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

Patrick Diehl^{1,3,4,*}, Prashant K. Jha², Hartmut Kaiser^{3,4}, Robert Lipton², Martin Levesque¹

- 1. Department for Multi scale Mechanics, Polytechnique Montreal
 - 2. Department of Mathematics, Louisiana State University
- 3. Center of Computation & Technology, Louisiana State University
 4. Stel|ar group
 - * patrickdiehl@lsu.edu (https://orcid.org/0000-0003-3922-8419)

April 2, 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 the focus on handle the fine-grain parallelism of this increasing amount of cores per computational node is needed. Asynchronous Many Task (AMT) run time systems represent an emerging paradigm for addressing fine-grain parallelism since 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 confirm with the C++ standard. Peridynamics is a non-local generalization of continuum mechanics tailored to address discontinuous displacement fields arising in fracture mechanics. As many non-local approaches, peridynamics requires considerable computing resources to solve practical problems. This paper investigates the implementation of a peridynamics EMU nodal discretization implementation in a asynchronous task-based fashion. The scalability of asynchronous task-based implementation is to be in agreement with theoretical estimations. In addition, to the scalabilty the code is convergent for implicit time integration and recovers theoretical solutions. Explicit time integration, convergence results are presented to showcase the agreement of results with theoretical claims in previous works.

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 [1, 2]. Through this shift of increasing cores per node, a new programming model with the focus on handle the fine-grain parallelism of this increasing amount of cores per computational node is needed.

Asynchronous Many Task (AMT) [3] run time systems represent an emerging paradigm for addressing fine-grain parallelism since they handle the increasing amount of threads per node and concurrency [4]. There are many solutions for task-based programming models available. These approaches can be distinct into three classes: (1) Library solutions, e.g. StarPU [5], Intel TBB [6], Argobots [7], Qthreads [8], Kokkos [9], and the C++ standard library for parallelism and concurrency (HPX) [10, 11, 12] (2) language extensions, e.g. Intel Cilk Plus [13] and OpenMP 3.0 [14], and (3) programming languages, e.g. Chapel [15], Intel ISPC [16], and X10 [17].

Most of the tasked-based libraries are based on C or C++ programming language. The C++ 11 programming Language standard laid the foundations for concurrency by introducing futures, that have shown to support the concept of futurization to enable a task-based parallelism [18]. In addition, the support for parallel execution with the so-called parallel algorithms were introduced in the C++ 17 standard [19].

HPX is an open source asynchronous many task run time system that focuses on high performance computing [12, 10, 11]. 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 is in strict adherence to the C++ 11 [18] and C++ 17 standard definitions [19].

The HPX library was recently utilized to parallelize a N-Body code and the asynchronous task-based implementation and was compared against non-AMT implementations [20, 21]. In many cases the HPX implementation outperformed the MPI/OpenMP implementation. The HPX-library has been utilized in an astrophysics applications for the time evolution of a rotating star on a Xeon Phi resulting in a speed up by factor of two with respect to a 24-core Skylake SP platform [22]. On the NERSC's Cori super computer the simulation of the Merger of two stars could achieves 96.8% parallel efficiency on 648,280 Intel Knight's landing cores [23] and 68.1% parallel efficiency on 2048 cores [24] of Swiss National Supercomputing Centre's Piz Daint super computer. In addition, a speedup up to 2.8x was shown on full system run on Piz Daint by replacing the Message Passing Interface with libfabric [25] for communication. Motivated by the acceleration seen for these computational codes here we address an application to computational fracture mechanics. We address fracture problems framed in terms of the peridynamic formulation. In this treatment we focus on how to implement peridynamics in a task-based fashion allowing us to be prepared for the new architecture designed for swift modern super computers.

Peridynamics is a non-local generalization of continuum mechanics, tailored to address discontinuous displacement fields that arise in fracture mechanics [26, 27, 28, 29, 30, 31]. Several peridynamics implementations utilizing the EMU nodal discretization [32] are available. Peridigm [33] and PDLammps [34] rely on the widely used Message Passing Interface (MPI) for the inter-node parallelization. Other approaches rely on acceleration cards, like OpenCL [35] and CUDA [36], for parallelization. These single device GPU approaches, however, cannot deal with large node clouds due to current hardware memory limitations on GPUs.

Computational methods for dynamic fracture problems require substantial computational resources beyond those used in classical linear elastodynamics problems. For nonlocal approaches to fracture given by the peridynamic formulation the cracks are part of the solution and emerge and dynamically interact with each other. Several authors have devised local-non-local bridging schemes to target domains where peridynamics calculations are required and domains where local approaches for linear elastodynamics yield acceptable results, to reduce the computational costs [37, 38, 39]. However, verification and validation of peridynamics codes [40] still require important computational resources [41, 42, 43, 44, 45] and the community would greatly benefit from efficient implementations.

This paper presents two peridynamics models discretized with the EMU nodal discretization making use of the features of AMT within HPX. We show in this paper how to take advantage of the fine-grain parallelism arising on modern supercomputers. In addition, the scalability of the algorithm is compared to its theoretical complexity (see Figure 17a and Figure 18a) and the promised work stealing of HPX is addressed (see Figure 17b and Figure 18b). 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 its concept, which are

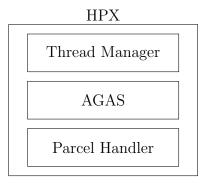


Figure 1: Run time components of HPX: Thread manager, Active Global Address Space (AGAS), and the Parcel handler. The thread manager provides a high-level API for the task-based programming and deals with the light-weight user level threads. The Active Global Address Space (AGAS) provides global identifiers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

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 within the concepts of HPX. The parallel implementation is validated against classical continuum mechanics and numerical analysis in Section 5. The actual computational time for the HPX peridynamics implementation is benchmarked against the theoretical computation time in Section 6. Section 7 concludes this work.

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

The HPX library [10, 11, 12] is a C++ standard compliant Asynchronous Many Task (AMT) run time system tailored for high performance applications (HPC) applications. Figure 1 shows the three components provided by HPX. First, the thread manager [11], which provides a high-level API for the task-based programming and deals with the light-weight user level threads. Second, the Active Global Address Space (AGAS) [11], which provides the global identifiers to hide the explicit message passing. So the access of a remote or local object is unified. Third, the parcel handler [46, 47] 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) protocol or the message passing Interface (MPI). For more implementation details about these components, we refer to [11, 46, 47] and for a general overview we refer to [3].

It 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, are 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

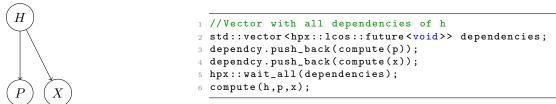


Figure 2: Example dependency graph.

