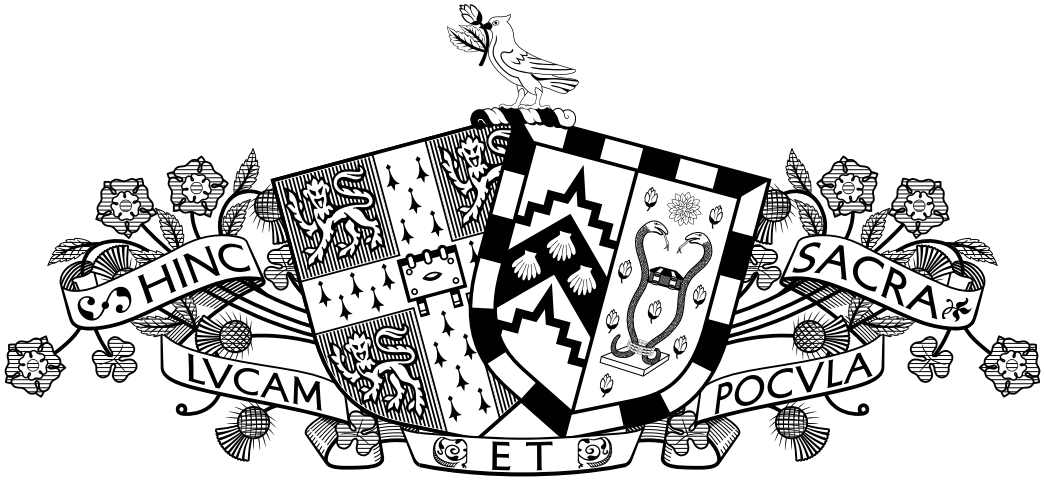
The Implementation of a High Performance Peridynamics solver for Probabilistic Peridynamics and the Modelling of Concrete Behaviour



Ben Boys

Supervisor: Dr. J.Orr

Prof. M.Girolami

Advisor: Prof. T.Dodwell

Department of Engineering
University of Cambridge

Gonville & Caius College May 2020

Acknowledgements

First and foremost, I would like to thank and acknowledge Professor Tim Dodwell from the Data-Centric Research Group at Exeter for his excellent weekly supervision and guidance during this project. Special thanks goes to the Data-Centric Engineering Group and HYDRA at the University of Exeter for allowing me unrestricted compute-time on their GPU cluster. Without the GPU it would not have been possible to produce the results in this report.

I would also like to thank Mark Hobbs for answering technical questions about developing a peridynamics solver, and for sharing his MATLAB code. I also thank my supervisor Dr John Orr for his help and advice.

I would like to thank Dr Greg Mingas and Dr Jim Madge for answering my technical questions and Jim especially for providing the refactoring and further optimisations of the sequential code to make the code ready for release to the research community. I have learned a great deal from talking to these scientists and working with them throughout the year.

Finally, I would like to thank Professor Mark Girolami for his continued tutoring and academic nurturing, and for putting me in to contact with these scientists.

Abstract

Peridynamics is a non-local continuum theory for describing the behaviour of material used especially when damage and crack nucleation or propagation is of interest; it circumvents spatial derivatives, which are undefined at the crack tip, by replacing the partial differential equations of continuum mechanics by integral partial differential equations. New parameters over classical continuum mechanics are needed to define the simplest peridynamic model. One of the challenges of peridynamics is selecting these parameters, which motivates a Bayesian parameter estimation approach. Current work is being done by the Data-Centric Engineering groups at the University of Exeter and the University of Cambridge to define a stochastic formulation of peridynamics (denoted Probabilistic Peridynamics) so as to obtain a statistical means of studying crack propagation in materials.

One of the major challenges associated with peridynamics is the large computational cost of simulations to achieve realistic results. This cost depends heavily on the coarseness of the spatial discretisation employed in the model. The only way around this challenge is by parallel processing the time integration in peridynamics. Parallel processing is one of the major trends in the computational mechanics community. Due to inherent limitations in processor design, manufacturers have shifted towards multi- and many- core architectures. Graphics Processing Units (GPUs) are gaining more and more popularity due to high availability and processing power as well as maturity of development tools and community experience.

In this work, the author demonstrates the implementation of a lightweight, open source OpenCL implementation of the 3D bond-based peridynamics model on a GPU platform. Step improvements in execution speed, ease of use and functionality over existing literature are presented in this report. The resulting solver is 2 - 5x faster than similar existing GPU implementations, and at least 6.0x faster than the OpenMP implementation being used by the Peridynamics Group at the University of Cambridge for the grid spacing used to simulate concrete failure (10mm). The GPU implementation is most efficient at the small to medium problem size range (10^2 to 10^4 particles). This has made numerical simulations fast enough to do Bayesian parameter estimation for medium sized peridynamics problems using MCMC (Markov Chain Monte Carlo). This has never before been attempted for peridynamics as given that a single peridynamics simulation was computationally costly, taking hundreds of thousands of samples was out of the question. Secondly, a

new stochastic formulation of peridynamics is presented by representing the dynamics through a single energy potential function and a stochastic differential equation to model the evolution of the peridynamic body to equilibrium. The output is a distribution of crack paths. The new parameters used to define the stochastic model were inferred using maximum likelihood estimation. This work only scrapes the surface of what is possible with parameter estimation techniques, and it is hoped that this solver will be used by the Peridynamics Group in Cambridge University Engineering Department and by researchers in the wider peridynamics community to achieve a data-driven approach to studying fracture.

Table of contents

Chapter 1

Introduction

Most reinforced concrete (RC) beams are prismatic, and for good reason: they are simple and economical to construct, and the design of prismatic beams is well covered by comprehensive design codes backed by decades of experimental research, so they are simple to verify as safe. However, structural material wastage in the order of 50% is common [1]. One reason for this is, in a prismatic beam, much of the material's strength is not fully utilised, even at failure. Using non-prismatic beams constructed with flexible formwork, as displayed in Figure 1.1, it is possible to build structures that use up to 40% less concrete than a prismatic beam of equivalent strength, resulting in significant embodied energy savings [2]. However, the non-linear behaviour of RC, as it softens and cracks during deflection and in failure, is hard to predict. If more efficient beam shapes, such as the non-prismatic beams produced by fabric-formwork, are to be considered for use in structures, then a better model of concrete behaviour, including fracture, is needed to predict the working deflections and failure loads of such shapes.

1.1 Modelling the fracture of materials

Fracture mechanics of materials and structures has been a long standing area of research in engineering, and people have made substantial efforts in order to understand material failures and alleviate potential dangers. Inglis (1913) derived stress concentration factors around an elliptical hole, Griffith (1920) recognised that extension of the crack was accompanied by a change in the total potential energy of the system. Williams (1957) gave a clear treatment of the singular stress field around a crack tip using Airy stress functions. Irwin (1956) extended the Griffith approach by developing the energy release rate.

