Research Article

An asynchronous and task-based implementation of peridynamics utilizing HPX—the C++ standard library for parallelism and concurrency

Patrick Diehl[1] · Prashant K. Jha[2] · Hartmut Kaiser[1] · Robert Lipton[3] · Martin Lévesque[4]

Received: 1 April 2020 / Accepted: 29 October 2020 / Published online: 4 December 2020

Abstract
On modern supercomputers, asynchronous many task systems are emerging to address the new architecture of computational nodes. Through this shift of increasing cores per node, a new programming model with focus on handling of the fine-grain parallelism with increasing amount of cores per computational node is needed. Asynchronous Many Task (AMT) run time systems represent a paradigm for addressing the fine-grain parallelism. They handle the increasing amount of threads per node and concurrency. HPX, a open source C++ standard library for parallelism and concurrency, is one AMT which is conforming to the C++ standard. Motivated by the impressive performance of asynchronous task-based parallelism through HPX to N-body problem and astrophysics simulation, in this work, we consider its application to the Peridynamics theory. Peridynamics is a non-local generalization of continuum mechanics tailored to address discontinuous displacement fields arising in fracture mechanics. Peridynamics requires considerable computing resources, owing to its non-local nature of formulation, offering a scope for improved computing performance via asynchronous task-based parallelism. Our results show that HPX-based peridynamic computation is scalable, and the scalability is in agreement with the theory. In addition to the scalability, we also show the validation results and the mesh convergence results. For the validation, we consider implicit time integration and compare the result with the classical continuum mechanics (CCM) (peridynamics under small deformation should give similar results as CCM). For the mesh convergence, we consider explicit time integration and show that the results are in agreement with theoretical claims in previous works.

Keywords Peridynamics · finite difference approximation · concurrency · parallelization · HPX · task-based programming · asynchronous many task systems

1 Introduction

Modern supercomputers' many core architectures, like field-programmable gate arrays (FPGAs) and Intel Knights Landing, provide more threads per computational node as before [45, 53]. Through this shift of increasing cores per node, a new programming model with the focus on handling of the fine-grain parallelism with increasing amount of cores per computational node is needed.

Asynchronous Many Task (AMT) [56] run time systems represent an emerging paradigm for addressing the fine-grain parallelism since they handle the increasing amount of threads per node and concurrency [43]. Existing task-based programming models can be classified into three classes: (1) Library solutions, e.g., StarPU [1], Intel TBB [42], Argobots [48], Qthreads [59], Kokkos [13], and HPX [25] (2) language extensions, e.g., Intel Cilk Plus [3] and OpenMP

✉ Patrick Diehl, patrickdiehl@lsu.edu | [1]Center of Computation and Technology, Louisiana State University, Baton Rouge, LA, USA. [2]Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX, USA. [3]Department of Mathematics, Louisiana State University, Baton Rouge, LA, USA. [4]Department of Multi scale Mechanics, Polytechnique Montreal, Montreal, QC, Canada.

4.0 [37], and (3) programming languages, e.g., Chapel [4], Intel ISPC [41], and X10 [5].

Most of the tasked-based libraries are based on C or C++ programming language. The C++ 11 programming language standard laid the foundation for concurrency by introducing futures, that have shown to support the concept of futurization to enable a task-based parallelism [54]. In addition, the support for parallel execution with the so-called parallel algorithms were introduced in the C++ 17 standard [55].

HPX is an open source asynchronous many task run time system that focuses on high performance computing [25]. HPX provides wait-free asynchronous execution and futurization for synchronization. It also features parallel execution policies utilizing a task scheduler, which enables a fine-grained load balancing parallelization and synchronization due to work stealing. HPX's application programming interface (API) is in strict adherence to the C++ 11 [54] and C++ 17 standard API definitions [55]. For example for the concept of futurization the `hpx:future` can be replaced by `std::future` without breaking the API.

The HPX library was recently utilized to parallelize a N-Body code using the asynchronous task-based implementation and was compared against non-AMT implementations [16, 28]. In many cases, the HPX implementation outperformed the MPI/OpenMP implementation. The HPX library has been utilized in an astrophysics application describing the time evolution of a rotating star. Test run on a Xeon Phi resulted in a speedup by factor of two with respect to a 24-core Skylake SP platform [40]. On the NERSC's Cori supercomputer, the simulation of the Merger of two stars could achieve 96.8% parallel efficiency on 648,280 Intel Knight's landing cores [17] and 68.1% parallel efficiency on 2048 cores [7] of Swiss National Supercomputing Centre's Piz Daint supercomputer. In addition, a speedup up to 2.8x was shown on full system run on Piz Daint by replacing the Message Passing Interface (MPI) with libfabric [15] for communication. Motivated by the acceleration seen for these problems, we consider the application of HPX to peridynamics theory.

Peridynamics is a non-local generalization of continuum mechanics, tailored to address discontinuous displacement fields that arise in fracture mechanics [12, 20]. Several peridynamics implementations utilizing the EMU nodal discretization [39] are available. Peridigm [38] and PDLammps [39] rely on the widely used MPI for the inter-node parallelization. Other approaches rely on acceleration cards, like OpenCL [36] and CUDA [9], for parallelization. These single device GPU approaches, however, cannot deal with large node clouds due to current hardware memory limitations on GPUs.

In peridynamics, each point in domain interacts with neighboring points within specified non-local length scale δ referred to as horizon. In numerical discretization, the mesh size is typically much smaller than the horizon, meaning that the mesh nodes, in addition to depending on the adjacent nodes of common finite element, depends on the mesh nodes within distance δ (horizon). This non-local nature of calculation makes the peridynamics-based simulation slow as compared to the local formulation such as linear elastodynamics. To reduce the computational burden, several versions of local to non-local coupling methods, in which the non-local calculations are restricted to smaller portions of domain and elsewhere local calculations arising from elastodynamics formulation, have been proposed [8]. While these methods reduce the cost considerably, the problem of efficient compute resource utilization for non-local calculations remains.

This work presents two peridynamics models discretized with the EMU nodal discretization making use of the features of AMT within HPX. We show in this work how to take advantage of the fine-grain parallelism arising on modern supercomputers. In addition, the speed-up S and the parallel efficiency E are shown in Figs. 17 and 18. The implementation is validated against results from classical continuum mechanics and numerical analysis to show that the novel task-based implementation is correct.

The paper is structured as follows. Section 2 briefly introduces HPX and associated key concepts utilized in the asynchronous task-based implementation. Section 3 reviews peridynamics models, and the EMU-ND discretization. Section 4 describes the proposed modular design and the implementation using HPX. Section 5 presents the validation and the mesh convergence results. The actual computational time for the HPX peridynamics implementation is benchmarked against the theoretical computation time in Sect. 6. Section 7 concludes this work.

2 HPX—an open source C++ standard library for parallelism and concurrency

The HPX library [25] is a C++ standard compliant Asynchronous Many Task (AMT) run time system tailored for high performance computing (HPC) applications. Figure 1 shows the three components provided by HPX. First is the *thread manager* [26], which provides a high-level API for the task-based programming and deals with the light-weight user level threads. Second is the *Active Global Address Space (AGAS)* [26], which provides the global identifiers to hide the explicit message passing. So the access of a remote or local object is unified. Third is the *parcel*

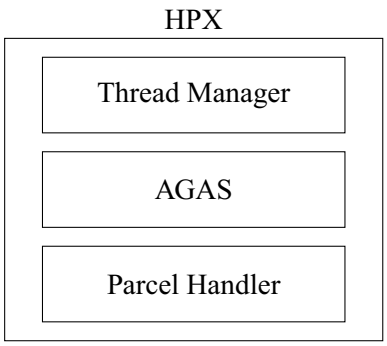


Fig. 1 Run time components of HPX: Thread manager, Active Global Address Space (AGAS), and the Parcel handler. The thread manager provides a high-level API for the task-based programming and deals with the lightweight user level threads. The Active Global Address Space (AGAS) provides global identifiers for all allocated memory. The parcel handler provides the communication between different computational nodes in the cluster

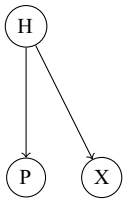
handler [2, 24] for communication between computational nodes in the cluster, which provides calls to remote computational nodes in a C++ fashion. The communication between the nodes is realized either by the transmission control protocol (tcp) or the message passing interface (MPI). For more implementation details about these components, we refer to [25] and for a general overview we refer to [56].

HPX provides well-known concepts such as data flow, futurization, and Continuation Passing Style (CPS), as well as new and unique overarching concepts. The combination of these concepts results in a unique model. The concept of futurization and parallel for loops, which are used to synchronize and parallelize, is recalled here.

2.1 Futurization

An important concept for synchronization provided by HPX is futurization. The API provides an asynchronous return type `hpx::lcos::future<T>`. This return type, a so-called future, is based on templates and hides the function call return type. The difference here is that the called function immediately returns, even if the return value is not computed. The return value of a future is accessible through the `.get()` operator that is an synchronous call and waits until the return type is computed. The standard-conforming API functions `hpx::wait_all`, `hpx::wait_any`, and `hpx::lcos::future<T>::then` are available for combining futures and generate parallel executions graphs [27].

Fig. 2 Example dependency graph




```
1 //Vector with all dependencies of h
2 std::vector<hpx::lcos::future<void>>
      dependencies;
3 dependcy.push_back(compute(p));
4 dependcy.push_back(compute(x));
5 hpx::wait_all(dependencies);
6 compute(h,p,x);
```

Listing 1: Modeling the dependency graph in Figure 2 with composition in HPX.

A typical example dependency graph is shown in Fig. 2. On the figure, *H* depends asynchronously on *P* and *X*. Listing 1 provides these dependencies resolutions within HPX. First, a `std::vector` is utilized to collect the dependencies. Second, the computations futures are added to this vector. Note, that the `compute` function returns a future in both cases, immediately, and the computations are executed asynchronously. Third, a barrier with `hpx::wait_all` for the dependencies has to be defined before *H* can be computed. HPX internally ensures that the function in line 6 is only called when the previous two futures computations are finished.

2.2 Parallelization

Consider the addition of *n* elements for the two vectors `p` and `x`, where the sum is stored piece-wise in vector `h`. Listing 2 shows the sequential approach for this computation, while Listing 3 shows the same computational task but the sum is executed in parallel. The `for` loop is replaced with `hpx::parallel::for_loop` which is conforming with the C++ 17 standard [55]. In other words, `hpx::parallel::for_loop` can be replaced by `std::for_each` which is currently an experimental feature in the GNU compiler collection 9 and Microsoft VS 2017 15.5. The first argument defines the execution policy while the second and third define the loop range. The last argument is the lambda function, executed in parallel for all *i* ranging from 0 to *z*. Note that only two lines of codes are required to execute the code in parallel. The parallel

for loop can be combined with futurization for synchronization. Therefore, the parallel execution policy is changed to `hpx::parallel::execution::par(hpx::parallel::execution::task)`.

```
//Sequential loop
for (size_t i=0,i<z;i++)
{
   h[i] = p[i]+x[i];
}
```

Listing 2: C++ code for storing the sum of two vectors sequentially in third vector.

```
//Synchronizing parallel for loop
hpx::parallel::for_loop(
hpx::parallel::execution::par,
0,z,[h,p,x](boost::uint64_t i)
{
   h[i] = p[i]+x[i];
});
```

Listing 3: HPX equivalent code for storing the sum of two vectors parallel in a third vector.