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

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

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

::future<T>::then are available for combining futures and generate parallel executions graphs [48].

A typical example dependency graph is shown in Figure 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 the previous two futures computations are finished.

2.2 Parallelization

Consider the addition of n elements for the two vectors \mathbf{p} and \mathbf{x} , where the sum is stored piece-wise in vector \mathbf{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 $\mathtt{hpx}::\mathtt{parallel}::\mathtt{for}_\mathtt{loop}$. 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 $\mathtt{hpx}::\mathtt{parallel}::\mathtt{execution}::\mathtt{par}(\mathtt{hpx}::\mathtt{parallel}::\mathtt{execution}::\mathtt{task})$.

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

```
//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.

```
std::vector<hpx::lcos::future<void>> dep;
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)
     p[i] = p[i]+1;
9 });
dep.push_back(hpx::parallel::for_loop(
hpx::parallel::execution::par(
hpx::parallel::execution::task),
13 0,z,[x](boost::uint64_t i)
     x[i] = x[i]-1;
15
16 });)
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)
     h[i] = p[i] + x[i];
23
24 });
26 hpx::wait_all(dep).then(f);
```

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

and then describes the sum's dependencies.

3 Peridynamics theory

In this section, we briefly introduce the key feature of peridyanmics theory essential for the implementation. For more details about PD we refer to [49, 50, 51] and for the utilized material models to [52, 41, 42].

Let a material domain be $\Omega_0 \subset \mathbb{R}^d$, for d=1,2, and 3. Peridynamics (PD) [53, 54] 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 Figure 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. The vector $\boldsymbol{\eta} := \mathbf{u}(t,\mathbf{X}') - \mathbf{u}(t,\mathbf{X})$ is called the bond-deformation vector and $\boldsymbol{\xi} := \mathbf{X}' - \mathbf{X}$ denotes the initial bond vector.

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

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

where **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 $\dot{\mathbf{u}}(0,\mathbf{X}) = v_0(\mathbf{X})$. In contrast to local problems, boundary conditions are non-local and are specified on a boundary layer or collar Ω_c that surrounds the domain Ω_0 .

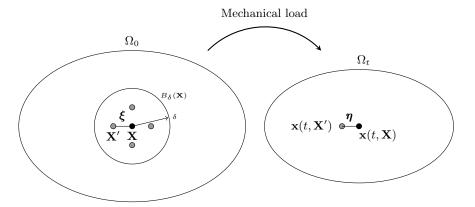


Figure 3: Left: Schematic representation of PD where the material point $\mathbf{X} \in \Omega$ interacts non-locally with all other material points inside $B_{\delta}(\mathbf{X})$. Right: Positions $\mathbf{x}(t, \mathbf{X})$ and $\mathbf{x}(t, \mathbf{X}')$ of the material points in the configuration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

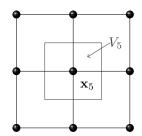


Figure 4: The initial positions of the discrete nodes $\mathbf{X} := {\mathbf{X}_i \in \mathbb{R}^3 : i = 1, ..., 9}$ in the reference configuration Ω_0 . For the discrete node \mathbf{x}_5 its surrounding volume V_5 is schematized.

The formulation given above depends on two-point interactions and is referred to as a bond-based peridynamics model. Bond based models allow for only two point non-local interactions and because of this the material's Poisson ratio is constrained to 0.25 [55, 56]. On the other hand, state-based peridynamics models [53] 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

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

Equation (3.2) is complemented by initial conditions $\mathbf{u}(0, \mathbf{X}) = \mathbf{u}_0(\mathbf{X})$ and $\dot{\mathbf{u}}(0, \mathbf{X}) = v_0(\mathbf{X})$. The description of initial boundary value problem is completed by the non-local boundary conditions described in the next section.

3.1 Discretization of peridynamics equations

Continuous and discontinuous Galerkin finite element methods [57], Gauss quadrature [58] and spatial discretization [59, 34] are different discretization approaches for PD. Owing to its efficient load distribution scheme, the method of Silling and Askari [60, 34], 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 $\mathbf{X} := {\mathbf{X}_i \in \mathbb{R}^3 : i = 1, ..., n}$ are considered to represent the material domain,

see Figure 4. The discrete nodal spacing h between \mathbf{X}_j and \mathbf{X}_i is given by $h = |\mathbf{X}_j - \mathbf{X}_i|$. 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 and $\sum_i^n V_i = V_{\Omega_0}$, 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 $\mathbf{X}_i \in \Omega_0$,

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

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

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

There are 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, [40]]. In both Equation (3.3) and (3.4) we replace V_j by V_jV_{ij} to correctly account for the volume.

4 Implementation of PeridynamicHPX

PeridynamicHPX is an modern C++ code utilizing HPX for parallelism. The classes design is modular and template-based to easily extend the code with new material models. The code's design is presented herein and parallelism and concurrency utilization within HPX is detailed.

4.1 Design with respect to modular expandability

Figure 5 shows the modular design class. PeridynamicHPX 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 YAML¹ file format. Each new Deck inherits the common functions from the AbstractDeck and is extended with the new problem/material specific attributes.

Second, the abstractions for bond-based and state-based materials are provided in the *Material* module. The nonlinear bond-based elastic material in Section 4.2.1 and the linear state-based elastic material of Section 4.2.2 were implemented. The abstract material must be inherited and the abstract methods, e.g. force and strain, are implemented if a new material model is to be implemented.

The different time integration schemes and the discretizations are considered third. All new *Problem* classes inherit their common and abstract functions from *AbstractProblem*. Note that a problem implementation takes the abstract classes as arguments. 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. For new problem classes, the user must 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.

 $^{^{1}\}mathrm{http://yaml.org/}$

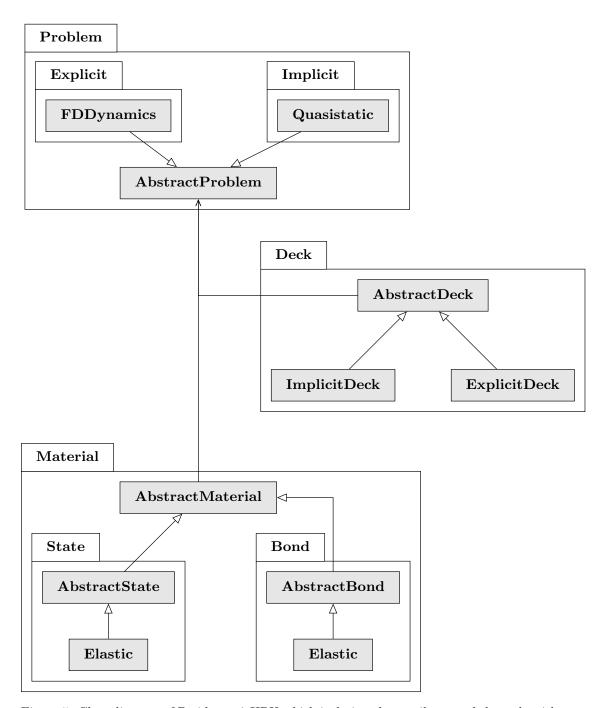


Figure 5: Class diagram of PeridynamicHPX which is designed to easily extend the code with new materials or discretizations. The functionality of the code is defined in three packages: Problem containing the different classes for discretizations, Deck which handles the input and output, and Material 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.

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 [41, 42].