In computational mechanics, accurate modelling of damage and fracture phenomena, including static and dynamic crack propagation, is still an active and open challenge among researchers. The main difficulty in such problems arises from the fact that the classical theory employs partial differential equations assuming that a body remains continuous as it deforms. The spatial derivatives in linear elastic fracture mechanics (LEFM) are

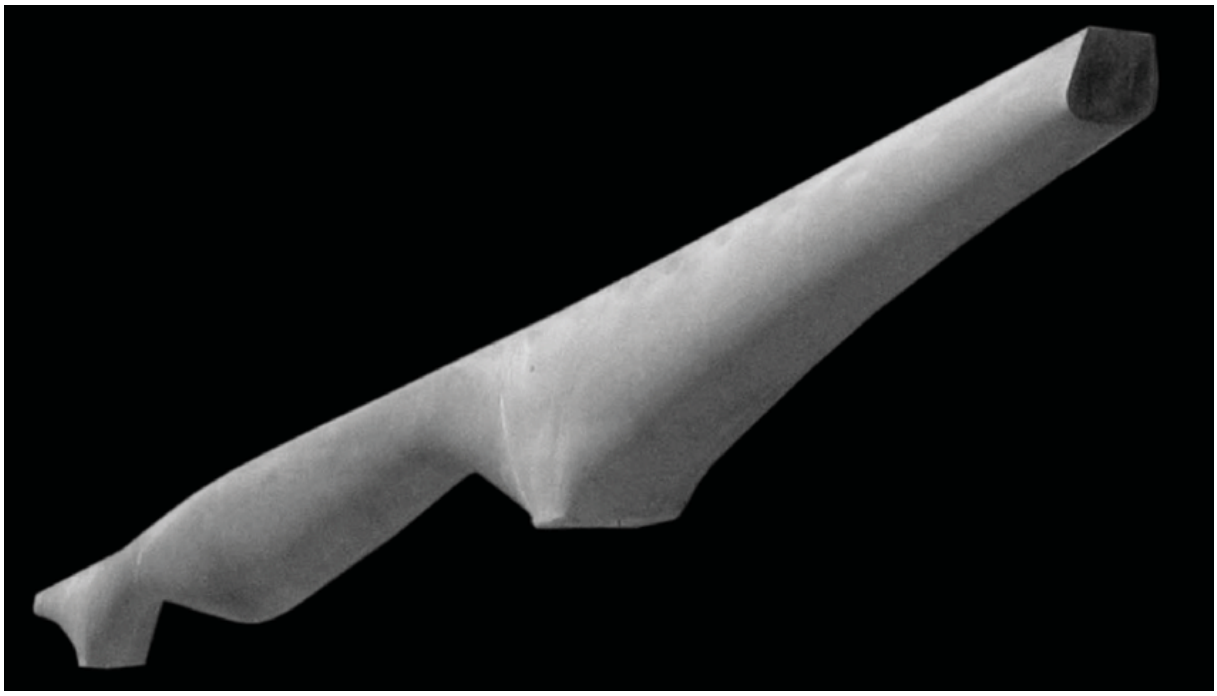


Fig. 1.1 An optimised design of RC beam courtesy of Mark West and The Centre for Architectural Structures and Technology (C.A.S.T.). The aesthetic curve of this beam follows the bending moment diagram of the structure, with additional depth at the supports needed to prevent shear failure.

undefined on the crack tip, where the displacement field is discontinuous, and stress fields are singular (while LEFM predicts infinite stresses at the crack tip, Irwin noted that in real materials stresses at the crack tip are finite because plasticity leads to relaxation of crack tip stresses).

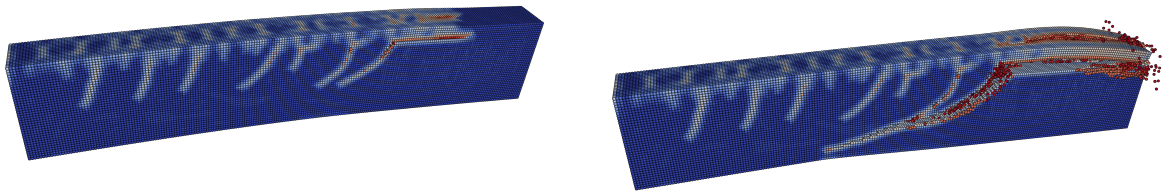
To correctly model the singular behaviour of stress fields at the crack tip, LEFM concepts have been integrated into Finite Element tools, but Finite element analysis (FEA) with traditional elements suffers from the following shortcomings [3]: (1) the interface between dissimilar materials is assumed to have zero thickness without any specific material properties; however, it presents a weak link and it is usually the location of failure. Therefore, it fails to appropriately model the interface between dissimilar materials. (2) Failure is a dynamic process, and it requires remeshing. Meshing is computationally costly and breaks down when multiple complex crack growth patterns develop. (3) Stress and strain fields are discontinuous, and mesh refinement does not necessarily ensure accurate stress fields near geometric and material discontinuities. (4) Finally, crack nucleation is not resolved. The analysis always requires a pre-existing crack.

In order to remedy or remove these shortcomings, Cohesive Zone Elements (CZE) and eXtended Finite Elements (XFEM) [4] [5] were developed; however, CZE requires a priori knowledge of the crack path. Although XFEM doesn't constrain the discontinuity to element boundaries, and can correctly model discontinuity without remeshing [6], it

still requires an external criteria for crack propagation. Thus, the results depend on the criteria employed in the analysis.

1.2 Peridynamics

The peridynamic theory of solid mechanics, introduced by Silling [7], overcomes some of the weaknesses of the existing methods, and is capable of identifying all failure modes without simplifying assumptions. Peridynamics predicts material failure in composite structures under general loading conditions and geometry. Damage is inherently calculated in a peridynamics analysis without special procedures, making progressive failure analysis more practical. It does this by reformulating classical mechanics into a non-local theory that is suitable for discontinuous problems such as crack propagation. The key feature of all peridynamic models is that the material is discretised into a set of points of finite mass, so the technique is meshless. In classical mechanics, Newton's second law is written as of a partial differential equation. By contrast, in peridynamics, Newton's second law is written as an integral-partial differential equation, eliminating spatial derivatives, and so is valid regardless of discontinuities. Therefore, peridynamics is useful to solve problems involving spontaneously nucleated discontinuities, without the need to pre-define crack tips. A comparison study between peridynamics, CZE, and XFEM techniques is given by Agwai et al. (2011) [8]. They showed that the crack speeds obtained from all three approaches are on the same order; however, the fracture paths obtained by using peridynamics are visibly closer to experimental results with respect to other two techniques. Sudden shear failure of a reinforced concrete beam is shown in Figure 1.2.



(a) Flexural cracks will form initially before a diagonal crack starts from the last flexural crack and turn gradually towards the applied load.

(b) During shear failure, sudden crack propagation of the diagonal crack is exhibited.

Fig. 1.2 Shear failure in a prismatic concrete beam. This figure was produced with the author's implementation of a peridynamics solver.

In spite of the appealing features of peridynamic models, they are computationally much more expensive than the other two methods. Moreover, capturing realistic numerical solutions with peridynamics which resemble experimental observation entails vast amounts

of computational resources [9]. To capture crack nucleation and growth accurately, a very fine discretization of the solution domain is required [7, 10].

One of the biggest complaints in peridynamics literature is that it is difficult to choose the additional parameters needed to define the model, such as the characteristic material length scale. This motivates a Bayesian parameter estimation approach on the deterministic model parameters. Further to this point, this work presents a stochastic reformulation of peridynamics that could account for uncertainty in the material failure at the macroscale arising from missing information at the microscale that defines the non-linear behaviour between the peridynamic bonds. Denoted Probabilistic Peridynamics, this formulation expresses the dynamics through a single energy potential function and a stochastic differential equation to model the evolution of the peridynamic body to equilibrium. The output is a distribution of crack paths, which could be adjusted by data.

