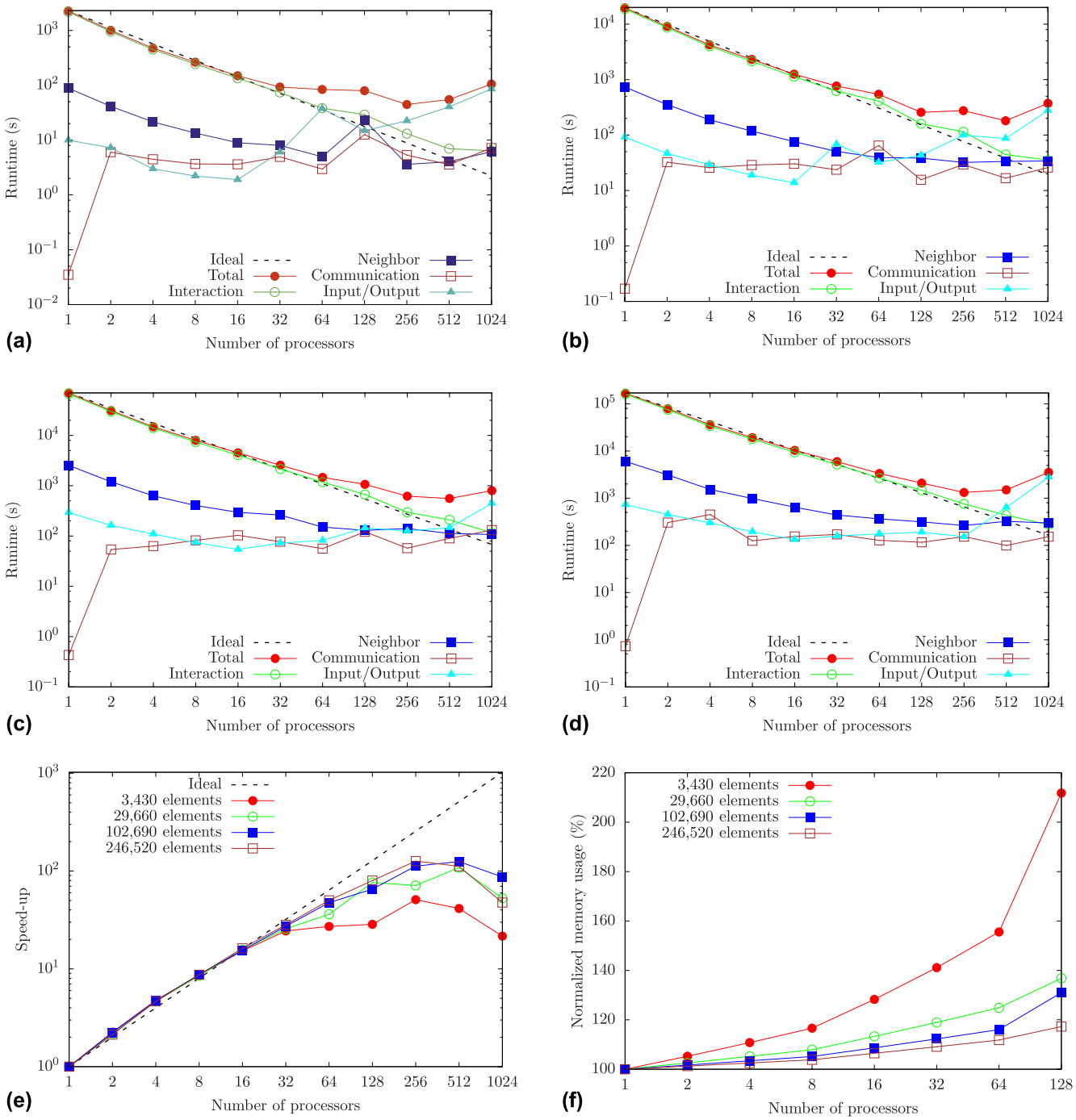


FIG. 6. (a–d) Runtime breakdown of all components as well as (e) speed-up and (f) normalized memory usage with respect to the serial CAC simulation for different simulations cell sizes in strong scaling.