```
1: future f1 =

2: parallel_for i < n do \trianglerightCompute force at mesh nodes

3: for j \in B_{\delta}(\mathbf{X}_i) do

4: \boldsymbol{\xi} = \mathbf{X}_j - \mathbf{X}_i

5: S = \frac{\mathbf{u}(\mathbf{X}_j) - \mathbf{u}(\mathbf{X}_i)}{|\boldsymbol{\xi}|} \cdot \frac{\boldsymbol{\xi}}{|\boldsymbol{\xi}|}

6: \trianglerightCompute force at \mathbf{X}_i as in [41, 42]

7: \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

8: \trianglerightCompute strain energy at \mathbf{X}_i as in [41, 42]

9: U(\mathbf{X}_i) + = \frac{1}{|B_{\delta}(\mathbf{0})|} J^{\delta}(|\boldsymbol{\xi}|) \psi(|\boldsymbol{\xi}|S^2) V_j/\delta

10: end for

11: end

12: hpx::wait_all(f1)
```

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 [61]. The implementation of the state-based material and the implicit time integration is adapted from [62]. 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 energy. For demonstration, we consider nonlinear bond-based peridynamic (NP) model introduced in [41, 42]. 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 Section 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.

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 [53]. Algorithm 2 shows the adapted algorithm [62] parallelized and synchronized with HPX. First, the weighted volume m is computed for each node in parallel. Second, the dilation θ is computed for each node in parallel. The internal force density and the strain energy can be computed independently from each other. Therefore, the execution policy px::parallel::execution::task is utilized to obtain futures back of these two loops. Third, the internal force is computed asynchronously. Fourth, the strain energy is computed asynchronously, when needed, e.g. for output. A synchronization for these two futures is needed before the force and strain energy are computed in future steps.

4.2.3 Implicit time integration

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

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

Algorithm 2 Computation of internal force and strain energy. Adapted from [62]

```
1: parallel_for i < n do \trianglerightCompute weighted volumes as in [53, 62]
               m_i = 0
               for j \in B_{\delta}(\mathbf{X}_i) do
  3:
                      \boldsymbol{\xi} = \mathbf{X}_j - \mathbf{X}_i
  4:
                       m_i + = |\boldsymbol{\xi}|^2 V_j
  5:
               end for
  6:
  7: end
  8: parallel_for i < n do \trianglerightCompute dilatation as in [53, 62]
  9:
               for j \in B_{\delta}(\mathbf{X}_i) do
10:
                      m{\xi} = \mathbf{X}_j - \mathbf{X}_i
m{\eta} = \mathbf{u}(\mathbf{X}_j) - \mathbf{u}(\mathbf{X}_i)
11:
12:
                       \underline{e} = |\boldsymbol{\xi} + \boldsymbol{\eta}| - |\boldsymbol{\xi}|
13:
                       \theta_i + = 3/m_i |\xi| \underline{e} V_j
14:
               end for
15:
17: parallel_for i < n do \trianglerightCompute internal forces as in [53, 62]
               for j \in B_{\delta}(\mathbf{X}_i) do
18:
                      egin{aligned} oldsymbol{\xi} &= \mathbf{X}_j - \mathbf{X}_i \ oldsymbol{\eta} &= \mathbf{u}(\mathbf{X}_j) - \mathbf{u}(\mathbf{X}_i) \ &\underline{e}^d &= e - (	heta_i | oldsymbol{\xi}|)/3 \end{aligned}
19:
20:
21:
                      \underline{t} = 3/m_i K \theta_i |\mathbf{\xi}| + (15\mu)/m_i \underline{e}^d
22:
                       \underline{M} = \eta + \xi/|\xi + \eta|
23:
                      \overline{\mathbf{f}_i} + = \underline{tM} V_j
24:
                       \mathbf{f}_j - = \underline{tM}V_i
25:
               end for
26:
27: end
```

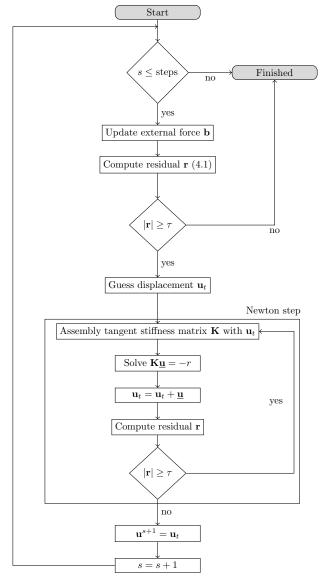


Figure 6: Flow chart of the implicit time integration scheme adapted from [62]. For each time step s the external force $\mathbf b$ is updated and the residual $\mathbf r$ is evaluated. When the norm of the residual is larger than the tolerance τ the displacement $\mathbf u^{s+1}$ is obtained via a Newton step.

Algorithm 3 Assembly of the tangent stiffness matrix by central finite difference. Adapted from [62]

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

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

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

is assembled as described in [62], (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 $\pm v$, where v is infinitesimally small. Line 9 computes the internal forces $f^{\pm v}$ and Line 13 evaluates the central difference to construct the stiffness matrix $\mathbf{K}(u)$. Note that the nodes 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) [63], a conjugate gradient method (CG), or a Galerkin method [64] could be used as well.

The high-performance open-source C++ library Blaze [65, 66] was used for matrix and vector operations. Blaze supports HPX for parallel execution and can be easily integrated. The library BlazeIterative² was used for solving $\mathbf{K}\underline{\mathbf{u}} = -\mathbf{r}$. The Biconjugate gradient stabilized method (BiCGSTAB) or the conjugate gradient (CG) solver were used for solving.

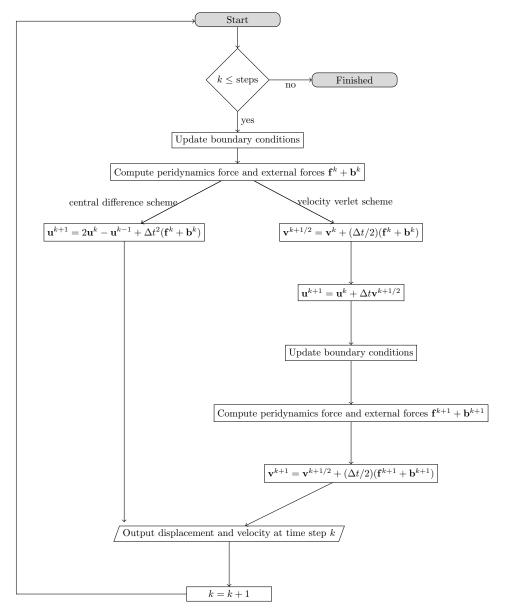


Figure 7: Flow chart of the explicit time integration scheme. For each time step k the boundary conditions are updated and the internal and external forces are computed. Depending on a central difference scheme or a velocity scheme the displacement \mathbf{u}^{k+1} and velocity \mathbf{v}^{k+1} is obtained.

Algorithm 4 Time integration using velocity-verlet scheme