The future can now be synchronized with other futures. Listing 4 shows an example for synchronization. Vectors `p` and `x` are independently manipulated before the pairwise sum is computed. Therefore, the execution policy is changed and the futures of the manipulations are synchronized with the third future in line 26. Here, the `hpx::wait_all` ensures that the manipulations are finished and `then` describes the sum's dependencies.

```
1  std::vector<hpx::lcos::future<void>> dep;
2
3  dep.push_back(hpx::parallel::for_loop(
4  hpx::parallel::execution::par(
5  hpx::parallel::execution::task),
6  0,z,[p](boost::uint64_t i)
7  {
8     p[i] = p[i]+1;
9  });
10 dep.push_back(hpx::parallel::for_loop(
11 hpx::parallel::execution::par(
12 hpx::parallel::execution::task),
13 0,z,[x](boost::uint64_t i)
14 {
15    x[i] = x[i]-1;
16 });)
17
18 hpx::lcos::future f = hpx::parallel::for_loop(
19 hpx::parallel::execution::par(
20 hpx::parallel::execution::task),
21 0,z,[h,p,x](boost::uint64_t i)
22 {
23    h[i] = p[i]+x[i];
24 });
25
26 hpx::wait_all(dep).then(f);
```

Listing 4: Example for the synchronization of three parallel for loops within HPX by using the concept of futurization.

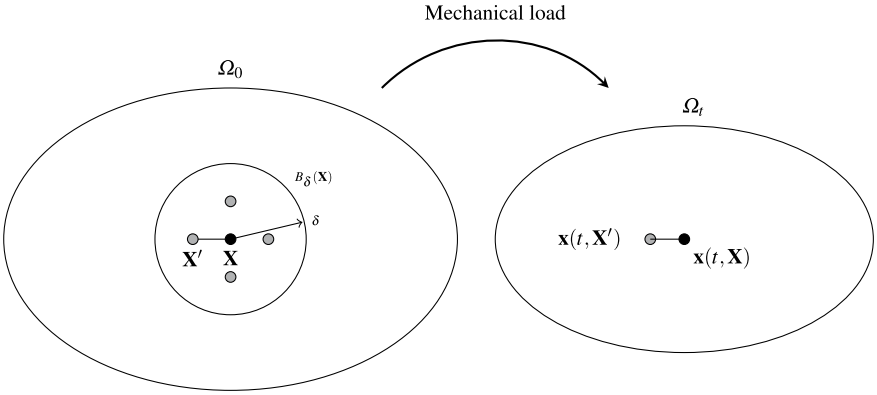


Fig. 3 Left: Schematic representation of PD where the material point $\mathbf{X} \in \Omega$ interacts non-locally with all other material points inside $B_\delta(\mathbf{X})$. Right: Positions $\mathbf{x}(t,\mathbf{X})$ and $\mathbf{x}(t,\mathbf{X}')$ of the material points in the configuration Ω_t

3 Peridynamics theory

In this section, we briefly introduce the key feature of peridynamics theory essential for the implementation. For more details about PD, we refer to [20] and for the utilized material models to [32, 33, 49].

Let a material domain be $\Omega_0 \subset \mathbb{R}^d$, for $d = 1, 2$, and 3. Peridynamics (PD) [49, 51] assumes that every material point $\mathbf{X} \in \Omega_0$ interacts non-locally with all other material points inside a horizon of length $\delta > 0$, as illustrated in Fig. 3. Let $B_\delta(\mathbf{X})$ be the sphere of radius δ centered at $\mathbf{X}$. When Ω_0 is subjected to mechanical loads, the material point $\mathbf{X}$ assumes position $\mathbf{x}(t,\mathbf{X}) = \mathbf{X} + \mathbf{u}(t,\mathbf{X})$, where $\mathbf{u}(t,\mathbf{X})$ is the displacement of material point $\mathbf{X}$ at time t.

Let $\mathbf{f}(t, \mathbf{u}(t,\mathbf{X}') - \mathbf{u}(t,\mathbf{X}), \mathbf{X}' - \mathbf{X})$ denote the peridynamic force as a function of time t, bond-deformation vector $\mathbf{u}(t,\mathbf{X}) - \mathbf{u}(t,\mathbf{X})$, and reference bond vector $\mathbf{X} - \mathbf{X}'$. The peridynamics equation of motion is given by

$$\varrho(\mathbf{X})\ddot{\mathbf{u}}(t,\mathbf{X}) = \int_{B_\delta(\mathbf{X})} \mathbf{f}(t, \mathbf{u}(t,\mathbf{X}') - \mathbf{u}(t,\mathbf{X}), \mathbf{X}' - \mathbf{X})\mathrm{d}\mathbf{X}' + \mathbf{b}(t,\mathbf{X}), \tag{3.1}$$

where $\mathbf{b}$ is the external force density, and $\varrho(\mathbf{X})$ is the material's mass density. Equation (3.1) is complemented by initial conditions $\mathbf{u}(0,\mathbf{X}) = \mathbf{u}_0(\mathbf{X})$ and $\dot{\mathbf{u}}(0,\mathbf{X}) = \mathbf{v}_0(\mathbf{X})$. In contrast to local problems, boundary conditions in peridynamics are defined over layer or collar Ω_c surrounding the domain Ω_0. Boundary conditions will be described in later sections when describing the numerical experiments.

The model in (3.1) depends on two-point interactions and is referred to as a bond-based peridynamics model. Bond-based model can only model the material with Poisson's ratio 0.25 [30, 31]. On the other hand, state-based peridynamics models [51] allow for multi-point non-local interactions and overcomes the restriction on the Poisson's ratio. It is conveniently formulated in terms of displacement dependent tensor valued functions. Let $\underline{T}[t,\mathbf{X}]$ be the peridynamic state at time t and material point $\mathbf{X}$. The peridynamics equation of motion for a state-based model is given by

$$\varrho(\mathbf{X})\ddot{\mathbf{u}}(t,\mathbf{X}) = \int_{B_\delta(\mathbf{X})} \big(\underline{T}[\mathbf{X},t]\langle\mathbf{X}' - \mathbf{X}\rangle - \underline{T}[\mathbf{X}',t]\langle\mathbf{X} - \mathbf{X}'\rangle\big)d\mathbf{X}' + \mathbf{b}(t,\mathbf{X}). \tag{3.2}$$

Equation (3.2) is complemented by initial conditions $\mathbf{u}(0,\mathbf{X}) = \mathbf{u}_0(\mathbf{X})$ and $\dot{\mathbf{u}}(0,\mathbf{X}) = \mathbf{v}_0(\mathbf{X})$.

3.1 Discretization of peridynamics equations

Continuous and discontinuous Galerkin finite element methods [6], Gauss quadrature [58] and spatial discretization [14, 39] are different discretization approaches for PD. Owing to its efficient load distribution scheme, the method of Silling and Askari [39, 50], the so-called EMU nodal discretization (EMU ND) is chosen for this implementation.

In the EMU ND scheme, the reference configuration is discretized and the discrete set of material points $\mathbf{X} := \{\mathbf{X}_i \in \mathbb{R}^d : i = 1, \ldots, n\}$ are considered to represent the material domain, see Fig. 4. The discrete neighborhood $B_\delta(\mathbf{X}_i)$ of the node $\mathbf{X}_i$ yields $B_\delta(\mathbf{X}_i) := \{\mathbf{X}_j | \, |\mathbf{X}_j - \mathbf{X}_i| \le \delta\}$. Each node $\mathbf{X}_i$ represents V_i volume such that $\sum_i^n V_i = V_{\Omega_0}$, where V_{Ω_0} is volume of whole domain. We denote nodal volume vector as $\mathbf{V}$.

The discrete bond-based equation of motion yields, for all $\mathbf{X}_i \in \Omega_0$,

$$\varrho(\mathbf{X}_i)\ddot{\mathbf{u}}(t,\mathbf{X}_i) = \sum_{B_\delta(\mathbf{X}_i)} \mathbf{f}(t, \mathbf{u}(t,\mathbf{X}_j) - u(t,\mathbf{X}_i), \mathbf{X}_j - \mathbf{X}_i)V_j + \mathbf{b}(t,\mathbf{X}_i) \tag{3.3}$$

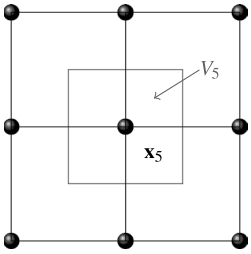


Fig. 4 The initial positions of the discrete nodes $\mathbf{X} := \{\mathbf{X}_i \in \mathbb{R}^2 : i = 1, \dots, 9\}$ in the reference configuration Ω_0. For the discrete node $\mathbf{x}_5$, its surrounding volume V_5 is shown schematically

and the discrete state-based equation of motion yields, for all $\mathbf{X}_i \in \Omega_0$,

$$\varrho(\mathbf{X}_i)\ddot{\mathbf{u}}(t, \mathbf{X}_i) = \sum_{B_\delta(\mathbf{X}_i)} \big(\underline{T}[\mathbf{X}_i, t]\langle \mathbf{X}_j - \mathbf{X}_i\rangle - \underline{T}[\mathbf{X}_j, t]\langle \mathbf{X}_i - \mathbf{X}_j\rangle\big) V_j + \mathbf{b}(t, \mathbf{X}_i). \tag{3.4}$$

Remark There could be nodes $\mathbf{X}_j$ in $B_\delta(\mathbf{X}_i)$ such that V_j is partially inside the ball $B_\delta(\mathbf{X}_i)$. For these nodes, we compute the correction in volume, V_{ij}, along the lines suggested in [Section 2, [47]]. In both Eqs. (3.3) and (3.4), we replace V_j by $V_j V_{ij}$ to correctly account for the volume.

4 Implementation of NLMech[1]

We introduce a modern C++ library, referred to as NLMech, that utilizes HPX for parallelism. The design is modular and template based making it easy to extend the code with new material models. We present the design of the code and describe the use of parallelism and concurrency based on HPX.

4.1 Design with respect to modular expandability

Figure 5 shows the modular design class. NLMech contains three modules that are affected by the discretization extensions and material models. First, the *Deck* module handles the loading of the discretization and the configuration in the YAML[2] file format. Each new Deck `class newDeck : public AbtstractDeck` inherits the common functions from the *AbstractDeck* and is extended with the new problem/material specific attributes.

Second, the abstractions for bond-based `class AbstractBond` and state-based `class AbstractState` materials are provided in the *Material* module. The nonlinear bond-based elastic material `class Elastic : public AbstractBond` in Sect. 4.2.1 and the linear state-based elastic material `class Elastic : public AbstractState` of Sect. 4.2.2 were implemented. The abstract material must be inherited and the methods, e.g., force and strain, are implemented if a new material model is to be implemented. Note that we tried to use as less as possible virtual functions to avoid some overhead which virtual function calls have.

The different time integration schemes and the discretizations are considered third. All new *Problem* classes inherit their common and abstract functions from *AbstractProblem*. The class `template<classT> class Quasistatic : public AbstractProblem` implements the implicit time integration in Sect. 4.2.3 and class `template<classT >class Explicit : public` implements the explicit time integration in Sect. 4.2.4. Note that a problem is a template class with the material definition as the template class. Thus, the specific implementation can therefore be used for state-based and bond-based materials.

The design aims to hide most of the HPX-specific features and making the code easy for adding new material models by implementing the abstract classes. Listing 5 shows how to run the implicit time integration and the explicit time integration using a elastic state-based material models. Note that we hide the inclusion of header files and parsing the command line arguments. For new problem classes, the user have to deal with the parallel for loops instead of using the C++ standard for loop. Thus, the code is accessible to users that do not have advanced programming experience in parallel computing, but still yields acceptable scalability without optimization.

[1] https://github.com/nonlocalmodels/NLMech.
[2] http://yaml.org/.