3.52 Å, the largest simulation cell has a size of 182.6 × 182.6 × 182.6 nm. As a result, the smallest and the largest simulation cells correspond to equivalent full atomistic models containing about 7.5 million and about 541.6 million atoms, respectively. Traction-free boundary conditions and {100} crystallographic orientations are applied along the three directions, with the EAM potential[72] describing the interatomic interactions. Instead of building the simulation cell from scratch, the CAC simulator the system configuration from pre-existing restart files. It followed that a dynamic run with the velocity Verlet algorithm was performed under an NVE ensemble. We remark that the results presented here are representative of dynamic runs using other options (qd and ld) because all of them adopt similar equations of motion. With a time step of 2 fs, each run consists of 200 steps,

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

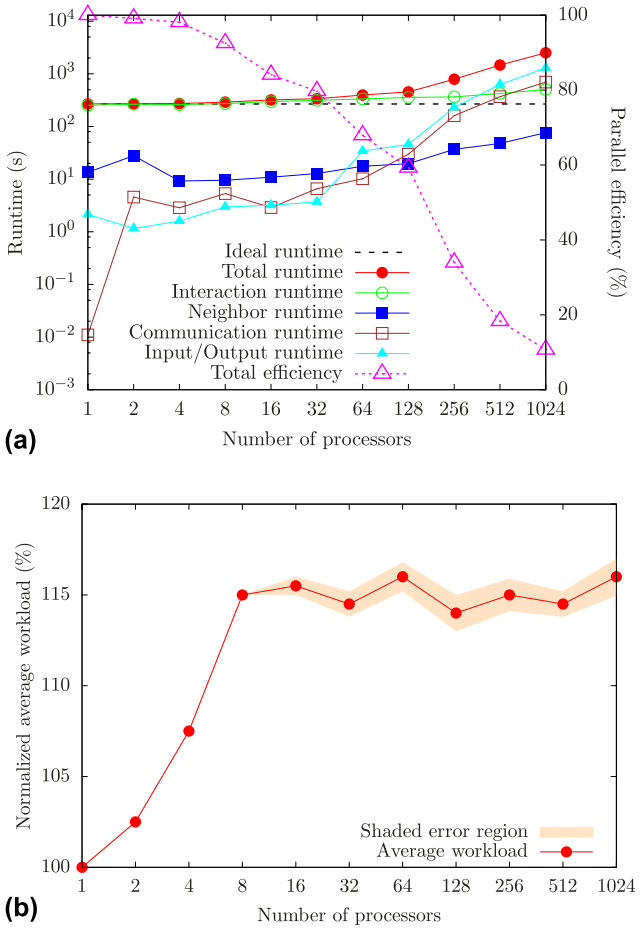


FIG. 7. (a) Runtime breakdown of all components and parallel efficiency and (b) normalized average workload with respect to the serial CAC simulation in weak scaling, with 492 elements per processor. In (b), the shaded error region represents the standard deviation of the workload of all processors.

with 1 input, 1 neighbor establishment, 2 neighbor updates, and 2 outputs.

For all simulation cells, when the number of processors is relatively small (<64 and <256 for the smallest and the largest simulation cells, respectively), the interaction and neighbor steps are shown to be the most CPU-intensive operations, with no major bottleneck caused by data input/output, similar to atomistic simulations. In all cases, when more than 1 processor is used, the inter-processor communication time is almost invariant with the number of processors. However, the runtime spent on input/ouput increases rapidly as the number of processors grows; it should be noted that the results may change significantly if different frequencies for output and neighbor updates are used. In addition, because of the shared elements, the total memory usage is no longer invariant but increases as more processors are used [Fig. 6(f)]. As expected, both parallel scalability and memory usage improve as the simulation cell size increases, as shown in Figs. 6(e) and 6(f).