```
1: ⊳Loop over time steps
  2: for 0 \le 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}
  3:
  4:
              parallel_for i < n do
  5:
                      ⊳Compute velocity
  6:
                     \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))
  7:
                     ⊳Compute displacement
  8:
                     \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)
  9:
10:
             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}
parallel_for i < n do \trianglerightLoop over nodes
11:
12:
13:
                      ⊳Compute velocity
14:
                     \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}_{ext}^{k+1}(\mathbf{X}_i))
15:
16:
17:
18: end for
```

4.2.4 Explicit time integration

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

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

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

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

$$(4.3)$$

to obtain the displacement \mathbf{u}^{k+1} for the time step k+1. Line 3 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

$$\mathbf{u}(t^{k+1}, \mathbf{X}_i) = 2\mathbf{u}(t^k, \mathbf{X}_i) - \mathbf{u}(t^{k-1}, \mathbf{X}_i) + \frac{\Delta t^2}{\varrho(\mathbf{X}_i)} \left[\mathbf{b}(t^k, \mathbf{X}_i) + \sum_{B_{\delta}(\mathbf{X}_i)} \mathbf{f}(t^k, \boldsymbol{\eta}, \boldsymbol{\xi}) \right]. \tag{4.4}$$

5 Validation of PeridynamicHPX

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

²https://github.com/tjolsen/BlazeIterative

Figure 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

5.1 Implicit

5.1.1 One dimensional tensile test

Consider the simple geometry of Figure 9 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 m² and a Young's modulus of 4 GPa, the resulting strain reads $\varepsilon_{\text{CCM}} = 1 \times 10^{-8}$ an the stress is of $\sigma_{\text{CCM}} = 40 \,\text{Pa}$. The strain energy density is given by $U_{\text{CCM}} = \sigma^2/(2E) = 2 \times 10^{-7} \,\text{Pa}$.

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 of force $F = -50 \,\mathrm{kN}$ 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 [67] as follows

$$\mathbf{u}_{x}(\mathbf{X}_{i}) = \frac{F}{EWT}(\mathbf{X}_{i_{x}} - W),$$

$$\mathbf{u}_{y}(\mathbf{X}_{i}) = -\frac{\nu F}{ET} \left(\frac{\mathbf{X}_{i_{y}}}{W} - \frac{1}{2} \right),$$
(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 $E = 4000\,\mathrm{MPa}$ 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.

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

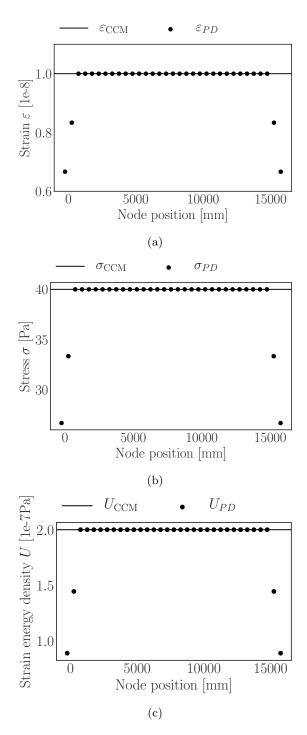


Figure 9: Comparison of strain ϵ (a), stress σ (b), and strain energy U (c) with classical continuum mechanics. Close to the boundary the surface effect influences the accuracy.

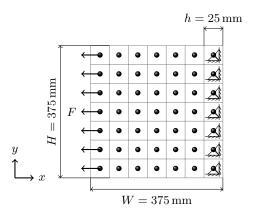


Figure 10: Sketch of the two dimensional tensile test. All nodes on the line of the right-hand side of the plate are clamped in x-direction and y-direction. A force of $-50 \,\mathrm{kN}$ is applied in x-direction to each node of the line on the left-hand side.

| Actual position | CCM | PD | Relative error |
|----------------------------|---|--|----------------------|
| x-direction y -direction | $0.18749\mathrm{m}$ $0.1875\mathrm{m}$ | $0.18744\mathrm{m}$ $0.1875\mathrm{m}$ | 4.9×10^{-5} |

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).

to [Section 6.1.2, [68]], 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, $\underline{C}h^{\alpha} \le ||\mathbf{u}_h - \mathbf{u}_{h'}|| = \bar{C}h^{\alpha}$ with $\underline{C} \le \bar{C}$ 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

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

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

$$\bar{\alpha} = \frac{\log(||\mathbf{u}_1 - \mathbf{u}_2||) - \log(||\mathbf{u}_2 - \mathbf{u}_3||)}{\log(r)}.$$
(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, [61]],

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

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

$$\psi(r) = C(1 - \exp[-\beta r]), \qquad C = \beta = 1.$$
 (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 [69]. 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, [61]],

$$f_{l}(t, u(t, X') - u(t, X), X' - X)) = \frac{2}{\delta^{2}} \int_{X - \delta}^{X + \delta} J^{\delta}(|X' - X|) \psi'(0) S(X', X) dX'.$$
(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 equation (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 Figure 11. Apply either one of the initial conditions:

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

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

$$u_0(X) = \exp[-|X - x_{c1}|^2/\beta]a + \exp[-|X - x_{c2}|^2/\beta]a, \quad v_0(X) = 0,$$
(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} .

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 the Figure 11 near t = 1.1 for both the initial conditions is due to the merging of two waves traveling towards each other. We show the displacement profile near t=1.1 in Figure 12 for IC 1 and Figure 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 Figure 11. This behavior is true for both the initial conditions. Similar results, not shown here, were obtained for the nonlinear model of equation (3.1). The convergence results presented in Figure 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 [42],