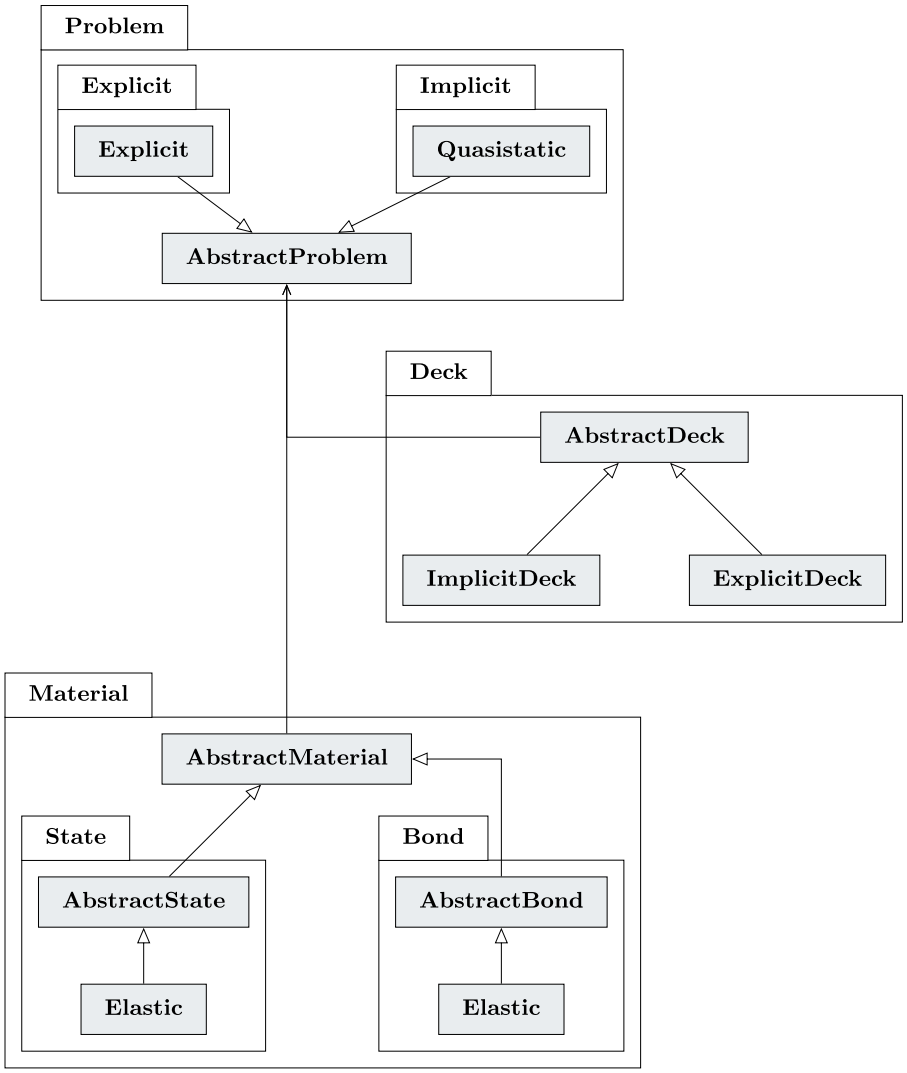


Fig. 5 Class diagram of NLMech which is designed to easily extend the code with new materials or discretizations. The functionality of the code is defined in three packages: *Problem* containing the different classes for discretizations, *Deck* which handles the input and output, and *Material* providing the different kind of material models. All packages provide an abstract class which needs to be inherited by all sub classes for extending the code

```
// Include the relevant headers

int main(int argc, char *argv[]) {

// Parse the command line arguments: filename

// Read the YAML config file and store all entities in the deck
IO::deck::ImplicitDeck* deck = new IO::deck::ImplicitDeck(
    filename);

// Run the implicit time integration with the elastic state
    material
problem::Quasistatic<material::state::Elastic>* problem =
          new problem::Quasistatic<material::state::Elastic>(deck
              );

// Read the YAML config file and store all entities in the deck
IO::deck::ExplicitDeck* deck2 = new IO::deck::ExplicitDeck(
    filename);

// Run the explicit time integration with the elastic state
    material
problem::Explicit<material::state::Elastic>* problem =
          new problem::Explicit<material::state::Elastic>(deck2);


return EXIT_SUCCESS;
}
```

Listing 5: Small example of the interfaces and functions to run the implicit time integration and the explicit time integration using a elastic state-based material model. Note that we hide the inclusion of header files and parsing the command line arguments.

4.2 Parallelization with HPX

The implementation of bond-based material and the explicit time integration is adapted from earlier one-dimensional code developed by the second author [22]. The implementation of the state-based material and the implicit time integration is adapted from [35]. These sequential algorithms are analyzed to make use of HPX parallelism. The use of HPX tools to achieve parallelism is discussed in the sequel.

4.2.1 Bond-based material

Algorithm 1 shows the use of HPX for computing the internal force and strain energy. For demonstration, we consider nonlinear bond-based peridynamic (NP) model introduced in [32, 33]. The same algorithm will work with commonly used bond-based PMB (prototype micro-elastic brittle) model. In Algorithm 1, S is the bond-strain, ψ is the pairwise potential function, and $J^{\delta}(r)$ is the influence function. In Sect. 5.2, we show specific form of ψ and J^{δ}. We read input file and initialize the material class with specified form of function ψ and J^{δ}. We compute and store the list of neighbors for each node in the reference configuration (initial configuration). HPX parallel for loop is used to compute the force and energy of each node in parallel, see Algorithm 1.

Algorithm 1 Computation of internal force for bond-based material. Here, S is the bond-strain, ψ is the nonlinear potential and $J^\delta(|\mathbf{X}|)$ is the influence function considered in [32, 33].

1: **parallel_for** $i < n$ **do** ▷Compute force at mesh nodes
2: **for** $j \in B_\delta(\mathbf{X}_i)$ **do**
3: $\boldsymbol{\xi} = \mathbf{X}_j - \mathbf{X}_i$
4: $S = \frac{\mathbf{u}(\mathbf{X}_j) - \mathbf{u}(\mathbf{X}_i)}{|\boldsymbol{\xi}|} \cdot \frac{\boldsymbol{\xi}}{|\boldsymbol{\xi}|}$
5: ▷Compute force at $\mathbf{X}_i$ as in [32, 33]
6: $\mathbf{f}(\mathbf{X}_i) += \frac{4}{\delta |B_\delta(\mathbf{0})|} J^\delta(|\boldsymbol{\xi}|) \psi'(|\boldsymbol{\xi}| S^2) S \frac{\boldsymbol{\xi}}{|\boldsymbol{\xi}|} V_j$
7: ▷Compute strain energy at $\mathbf{X}_i$ as in [32, 33]
8: $U(\mathbf{X}_i) += \frac{1}{|B_\delta(\mathbf{0})|} J^\delta(|\boldsymbol{\xi}|) \psi(|\boldsymbol{\xi}| S^2) V_j / \delta$
9: **end for**
10: **end**

4.2.2 Linearly elastic state-based material

The internal force density and strain energy are computed for a state-based elastic peridynamic solid material, as described in [51]. Algorithm 2 shows the adapted algorithm [35] parallelized and synchronized with HPX. First, the weighted volume m is computed for each node using a HPX parallel for loop. Second, the dilation θ is computed for each node a HPX parallel for loop. Note that the dilation θ depends on the weighted volume m, and therefore, we cannot use asynchronous code execution here. The internal force density and the strain energy can be computed independently of each other. Therefore, the execution policy `hpx::parallel::execution::task` is utilized to obtain futures `f1` and `f2` back of these two loops to compute the loops asynchronously, see Line 17 and Line 30. Note that the strain energy is optional and is only computed if requested, e.g., for output. A synchronization for these two futures is needed before the force and strain energy are computed in future steps, see Line 42.

Algorithm 2 Computation of internal force and strain energy. Adapted from [35]

```
parallel_for i < n do ▷Compute weighted volumes as in [35, 51]
    m_i = 0
    for j ∈ B_δ(X_i) do
        ξ = X_j − X_i
        m_i += |ξ|^2 V_j
    end for
end
parallel_for i < n do ▷Compute dilatation as in [35, 51]
    θ_i = 0
    for j ∈ B_δ(X_i) do
        ξ = X_j − X_i
        η = u(X_j) − u(X_i)
        e = |ξ + η| − |ξ|
        θ_i += 3/m_i |ξ| e V_j
    end for
end
future f1 =
parallel_for i < n do ▷Compute internal forces as in [35, 51]
    for j ∈ B_δ(X_i) do
        ξ = X_j − X_i
        η = u(X_j) − u(X_i)
        e^d = e − (θ_i|ξ|)/3
        t = 3/m_i K θ_i |ξ| + (15μ)/m_i e^d
        M = (ξ + η)/|ξ + η|
        f_i += t M V_j
        f_j −= t M V_i
    end for
end
▷Note that the strain energy is optional and only computed if requested, e.g. for output
future f2 =
parallel_for i < n do ▷Compute strain energy as in [51]
    for j ∈ B_δ(X_i) do
        ξ = X_j − X_i
        η = u(X_j) − u(X_i)
        M_ji = (ξ + η)/|ξ + η|
        ξ = X_i − X_j
        η = u(X_i) − u(X_j)
        M_ij = (ξ + η)/|ξ + η|
        e_i = 1/2 M_ij M_ji α V_j
    end for
end
wait{f1,f2}
```

4.2.3 Implicit time integration

Figure 6 shows the implicit integration implementation flowchart. The external force $\mathbf{b}$ is updated for each load step s. Next, the residual

$$\mathbf{r} = \sum_{\Omega_0} \mathbf{f}(t, x_i) + \mathbf{b}(t, x_i) \tag{4.1}$$

and its l_2 norm are computed and compared against the tolerance τ. If the residual is too large, the tangent stiffness matrix

$$\mathbf{K}_{ij} \approx \frac{\mathbf{f}(x_i, u_i + \epsilon^j) - \mathbf{f}(x_i, u_i - \epsilon^j)}{2\epsilon}. \tag{4.2}$$

is assembled as described in [35], (see Algorithm 3). The displacement of the previous load step was used to assembly the first matrix $\mathbf{K}(u)$. Line 6 perturbs the displacement by $\pm\upsilon$, where υ is infinitesimally small. Line 9 computes the internal forces $f^{\pm\upsilon}$ and Line 13 evaluates the central difference to construct the stiffness matrix $\mathbf{K}(u)$. Note that the node's neighborhood B_δ is represented and has several zero entries where nodes do not have neighbors. Next, the guessed displacement is updated with the solution from solving $\mathbf{K}\underline{\mathbf{u}} = -\mathbf{r}$. The residual is evaluated once and the Newton method is iterated until $|r| \leq \tau$. Note that a dynamic relaxation method (DR) [29], a conjugate gradient method (CG), or a Galerkin method [57] could be used as well.

The high-performance open-source C++ library Blaze [18, 19] was used for matrix and vector operations. Blaze supports HPX for parallel execution and can be easily integrated. The library BlazeIterative[3] was used for solving $\mathbf{K}\underline{\mathbf{u}} = -\mathbf{r}$. The biconjugate gradient stabilized method (BiCGSTAB) or the conjugate gradient (CG) solver was used for solving.