In weak scaling, the number of processors in one simulation varies from 1 to 1024, with about 492 elements per processor. As a result, the largest simulation cell has a size of 231.9 × 231.9 × 231.9 nm, with 503,808 elements, corresponding to 1,106,866,176 interpolated atoms, which is more than double the previous largest EAM-based CAC models that contained about 552.7 million atoms.[16] Other simulation settings are the same as those in strong scaling. It is found that the runtime spent on data loading can be negligible compared to the very long runtime of the CAC simulation on small or medium number of processors. Nonetheless, as the number of processors increases, the runtime for interprocessor communication and input/output increases much more quickly than that for interatomic interactions and establishing/updating neighbors, as shown in Fig. 7(a). In particular, with 1024 processors, the runtime for input/output, which is not fully parallelized as discussed in Sec. IV.A, constitutes more than half the total runtime; in other words, the input/output procedure may become a new bottleneck for achieving high parallel scalability according to Amdahl's law.[73] In this regard, we will conduct further investigation to improve the parallel scalability of the CAC simulator as part of our future work. We remark that the processor workload is well balanced, as shown in Fig. 7 (b).

V. PyCAC SOFTWARE

In this section, we present the features, the Python scripting interface as well as the compilation and execution of the PyCAC software. More specific details of PyCAC, including the input script format and some example problems, can be found at www.pycac.org.

A. Features

The PyCAC code can simulate monatomic pure FCC and pure BCC metals using the LJ and EAM interatomic potentials in conservative systems. In the coarse-grained domain, 3D trilinear rhombohedral elements are used to accommodate dislocations in 9 out of 12 sets of $\{111\}\langle 110\rangle$ slip systems in an FCC lattice as well as 6 out of 12 sets of $\{110\}\langle 111\rangle$ slip systems in a BCC lattice, as shown in Figs. 3(c) and 3(d).

We remark that there is no theoretical challenge in extending the current PyCAC code to admit more slip systems, crystal structures, interatomic potentials, element types, and materials. In fact, AFT was originally designed with polyatomic crystalline materials in mind,[74,75] and CAC has been applied to a 1D polyatomic crystal with trilinear[76] and nonlinear shape functions,[36] an ideal brittle FCC crystal with 2D triangular elements[38] or with 3D tetrahedral and 3D pyramid elements,[39] silicon using the SW potential,[43] a 2D Cu single crystal with 2D quadrilateral elements,[16] 2D FCC Cu single

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

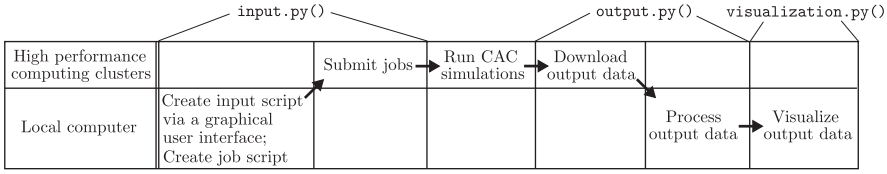


FIG. 8. Python scripting interface scheme. First, input.py() creates CAC simulation cells via a graphical user interface on a local computer and submits jobs via job schedulers on high performance computing clusters; then CAC simulations compiled from the underlying Fortran code are run following Figs. 1 or 5; once the simulations finish, output.py() downloads the output data from clusters before processing them locally using the data analyzer; in the end, visualization.py() employs some visualization software to render CAC simulation results.

crystal[47] and polycrystal[48] with 2D quadrilateral elements, and 2D superlattices with 2D quadrilateral elements[49] as well as strontium titanate using a rigid-ion potential with 3D cubic elements.[42,44,45] These simulations were conducted using other versions of CAC.

B. Python scripting interface

In PyCAC, the Python scripting interface is a Python wrapper module for CAC's underlying Fortran 2008 code, allowing handling of the program's input and output as well as links to some external visualization software. Written in Python 3 and contained in pycac.py, the Python module provides a robust user interface to facilitate parametric studies via CAC simulations without interacting with the Fortran code and to improve handling of input, output, and visualization options, as illustrated in Fig. 8. Mainly working on local computers, the Python module serves as an interface with high performance computing clusters. In particular, the module consists of three main components:

(i) input.py() for generation and manipulation of CAC simulation cells on local computer via a graphical user interface, as well as submitting jobs via job schedulers on high performance computing clusters, e.g., those on NSF XSEDE.[71] A CAC simulation requires at least two types of files as input: an input script and the analytic or tabulated interatomic potential files. If the users intend to continue a previous run, a restart file with previously saved system configuration needs to be provided as well. Note that some or all elements from a previous run can be refined into atomic scale upon reading the restart file. This is useful, for example, to correctly simulate the migration of a dislocation whose pathway is not aligned with the interelemental gap.[15]