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

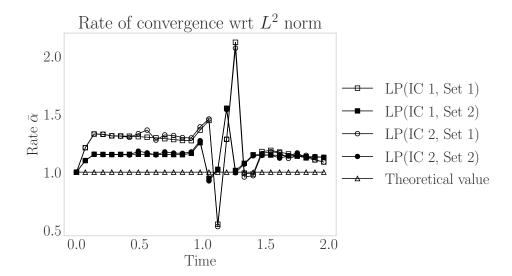


Figure 11: Time vs rate of convergence with respect to mesh size. These results are for linear peridynamic force (LP). Set 1 corresponds to convergence rate obtained from solutions of mesh sizes $\{h_1, h_2, h_3\}$ and set 2 corresponds to convergence rate obtained from solutions of mesh sizes $\{h_2, h_3, h_4\}$. 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.

 $\psi(r)$ 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}_l .

Let $\Omega = [0,1]^2$ be the material domain with a horizon $\delta = 0.1$. The 2-d vector **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 Figure 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

$$u_{0,1}(X_1, X_2) = \exp[-(|X_1 - x_{c,1}|^2 + |X_2 - x_{c,2}|^2)/\beta]a_1,$$

$$u_{0,2}(X_1, X_2) = \exp[-(|X_1 - x_{c,1}|^2 + |X_2 - x_{c,2}|^2)/\beta]a_2,$$

$$v_{0,1}(X_1, X_2) = 0, v_{0,2}(X_1, X_2) = 0,$$
(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):

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

Initial condition 2(IC 2):

$$\mathbf{a} = (0.0002, 0.0007), \mathbf{x}_c = (0.25, 0.25), \text{ and } \beta = 0.01.$$
 (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 [70, 61].

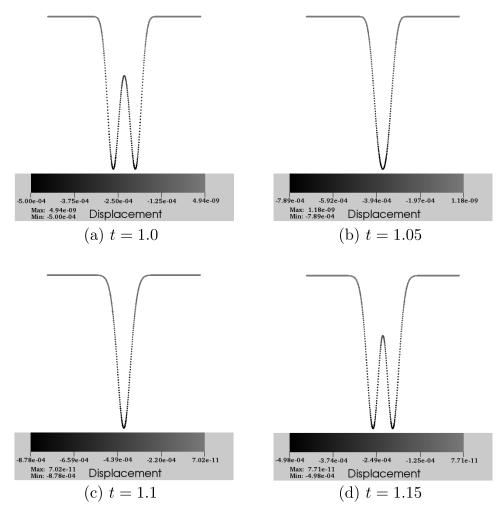


Figure 12: Displacement profile for mesh size $h = \delta/8$. Results are for IC 1 and for linear peridynamic (LP). In (a) the waves are traveling towards each other. In (b) and (c) we see increase in amplitude after merging. In (d) two waves are traveling away from each other.

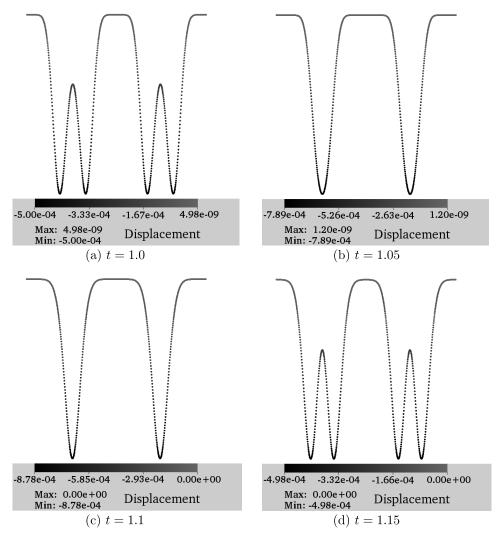


Figure 13: Displacement profile for mesh size $h = \delta/8$. Results are for IC 2 and for linear peridynamic (LP). In (a) the two waves in left side (X=0.25) and right side (X=0.75) approach towards each other. (b) and (c) correspond to intermediate time before the wave divides into two smaller waves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

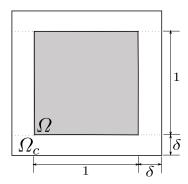


Figure 14: Square domain (grey area) $\Omega = [0,1]^2$ and a nonlocal boundary $\Omega_c = [-\delta, 1+\delta]^2 - [0,1]^2$. The area outside Ω and within the outer boundary is Ω_c . Dashed lines show the division of Ω_c into left, right, bottom, and top parts.

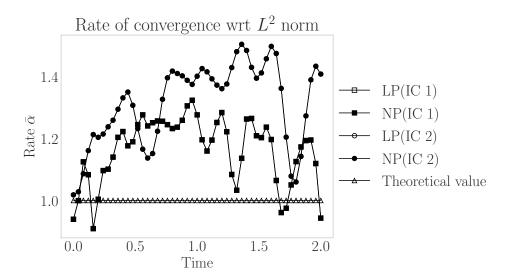


Figure 15: Time vs rate of convergence with respect to mesh size. The boundary condition is $\mathbf{u} = (0,0)$ on the non-local boundary $\Omega_c = [-\delta, 1+\delta]^2 - [0,1]^2$. IC 1 and IC 2 refer to the two initial conditions described in Equation (5.11) and Equation (5.12). The rate of convergence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

6 Benchmark for PeridynamicHPX

6.1 Implicit

The benchmark is realized by comparing the the computational time to the theoretical complexity. The theoretical complexity is the number of operations an algorithm requires to perform its task. The computational time is the measured physical time the program required to complete its tasks.

The algorithm in Figure 6 features several loops for which the maximal amount of iterations can be estimated with $\mathcal{O}(n^2)$, with n being the number of discrete nodes. The computational complexity of the conjugate gradient (CG) solver for k iterations can be estimated with $\mathcal{O}(kn^2)$ [71]. The complexity for the computational time can be approximated with $\mathcal{O}(\frac{n^2+kn}{p})$, where p is the amount of CPUs.

The test case of Section 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 PeridynamicHPX were compiled with gcc 6.2.1, boost 1.61 and blaze 3.2 libraries were used.

Figure 17a shows the obtained computational time for up to 6 CPUs. The figure shows that the computational time and theoretical complexity followed similar trends. Note that only parallel for loops, synchronization, and futurization were utilized for parallelization.

Figure 17b shows the CPUs idle-rates. The idle-rate is obtained with the performance counters within HPX and measured, in percentage, how long the CPU was idling with respect to the overall computation time. Here, the idle-rate is 0.01%, for the default execution policy. These observations suggest that the implicit time integration seems to scale with the same trend as the theoretical complexity without any code optimization. More sophisticated execution policies (e.g. dynamic or adaptive chunk size) could be applied, to decrease the computational time.

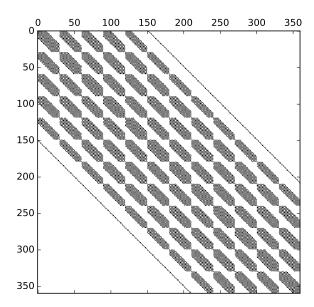


Figure 16: Nonzero matrix elements (20436) of the tangent stiffness matrix K with the condition number $\kappa(K) = \frac{|K|_{L_2}}{|K^{-1}|_{L_2}} = 90.688$ as defined in [72].

6.2 Explicit

The setup presented in Section 5.2.2 was discretized with 196249 nodes and an horizon of $0.05\,\mathrm{m}$ 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 PeridynamicHPX were compiled with gcc 7.2, boost 1.61 and blaze 3.2 libraries.