Algorithm 3 Assembly of the tangent stiffness matrix by central finite difference. Adapted from [35]

1: $\mathbf{K}^{d\cdot n\times d\cdot n} = 0$ ▷Set matrix to zero
2: **parallel_for** $i < n$ **do**
3: **for** $i \in \{B_\delta(\mathbf{X}_i), i\}$ **do**
4: ▷Evaluate force state under perturbations of displacement
5: **for each** displacement degree of freedom r at node j **do**
6: $\underline{T}[\mathbf{X}_i](\mathbf{u} + \upsilon^r)$
7: $\underline{T}[\mathbf{X}_i](\mathbf{u} - \upsilon^r)$
8: **for** $k \in B_\delta(\mathbf{X}_i)$ **do**
9: $\mathbf{f}^{\upsilon+} = \underline{T}^{\upsilon+}\langle \mathbf{X}_k - \mathbf{X}_i\rangle V_i V_j$
10: $\mathbf{f}^{\upsilon-} = \underline{T}^{\upsilon-}\langle \mathbf{X}_k - \mathbf{X}_i\rangle V_i V_j$
11: $\mathbf{f}_{\text{diff}} = \mathbf{f}^{\upsilon+} - \mathbf{f}^{\upsilon-}$
12: **for each** degree of freedom s at node k **do**
13: $\mathbf{K}_{sr} += {}^{f_{\text{diff}_s}}/_{2\upsilon}$
14: **end for**
15: **end for**
16: **end for**
17: **end for**
18: **end**

4.2.4 Explicit time integration

Figure 7 shows the flowchart for the explicit time integration. Algorithm 4 outlines the steps to implement the velocity-verlet scheme

$$\mathbf{v}(t^{k+1/2}, \mathbf{X}_i) = \mathbf{v}(t^k, \mathbf{X}_i) + \frac{(\Delta t/2)}{\rho(\mathbf{X}_i)} \left[\mathbf{b}(t^k, \mathbf{X}_i) + \sum_{B_\delta(\mathbf{X}_i)} \mathbf{f}(t^k, \boldsymbol{\eta}, \boldsymbol{\xi})\right], \tag{4.3}$$

$$\mathbf{u}(t^{k+1}, \mathbf{X}_i) = \mathbf{u}(t^k, \mathbf{X}_i) + \Delta t \mathbf{v}(t^{k+1/2}, \mathbf{X}_i), \tag{4.4}$$

$$\mathbf{v}(t^{k+1}, \mathbf{X}_i) = \mathbf{v}(t^{k+1/2}, \mathbf{X}_i) + \frac{(\Delta t/2)}{\rho(\mathbf{X}_i)} \left[\mathbf{b}(t^{k+1}, \mathbf{X}_i) + \sum_{B_\delta(\mathbf{X}_i)} \mathbf{f}(t^{k+1}, \boldsymbol{\eta}, \boldsymbol{\xi})\right] \tag{4.5}$$

to obtain the displacement $\mathbf{u}^{k+1}$ for the time step $k+1$. Line 4 calls a function of either bond-based Material or state-based Material class to compute the forces and energies corresponding to displacement $\mathbf{u}^k$. The velocity-verlet algorithm is used to compute the velocity at $k+1/2$ and displacement $\mathbf{u}^{k+1}$. Line 12 invokes the material class again to compute the forces at new displacements $\mathbf{u}^{k+1}$. The velocity at $k+1$is computed with the updated forces. The central different scheme is given by

$$\mathbf{u}(t^{k+1}, \mathbf{X}_i) = 2\mathbf{u}(t^k, \mathbf{X}_i) - \mathbf{u}(t^{k-1}, \mathbf{X}_i) + \frac{\Delta t^2}{\rho(\mathbf{X}_i)} \left[\mathbf{b}(t^k, \mathbf{X}_i) + \sum_{B_\delta(\mathbf{X}_i)} \mathbf{f}(t^k, \boldsymbol{\eta}, \boldsymbol{\xi})\right]. \tag{4.6}$$

[3] https://github.com/tjolsen/BlazeIterative.

Algorithm 4 Time integration using velocity-verlet scheme

▷Loop over time steps
for $0 \leq k < N$ **do**
  ▷Compute peridynamic force $\mathbf{f}^k$, body force $\mathbf{b}^k$,
  ▷external force $\mathbf{f}^k_{ext}$, and total energy U^k_{total} as shown in Algorithm 1
  parallel_for $i < n$ **do**
    ▷Compute velocity as shown in Equation (4.3)
    $\mathbf{v}^{k+1/2}(\mathbf{X}_i) = \mathbf{v}^k(\mathbf{X}_i) + (\Delta t/2)(\mathbf{f}^k(\mathbf{X}_i) + \mathbf{b}^k(\mathbf{X}_i) + \mathbf{f}^k_{ext}(\mathbf{X}_i))$
    ▷Compute displacement as shown in Equation (4.4)
    $\mathbf{u}^{k+1}(\mathbf{X}_i) = \mathbf{u}^k(\mathbf{X}_i) + \Delta t \mathbf{v}^{k+1/2}\mathbf{f}(\mathbf{X}_i)$
  end
  Update boundary condition for time $t = (k+1)\Delta t$
  ▷Compute $\mathbf{f}^{k+1}$, $\mathbf{b}^{k+1}$, $\mathbf{f}^{k+1}_{ext}$, and U^{k+1}_{total} as shown in Algorithm 1.
  parallel_for $i < n$ **do** ▷Loop over nodes
    ▷Compute velocity as shown in Equation (4.5)
    $\mathbf{v}^{k+1}(\mathbf{X}_i) = \mathbf{v}^{k+1/2}(\mathbf{X}_i) + (\Delta t/2)(\mathbf{f}^{k+1}(\mathbf{X}_i) + \mathbf{b}^{k+1}(\mathbf{X}_i) + \mathbf{f}^{k+1}_{ext}(\mathbf{X}_i))$
  end

end for

5 Validation of NLMech

In this section, we demonstrate the convergence of HPX implementations for both implicit and explicit schemes.

5.1 Implicit

5.1.1 One-dimensional tensile test

Consider the simple geometry of Fig. 8 for comparing the one dimensional implicit time integration against a classical continuum mechanics (CCM) solution. The node on the left-hand side is clamped and displacement is set to zero. A force F is applied to the node at the right-hand side.

The strain, stress, and strain energy for this configuration are compared with the values obtained from classical continuum mechanics (CCM) where $\sigma = E \cdot \varepsilon$, where σ is the stress, E is the Young's modulus and ε is the strain. The stress $\sigma = {}^{F}/_{A}$ is then defined by the force F per cross section A. Thus, the strain is obtained by $\varepsilon = {}^{\sigma}/_{E} = {}^{F}/_{(AE)}$. For a force F of 40 N, a cross section of 1 m^2 and a Young's modulus of 4 GPa, the resulting strain reads $\varepsilon_{\text{CCM}} = 1 \times 10^{-8}$ an the stress is of $\sigma_{\text{CCM}} = 40$ Pa. The strain energy density is given by $U_{\text{CCM}} = {}^{\sigma^2}/_{(2E)} = 2 \times 10^{-7}$ Pa.

The bar was discretized with 33 nodes with a nodal spacing h of 500 mm. The horizon was $\delta = 1000$ mm. The tolerance τ for the Biconjugate gradient stabilized method (BiCGSTAB) was 1×10^{-9}. Figure 9 shows that stresses, strains and strain energy match perfectly the theoretical values inside the bar, but all these quantities diverge at the boundaries. These effects are the well-known surface effects within the EMU nodal discretization.

5.1.2 Two-dimensional tensile test

Figure 10 shows the two-dimensional tensile benchmark. The line of nodes of the right-hand side of the plate are clamped in x-direction and y-direction. On each of the nodes of the line at the left-hand side, a force $F = -50$ kN in x-direction was applied. The displacement of a node $\mathbf{X}_i$ for a tensile behavior can be derived using the Airy stress function [46] as follows

$$\begin{aligned} \mathbf{u}_x(\mathbf{X}_i) &= \frac{F}{EWT}(\mathbf{X}_{i_x} - W), \\ \mathbf{u}_y(\mathbf{X}_i) &= -\frac{\nu F}{ET}\left(\frac{\mathbf{X}_{i_y}}{W} - \frac{1}{2}\right), \end{aligned} \tag{5.1}$$

where F is the applied force and W and H and T are, respectively, the plate's width and height and thickness. Note that we assume a thickness $T = 1$, mm. $\mathbf{X}_{i_x}, \mathbf{X}_{i_y}$ denotes the x and y coordinate of node $\mathbf{X}_i$. A Young's modulus of $E = 4000$ MPa and a Poisson's ratio of $\varrho = 0.3$ was used.

H and W were set to 375 mm and $h = 25$ mm. The tolerance for the BiCGSTAB solver was $\tau = 1 \times 10^{-3}$. The m_d-value was, 4, which means that $2m_d + 1$ nodes are within $[\mathbf{X}_{i_x} - \delta, \mathbf{X}_{i_x} + \delta]$ and $[\mathbf{X}_{i_y} - \delta, \mathbf{X}_{i_y} + \delta]$. Table 1 lists the actual position at the node in the center of the plate x_{mid} and its comparison with the one from CCM (5.1). The relative error for the actual position in x-direction is sufficiently small. With the applied boundary conditions, the displacement at the point in the center of the plate in y-direction is zero.

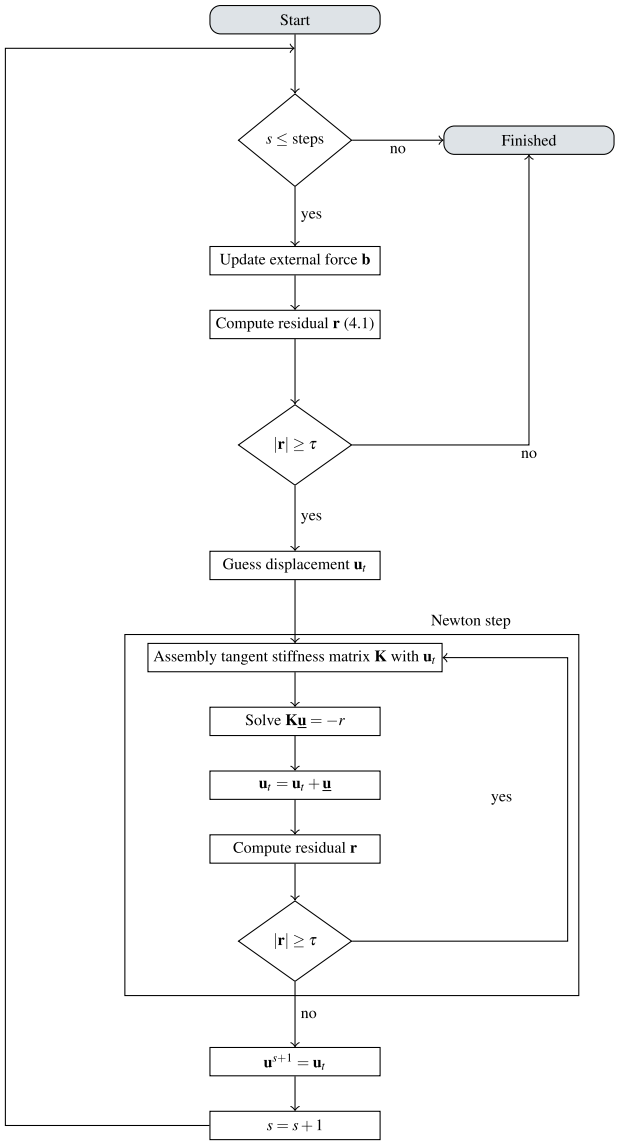


Fig. 6 Flowchart of the implicit time integration scheme adapted from [35]. For each time step s, the external force $\mathbf{b}$ is updated and the residual $\mathbf{r}$ is evaluated. When the norm of the residual is larger than the tolerance τ, the displacement $\mathbf{u}^{s+1}$ is obtained via a Newton step

5.2 Explicit

For the explicit scheme, we present numerical verification of convergence of the approximation. We study the rate at which numerical approximation converge with respect to the mesh size. Similar to [Section 6.1.2, [23]], we derive the formula to compute the rate of convergence numerically. We denote the L^2 norm of function $\mathbf{f} : \Omega_0 \to \mathbb{R}^d$ as $||\mathbf{f}|| := \sqrt{\int_{\Omega_0} |\mathbf{f}(\mathbf{X})|^2 d\mathbf{X}}$, where $d\mathbf{X}$ is the infinitesimal length (1-d), area (2-d), or volume (3-d) element for given dimension d.