The research focus of this work lies in solving a the following problem: to decrease the compute time of peridynamics simulations so that parameter estimation, such as maximum likelihood, is possible. To do maximum likelihood parameter estimation, the sample time of a single simulation needs to be on the order of seconds. To estimate integrals with Markov Chain Monte Carlo (MCMC) methods that define posterior distributions over model parameters requires sample rates that are orders of magnitudes faster still. In comparison, on a consumer grade computer, current PD software often takes hours to run a single simulation[11].

1.3 Computational demand

Due to the computational demand of peridynamics, and considering limitations in design of CPUs [12], even the fastest serial CPU codes are restricted to low node counts and therefore unrealistic results [13]. There are some ways of gaining marginal performance in serial codes, such as vectorisation, which is explored in Appendix A.1, or by avoiding the need to use square roots when calculating Euclidean distances between the nodes by using a Taylor expansion approximation [14], however, parallelisation is the only way to obtain realistic simulations in reasonable compute time.

Several parallel peridynamics implementations are already available, such as Peridigm[15] and PDLAMMPS [16]. Peridigm was developed at Sandia National Laboratories for parallel multi-physics simulations. Peridigm is built in C++, and as a result, has many dependencies and an 87 page installation manual [17]. Peridigm and PDLammps rely on the widely used Message Passing Interface (MPI) for the inter-node parallelisation on CPUs. It has been claimed that single device GPU approaches cannot deal with large particle clouds due to current hardware memory limitations on GPUs [18]. This is no longer true, and depends on the particular device in use; GPU technology is developing at

a fast rate, and high end market consumer cards, such as the NVIDIA GEFORCE RTX 2080 Ti, contain as much as 11Gb of onboard memory and can handle millions of degrees of freedom, as shown in Section 6.2 of this report.

When it comes to using GPUs, the works to note include using GPU acceleration for building the horizons data structure [19, 20], GPU-assisted MATLAB code [21] and using OpenACC directives for parallelizing the inner loops in existing CPU codes [22, 23]. However OpenACC is not an attractive option, as it is currently only supported by the PGL compiler and it must be used along side, and not instead of the major players in the field of general-purpose GPU computing, OpenCL or CUDA.

CUDA, developed by NVIDIA, was the first set of tools that were adopted by the engineering and scientific community for harnessing the power of GPUs. An open standard in OpenCL was established in 2009 by a series of companies, led by Apple. The advantage of OpenCL over CUDA is that the code written based on this standard is guaranteed to run on every OpenCL certified platform, ranging from cell phones to GPUs and multi-core CPUs.

There is only one work of note that uses OpenCL for inter-node parallelisation during time-integration: in their paper, Mossaiby et al. present a native OpenCL implementation of bond-based peridynamics, capable of handling 3D problems with crack nucleation and branching as well as existing pre-cracks[24]. They compare the performance of the code with a highly-tuned, parallelised OpenMP code on some benchmark problems with several millions of degrees of freedom. The result is that the OpenCL code outperforms the OpenMP parallel code by a factor of roughly 3 to 6, and outperforms an optimised serial code by 40 to 90 times depending on the size of the problem. The author's code employs local GPU memory optimisations to further outperform this OpenCL code by another 2 to 5 times, depending on the size of the problem, making Bayesian parameter estimation possible for the first time.

The functionality of the code is demonstrated in this report by attempting to model concrete behaviour with a simplistic constitutive model, and comparing failure loads from different meshing methods, mesh spacing and load control to results from lab tests.

1.4 Outline of this work

The main research goal of this project was to develop a fast, easy to install and research friendly peridynamics solver in python, and using OpenCL for GPU acceleration. The author's code can be accessed at the GitHub repository github.com/alan-turing-institute/Probabilistic-Peridynamics/tree/feature/pymc3. The solver has only 4 dependencies: meshio, numpy, scipy and pyopencl. The advantage of OpenCL is that it does not

require a dedicated graphics card to run the code, so the code can be run on any device with a CPU (with or without onboard graphics).

The chapters of this report cover the following topics:

Chapter 2 - Bond-based Peridynamic theory will recap the underlying theory of the simplest peridynamics model, the *bond-based* peridynamic model, which is the peridynamics model that was used in this work. This section describes the constitutive models used to model concrete behaviour.

Chapter 3 - Probabilistic Peridynamics discretises the peridynamics equation of motion that is defined in Chapter 2 for implementation in a numerical solver and presents a reformulation of peridynamics as a stochastic differential equation, so that random crack growth be modelled. Two stochastic parameters are introduced, which describe the standard deviation and length scale of Brownian motion which is applied to the particles.

Chapter 4 - Implementation contains details of the implementation including how the speedup over existing literature was achieved, and validates the code by comparing the peridynamics solution to a classical continuum mechanics solution for a 2D tensile test.

Chapter 5 - Experimental setup details the computational experiments used to test the code, and demonstrates a method of parameter estimation from damage data for both deterministic and probabilistic peridynamics parameters, using MCMC and maximum likelihood, respectively. This section presents samples from a forward distribution of crack paths for a 3D tensile problem.

Chapter 6 - Deterministic results and discussion contains the results of the deterministic code when applied to two distinct concrete canteliver beam problems. This section will benchmark the solver against a similar existing GPU solver. This report shows that a 2 to 5 times performance gain over an existing OpenCL peridynamics code has been achieved over all problem sizes.

Chapter 6.3 - Probabilistic results and discussion contains the results of basic parameter estimation using damage data from a crack sample obtained in Chapter 5. Maximum likelihood parameter estimation was used to recover the newly introduced stochastic parameters, and MCMC was used to find a posterior distribution over the deterministic parameters in the deterministic setting.

Chapter 7 - Conclusions and future work writes down the conclusions that can be made from this report, with suggestions of future work that should be done to further develop Probabilistic Peridynamics.

Chapter 2

Bond-Based Peridynamic Theory

This section briefly presents the bond-based peridynamic theory and the constitutive models used to model concrete behaviour. More detailed explanations can be found in the literature [7, 25].

2.1 Peridynamic continuum model

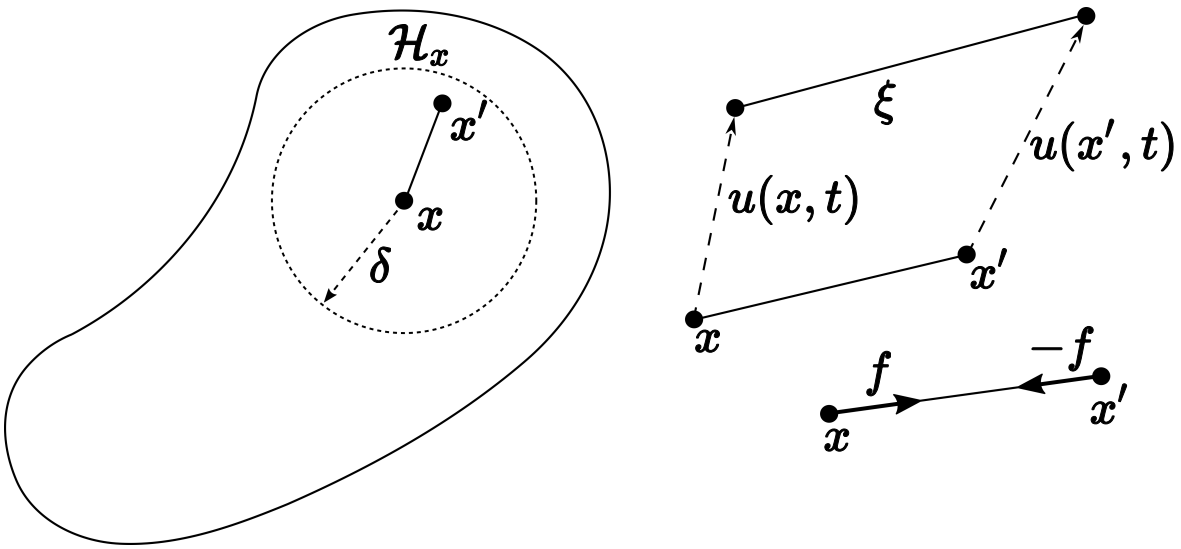


Fig. 2.1 The peridynamic body (left) and the kinematics of a particle pair and the bond-based pairwise force function (right).