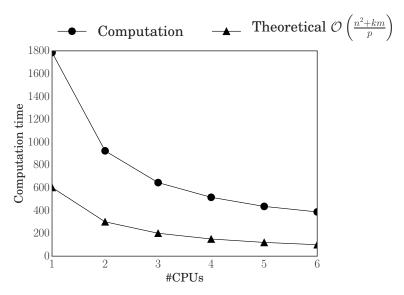
Figure 18a shows the measured computational time for up to 8 CPUs. The algorithm in Figure 7 combines several loops over twice the amount of nodes n, which can be estimated by $\mathcal{O}(n^2)$ and for parallel for loops by $\mathcal{O}(\frac{n^2}{p})$. The computational time shows the same behavior as the theoretical complexity.

Figure 18b shows the idle rate. The idle rate is independent of the amount of CPUs and work is well distributed with the default execution policy.

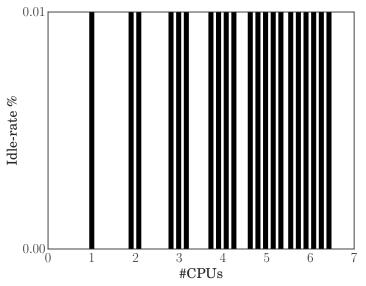
7 Conclusion

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

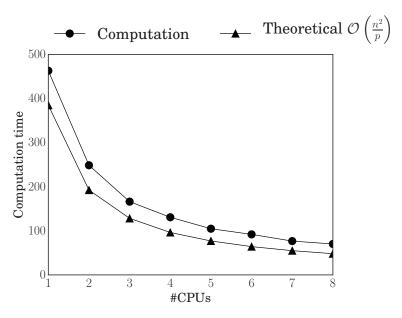


(a) Computational time with increasing amount of CPUs for the test case in Section 5.1.2.

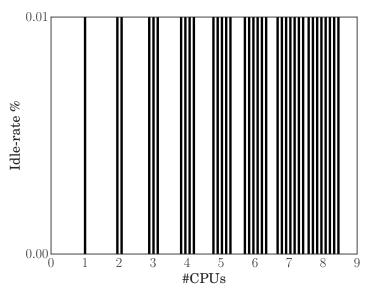


(b) Idle rate with increasing CPUs for the test case in Section 5.1.2.

Figure 17: The computational time with increasing increasing amount of CPUs (a) and the idlerate of each CPU which means the percentage of overall computational time a CPU did not do any work (b).



(a) Computational time with increasing CPUs for the test case in Section 5.2.



(b) Idle rate with increasing amount of CPUs for the test case in Section 5.2.

Figure 18: The computational time with increasing increasing amount of CPUs (a) and the idlerate of each CPU which means the percentage of overall computational time a CPU did not do any work (b).

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. 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.

Acknowledgment

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".

References

- [1] H. Sutter, The free lunch is over: A fundamental turn toward concurrency in software, Dr. Dobbs journal 30 (3) (2005) 202–210.
- [2] P. E. Ross, Why cpu frequency stalled, IEEE Spectrum 45 (4) (2008).
- [3] P. Thoman, K. Dichev, T. Heller, R. Iakymchuk, X. Aguilar, K. Hasanov, P. Gschwandtner, P. Lemarinier, S. Markidis, H. Jordan, et al., A taxonomy of task-based parallel programming technologies for high-performance computing, The Journal of Supercomputing 74 (4) (2018) 1422–1434.
- [4] I. Raicu, I. T. Foster, Y. Zhao, Many-task computing for grids and supercomputers, in: 2008 Workshop on Many-Task Computing on Grids and Supercomputers, 2008, pp. 1–11. doi:10.1109/MTAGS.2008.4777912.
- [5] StarPU A Unified Runtime System for Heterogeneous Multicore Architectures, http://runtime.bordeaux.inria.fr/StarPU/ (2013).
- [6] Intel, Intel Thread Building Blocks 4.4 (2016). URL http://www.threadingbuildingblocks.org
- [7] S. Seo, A. Amer, P. Balaji, C. Bordage, G. Bosilca, A. Brooks, P. H. Carns, A. Castellx00F3, D. Genet, T. Hérault, S. Iwasaki, P. Jindal, L. V. Kalx00E9, S. Krishnamoorthy, J. Lifflander, H. Lu, E. Meneses, M. Snir, Y. Sun, K. Taura, P. H. Beckman, Argobots: A lightweight low-level threading and tasking framework, IEEE Transactions on Parallel and Distributed Systems 29 (2018) 512–526.
- [8] The Qthread Library, http://www.cs.sandia.gov/qthreads/ (2014).
- [9] H. C. Edwards, C. R. Trott, D. Sunderland, Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Journal of Parallel and Distributed Computing 74 (12) (2014) 3202 3216, domain-Specific Languages and High-Level Frameworks for High-Performance Computing. doi:https://doi.org/10.1016/j.jpdc.2014.07.003. URL http://www.sciencedirect.com/science/article/pii/S0743731514001257
- [10] A. Tabbal, M. Anderson, M. Brodowicz, H. Kaiser, T. Sterling, Preliminary design examination of the parallex system from a software and hardware perspective, ACM SIGMETRICS Performance Evaluation Review 38 (4) (2011) 81–87.

- [11] H. Kaiser, T. Heller, B. Adelstein-Lelbach, A. Serio, D. Fey, 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, 2014, p. 6.
- [12] T. Heller, P. Diehl, Z. Byerly, J. Biddiscombe, H. Kaiser, HPX An open source C++ Standard Library for Parallelism and Concurrency, in: Proceedings of OpenSuCo 2017, Denver, Colorado USA, November 2017 (OpenSuCo17), 2017, p. 5.
- [13] Intel(R) Cilk(tm) Plus, http://software.intel.com/en-us/intel-cilk-plus (2014).
- [14] OpenMP V4.0 Specification, http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf (2013).
- [15] B. L. Chamberlain, D. Callahan, H. P. Zima, Parallel programmability and the Chapel language, International Journal of High Performance Computing Applications 21 (2007) 291–312.
- [16] Intel SPMD Program Compiler, http://ispc.github.io/(2011-2012).
- [17] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, V. Sarkar, 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, NY, USA, 2005, pp. 519–538. doi:10.1145/1094811.1094852.
 URL http://doi.acm.org/10.1145/1094811.1094852
- [18] The C++ Standards Committee, ISO International Standard ISO/IEC 14882:2011, Programming Language C++, Tech. rep., Geneva, Switzerland: International Organization for Standardization (ISO)., http://www.open-std.org/jtc1/sc22/wg21 (2011).
- [19] The C++ Standards Committee, ISO International Standard ISO/IEC 14882:2017, Programming Language C++, Tech. rep., Geneva, Switzerland: International Organization for Standardization (ISO)., http://www.open-std.org/jtc1/sc22/wg21 (2017).