Let $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ are three approximate displacement fields corresponding to the meshes of size h_1, h_2, h_3. Let $\mathbf{u}$ be the exact solution. We assume that for $h' < h$, $\underline{C}h^\alpha \leq ||\mathbf{u}_h - \mathbf{u}_{h'}|| = \bar{C}h^\alpha$ with $\underline{C} \leq \bar{C}$ and $\alpha > 0$. We fix the ratio of mesh sizes as $\frac{h_1}{h_2} = \frac{h_2}{h_3} = r$, where r is fixed number. We can show then

$$\alpha \leq \frac{\log(||\mathbf{u}_1 - \mathbf{u}_2||) - \log(||\mathbf{u}_2 - \mathbf{u}_2||) + \log(\bar{C}) - \log(\underline{C})}{\log(r)}. \tag{5.2}$$

So an upper bound on the convergence rate is at least as big as

$$\bar{\alpha} = \frac{\log(||\mathbf{u}_1 - \mathbf{u}_2||) - \log(||\mathbf{u}_2 - \mathbf{u}_3||)}{\log(r)}. \tag{5.3}$$

$\bar{\alpha}$ provides an estimate for rate of convergence.

We now present the convergence results for explicit scheme.

5.2.1 One dimensional

In 1-d the strain between material point X, X' is given by $S(X', X) = \frac{u(X') - u(X)}{|X' - X|}$. We consider a nonlinear peridynamic force between material point X, X' of the form, see [Section 2.1, [22]],

$$f(t, u(t, X') - u(t, X), X' - X)) = \frac{2}{\delta^2} \int_{X-\delta}^{X+\delta} J^\delta(|X' - X|) \psi'(|X' - X|S(X', X)^2) S(X', X) \mathrm{d}X'. \tag{5.4}$$

$\psi : \mathbb{R}^+ \to \mathbb{R}$ is the nonlinear potential which is smooth, positive, and concave. We set function ψ as

$$\psi(r) = C(1 - \exp[-\beta r]), \qquad C = \beta = 1. \tag{5.5}$$

With this choice of potential, we effectively model a bond which is elastic for small strains and softens for large strains. As $S \to \infty$, $\psi'(r) = C\beta \exp[-\beta r] \to 0$, and therefore, the pairwise force f between two points X, X' go to zero as the bond-strain $S(X', X)$ gets larger and larger. The cumulative effect is that the material behaves like a elastic material under small deformation and cracks develop in regions where the deformation is large, see [34]. The influence function J^δ is of the form: $J^\delta(r) = J(r/\delta)$, where $J(r) = c_1 r \exp(-r^2/c_2)$, $c_1 = 1$, and $c_2 = 0.4$.

The linear peridynamic force is given by, see [Section 2.1, [22]],

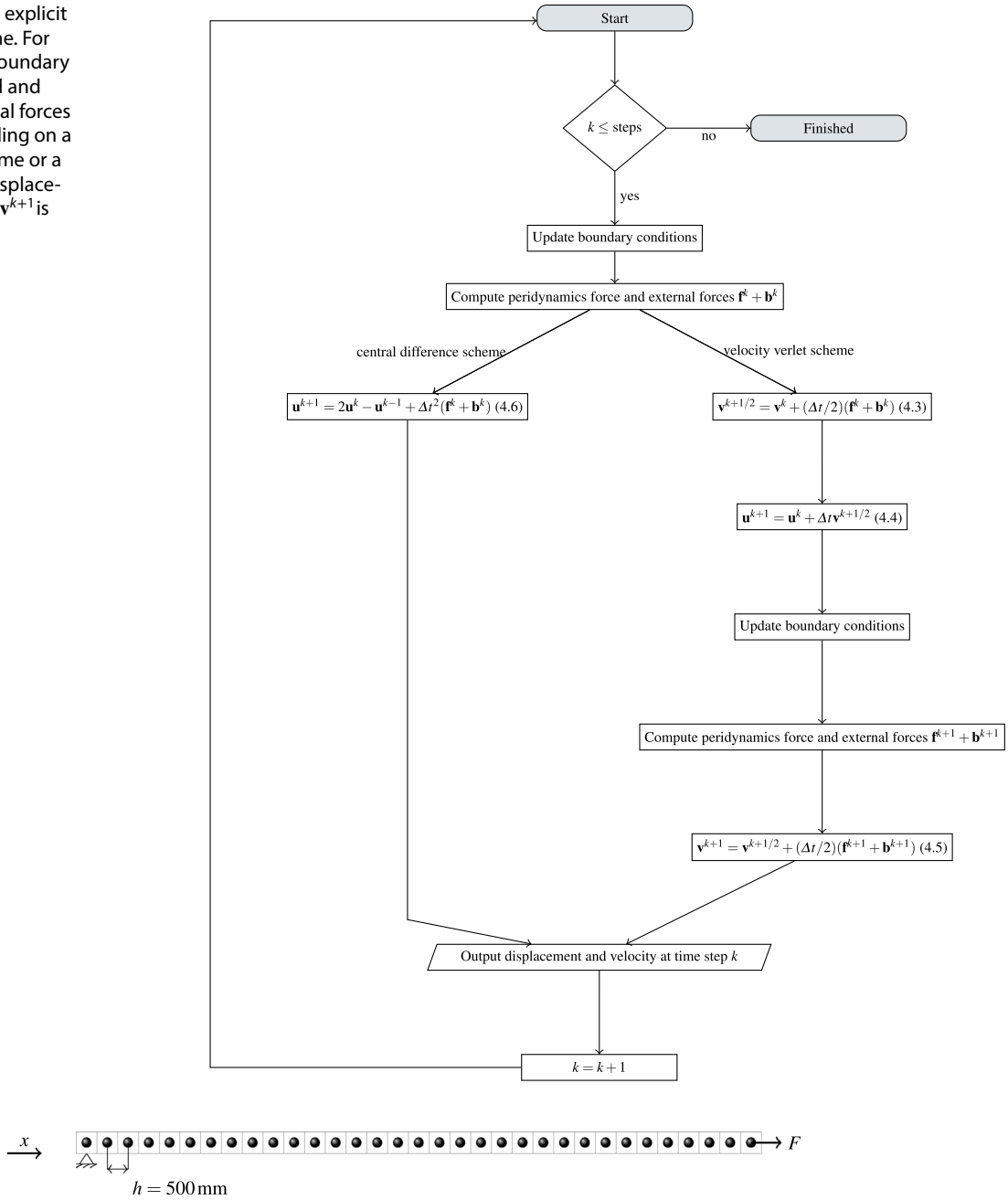


Fig. 7 Flowchart of the explicit time integration scheme. For each time step k, the boundary conditions are updated and the internal and external forces are computed. Depending on a central difference scheme or a velocity scheme, the displacement $\mathbf{u}^{k+1}$ and velocity $\mathbf{v}^{k+1}$ is obtained

Fig. 8 Sketch of the one dimensional bar benchmark test. The node on the left-hand side is clamped and its displacement is set to zero. A force F is applied on the node at the right-hand side. Adapted from [11]

$$f_I(t, u(t,X') - u(t,X), X' - X)) = \frac{2}{\delta^2} \int_{X-\delta}^{X+\delta} J^\delta(|X' - X|)\psi'(0)S(X', X)\mathrm{d}X'. \tag{5.6}$$

From (5.5), we have $\psi'(0) = C\beta$.

Let $\Omega = [0, 1]$ be the material domain with an horizon $\delta = 0.01$. The time domain is $[0, 2]$ with a time step $\Delta t = 10^{-5}$. Consider four mesh sizes $h_1 = \delta/2$, $h_2 = \delta/4$, $h_3 = \delta/8$, and $h_4 = \delta/16$, and compute Eq. (5.3) for two sets $\{h_1, h_2, h_3\}$ and $\{h_2, h_3, h_4\}$ of mesh sizes. The boundary conditions are those described in Fig. 11. Apply either one of the initial conditions:

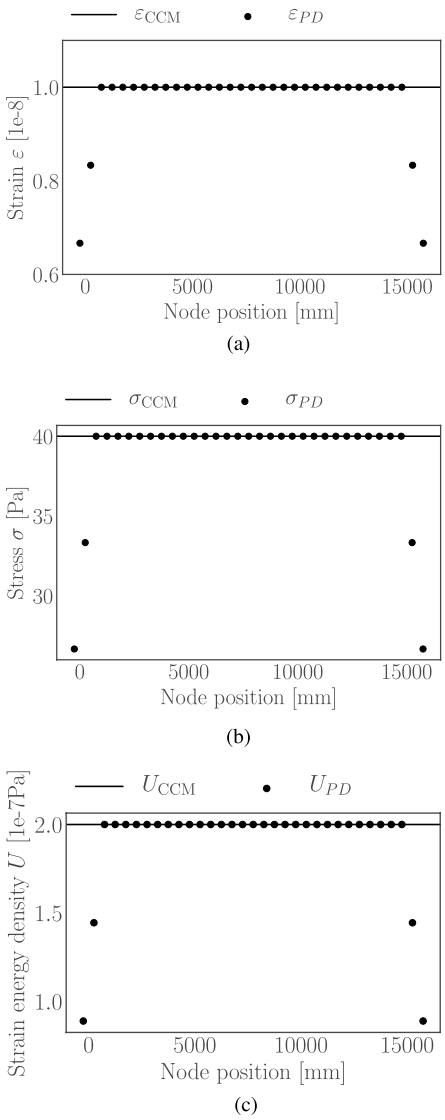


Fig. 9 Comparison of strain ϵ **a**, stress σ **b**, and strain energy U **c** with classical continuum mechanics. Close to the boundary the surface effect influences the accuracy

Initial condition 1(IC 1): Let the initial condition on the displacement u_0 and the velocity v_0 be given by

$$u_0(X) = \exp[-|X - x_c|^2/\beta]a, \quad v_0(X) = 0, \tag{5.7}$$

with $x_c = 0.5, a = 0.001, \beta = 0.003$. u_0 is the Gaussian function centered at x_c.

Initial condition 2(IC 2): The initial condition u_0 and v_0 are described as

$$\begin{aligned} u_0(X) &= \exp[-|X - x_{c1}|^2/\beta]a \\ &\quad + \exp[-|X - x_{c2}|^2/\beta]a, \quad v_0(X) = 0, \end{aligned} \tag{5.8}$$

with $x_{c1} = 0.25, x_{c2} = 0.75, a = 0.001, \beta = 0.003$. u_0 is the sum of two Gaussian functions centered at x_{c1} and x_{c2}.

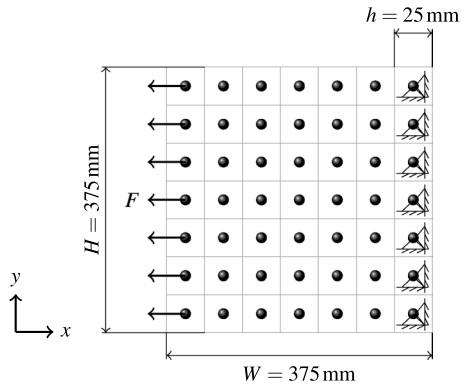


Fig. 10 Sketch of the two-dimensional tensile test. All nodes on the line of the right-hand side of the plate are clamped in *x*-direction and *y*-direction. A force of −50 kN is applied in $x-$*direction* to each node of the line on the left-hand side. Adapted from [10]

Table 1 Comparison of the actual position in meters of the node in the center of the plate obtained in the simulation with those from classical continuum mechanics (5.1)

Actual position	CCM	PD	Relative error
x-direction	0.187 49 m	0.187 44 mm	4.9×10^{-5}
y-direction	0.1875 m	0.1875 m	0

Figure 11 shows the rate of convergence $\bar{\alpha}$ as a function of time for solutions having the initial conditions 1 and 2. The convergence rate for $\{h_1, h_2, h_3\}$ and $\{h_2, h_3, h_4\}$ follows the same trend. The bump in Fig. 11 near $t = 1.1$ for both the initial conditions is due to the merging of two waves traveling toward each other. We show the displacement profile near $t = 1.1$ in Fig. 12 for IC 1 and Fig. 13 for IC 2. This behavior is particularly difficult to capture and requires much finer mesh. For the rapidly varying (spatially) displacement field, the length scale at which the displacement varies is small and requires very fine mesh size. This is the reason we see a better rate for Set 2 $\{h_2, h_3, h_4\}$ as compared to Set 1 $\{h_1, h_2, h_3\}$ in Fig. 11. This behavior is true for both the initial conditions. Similar results, not shown here, were obtained for the nonlinear model of Eq. (3.1). The convergence results presented in Fig. 11 agree with the theoretical convergence rate, which suggests that the implementation is robust.