(ii) output.py() for downloading the CAC simulation output data from clusters before processing them locally. A CAC simulation outputs three main types of files: (i) vtk files in the legacy ASCII format containing elemental, nodal, and atomic information (positions, energies, forces, stresses, etc.); (ii) restart files containing system configuration information; and (iii) a log file recording relevant lumped simulation data, e.g., system force/energy and number of elements/atoms, during a run. It follows that the output.py() component reads the vtk files and interpolates all atoms inside the elements in the coarse-grained domain. These interpolated atoms, together with the real atoms in the atomistic domain (also read from the vtk files), are used to generate LAMMPS dump files that can be visualized by atomistic model viewers and/or read by LAMMPS directly to carry out equivalent fully-resolved atomistic simulations. CAC simulation results may also be processed via the data analyzer which will be presented in the near future.

(iii) visualization.py() for integrating some visualization software. Currently, the dump files and vtk files are visualized by OVITO[58] and ParaView,[63] respectively, as discussed in Sec. III.D.

C. Compilation and Execution

In PyCAC, the CAC algorithm is implemented in Fortran 2008 and parallelized with MPI-3, wrapped by a Python scripting interface written in Python 3. Therefore, the PyCAC code can be compiled with any MPI, Fortran, and Python compilers that support MPI-3, Fortran 2008, and Python 3, respectively. For example, the parallel scalability benchmark simulations in Figs. 6 and 7 were run using an executable compiled by Intel Fortran compiler version 17.0 wrapped by MVAPICH2 version 2.3.

Potential users may explore PyCAC by running simulations via the MATerials Innovation Network (MATIN, https://matin.gatech.edu),[77] a cloud-based e-collaboration platform for accelerating materials innovation developed at the Georgia Institute of Technology (GT) with support of the GT Institute for Materials. While MATIN is focused on emergent field of materials data science and informatics, it offers a comprehensive range of functionality targeting the materials science and engineering domain. After signing up for a free account, a user can run the PyCAC code via MATIN on GT's Partnership for an Advanced Computing Environment (PACE) cluster or on NSF XSEDE.[a]

VI. CONCLUSIONS

In this work, we present a novel implementation of the CAC method—PyCAC—an efficient, user-friendly, and extensible CAC simulation environment which is

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

implemented mainly in Fortran 2008 and wrapped with a Python scripting interface. First, the theoretical background of the CAC method was briefly reviewed. Then, the serial algorithms of dynamic, quasistatic, and hybrid CAC, along with some programming techniques used in the PyCAC code were discussed. In addition, the parallel algorithm/scalability of the CAC simulator and some specific details of the PyCAC software were presented. Additional information of PyCAC can be found at www.pycac.org.

While the CAC method has a high coarse-graining efficiency[8] and the PyCAC software shows good scaling performance, there is still room for improvement in the code performance by parallelizing the input/output. Future development of code features will involve implementation of more types of finite elements and interatomic potentials to accommodate more crystalline materials such as multicomponent and polyatomic materials. New types of finite elements will also be used to replace the filled-in atoms at the otherwise zigzag simulation cell boundaries to reduce the computational cost, particularly in the case of PBCs. Other versions of the CAC simulator, e.g., the one in Ref. 78, may be incorporated into PyCAC. Future extensions also include developing adaptive mesh refinement schemes for dislocation migration, in which case the evaluation point density may evolve with time, requiring more complicated algorithms to dynamically re-balance the processor workload.