- [20] Z. Khatami, H. Kaiser, P. Grubel, A. Serio, J. Ramanujam, A massively parallel distributed n-body application implemented with hpx, in: 2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), IEEE, 2016, pp. 57–64.
- [21] T. Heller, H. Kaiser, A. Schäfer, D. Fey, 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, 2013, p. 1.
- [22] D. Pfander, G. Daiß, D. Marcello, H. Kaiser, D. Pflüger, Accelerating octo-tiger: Stellar mergers on intel knights landing with hpx, in: Proceedings of the International Workshop on OpenCL, IWOCL '18, ACM, New York, NY, USA, 2018, pp. 19:1–19:8. doi:10.1145/3204919.3204938.
 URL http://doi.acm.org/10.1145/3204919.3204938
- [23] T. Heller, B. A. Lelbach, K. A. Huck, J. Biddiscombe, P. Grubel, A. E. Koniges, M. Kretz, D. Marcello, D. Pfander, A. Serio, J. Frank, G. C. Clayton, D. Pflger, D. Eder, H. Kaiser, Harnessing billions of tasks for a scalable portable hydrodynamic simulation of the merger of two stars, The International Journal of High Performance Computing Applications 0 (0) (0) 1094342018819744. arXiv:https://doi.org/10.1177/1094342018819744, doi:10.1177/1094342018819744.
 - URL https://doi.org/10.1177/1094342018819744
- [24] G. Dai, P. Amini, J. Biddiscombe, P. Diehl, J. Frank, K. Huck, H. Kaiser, D. Marcello, D. Pfander, D. Pflger, From piz daint to the stars: Simulation of stellar mergers using highlevel abstractions (2019). arXiv:1908.03121.

- [25] P. Grun, S. Hefty, S. Sur, D. Goodell, R. D. Russell, H. Pritchard, J. M. Squyres, A brief introduction to the openfabrics interfaces-a new network api for maximizing high performance application efficiency, in: 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, IEEE, 2015, pp. 34–39.
- [26] W. Gerstle, N. Sau, S. Silling, Peridynamic modeling of concrete structures, Nuclear engineering and design 237 (12) (2007) 1250–1258.
- [27] Y. D. Ha, F. Bobaru, Studies of dynamic crack propagation and crack branching with peridynamics, International Journal of Fracture 162 (1-2) (2010) 229–244.
- [28] D. J. Littlewood, Simulation of dynamic fracture using peridynamics, finite element modeling, and contact, in: Proceedings of the ASME 2010 International Mechanical Engineering Congress and Exposition (IMECE), 2010.
- [29] M. Ghajari, L. Iannucci, P. Curtis, A peridynamic material model for the analysis of dynamic crack propagation in orthotropic media, Computer Methods in Applied Mechanics and Engineering 276 (2014) 431–452.
- [30] G. Zhang, Q. Le, A. Loghin, A. Subramaniyan, F. Bobaru, Validation of a peridynamic model for fatigue cracking, Engineering Fracture Mechanics 162 (2016) 76–94.
- [31] P. Diehl, S. Prudhomme, M. Lévesque, A review of benchmark experiments for the validation of peridynamics models, Journal of Peridynamics and Nonlocal Modeling (Feb 2019). doi: 10.1007/s42102-018-0004-x. URL https://doi.org/10.1007/s42102-018-0004-x
- [32] S. A. Silling, Peridynamic modeling of the kalthoff-winkler experiment, Tech. rep. (2002).
- [33] M. Parks, D. Littlewood, J. Mitchell, S. Silling, Peridigm Users Guide, Tech. Rep. SAND2012-7800, Sandia National Laboratories (2012).
- [34] M. L. Parks, R. B. Lehoucq, S. J. Plimpton, S. A. Silling, Implementing peridynamics within a molecular dynamics code, Computer Physics Communications 179 (11) (2008) 777–783.
- [35] F. Mossaiby, A. Shojaei, M. Zaccariotto, U. Galvanetto, Opencl implementation of a high performance 3d peridynamic model on graphics accelerators, Computers & Mathematics with Applications 74 (8) (2017) 1856 1870. doi:https://doi.org/10.1016/j.camwa.2017.06.045.
- [36] P. Diehl, Implementierung eines Peridynamik-Verfahrens auf GPU, Diplomarbeit, Institute of Parallel and Distributed Systems, University of Stuttgart (2012).
- [37] M. Zaccariotto, T. Mudric, D. Tomasi, A. Shojaei, U. Galvanetto, Coupling of fem meshes with peridynamic grids, Computer Methods in Applied Mechanics and Engineering 330 (2018) 471 497. doi:https://doi.org/10.1016/j.cma.2017.11.011.
- [38] B. Kilic, E. Madenci, Coupling of peridynamic theory and finite element method, in: 50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference 17th AIAA/ASME/AHS Adaptive Structures Conference 11th AIAA No, 2009, p. 2395.
- [39] U. Galvanetto, T. Mudric, A. Shojaei, M. Zaccariotto, An effective way to couple fem meshes and peridynamics grids for the solution of static equilibrium problems, Mechanics Research Communications 76 (2016) 41–47.
- [40] P. Seleson, D. J. Littlewood, Convergence studies in meshfree peridynamic simulations, Computers & Mathematics with Applications 71 (11) (2016) 2432–2448.
- [41] R. Lipton, Dynamic brittle fracture as a small horizon limit of peridynamics, Journal of Elasticity 117 (1) (2014) 21–50.

- [42] R. Lipton, Cohesive dynamics and brittle fracture, Journal of Elasticity 124 (2) (2016) 143–191.
- [43] E. Emmrich, R. B. Lehoucq, D. Puhst, Peridynamics: a nonlocal continuum theory, in: Meshfree Methods for Partial Differential Equations VI, Springer, 2013, pp. 45–65.