In bond-based peridynamics, the acceleration of any particle at $\boldsymbol{x}$ in the body $\mathcal{R}$ at the time t is found from the peridynamics equation of motion

$$\rho\ddot{\boldsymbol{u}}(\boldsymbol{x},t) = \int_{\mathcal{H}_x} \boldsymbol{f}(\boldsymbol{u}(\boldsymbol{x}',t) - \boldsymbol{u}(\boldsymbol{x},t), \boldsymbol{x}' - \boldsymbol{x}) dV_{\boldsymbol{x}'} + \boldsymbol{b}(\boldsymbol{x},t), \tag{2.1}$$

where the particle interacts directly with the points in its family $\mathcal{H}_x$, centred on the particle, and defined by the radius, namely the *horizon* $\delta \geq 0$, which describes a characteristic length scale specific to the mechanics of the material. This makes peridynamics a *non-local*

model which assumes that pair particles in a continuum interact with each other across a finite distance through a non-linear *bond*. The concept of a bond that extends over a finite distance is a fundamental difference between the peridynamic theory and the classical theory, which is based on the idea of contact forces between particles that are in direct contact, in a continuum. $\boldsymbol{u}$ is the displacement vector field, $\boldsymbol{b}$ is a prescribed body force density field, ρ is mass density in the body, and $\boldsymbol{f}$ is a *pairwise force function*, whose value is the force vector (per unit volume squared) that the particle $\boldsymbol{x}'$ exerts on the particle $\boldsymbol{x}$,

$$\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{u}(\boldsymbol{x}', t), t), \;\; \boldsymbol{x}' \in \mathcal{R} : ||\boldsymbol{x}' - \boldsymbol{x}|| \leq \delta, \tag{2.2}$$

$\boldsymbol{f}$ contains the constitutive model of the material. Anisotropy may be included through a dependence of $\boldsymbol{f}$ on the direction of the bond $\boldsymbol{x}' - \boldsymbol{x}$ in the initial configuration, but won't be explored in this report.

Silling [25] showed that for an isotropic material, $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{\eta}, \ell_0)$, where ℓ_0 is the original bond length,

$$\ell_0 = \|\boldsymbol{x}' - \boldsymbol{x}\|_2, \tag{2.3}$$

and $\boldsymbol{\eta}$ is the relative displacement at time t,

$$\boldsymbol{\eta} = \boldsymbol{u}(\boldsymbol{x}', t) - \boldsymbol{u}(\boldsymbol{x}, t). \tag{2.4}$$

Let $\boldsymbol{\xi}$ be the bond vector at time t,

$$\boldsymbol{\xi} = \boldsymbol{y}(\boldsymbol{x}', t) - \boldsymbol{y}(\boldsymbol{x}, t), \tag{2.5}$$

,

where $\boldsymbol{y}$ is the position of the particle at time t,

$$\boldsymbol{y}(\boldsymbol{x}, t) = \boldsymbol{x} + \boldsymbol{u}(\boldsymbol{x}, t). \tag{2.6}$$

For brevity the length of the bond at time t is defined as,

$$\ell = \|\boldsymbol{\xi}\|_2. \tag{2.7}$$

Silling defined a *microelastic* material as one which the pairwise force function is derivable from a scalar-valued *micropotential*, $\hat{w}$,

$$\boldsymbol{f} = \nabla_{\boldsymbol{\xi}} \hat{w}(\ell, \ell_0) = \hat{\boldsymbol{\xi}} f(\ell, \ell_0), \tag{2.8}$$

where $\hat{\boldsymbol{\xi}} = \boldsymbol{\xi}/\ell$ is the unit direction vector of the bond at time t, so that the force density is a scalar value projected in the direction of the bond.

The micropotential is the energy in a single bond and has dimensions of energy per unit volume squared, and only depends on $\boldsymbol{\xi}$ through the *scalar* valued distance between the deformed points, ℓ. Therefore, the interaction between any two points in a microelastic material may be thought of as an elastic (and possibly nonlinear) spring. The energy per unit volume in the body at a given point (i.e., the local strain energy density) is therefore found from

$$W = \frac{1}{2}\int_{\mathcal{H}_{\boldsymbol{x}}} \hat{w}(\ell, \ell_0) dV. \tag{2.9}$$

The factor of 1/2 appears because each endpoint of a bond "owns" only half the energy in the bond. If a body is composed of a microelastic material, work done on it by external forces is stored in recoverable form in much the same way as in the classical theory of elasticity.

Modelling of composite materials, such as reinforced concrete, may be included through dependence of $\boldsymbol{f}$ on the initial position of the particles defining the bond, $\boldsymbol{x}$ and $\boldsymbol{x'}$, and through a constitutive models that define a scalar valued micropotential function for the different materials in the composite, as well as defining a constitutive model for the interface between those materials.

2.2 Constitutive model

The constitutive behaviour of a material in bond-based peridynamics is expressed through the definition of the scalar values micro-potential $\hat{w}(\ell, \ell_0)$, which in turn defines the scalar valued force density $f(\ell, \ell_0)$.

Under a general deformation of each particle the strain of the connection (stretch over original length) is given by

$$s = \frac{\ell - \ell_0}{\ell_0}$$

Silling and Askari [25] introduced the first peridynamic constitutive law, known as the *prototype micro-elastic brittle* (PMB) model, defining the bond strain / force density curve projected in the axial direction of the bond $\hat{\boldsymbol{\xi}}$, by

$$f = \hat{\boldsymbol{\xi}} \cdot \nabla_{\boldsymbol{\xi}} \hat{w}(\ell, \ell_0) := cs\mu. \tag{2.10}$$

Here the constant c defines the micromodulus, also known as the peridynamic bond stiffness, which is defined as