ACKNOWLEDGMENTS

These results are based upon work supported by the National Science Foundation as a collaborative effort between Georgia Tech (CMMI-1232878) and University of Florida (CMMI-1233113). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors thank Dr. Jinghong Fan, Dr. Qian Deng, Dr. Shengfeng Yang, Dr. Xiang Chen, Mr. Rui Che, Mr. Weixuan Li, and Mr. Ji Rigelesaiyin for helpful discussions, and Dr. Aleksandr Blekh for arranging execution of PyCAC via MATIN. The work of SX was supported in part by Georgia Tech Institute for Materials and in part by the Elings Prize Fellowship in Science offered by the California NanoSystems Institute (CNSI) on the UC Santa Barbara campus. SX also acknowledges support from the Center for Scientific Computing from the CNSI, MRL: an NSF MRSEC (DMR-1121053). LX acknowledges the support from the Department of Energy, Office of Basic Energy Sciences under Award Number DE-SC0006539. The work of LX was also supported in part by the National Science Foundation under Award Number CMMI-1536925. DLM is grateful for the additional support of the Carter N. Paden, Jr. Distinguished Chair in Metals Processing. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

END NOTE

a. Currently available resource on XSEDE is San Diego Supercomputer Center's Comet cluster; integration with other XSEDE resources is planned for the future. Users desiring to run PyCAC (and other available materials informatics tools) for large-scale simulation/modeling projects should have their own compute/storage allocations on PACE or XSEDE and contact MATIN Project Lead (Aleksandr Blekh, aleksandr.blekh@gatech.edu) to discuss relevant integration and collaboration.

REFERENCES

1. F.F. Abraham, J.Q. Broughton, N. Bernstein, and E. Kaxiras: Spanning the continuum to quantum length scales in a dynamic simulation of brittle fracture. *Europhys. Lett.* **44**, 783–787 (1998).
2. D.L. McDowell: A perspective on trends in multiscale plasticity. *Int. J. Plast.* **26**, 1280–1309 (2010).
3. V. Bulatov, F.F. Abraham, L. Kubin, B. Devincre, and S. Yip: Connecting atomistic and mesoscale simulations of crystal plasticity. *Nature* **391**, 669–672 (1998).
4. D.E. Spearot and M.D. Sangid: Insights on slip transmission at grain boundaries from atomistic simulations. *Curr. Opin. Solid State Mater. Sci.* **18**, 188–195 (2014).
5. R. Phillips: Multiscale modeling in the mechanics of materials. *Curr. Opin. Solid State Mater. Sci.* **3**, 526–532 (1998).
6. E.B. Tadmor and R.E. Miller: *Modeling Materials: Continuum, Atomistic and Multiscale Techniques* (Cambridge University Press, New York, 2012).
7. L. Xiong, G. Tucker, D.L. McDowell, and Y. Chen: Coarse-grained atomistic simulation of dislocations. *J. Mech. Phys. Solids*, **59**, 160–177 (2011).
8. S. Xu, R. Che, L. Xiong, Y. Chen, and D.L. McDowell: A quasistatic implementation of the concurrent atomistic-continuum method for FCC crystals. *Int. J. Plast.* **72**, 91–126 (2015).
9. Y. Chen, J. Zimmerman, A. Krivtsov, and D.L. McDowell: Assessment of atomistic coarse-graining methods. *Int. J. Eng. Sci.* **49**, 1337–1349 (2011).
10. Y. Chen, J. Lee, and L. Xiong: A generalized continuum theory and its relation to micromorphic theory. *J. Eng. Mech.* **135**, 149–155 (2009).
11. S. Xu: The concurrent atomistic-continuum method: Advancements and applications in plasticity of face-centered cubic metals. Ph.D. thesis, Georgia Institute of Technology, 2016.
12. J. Knap and M. Ortiz: An analysis of the quasicontinuum method. *J. Mech. Phys. Solids* **49**, 1899–1923 (2001).
13. B. Eidel and A. Stukowski: A variational formulation of the quasicontinuum method based on energy sampling in clusters. *J. Mech. Phys. Solids* **57**, 87–108 (2009).
14. S. Xu, L. Xiong, Y. Chen, and D.L. McDowell: An analysis of key characteristics of the Frank–Read source process in FCC metals. *J. Mech. Phys. Solids* **96**, 460–476 (2016).
15. S. Xu, L. Xiong, Q. Deng, and D.L. McDowell: Mesh refinement schemes for the concurrent atomistic-continuum method. *Int. J. Solids Struct.* **90**, 144–152 (2016).
16. L. Xiong, J. Rigelesaiyin, X. Chen, S. Xu, D.L. McDowell, and Y. Chen: Coarse-grained elastodynamics of fast moving dislocations. *Acta Mater.* **104**, 143–155 (2016).
17. S. Xu, L. Xiong, Y. Chen, and D.L. McDowell: Shear stress- and line length-dependent screw dislocation cross-slip in FCC Ni. *Acta Mater.* **122**, 412–419 (2017).