5.2.2 Two dimensional

In higher dimension, the strain is given by $S(\mathbf{X}', \mathbf{X}) = \frac{\mathbf{u}(\mathbf{X}') - \mathbf{u}(\mathbf{X})}{|\mathbf{X}' - \mathbf{X}|} \cdot \frac{\mathbf{X}' - \mathbf{X}}{|\mathbf{X}' - \mathbf{X}|}$. The nonlinear peridynamic force in 2-d is given by, see [33],

$$\mathbf{f}(t, \mathbf{u}(t, \mathbf{X}') - \mathbf{u}(t, \mathbf{X}), \mathbf{X}' - \mathbf{X})) = \frac{4}{\delta |B_\delta(\mathbf{0})|} \int_{B_\delta(\mathbf{X})} J^\delta(|\mathbf{X}' - \mathbf{X}|) \psi'(|\mathbf{X}' - \mathbf{X}| S(\mathbf{X}', \mathbf{X})^2) S(\mathbf{X}', \mathbf{X}) \frac{\mathbf{X}' - \mathbf{X}}{|\mathbf{X}' - \mathbf{X}|} \mathrm{d}\mathbf{X}'. \tag{5.9}$$

$\psi(r)$ is given by (5.5). We set $J^\delta(r) = 1$ if $r < 1$ and 0 otherwise. Similar to the 1-d case, if we substitute $\psi'(0)$ in place of $\psi'(|\mathbf{X}' - \mathbf{X}| S(\mathbf{X}', \mathbf{X})^2)$ we will get the expression of linear peridynamic force $\mathbf{f}_i$.

Let $\Omega = [0, 1]^2$ be the material domain with a horizon $\delta = 0.1$. The 2-d vector $\mathbf{X}$ is written $\mathbf{X} = (X_1, X_2)$ where X_1 and X_2 are the components along the x and y axes. The time domain is taken to be [0, 2] and $\Delta t = 10^{-5}$ is the time step. The influence function is $J^\delta(r) = 1$ for $r < \delta$ and 0 otherwise. The rate $\bar{\alpha}$ is computed for three mesh sizes $h = \delta/2, \delta/4, \delta/8$.

We fix $\mathbf{u} = (0, 0)$ on the collar Ω_c around domain Ω, see Fig. 14. Let the initial condition on displacement vector $\mathbf{u}_0 = (u_{0,1}, u_{0,2})$ and velocity vector $\mathbf{v}_0 = (v_{0,1}, v_{0,2})$ be

$$\begin{aligned} u_{0,1}(X_1, X_2) &= \exp[-(|X_1 - x_{c,1}|^2 + |X_2 - x_{c,2}|^2)/\beta] a_1, \\ u_{0,2}(X_1, X_2) &= \exp[-(|X_1 - x_{c,1}|^2 + |X_2 - x_{c,2}|^2)/\beta] a_2, \\ v_{0,1}(X_1, X_2) &= 0, v_{0,2}(X_1, X_2) = 0, \end{aligned} \tag{5.10}$$

where $\mathbf{a} = (a_1, a_2)$ and $\mathbf{x}_c = (x_{c,1}, x_{c,2})$ are 2-d vectors and β is a scalar parameter. Two different types of initial conditions are considered:

Initial condition 1(IC 1):

$$\mathbf{a} = (0.001, 0), \mathbf{x}_c = (0.5, 0.5), \text{ and } \beta = 0.003. \tag{5.11}$$

Initial condition 2(IC 2):

$$\mathbf{a} = (0.0002, 0.0007), \mathbf{x}_c = (0.25, 0.25), \text{ and } \beta = 0.01. \tag{5.12}$$

Figure 15 shows $\bar{\alpha}$ with respect to time for the nonlinear (NP) and linear (LP) peridynamics solutions. Solutions appear to converge at a rate above theoretically predicted rate of 1 for the nonlinear model with boundary conditions considered in this problem, see [21, 22].

6 Benchmark for NLMech

6.1 Implicit

The test case of Sect. 5.1.2 served as benchmark for the two-dimensional implicit time integration. Figure 16 shows the 20436 nonzero entries of the tangent stiffness matrix $K^{360\times360}$ with the condition number $\kappa(K) = 90.688$. The solver required 28 iterations. The benchmark was run on Fedora 25 with kernel 4.8.10 on Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz with up to 6 physical cores. HPX (version *bd2f240 b1565b4*) and NLMech were compiled with gcc 6.2.1, boost 1.61 and blaze 3.2 libraries were used.

The speed-up $S(p) = {}^{T_1}/_{T_p}$ [44] with respect to the computational time on one node T_1 is shown in Fig. 17a for $p = [1, 2, 3, \ldots, 6]$ where p is the number of CPUs. The straight line shows the optimal speed-up, meaning that we assume if we go from one CPU to two CPUs the code gets two time faster and so. The lines with the circle markers shows the speed-up with respect to one single CPU. Here, up to three CPUs, we are close to the optimal speed-up. Later we divergence from the optimal speed-up probably due to the strong scaling and the fixed problem size. In addition, the parallel efficiency $E(p) = {}^{S(p)}/_{p}$ [44] is shown in Fig. 17b for $p = [1, 2, 3, \ldots, 6]$ where p is the number of CPUs. The parallel efficiency is some indication for the fraction of time for which the CPU is doing some computation. The parallel efficiency is above 0.9 for up to three CPUs. For four to five CPUs, the parallel efficiency decays to 0.8 and for six CPUs the efficiency is around 0.75. More sophisticated execution policies (e.g., dynamic or adaptive chunk size) could be applied, to decrease the computational time. Note that only parallel for loops, synchronization, and futurization were utilized for parallelization.

6.2 Explicit

The setup presented in Sect. 5.2.2 was discretized with 196249 nodes and an horizon of 0.05 m and $m_d = 20$ as chosen. The benchmark was run on CentOS 7 with kernel 3.10.0 on Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz. HPX (version *82f7b281*) and NLMech were compiled with gcc 7.2, boost 1.61 and blaze 3.2 libraries.

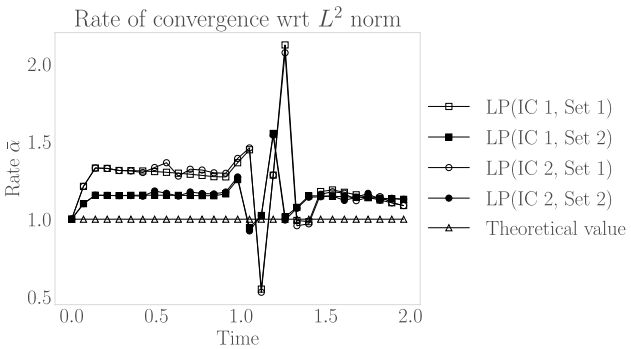


Fig. 11 Time vs rate of convergence with respect to mesh size. These results are for linear peridynamic force (LP). Set 1 corresponds to convergence rate obtained from solutions of mesh sizes $\{h_1, h_2, h_3\}$ and set 2 corresponds to convergence rate obtained from solutions of mesh sizes $\{h_2, h_3, h_4\}$. The boundary condition is $u = 0$ on non-local boundary $\Omega_c = [-\delta, 0] \cup [1, 1 + \delta]$. These results validate the implementation of the explicit scheme for peridynamics in one dimension

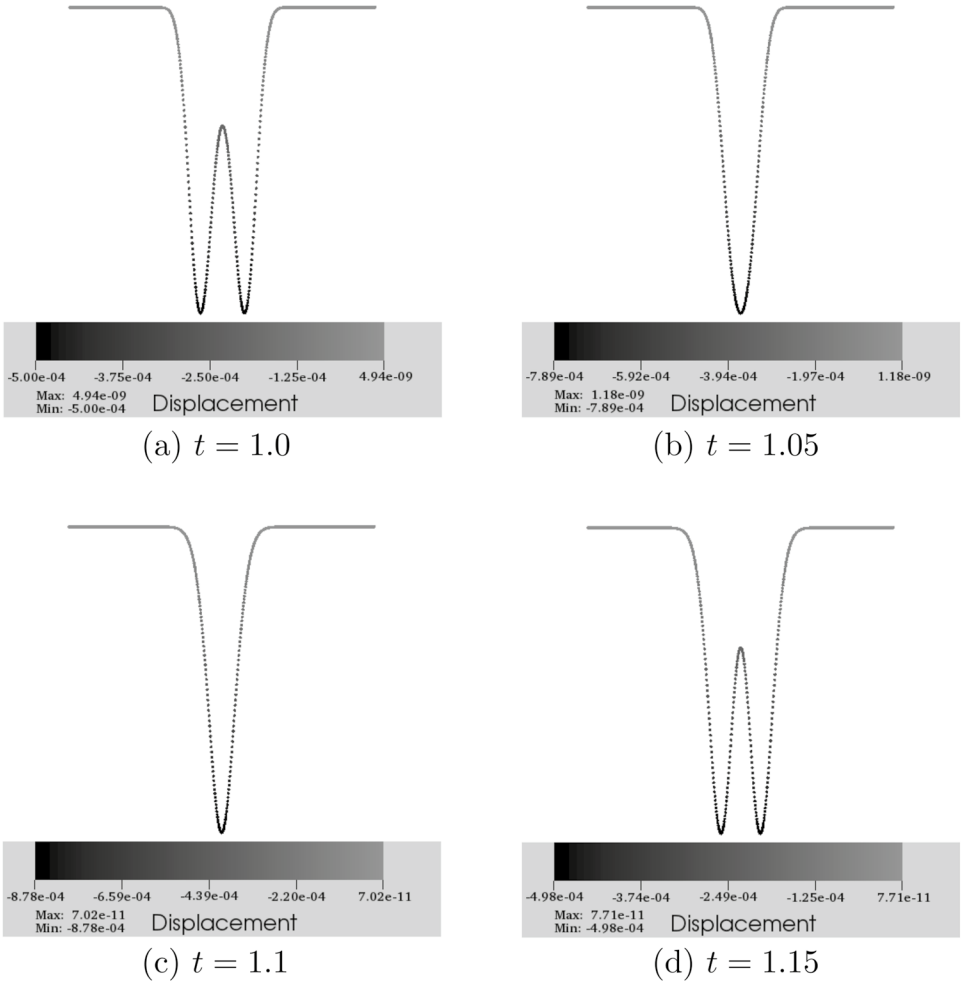


Fig. 12 Displacement profile for mesh size $h = \delta/8$. Results are for IC 1 and for linear peridynamic (LP). In **a** the waves are traveling toward each other. In **b** and **c**, we see increase in amplitude after merging. In **d** two waves are traveling away from each other

The speed-up $S(p) = {}^{T_1}/_{T_p}$ [44] with respect to the computational time on one node T_1 is shown in Fig. 17a for $p = [1, 2, 3, \ldots, 8]$ where p is the number of CPUs. The straight line shows the optimal speed-up, meaning that we assume if we go from one CPU to two CPUs the code gets two time faster and so. The lines with the circle markers show the speed-up with respect to one single CPU. Here, up to three CPUs, we are close to the optimal speed-up. Later we divergence from the optimal speed-up probably due to the strong scaling and the fixed problem size. In addition, the parallel efficiency $E(p) = {}^{S(p)}/_{p}$ [44] is shown in Fig. 18a for $p = [1, 2, 3, \ldots, 8]$ where p is the number of CPUs. The parallel efficiency is some indication for the fraction of time for which the CPU is doing some computation. The parallel efficiency is above 0.9 for up to three CPUs. For four to five CPUs, the parallel efficiency decays to 0.85 and for six CPUs the efficiency is around 0.82. For seven CPUs, the efficiency increases again to 0.84. For eight CPUs, the efficiency decays again to 0.81. The behavior of the efficiency rate might be related to the fix problem six of the strong scaling. More sophisticated execution policies (e.g., dynamic or adaptive chunk size) could be applied and larger problem sizes, to decrease the computational time. Note that only parallel for loops, synchronization, and futurization were utilized for parallelization.