- [44] S. A. Silling, R. B. Lehoucq, Convergence of peridynamics to classical elasticity theory, Journal of Elasticity 93 (1) (2008) 13–37.
- [45] B. Aksoylu, Z. Unlu, Conditioning analysis of nonlocal integral operators in fractional sobolev spaces, SIAM Journal on Numerical Analysis 52 (2014) 653–677.
- [46] J. Biddiscombe, T. Heller, A. Bikineev, H. Kaiser, Zero Copy Serialization using RMA in the Distributed Task-Based HPX runtime, in: 14th International Conference on Applied Computing, IADIS, International Association for Development of the Information Society, 2017.
- [47] H. Kaiser, M. Brodowicz, T. Sterling, Parallex an advanced parallel execution model for scaling-impaired applications, in: Parallel Processing Workshops, 2009. ICPPW'09. International Conference on, IEEE, 2009, pp. 394–401.
- [48] H. Kaiser, T. Heller, D. Bourgeois, D. Fey, 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, NY, USA, 2015, pp. 29–37.
- [49] F. Bobaru, J. T. Foster, P. H. Geubelle, S. A. Silling, Handbook of peridynamic modeling, CRC Press, 2016.
- [50] E. Madenci, E. Oterkus, Peridynamic theory and its applications, Vol. 17, Springer, 2014.
- [51] A. Javili, R. Morasata, E. Oterkus, S. Oterkus, Peridynamics review, Mathematics and Mechanics of Solids 24 (11) (2019) 3714–3739.
- [52] S. A. Silling, Reformulation of elasticity theory for discontinuities and long-range forces, Journal of the Mechanics and Physics of Solids 48 (1) (2000) 175–209.
- [53] S. A. Silling, M. Epton, O. Weckner, J. Xu, E. Askari, Peridynamic states and constitutive modeling, Journal of Elasticity 88 (2) (2007) 151–184.
- [54] S. A. Silling, Reformulation of elasticity theory for discontinuities and long-range forces, Journal of the Mechanics and Physics of Solids 48 (1) (2000) 175–209.
- [55] I. A. Kunin, Elastic Media with Microstructure I: One-Dimensional Models (Springer Series in Solid-State Sciences), softcover reprint of the original 1st ed. 1982 Edition, Springer, 2011.
- [56] I. A. Kunin, Elastic Media with Microstructure II: Three-Dimensional Models (Springer Series in Solid-State Sciences), softcover reprint of the original 1st ed. 1983 Edition, Springer, 2012.
- [57] X. Chen, M. Gunzburger, Continuous and discontinuous finite element methods for a peridynamics model of mechanics, Computer Methods in Applied Mechanics and Engineering 200 (9) (2011) 1237–1250.
- [58] O. Weckner, E. Emmrich, Numerical simulation of the dynamics of a nonlocal, inhomogeneous, infinite bar, Journal of Computational and Applied Mechanics 6 (2) (2005) 311–319.
- [59] E. Emmrich, O. Weckner, The peridynamic equation and its spatial discretisation, Mathematical Modelling and Analysis 12 (1) (2007) 17–27.

- [60] S. A. Silling, E. Askari, A meshfree method based on the peridynamic model of solid mechanics, Computers & structures 83 (17) (2005) 1526–1535.
- [61] P. K. Jha, R. Lipton, Numerical convergence of nonlinear nonlocal continuum models to local elastodynamics, International Journal for Numerical Methods in Engineering 114 (13) (2018) 1389-1410. doi:10.1002/nme.5791. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.5791
- [62] D. J. Littlewood, Roadmap for Peridynamic Software Implementation, Tech. Rep. 2015-9013, Sandia National Laboratories (2015).
- [63] B. Kilic, Peridynamic Theory for Progressive Failure Prediction in Homogeneous and Heterogeneous Materials, The University of Arizona., 2008.
- [64] H. Wang, H. Tian, A fast galerkin method with efficient matrix assembly and storage for a peridynamic model, Journal of Computational Physics 231 (23) (2012) 7730 7738. doi: https://doi.org/10.1016/j.jcp.2012.06.009.
 URL http://www.sciencedirect.com/science/article/pii/S0021999112003166
- [65] K. Iglberger, G. Hager, J. Treibig, U. Rüde, Expression templates revisited: a performance analysis of current methodologies, SIAM Journal on Scientific Computing 34 (2) (2012) C42–C69.
- [66] K. Iglberger, G. Hager, J. Treibig, U. Rüde, High performance smart expression template math libraries, in: 2012 International Conference on High Performance Computing Simulation (HPCS), 2012, pp. 367–373. doi:10.1109/HPCSim.2012.6266939.
- [67] M. H. Sadd, Elasticity: theory, applications, and numerics, Academic Press, 2009.
- [68] P. K. Jha, R. Lipton, Numerical convergence of finite difference approximations for state based peridynamic fracture models, Computer Methods in Applied Mechanics and Engineering (2019) (March 2019). doi:10.1016/j.cma.2019.03.024. URL https://doi.org/10.1016/j.cma.2019.03.024
- [69] R. P. Lipton, R. B. Lehoucq, P. K. Jha, Complex fracture nucleation and evolution with nonlocal elastodynamics, Journal of Peridynamics and Nonlocal Modeling 1 (2) (2019) 122– 130. doi:10.1007/s42102-019-00010-0. URL https://doi.org/10.1007/s42102-019-00010-0
- [70] P. K. Jha, R. Lipton, Numerical analysis of nonlocal fracture models in hölder space, SIAM Journal on Numerical Analysis 56 (2) (2018) 906-941. doi:10.1137/17M1112236. URL https://doi.org/10.1137/17M1112236
- [71] J. R. Shewchuk, et al., An introduction to the conjugate gradient method without the agonizing pain (1994).
- [72] G. Strang, Linear Algebra and Its Applications), Academic Press, Inc., 1980.