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

18. L. Xiong, S. Xu, D.L. McDowell, and Y. Chen: Concurrent atomistic-continuum simulations of dislocation-void interactions in fcc crystals. *Int. J. Plast.* **65**, 33–42 (2015).
19. S. Xu, L. Xiong, Y. Chen, and D.L. McDowell: Validation of the concurrent atomistic-continuum method on screw dislocation/stacking fault interactions. *Crystals* **7**, 120 (2017).
20. S. Xu, L. Xiong, Y. Chen, and D.L. McDowell: Edge dislocations bowing out from a row of collinear obstacles in Al. *Scr. Mater.* **123**, 135–139 (2016).
21. S. Xu, L. Xiong, Y. Chen, and D.L. McDowell: Sequential slip transfer of mixed-character dislocations across Σ3 coherent twin boundary in FCC metals: A concurrent atomistic-continuum study. *npj Comput. Mater.* **2**, 15016 (2016).
22. S. Xu, L. Xiong, Y. Chen, and D.L. McDowell: Comparing EAM potentials to model slip transfer of sequential mixed character dislocations across two symmetric tilt grain boundaries in Ni. *JOM* **69**, 814–821 (2017).
23. Y. Chen and J. Lee: Atomistic formulation of a multiscale field theory for nano/micro solids. *Philos. Mag.* **85**, 4095–4126 (2005).
24. Y. Chen: Reformulation of microscopic balance equations for multiscale materials modeling. *J. Chem. Phys.* **130**, 134706 (2009).
25. J.H. Irving and J.G. Kirkwood: The statistical mechanical theory of transport processes. IV. The equations of hydrodynamics. *J. Chem. Phys.* **18**, 817–829 (1950).
26. C. Kittel: *Introduction to Solid State Physics*, 8th ed. (Wiley, Hoboken, NJ, 2004).
27. L. Xiong, Y. Chen, and J.D. Lee: Atomistic simulation of mechanical properties of diamond and silicon carbide by a field theory. *Modell. Simul. Mater. Sci. Eng.* **15**, 535–551 (2007).
28. L. Xiong and Y. Chen: Coarse-grained simulations of single-crystal silicon. *Modell. Simul. Mater. Sci. Eng.* **17**, 035002 (2009).
29. Y. Chen: The origin of the distinction between microscopic formulas for stress and Cauchy stress. *Europhys. Lett.* **116**, 34003 (2016).
30. Y. Chen and A. Diaz: Local momentum and heat fluxes in transient transport processes and inhomogeneous systems. *Phys. Rev. E* **94**, 053309 (2016).
31. W.C. Swope, H.C. Andersen, P.H. Berens, and K.R. Wilson: A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *J. Chem. Phys.* **76**, 637–649 (1982).
32. D. Sheppard, R. Terrell, and G. Henkelman: Optimization methods for finding minimum energy paths. *J. Chem. Phys.* **128**, 134106 (2008).
33. A. Brünger, C.L. Brooks, III, and M. Karplus: Stochastic boundary conditions for molecular dynamics simulations of ST2 water. *Chem. Phys. Lett.* **105**, 495–500 (1984).
34. D.J. Evans and G. Morriss: *Statistical Mechanics of Nonequilibrium Liquids*, 2nd ed. (Cambridge University Press, Cambridge, 2008).
35. M.E. Tuckerman: *Statistical Mechanics: Theory and Molecular Simulation*, 1st ed. (Oxford University Press, Oxford, New York, 2010).
36. X. Chen, A. Diaz, L. Xiong, D.L. McDowell, and Y. Chen: Passing waves from atomistic to continuum. *J. Comput. Phys.* **354**, 393–402 (2018).
37. H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola, and J.R. Haak: Molecular dynamics with coupling to an external bath. *J. Chem. Phys.* **81**, 3684–3690 (1984).
38. Q. Deng, L. Xiong, and Y. Chen: Coarse-graining atomistic dynamics of brittle fracture by finite element method. *Int. J. Plast.* **26**, 1402–1414 (2010).
39. Q. Deng and Y. Chen: A coarse-grained atomistic method for 3D dynamic fracture simulation. *Int. J. Multiscale Comput. Eng.* **11**, 227–237 (2013).