7 Conclusion

Bond-based and state-based elastic peridynamic material models and implicit and explicit time integration schemes were implemented within a asynchronous many task run time system. These run time systems, like HPX, are essential for utilizing the full performance of the cluster with many cores on a single node.

One important part of the design was the modular expandability for the extensions. New material models can be integrated into the code by inheriting the functions of an abstract class. Consequently, only the material-specific functionality, like forces or strain, is provided by the user and implementation details for the parallelism and concurrency are hidden from the user as much as possible.

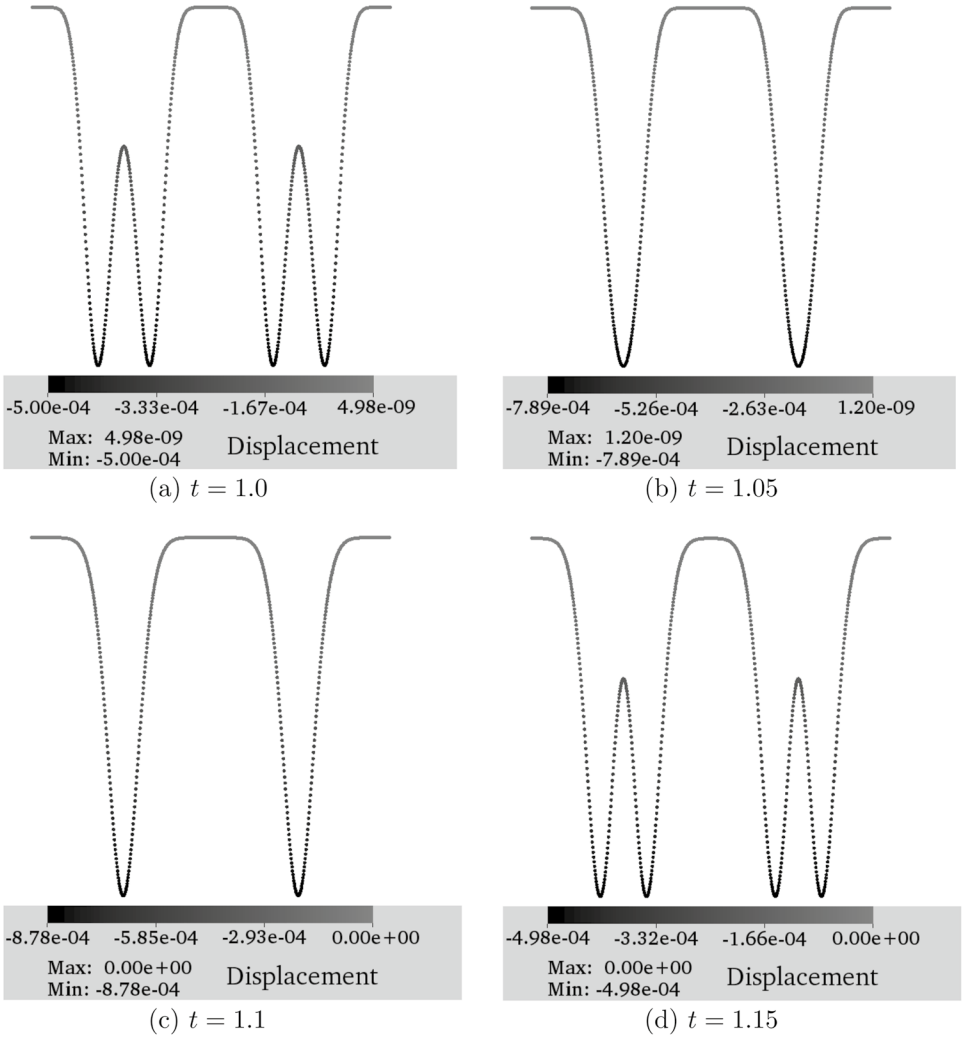


Fig. 13 Displacement profile for mesh size $h = \delta/8$. Results are for IC 2 and for linear peridynamic (LP). In **a** the two waves in left side (X=0.25) and right side (X=0.75) approach toward each other. **b**, **c** correspond to intermediate time before the wave divides into two smaller waves moving in opposite direction in (**d**)

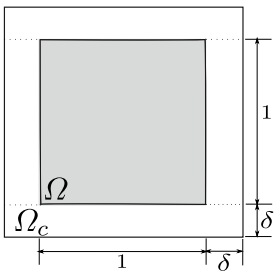


Fig. 14 Square domain (gray area) $\Omega = [0, 1]^2$ and a nonlocal boundary $\Omega_c = [-\delta, 1 + \delta]^2 - [0, 1]^2$. The area outside Ω and within the outer boundary is Ω_c. Dashed lines show the division of Ω_c into left, right, bottom, and top parts

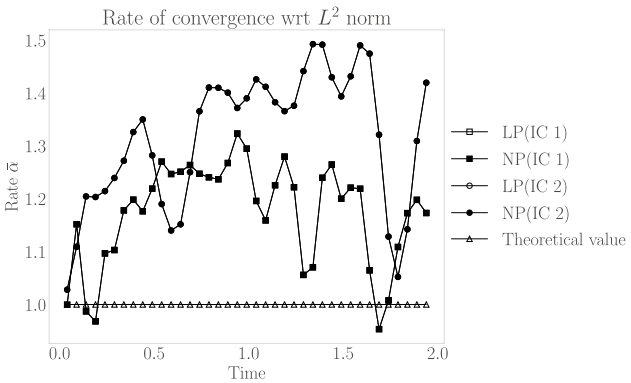


Fig. 15 Time vs rate of convergence with respect to mesh size. The boundary condition is $\mathbf{u} = (0, 0)$ on the non-local boundary $\Omega_c = [-\delta, 1 + \delta]^2 - [0, 1]^2$. IC 1 and IC 2 refer to the two initial conditions described in Eqs. (5.11) and (5.12). The rate of convergence is similar for linear (LP) and nonlinear (NP) peridynamics

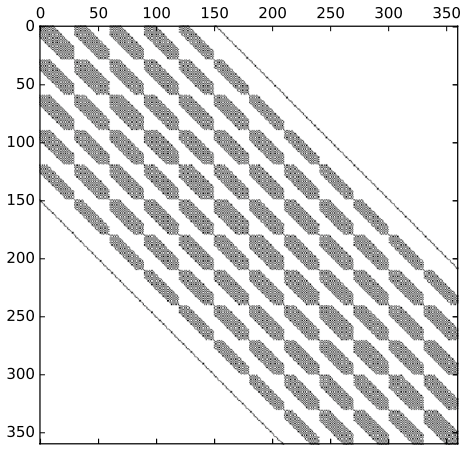
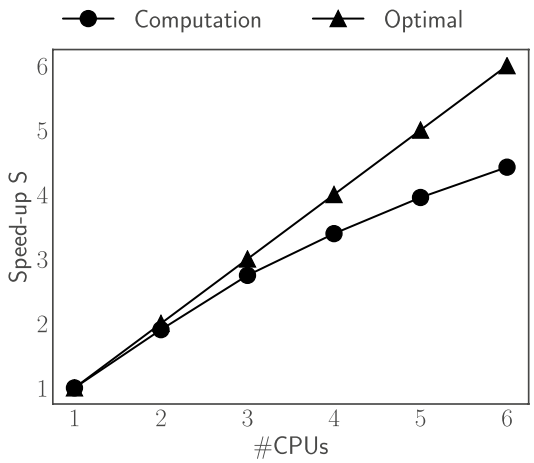


Fig. 16 Nonzero matrix elements (20436) of the tangent stiffness matrix K with the condition number $\kappa(K) = {}^{|K|_{L_2}}/_{|K^{-1}|_{L_2}} = 90.688$ as defined in [52]

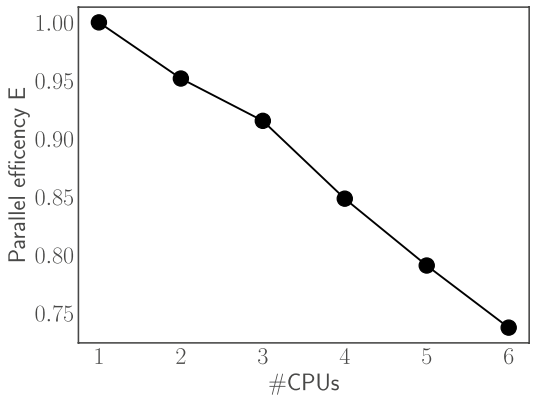
Additional HPX-related functionality needs to be considered for the extension to other integration schemes.

Materials models and the different time integration schemes were validated against theoretical solutions and classical continuum mechanics. All are in good agreement with reference solutions. The convergence rate was shown to be closer to theoretical value, which suggests that the code behaves as expected. The solutions converge to the exact solution at a rate of 1.

The code scaling with respect to computational resource is important and our benchmark results show that the scaling achieved is very close to the theoretical estimates. In addition, the speed-up S and the parallel efficiency E are reasonable for a non-optimized code using default execution policies. Both integration schemes were compared against theoretical estimations. The trend of the theoretical estimates fits with measured computational time and both integration schemes scale with increasing amounts of CPUs. These results were obtained by the default execution policies without any optimization.

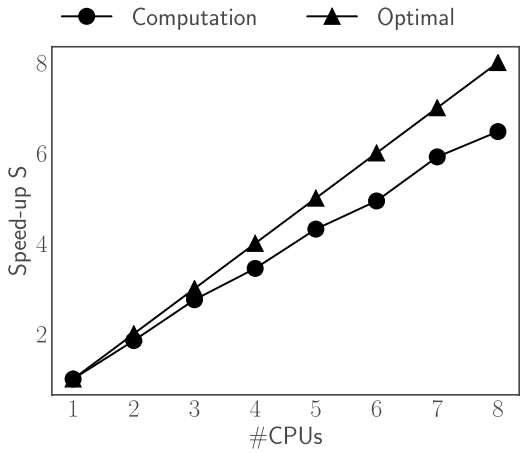


(a) Speed-up $\mathsf{S}(p) = {}^{T_1}/_{T_p}$ with respect to the computational time on one node T_1 for $p = [1,2,3,\ldots,6]$ where p is the number of CPUs. Note that strong scaling was used.

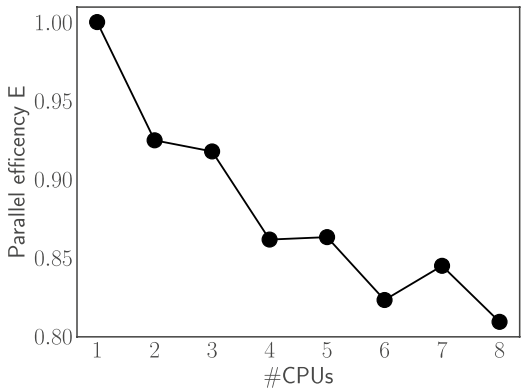


(b) Parallel efficiency $\mathsf{E}(p) = {}^{S(p)}/_{p}$ for $p = [1,2,3,\ldots,6]$ where p is the number of CPUs. The parallel efficiency is some indication for the fraction of time for which the CPU is doing some computation.

Fig. 17 The speed-up S **a** and parallel efficiency E **b** for the test case in Sect. 5.1.2 using the implicit scheme. Note that all presented results are for strong scaling



(a) Speed-up $S(\text{CPUs}) = {}^{T_1}/_{T_{\text{CPUs}}}$ with respect to the computational time on one node T_1 for CPUS $= [1,2,3,\ldots,8]$ where p is the number of CPUs.



(b) Parallel efficiency $E(p) = {}^{S(p)}/_{p}$ for $p = [1,2,3,\ldots,6]$ where p is the number of CPUs. The parallel efficiency is some indication for the fraction of time for which the CPU is doing some computation.