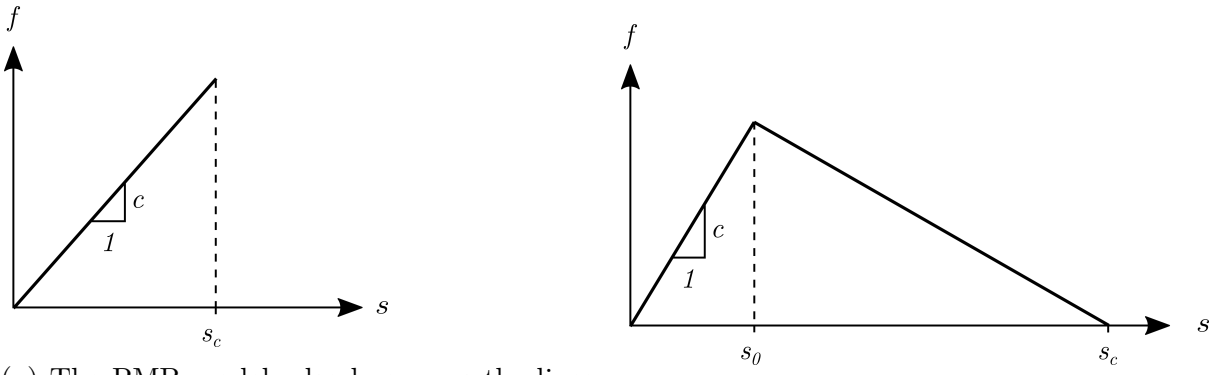
$$c = \frac{18K}{\pi\delta^4}, \tag{2.11}$$

for some material bulk modulus K and horizon δ. The micromodulus c is determined by equating the strain energy density of a body under isotropic extension in the classical theory [25] or by numerical calculations [26]. For a pure shear state there is also an

equivalence of strain energy, and a second expression for c can be determined. By equating the two expressions for c it is found that the Poisson's ratio ν is limited to 1/4 for the 3D case. This restriction on the Poisson's ratio has been addressed by the state-based peridynamic theory Silling et al. [27]. The bond-based model has also been revised to overcome the Poisson's ratio restriction[28]. Despite this limitation, the PMB model, which was proposed for capturing the behaviour of ideal-brittle materials, such as glass and ceramics, has been used to accurately capture complex fracture patterns [29] [30].

The other new material parameter introduced in (2.10) is s_c. This defines a *critical strain* of the bond, which, once exceeded, the bond irreversibly breaks. The PMB model is shown in Figure 2.2a. μ is a history-dependent scalar valued function that tracks if the bond has broken, and takes on values[25] of either 1 or 0,

$$\mu = \begin{cases} 1 & \text{for} \quad s < s_c \text{ and } \max_{t' \in [0,t]} s(t') < s_c \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$



(a) The PMB model, also known as the linear damage model.

(b) The bi-linear damage model.

Fig. 2.2 Two constitutive models which define the bond force density function.

Damage is introduced into the model to capture failure behaviour. Local damage $\phi(\boldsymbol{x}, t)$, for an isotropic material, is defined[25] as

$$\phi(\boldsymbol{x}, t) = 1 - \frac{\int_{\mathcal{H}_{\boldsymbol{x}}} \mu(\boldsymbol{x}, t) dV}{\int_{\mathcal{H}_{\boldsymbol{x}}} dV}. \tag{2.13}$$

in which $\boldsymbol{x}$ is included as an argument of μ as a reminded that it is a function of the position in the body. Note that $0 \leq \phi \leq 1$, with 0 representing virgin material, and 1 representing complete disconnection of a point from all of the points with which it initially interacted. To completely separate the two halves of the body across the fracture surface requires breaking all the bonds that initially connected points in the opposite halves, and will result in a damage of $\phi \approx 0.5$ at those points on the fracture surface.

Silling and Askari [25] derived the critical stretch value in terms of the material fracture energy release rate G_F. A peridynamic body is completely separated into two halves and

all the bonds that initially crossed the fracture surface are broken. By determining the energy required to break a single bond, the energy per unit fracture area for complete separation of a peridynamic body can be determined. The energy release rate G_F is a material parameter and can be determined experimentally. The critical stretch value s_c for a three-dimensional peridynamic material is defined[25] by

$$s_c = \sqrt{\frac{5G_F}{6E\delta}}. \tag{2.14}$$

2.3 Constitutive models for reinforced concrete

2.3.1 Concrete constitutive model

The PMB model was used to produce the results in section 6.2 of this report. As stated in (2.14), the critical stretch of the PMB model is dependent on the material fracture energy G_F. The selection of G_F is a major source of uncertainty in the peridynamic model and the value chosen will have a significant impact on the numerical results. Concrete material properties are typically determined through the testing of concrete compressive strength f_c. Additional material parameters such as elastic modulus E and fracture energy G_F are rarely tested and must be determined from empirical formulas. At present, there is no standard theory relating concrete compressive strength to fracture energy, and Neville [31] states that fracture mechanics has not succeeded in developing parameters that adequately quantify the resistance of concrete to cracking. The value of fracture energy, $G_F = 100\text{N/m}$ has been taken from Yang et al. [32]. The value of Young's modulus, $E = 22.0\text{GPa}$. The value of bulk modulus was taken as $K = E/(3(1-2\nu))$ [33]. The Poisson's ratio for concrete is usually in the range of 0.1 to 0.2, but a value of $\nu = 0.25$ is taken, for the reason discussed in Section 2.2.

Whilst the PMB model has been applied to modelling concrete structures [34, 35], the use of linear damage models for concrete is primarily suited to the analysis of large structures (e.g. large dams). This is because in the fracture of large structures, the size of the fracture process zone (FPZ) is insignificant when compared the size of the structure [36] and the fracture behaviour is in the LEFM regime.

In small and medium sized concrete structures, such as beams, concrete exibits non-linear strain softening behaviour when in tension. The linear PMB damage model does not reproduce this softening behaviour seen in experimental load-displacement curves, since the constitutive law is linear.

Gerstle et al. [34] recognised this limitation as proposed a bilinear damage model, illustrated in Figure 2.2b but no details were provided for determining the value of the bond stretch at the linear elastic limit, s_0, or the value of the bond stretch at failure

s_c. Various and differing methods of determining these parameters has been proposed [32, 37]. The difficulty in choosing these additional parameters further motivates the use of a Bayesian parameter estimation approach, however, for simplicity, the use bilinear damage model was not explored in this report, which limits the predictive capability of the model.

2.3.2 Steel constitutive model

Broadly speaking, reinforced concrete structures have two types of failure: failure in bending, where the steel reinforcement in tension has fully yielded, and failing in shear, where the steel reinforcement has not yielded before the shear failure of the concrete. This report has restricted the analysis to concrete structures that fail in shear, and therefore the steel peridynamic bonds will behave in a linear elastic manner and not exceed the critical stretch, so that a plastic constituative model for the steel reinforcement does not have to be defined. The value of concrete critical stretch has been calculated using a value of $G_F = 13000$ N/m and the Young's modulus has been taken as $E = 210$ GPa which are mid-range values for carbon steel [33]. The value of bulk modulus was taken as $K = E/(3(1-2\nu))$ [33] where $\nu = 0.25$, as discussed in section 2.2.

2.3.3 Interface constitutive model

In composite peridynamic models, each particle, which represents a lumped volume, is assigned a material. When a bond connects a steel bond to a concrete bond, it gives the opportunity to define stiffnesses and critical stretches that are particular to the interface between these materials.

The bond between the steel reinforcing bars and concrete displays complex behaviour such as toughening effect due to the laps of steel bars and bond-slip. In peridynamics, there is limited literature that explores a steel-concrete constitutive model. Gerstle et al. [38] modelled a reinforced concrete lap splice problem and modelled the toughening effect of the ribs on the steel bars, increased the bond stiffness and bond critical stretch to three times that of the concrete bonds. For the purposes of this report, this assumption was also made to produce the results in section 6.2.

Chapter 3

Probabilistic Peridynamics

In this section, the deterministic peridynamic formulation is discretised, and the correction factors needed for implementation into a numerical solver are outlined. A stochastic reformulation of peridynamics with an equation for its stochastic dynamics is presented.