40. L. Xiong and Y. Chen: Coarse-grained atomistic modeling and simulation of inelastic material behavior. *Acta Mech. Solida Sin.* **25**, 244–261 (2012).
41. L. Xiong, Q. Deng, G. Tucker, D.L. McDowell, and Y. Chen: A concurrent scheme for passing dislocations from atomistic to continuum domains. *Acta Mater.* **60**, 899–913 (2012).
42. S. Yang, L. Xiong, Q. Deng, and Y. Chen: Concurrent atomistic and continuum simulation of strontium titanate. *Acta Mater.* **61**, 89–102 (2013).
43. L. Xiong, D.L. McDowell, and Y. Chen: Nucleation and growth of dislocation loops in Cu, Al, and Si by a concurrent atomistic-continuum method. *Scr. Mater.* **67**, 633–636 (2012).
44. S. Yang, N. Zhang, and Y. Chen: Concurrent atomistic-continuum simulation of polycrystalline strontium titanate. *Philos. Mag.* **95**, 2697–2716 (2015).
45. S. Yang and Y. Chen: Concurrent atomistic and continuum simulation of bi-crystal strontium titanate with tilt grain boundary. *Proc. R. Soc. London, Ser. A* **471**, 20140758 (2015).
46. L. Xiong, D.L. McDowell, and Y. Chen: Sub-THz phonon drag on dislocations by coarse-grained atomistic simulations. *Int. J. Plast.* **55**, 268–278 (2014).
47. X. Chen, L. Xiong, D.L. McDowell, and Y. Chen: Effects of phonons on mobility of dislocations and dislocation arrays. *Scr. Mater.* **137**, 22–26 (2017).
48. X. Chen, W. Li, L. Xiong, Y. Li, S. Yang, Z. Zheng, D.L. McDowell, and Y. Chen: Ballistic-diffusive phonon heat transport across grain boundaries. *Acta Mater.* **136**, 355–365 (2017).
49. X. Chen, W. Li, A. Diaz, Y. Li, Y. Chen, and D.L. McDowell: Recent progress in the concurrent atomistic-continuum method and its application in phonon transport. *MRS Commun.*, **7**, 785–797 (2017).
50. S. Chapra and R. Canale: *Numerical Methods for Engineers*, 6th ed. (McGraw-Hill Science/Engineering/Math, Boston, 2009).
51. E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, and P. Gumbsch: Structural relaxation made simple. *Phys. Rev. Lett.* **97**, 170201 (2006).
52. B. Eidel, A. Hartmaier, and P. Gumbsch: Atomistic simulation methods and their application on fracture. In *Multiscale Modelling of Plasticity and Fracture by Means of Dislocation Mechanics*, 1st ed., P. Gumbsch and R. Pippan, eds.; CISM International Centre for Mechanical Sciences (Springer, Vienna, 2010); pp. 1–57. doi: 10.1007/978-3-7091-0283-1_1.
53. E.B. Tadmor and R.E. Miller: *Modeling Materials: Continuum, Atomistic and Multiscale Techniques*, 1st ed. (Cambridge University Press, Cambridge, New York, 2012).
54. M.P. Allen and D.J. Tildesley: *Computer Simulation of Liquids* (Oxford University Press, New York, 1989).
55. L. Verlet: Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Phys. Rev.* **159**, 98–103 (1967).
56. J.E. Jones: On the determination of molecular fields. II. From the equation of state of a gas. *Proc. R. Soc. London, Ser. A* **106**, 463–477 (1924).
57. M.S. Daw and M.I. Baskes: Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B*, **29**, 6443–6453 (1984).
58. A. Stukowski: Visualization and analysis of atomistic simulation data with OVITO—The open visualization tool. *Modell. Simul. Mater. Sci. Eng.* **18**, 015012 (2010).
59. J. Li: AtomEye: An efficient atomistic configuration viewer. *Modell. Simul. Mater. Sci. Eng.* **11**, 173 (2003).
60. W. Humphrey, A. Dalke, and K. Schulten: VMD: Visual molecular dynamics. *J. Mol. Graphics* **14**, 33–38 (1996).
61. C. Begau, A. Hartmaier, E.P. George, and G.M. Pharr: Atomistic processes of dislocation generation and plastic deformation during nanoindentation. *Acta Mater.* **59**, 934–942 (2011).