Fig. 18 The speed-up *S* **a** and parallel efficiency *E*. Note that all presented results are for strong scaling

Funding This material is based upon work supported by the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number "W911NF1610456." This work was partly funded by DTIC Contract "FA8075-14-D-0002/0007" and the Center of Computation & Technology at Louisiana State University.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Augonnet C, Thibault S, Namyst R, Wacrenier PA (2009) StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. European conference on parallel processing. Springer, New York, pp 863–874
2. Biddiscombe J, Heller T, Bikineev A, Kaiser H (2017) Zero copy serialization using RMA in the distributed task-based HPX runtime. In: Proceedings of the 14th international conference on applied computing. IADIS, International Association for Development of the Information Society
3. Blumofe RD, Joerg CF, Kuszmaul BC, Leiserson CE, Randall KH, Zhou Y (1995) Cilk: an efficient multithreaded runtime system. In: Proceedings of the fifth ACM SIGPLAN symposium on principles and practice of parallel programming (PPoPP). Santa Barbara, California, pp 207–216
4. Chamberlain BL, Callahan D, Zima HP (2007) Parallel programmability and the chapel language. Int J High Perform Comput Appl 21:291–312
5. Charles P, Grothoff C, Saraswat V, Donawa C, Kielstra A, Ebcioglu K, von Praun C, Sarkar V (2005) X10: an object-oriented approach to non-uniform cluster computing. In: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOPSLA '05. ACM, New York, pp. 519–538
6. Chen X, Gunzburger M (2011) Continuous and discontinuous finite element methods for a peridynamics model of mechanics. Comput Methods Appl Mech Eng 200(9):1237–1250
7. Daiß G, Amini P, Biddiscombe J, Diehl P, Frank J, Huck K, Kaiser H, Marcello D, Pfander D, Pfüger D (2019) From Piz Daint to the stars: simulation of stellar mergers using high-level abstractions. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, pp. 1–37
8. D'Elia M, Li X, Seleson P, Tian X, Yu Y (2019) A review of local-to-nonlocal coupling methods in nonlocal diffusion and nonlocal mechanics. arXiv preprint arXiv:1912.06668
9. Diehl P (2012) Implementierung eines Peridynamik-Verfahrens auf GPU. Diplomarbeit, Institute of Parallel and Distributed Systems, University of Stuttgart
10. Diehl P (2020) Validation 2d. https://doi.org/10.6084/m9.figshare.12665333.v1
11. Diehl P (2020) Validation of a one-dimensional bar. https://doi.org/10.6084/m9.figshare.12343991.v1
12. Diehl P, Prudhomme S, Lévesque M (2019) A review of benchmark experiments for the validation of peridynamics models. J Peridyn Nonlocal Model 1:14–35
13. Edwards HC, Trott CR, Sunderland D (2014) Kokkos: enabling manycore performance portability through polymorphic memory access patterns. J Parallel Distrib Comput 74(12):3202–3216
14. Emmrich E, Weckner O (2007) The peridynamic equation and its spatial discretisation. Math Model Anal 12(1):17–27
15. Grun P, Hefty S, Sur S, Goodell D, Russell RD, Pritchard H, Squyres JM (2015) A brief introduction to the openfabrics interfaces-a new network api for maximizing high performance application efficiency. In: Proceedings of the 2015 IEEE 23rd annual symposium on high-performance interconnects. IEEE, pp 34–39
16. Heller T, Kaiser H, Schäfer A, Fey D (2013) Using HPX and LibGeoDecomp for scaling HPC applications on heterogeneous supercomputers. In: Proceedings of the workshop on latest advances in scalable algorithms for large-scale systems. ACM, p 1
17. Heller T, Lelbach BA, Huck KA, Biddiscombe J, Grubel P, Koniges AE, Kretz M, Marcello D, Pfander D, Serio A, Frank J, Clayton GC, Pflger D, Eder D, Kaiser H. Harnessing billions of tasks for a scalable portable hydrodynamic simulation of the merger of two stars. Int J High Perform Comput Appl
18. Iglberger K, Hager G, Treibig J, Rüde U (2012) Expression templates revisited: a performance analysis of current methodologies. SIAM J Sci Comput 34(2):C42–C69
19. Iglberger K, Hager G, Treibig J, Rüde U (2012) High performance smart expression template math libraries. In: Proceedings of the

2012 international conference on high performance computing simulation (HPCS), pp 367–373
20. Javili A, Morasata R, Oterkus E, Oterkus S (2019) Peridynamics review. Math Mech Solids 24(11):3714–3739
21. Jha PK, Lipton R (2018) Numerical analysis of nonlocal fracture models in hölder space. SIAM J Numer Anal 56(2):906–941
22. Jha PK, Lipton R (2018) Numerical convergence of nonlinear nonlocal continuum models to local elastodynamics. Int J Numer Methods Eng 114(13):1389–1410
23. Jha PK, Lipton R (2019) Numerical convergence of finite difference approximations for state based peridynamic fracture models. Comput Methods Appl Mech Eng 351:184–255
24. Kaiser H, Brodowicz M, Sterling T (2009) Parallex an advanced parallel execution model for scaling-impaired applications. In: Parallel processing workshops, 2009. ICPPW'09. IEEE, pp 394–401
25. Kaiser H, Diehl P, Lemoine AS, Lelbach BA, Amini P, Berge A, Biddiscombe J, Brandt SR, Gupta N, Heller T, Huck K, Khatami Z, Kheirkhahan A, Reverdell A, Shirzad S, Simberg M, Wagle B, Wei W, Zhang T (2020) HPX: the C++ standard library for parallelism and concurrency. J Open Sour Softw 5(53):2352 10.21105/joss.02352
26. Kaiser H, Heller T, Adelstein-Lelbach B, Serio A, Fey D (2014) Hpx: a task based programming model in a global address space. In: Proceedings of the 8th international conference on partitioned global address space programming models. ACM, p 6
27. Kaiser H, Heller T, Bourgeois D, Fey D (2015) Higher-level parallelization for local and distributed asynchronous task-based programming. In: Proceedings of the first international workshop on extreme scale programming models and middleware, ESPM '15. ACM, New York, pp 29–37
28. Khatami Z, Kaiser H, Grubel P, Serio A, Ramanujam J (2016) A massively parallel distributed n-body application implemented with hpx. In: Proceedings of the 2016 7th workshop on latest advances in scalable algorithms for large-scale systems (ScalA). IEEE, pp 57–64
29. Kilic B (2008) Peridynamic theory for progressive failure prediction in homogeneous and heterogeneous materials. The University of Arizona
30. Kunin IA (2011) Elastic media with microstructure I: one-dimensional models (Springer Series in Solid-State Sciences), softcover reprint of the original, 1st ed, 1982nd edn. Springer, New York
31. Kunin IA (2012) Elastic media with microstructure II: three-dimensional models (Springer Series in Solid-State Sciences), softcover reprint of the original, 1st ed, 1983rd edn. Springer, New York
32. Lipton R (2014) Dynamic brittle fracture as a small horizon limit of peridynamics. J Elast 117(1):21–50
33. Lipton R (2016) Cohesive dynamics and brittle fracture. J Elast 124(2):143–191
34. Lipton RP, Lehoucq RB, Jha PK (2019) Complex fracture nucleation and evolution with nonlocal elastodynamics. J Peridyn Nonlocal Model 1(2):122–130
35. Littlewood DJ (2015) Roadmap for Peridynamic Software Implementation. Tech Rep 2015-9013, Sandia National Laboratories
36. Mossaiby F, Shojaei A, Zaccariotto M, Galvanetto U (2017) Opencl implementation of a high performance 3d peridynamic model on graphics accelerators. Comput Math Appl 74(8):1856–1870
37. OpenMP Architecture Review Board: OpenMP V5.0 Specification (2018). https://www.openmp.org/wp-content/uploads/OpenMPRef-5.0-0519-web.pdf
38. Parks M, Littlewood D, Mitchell J, Silling S (2012) Peridigm Users' Guide. Tech Rep SAND2012-7800, Sandia National Laboratories
39. Parks ML, Lehoucq RB, Plimpton SJ, Silling SA (2008) Implementing peridynamics within a molecular dynamics code. Comput Phys Commun 179(11):777–783
40. Pfander D, Daiß G, Marcello D, Kaiser H, Pflüger D (2018) Accelerating octo-tiger: stellar mergers on intel knights landing with HPX. In: Proceedings of the international workshop on OpenCL, IWOCL '18. ACM, New York, pp 19:1–19:8
41. Pharr M, Mark WR (2012) ISPC: a SPMD compiler for high-performance cpu programming. In: Proceedings of the 2012 innovative parallel computing (InPar), pp 1–13. https://doi.org/10.1109/InPar.2012.6339601
42. Pheatt C (2008) Intel®R threading building blocks. J Comput Sci Coll 23(4):298
43. Raicu I, Foster IT, Zhao Y (2008) Many-task computing for grids and supercomputers. In: Proceedings of the 2008 workshop on many-task computing on grids and supercomputers, pp 1–11
44. Rodgers DP (1985) Improvements in multiprocessor system design. ACM SIGARCH Comput Archit News 13(3):225–231
45. Ross PE (2008) Why cpu frequency stalled. IEEE Spectrum 45(4)
46. Sadd MH (2009) Elasticity: theory, applications, and numerics. Academic Press, Cambridge
47. Seleson P, Littlewood DJ (2016) Convergence studies in meshfree peridynamic simulations. Comput Math Appl 71(11):2432–2448
48. Seo S, Amer A, Balaji P, Bordage C, Bosilca G, Brooks A, Carns PH, Castell A, Genet D, Hérault T, Iwasaki S, Jindal P, Kalx LV, Krishnamoorthy S, Lifflander J, Lu H, Meneses E, Snir M, Sun Y, Taura K, Beckman PH (2018) Argobots: a lightweight low-level threading and tasking framework. IEEE Trans Parallel Distrib Syst 29:512–526
49. Silling SA (2000) Reformulation of elasticity theory for discontinuities and long-range forces. J Mech Phys Solids 48(1):175–209
50. Silling SA, Askari E (2005) A meshfree method based on the peridynamic model of solid mechanics. Comput Struct 83(17):1526–1535
51. Silling SA, Epton M, Weckner O, Xu J, Askari E (2007) Peridynamic states and constitutive modeling. J Elast 88(2):151–184
52. Strang G (1980) Linear algebra and its applications. Academic Press Inc., Cambridge
53. Sutter H (2005) The free lunch is over: a fundamental turn toward concurrency in software. Dr. Dobb's J 30(3):202–210
54. The C++ Standards Committee: ISO International Standard ISO/IEC 14882:2011 (2011) Programming Language C++. Technical report., Geneva, Switzerland: International Organization for Standardization (ISO). http://www.open-std.org/jtc1/sc22/wg21
55. The C++ Standards Committee: ISO International Standard ISO/IEC 14882:2017 (2017) Programming Language C++. Technical report, Geneva, Switzerland: International Organization for Standardization (ISO). http://www.open-std.org/jtc1/sc22/wg21
56. Thoman P, Dichev K, Heller T, Iakymchuk R, Aguilar X, Hasanov K, Gschwandtner P, Lemarinier P, Markidis S, Jordan H et al (2018) A taxonomy of task-based parallel programming technologies for high-performance computing. J Supercomput 74(4):1422–1434
57. Wang H, Tian H (2012) A fast Galerkin method with efficient matrix assembly and storage for a peridynamic model. J Comput Phys 231(23):7730–7738
58. Weckner O, Emmrich E (2005) Numerical simulation of the dynamics of a nonlocal, inhomogeneous, infinite bar. J Comput Appl Mech 6(2):311–319
59. Wheeler KB, Murphy RC, Thain D (2008) Qthreads: an api for programming with millions of lightweight threads. In: Proceedings of the 2008 IEEE international symposium on parallel and distributed processing. IEEE, pp 1–8