3.1 Discretised peridynamics

As before, let a body in it's initial configuration occupy a domain $\mathcal{R} \subset \mathbb{R}^d$. Computationally, in the mesh-free approach of Silling and Askari[25] the material can be represented by n particles, with the initial coordinates $\mathbf{x}_i \in \mathcal{R}$, displacement vector $\mathbf{u}_i(t) \in [0, T] \times \mathbb{R}^d$ and associated lumped volumes dV_i. Furthermore, a given particle is subject to an externally applied force density $\mathbf{b}_i \in \mathbb{R}^d$. At any point in time t the position of a particle is

$$\mathbf{y}_i(t) = \mathbf{x}_i + \mathbf{u}_i(t) \quad \text{for} \quad t > 0.$$

As aforementioned, the peridynamic formulation is a *non-local* model. The i^{th} particle interacts with the all particles within the index set

$$\mathcal{H}_i := \{j \mid \|\mathbf{x}_i - \mathbf{x}_j\| < \delta\},$$

which is called the *family* of particle $\mathbf{x}_i$. The user defined parameter $\delta \geq 0$ is as before, the *horizon*, which describes a characteristic length scale specific to the mechanics of the material.

As described in section, 2.1, between a pair of particles i and j, which reside in each others horizons, a non-linear 'bond' exists, defined by the scalar-valued micropotential $\hat{w}_{ij}(\ell, \ell_0, \theta) > 0$, where θ are the constitutive model parameters, and the connection between two particles has an original length of

$$\ell_{0,ij} = \ell_{0,ji} = \|\mathbf{x}_i - \mathbf{x}_j\|_2 \tag{3.1}$$

at which the connection stores no strain energy. The force that results from the strain of the connection is in the axial direction of the bond, $\boldsymbol{\xi}_{ij} = \mathbf{y}_j(t) - \mathbf{y}_i(t)$. The connection between two particles at a point in time t is

$$\ell_{ij} = \ell_{ji} = \|\mathbf{y}_i(t) - \mathbf{x}_j(t)\|_2 \tag{3.2}$$

There is a choice at this stage how to progress with the numerical integration. Two approaches laid out below involve implicit schemes that reach equivalent steady state solutions in the deterministic setting.

3.1.1 Deterministic Euler-Cromer implementation

The discretised peridynamics equation of motion approximates the integral in Equation (2.1) by the sum

$$\rho\ddot{\mathbf{u}}(\mathbf{x}, t) = \sum_{j \in \mathcal{H}_i} \mathbf{f}_{ij}(\boldsymbol{\xi}) dV_j + \mathbf{b}(\mathbf{x}, t). \tag{3.3}$$

The peridynamics system with the PMB model is a lumped mass-spring system that has no energy dissipation. Therfore, an energy dissipation term, known as dynamic relaxation, must be added to (3.3), so that the solution will converge to a steady state solution. The method relies on the introduction of an artificial damping term in the equation of motion,

$$\rho\ddot{\mathbf{u}} + \eta\dot{\mathbf{u}} = \sum_{j \in \mathcal{H}_i} \mathbf{f}_{ij}(\boldsymbol{\xi}) dV_j + \mathbf{b}(\mathbf{x}, t). \tag{3.4}$$

Simulating the evolution of the model from its initial conditions requires a numerical time integration scheme. The choice is between *explicit* schemes, and *implicit* schemes[39]. Explicit schemes are most commonly used in peridynamics solvers because they more suited to parallel computation, since no solve of a system of equations is required. The specific method used was Euler-Cromer. A central difference method such as velocity-Verlet is often used for peridynamics solvers but Euler-Cromer was found to be suitably stable and is simpler to implement. Such Euler-Cromer algorithm leads to the following difference equations

$$\dot{\mathbf{u}}_i(t{+}1) = \dot{\mathbf{u}}_i(t) + \ddot{\mathbf{u}}_i(t{+}1)\boldsymbol{\Delta}t \tag{3.5}$$

$$\mathbf{u}_i(t{+}1) = \mathbf{u}_i(t) + \dot{\mathbf{u}}_i(t{+}1)\boldsymbol{\Delta}t \tag{3.6}$$

3.1.2 A stochastic reformulation of peridynamic theory

An equally valid model for the evolution of the elastic body is to assume that the system evolves to a configuration which minimises the total potential energy, $V(\mathbf{U})$, which is a

sum of the strain energy in all connecting bonds with a given particle, and the work done by an external load, $\mathbf{b}_i$, acting on a given particle. Therefore the peridynamic model can be expressed as a deterministic gradient flow

$$\eta\, \partial_t \mathbf{U} = -\nabla V(\mathbf{U}) \quad \text{for} \quad t > 0 \quad \text{and} \quad \mathbf{U}(0) \quad \text{given}, \tag{3.7}$$

where $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_n]^T$ is the complete solution vector, and η (not $\boldsymbol{\eta}$ from Chapter 2) is a parameter which has the physical interpretation as the damping/viscosity of the material: responsiveness of the material to non-equilibrium in forces acting on it. The potential function is

$$V(\mathbf{U}, \theta) = \sum_{i=1}^{N} \left[\left(\frac{1}{2} \sum_{j \in \mathcal{H}_i} \hat{w}_{ij}(\boldsymbol{\xi}_{ij}, \theta) dV_j \right) - \mathbf{b}_i^T \mathbf{u}_i \right]. \tag{3.8}$$

The first term is the sum of (half the) strain energy in all connecting bonds with a given particle, and the discretised version of Equation (2.9). The second term is the work done by an external force density $\mathbf{b}_i$ acting on a particle. The deterministic gradient flow for a particle $\mathbf{x}_k$ is evaluated as

$$\begin{aligned}
\nabla V(\mathbf{U})_k = \frac{\partial V}{\partial \mathbf{u}_k} &= \frac{\partial \boldsymbol{\xi}_{ij}}{\partial \mathbf{u}_k} \frac{\partial V}{\partial \boldsymbol{\xi}_{ij}} \\
&= \frac{\partial (\mathbf{u}_j - \mathbf{u}_i)}{\partial \mathbf{u}_k} \sum_{i=1}^{N} \left[\left(\frac{1}{2} \sum_{j \in \mathcal{H}_i} \boldsymbol{f}(\boldsymbol{\xi}_{ij}, \theta) dV_j \right) - \mathbf{b}_i \right] \\
&= \sum_{i=1}^{N} \left(\frac{1}{2} \sum_{j \in \mathcal{H}_i} \frac{\partial \mathbf{u}_j}{\partial \mathbf{u}_k} \boldsymbol{f}(\boldsymbol{\xi}_{ij}, \theta) dV_j \right) - \sum_{i=1}^{N} \frac{\partial \mathbf{u}_i}{\partial \mathbf{u}_k} \left(\frac{1}{2} \sum_{j \in \mathcal{H}_i} \boldsymbol{f}(\boldsymbol{\xi}_{ij}, \theta) dV_j \right) - \mathbf{b}_k \\
&= \frac{1}{2} \sum_{i=1}^{N} \boldsymbol{f}(\boldsymbol{\xi}_{ik}, \theta) dV_k - \frac{1}{2} \sum_{j \in \mathcal{H}_k} \boldsymbol{f}(\boldsymbol{\xi}_{kj}, \theta) dV_j - \mathbf{b}_k \\
&= -\frac{1}{2} \sum_{i \in \mathcal{H}_k} \boldsymbol{f}(\boldsymbol{\xi}_{ki}, \theta) dV_i - \frac{1}{2} \sum_{j \in \mathcal{H}_k} \boldsymbol{f}(\boldsymbol{\xi}_{kj}, \theta) dV_j - \mathbf{b}_k \quad \text{since } \boldsymbol{f}(\boldsymbol{\xi}) = -\boldsymbol{f}(-\boldsymbol{\xi}) \\
&= -\sum_{i \in \mathcal{H}_k} \boldsymbol{f}(\boldsymbol{\xi}_{ki}, \theta) dV_i - \mathbf{b}_k, \qquad \text{by putting } i = j
\end{aligned} \tag{3.9}$$

which is the force density acting on the particle, and equal to 0 for a particle in static equilibrium.