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press

62. C. Begau, J. Hua, and A. Hartmaier: A novel approach to study dislocation density tensors and lattice rotation patterns in atomistic simulations. *J. Mech. Phys. Solids* **60**, 711–722 (2012).
63. W. Schroeder, K. Martin, and B. Lorensen: *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 4th ed. (Kitware, Clifton Park, New York, 2006).
64. W. Gropp, T. Hoefler, R. Thakur, and E. Lusk: *Using Advanced MPI: Modern Features of the Message-Passing Interface*, 1st ed. (The MIT Press, Cambridge, Massachusetts, 2014).
65. S. Plimpton: Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* **117**, 1–19 (1995).
66. E.B. Tadmor, M. Ortiz, and R. Phillips: Quasicontinuum analysis of defects in solids. *Philos. Mag. A* **73**, 1529–1563 (1996).
67. O. Pearce, T. Gamblin, B.R. de Supinski, T. Arsenlis, and N.M. Amato: Load balancing N-body simulations with highly non-uniform density. In *Proceedings of the 28th ACM International Conference on Supercomputing, ICS'14* (ACM, New York, NY, 2014); pp. 113–122.
68. F. Pavia and W.A. Curtin: Parallel algorithm for multiscale atomistic/continuum simulations using LAMMPS. *Modell. Simul. Mater. Sci. Eng.* **23**, 055002 (2015).
69. E. Biyikli and A.C. To: Multiresolution molecular mechanics: Implementation and efficiency. *J. Comput. Phys.* **328**, 27–45 (2017).
70. A. Hunter, F. Saied, C. Le, and M. Koslowski: Large-scale 3D phase field dislocation dynamics simulations on high-performance architectures. *Int. J. High Perform. Comput. Appl.* **25**, 223–235 (2011).
71. J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G.D. Peterson, R. Roskies, J.R. Scott, and N. Wilkins-Diehr: XSEDE: Accelerating scientific discovery. *Comput. Sci. Eng.* **16**, 62–74 (2014).
72. Y. Mishin, D. Farkas, M.J. Mehl, and D.A. Papaconstantopoulos: Interatomic potentials for monoatomic metals from experimental data and ab initio calculations. *Phys. Rev. B* **59**, 3393–3407 (1999).
73. G.M. Amdahl: Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18–20, 1967, Spring Joint Computer Conference, AFIPS'67* (ACM, Spring, New York, NY, 1967); pp. 483–485.
74. Y. Chen and J.D. Lee: Connecting molecular dynamics to micromorphic theory. II. Balance laws. *Phys. A* **322**, 377–392 (2003).
75. Y. Chen and J.D. Lee: Connecting molecular dynamics to micromorphic theory. I. Instantaneous and averaged mechanical variables. *Phys. A* **322**, 359–376 (2003).
76. L. Xiong, X. Chen, N. Zhang, D.L. McDowell, and Y. Chen: Prediction of phonon properties of 1D polyatomic systems using concurrent atomistic-continuum simulation. *Arch. Appl. Mech.* **84**, 1665–1675 (2014).
77. S.R. Kalidindi, D.B. Brough, S. Li, A. Cecen, A.L. Blekh, F.Y.P. Congo, and C. Campbell: Role of materials data science and informatics in accelerated materials innovation. *MRS Bull.* **41**, 596–602 (2016).
78. H. Chen, S. Xu, W. Li, J. Rigelesaiyin, T. Phan, and L. Xiong: A spatial decomposition parallel algorithm for a concurrent atomistic-continuum simulator and its preliminary applications, *Comput. Mater. Sci*. **144**, 1–10 (2018).

Supplementary Material

To view supplementary material for this article, please visit https://doi.org/10.1557/jmr.2018.8.

https://doi.org/10.1557/jmr.2018.8 Published online by Cambridge University Press