Central to the idea of Probabilistic Peridynamics, is to reformulate the discretised peridynamics equations by adding Brownian motion to each particle[40], thus allowing us to capture our uncertainty in the material failure at the macroscale, from missing information at the microscale. This additive Brownian forcing can be seen as a bias corrector for the model, for which could be inferred. The output would be a data-adjusted peridynamic formulation.

The corresponding stochastic dynamics is an overdamped Langevin diffusion [40], given by

$$\mathbf{dU} = -\eta^{-1}\mathcal{C}\nabla V(\mathbf{U})dt + \sqrt{2\mathcal{C}}\, d\mathbf{B} \quad \text{for} \quad t > 0 \quad \text{and} \quad \mathbf{u}(0) \quad \text{given.} \tag{3.10}$$

where $\mathbf{B}(\sigma)$ denotes n-dimensional Brownian motion in $\mathbb{R}^d$ with $\mathbf{dB} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}_d)$ and $\mathcal{C}(l)$ defines a covariance function which is a function of l which is a length scale, for say, a square exponential kernel

$$\mathcal{C}_{ij}(x_i, x_j) = \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T\mathbf{M}(\mathbf{x}_i - \mathbf{x}_j)), \tag{3.11}$$

where $\mathbf{M} = \text{diag}(l^{-2})$.

This equation is the update equation for an Euler time-integration scheme, which is discretised in time in a similar way to Equations (3.5) and (3.6). The value of $\eta = 1$ can be set arbitrarily since this parameter is paired with the time step size Δt, therefore the size of the timstep will control the speed at which strain waves propagate through the material. A larger time-step will result in a faster strain wave. The time-step should be made large enough such that the strain wave propagates across the whole structure in less than one time step, and is still stable. It was found for some problems that no Δt could be used such that this criterion could be satisfied. This was because at the desired strain wave propagation speeds, the accumulated error introduced by the Euler integration was too large and overwhelmed the calculation giving instability. Using a more accurate higher order integrator solves this problem. As such, a variety of adaptive and non-adaptive integration methods have been implemented, including Runge-Kutta-4, Dormand-Prince and Heun-Euler methods. These higher order methods contain multiple updates of displacement, so the Brownian forcing is applied at the final update step.

Note that Equation (3.7) is the same as Equation (3.4) except inertial effects are neglected. When using peridynamics to study quasi-static loading, such as on concrete beams, this simplification is justified, however, if the simulator was to be expanded to solve impact problems, then the inertial term should be included as follows

$$\partial_t \begin{bmatrix} \boldsymbol{U} \\ \rho\dot{\boldsymbol{U}} \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{U}} \\ -\nabla V(\boldsymbol{U}, \dot{\boldsymbol{U}}) \end{bmatrix}. \tag{3.12}$$

3.2 Spatial discretisation

Although peridynamics is a meshless method in the sense that its particles behave as lumped masses attached to springs, a Cartesian cubic (regular) or tetrahedral (regular or irregular) mesh can be used to define positions of the peridynamic particles on the vertices

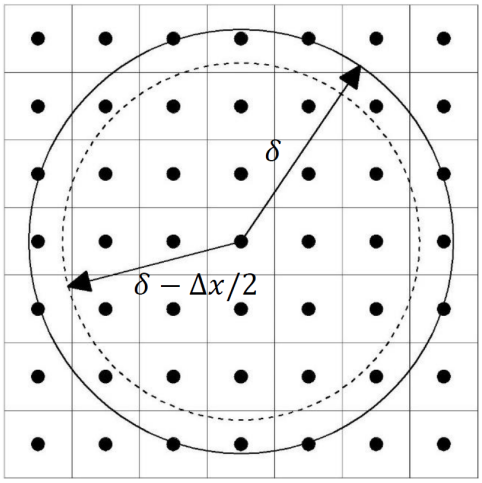
of the mesh. Existing solution schemes for peridynamic models have largely been based on cubic Cartesian meshes since it is easy to define the volumes lumped at the particles as the total volume of the structure divided by the number of particles, however, the regularity of these grids can introduce a directional bias into solutions, resulting in fracture patterns that artificially track symmetries in the mesh. To alleviate this behavior, it has been suggested that irregular quadrature point distributions, such as a centroidal Voronoi tessellation (CVT) are used. Henke et al.[41] explore the alternative of CVT generators to tensor product grids and found they achieve similar accuracy, yet possess a robustness to small position perturbations that the latter lacks. The regularity of the cubic meshes was problematic, resulting in cracks that propagate along lines of symmetry in the mesh, while fracture patterns on the CVT generators contained complex branching patterns that compared favorably to the results obtained at higher resolutions. The particle volumes of these irregular meshes can be calculated using Veronoi decomposition, but can be well approximated with a less computationally expensive and simpler Delaunay decomposition directly from a finite element mesh to calculate the volumes of the tetrahedra, with vertices that define the tetrahedra being allocated equal proportions of that volume. Both irregular and regular discretisation are explored in this report, and mesh dependency is explored in Section 6.2.

3.3 Spatial integration

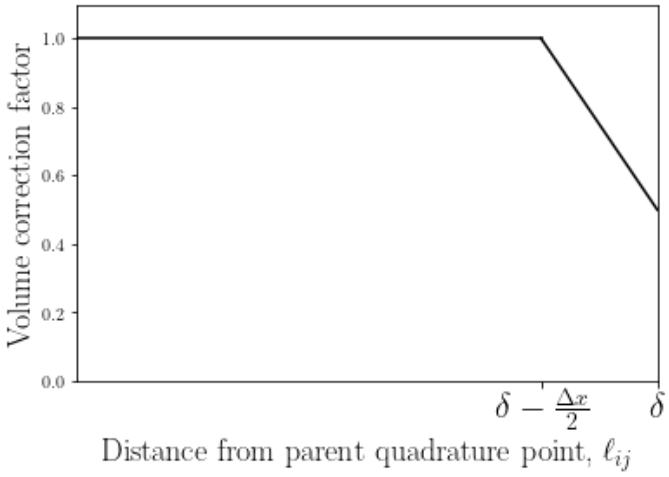
Both of the numerical integrators in 3.1.1 and 3.1.2 contain a numerical integration of the pairwise force function $\mathbf{f}$ over the horizon of a particle. The particle $\mathbf{x}_j$ being used as a quadrature point for the integration, and for some particles, a fraction of the cell volume will sit across the horizon boundary. Using the whole cell volume will lead to errors in the spatial integration. This is corrected by the volume correction factor. Seleson [42] provided a detailed analysis of the various methods available for volume correction. This work has implemented a volume correction procedure introduced by Parks et al [43] for its simplicity. There are examples in the literature where volume correction is ignored for solver speed reasons [14]. However, it was found that correcting for numerical integration in this way resulted in non-negligible changes in deflection and failure load of the test cases. It was implemented without a significant slow down in the speed of the solver.

3.4 Surface correction factors

The derivation of the peridynamic bond stiffness in Equation (2.11) assumes that a horizon, $\mathcal{H}_x$ is completely embedded in the body $\mathcal{R}$. This assumption does not hold for particles within a distance δ of the surface of the body. Therefore, to correct a surface softening



(a) The volume correction factor is applied to a regular discretisation with $\delta - \frac{\Delta x}{2} < \ell_{ij} < \delta$



(b) Volume correction factor against distance adapted from Liu & Hong, 2012.

Fig. 3.1 The volume correction factor applies a linear interpolation of the volume of

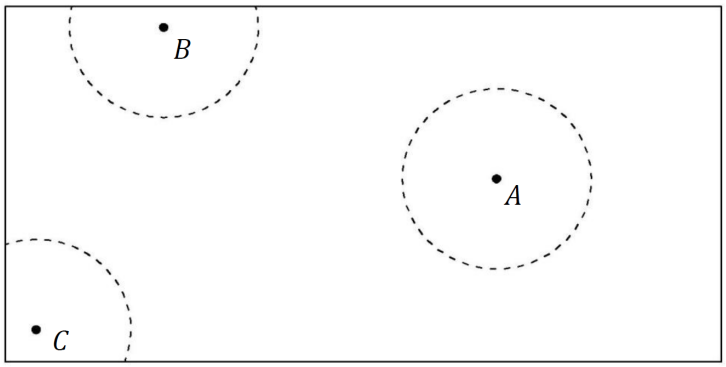


Fig. 3.2 Particle horizons. The horizon of Material Particle A is contained completely within the material bulk. The horizon of Material Particle B and C lies across the boundary of the peridynamic body. Image courtesy of Mark Hobbs.

effect, a surface correction must be applied. The volume method[37] applies a corrected stiffness

$$c' = \lambda c = \frac{2V_{\mathcal{H}_0}}{V_{\mathcal{H}_i} + V_{\mathcal{H}_j}} c \tag{3.13}$$

to the bond stiffness c of bond $\mathbf{x}_{ij}$, where $V_0 = \pi\delta^2$ in 2D, or $\frac{4}{3}\pi\delta^3$ in 3D. For example, in Figure 3.2, in 2D, the correction factor for a bond connecting a particle whose horizon is fully embedded to particle B, on an edge, or C, on a corner is $\lambda = \frac{4}{3}$ and $\lambda = \frac{8}{5}$ respectively.

Chapter 4

Implementation

It was proposed that if a peridynamic code has few dependencies, is easy to use and above all is fast, then the user-base and quality of research in peridynamics will increase. To that end, with the help of Professor Tim Dodwell, who leads the Data Centric Engineering Group at Exeter, and Dr Jim Madge, Research Software Engineer at the Alan Turing Institute, the author has developed an open-source GPU code that uses OpenCL for GPU acceleration, and is wrapped in python for the user input. All of the code used to produce the results in this report was written exclusively by the author and can be accessed on github.com/alan-turing-institute/Probabilistic-Peridynamics/tree/feature/pymc3. An archived version of the code, up to date as this report was written is made available at github.com/bb515/probabilistic-peridynamics-project. The GPU communicates with the host device (a CPU) with the OpenCL API. Since the front end of the code is written in python in order to make it accessible for research, the python API, pyopencl, was chosen.

4.1 Vectorisation

It has been shown that the numerical implementations in section 3.1.2 and 3.1.1 require a calculation of the resultant force density on each particle $\mathbf{x}_i$, $\sum_{j \in \mathcal{H}_i} \boldsymbol{f}(\boldsymbol{\xi}_{ji}, \theta) dV_i + \mathbf{b}_i$ (Equation (3.9)). This is the most expensive calculation for any peridynamics solver, and contributes to more than 95% of the runtime of a peridynamics integrator. This is because of the nested for loop required to calculate the force density Equation (3.9) for each particle, furthermore, the calculation of the pairwise force function requires the bond length ℓ from Equation(3.2), which involves an expensive square root operation. Therefore to write a fast implementation, the calculation and summation of the bond force density contributions must be tackled.

By vectorising the numerical calculations, it is simple to show that the force density contributions of the bonds can be decomposed into a series of element-wise multiplications and additions of the current particle coordinates $\mathbf{y}$ and initial particle coordinates $\mathbf{x}$ over a bond index space (a matrix) in $\mathbb{R}^{n \times n}$, where the rows and column indices represent particles.

The force density contributions in this bond index space are reduced to a resultant vector that represents the force density on the particles in $\mathbb{R}^{n\times 1}$. This is shown in Appendix A.1. In a serial peridynamics solver, each dimension of the bond index space represents a 'for' loop, with the row 'for' loop nested inside the column 'for' loop, corresponding to calculating the bond force density contributions and the particle force densities respectively. A single threaded CPU implementation calculates each of the elements of this index space in series, when in fact they could be calculated in parallel, since **the calculation of the contribution to force density of one bond is independent of another bond.**

Therefore, peridynamics is ideal for GPU acceleration, since each bond calculation can be done on one of the many concurrent threads in a GPU. This is called *loop-unrolling*. A function which is written in OpenCL (using C) which takes advantage of loop-unrolling is called a *compute kernel*.

4.1.1 N-dimensionality over work items

Loop un-rolling can be done easily in OpenCL. Existing OpenCL peridynamics codes have exploited this trick, as shown in the compute kernel pseudo-code in Appendix A.2, which was adapted [1] from the work done by Mossaiby et al. [24], where they have un-rolled only the outer loop over the particles. This compute kernel is denoted 'CalcBondForce1' and was used to benchmark the author's improvements.

It is possible to un-roll the nested loops by taking advantage of the multi-dimensional nature of OpenCL code. The structure of OpenCL is N dimensional in the sense that algorithms can be written in an N dimensional index space that work items are mapped to. This structure makes it easier to implement 2D and 3D algorithms, such as a convolution with a 3D kernel in a convolutional neural network (CNN), but cannot be exploited for speed. In peridynamics, a nested 'for' loop is unrolled so its algorithms are written in a 2D structure, as summarised in the lines of pseudo-code in Algorithm 1.

[1] The compute kernels by Mossaiby et al. were extended with numerous additional features, such as the correction factors discussed in section 3.3 and 3.4, ability to model composite materials and apply general boundary conditions. It should be noted that these features are all necessary ones for a basic peridynamics simulator, but are not expected to change the speed of the code significantly (it amounts to a indexing a stiffness array where a constant stiffness would have been used, for example).

Algorithm 1 Pseudocode of multi-dimensional loop unrolling in an OpenCL kernel.

```
procedure CalcBondForce2(U, X, Stiffnesses, Volumes)
    const int i = get_global_id(0)                    ▷ Get parent particle
    const int j = get_global_id(1)                    ▷ Get j ∈ H_i
    if i < NUMBER_OF_PARTICLES && j < MAX_HORIZON_LENGTH then
        double bond_forces[i * HORIZON_LENGTH + j] = bond_force
    end if
end procedure
```

This is the premise behind the first optimised compute kernel, 'CalcBondForce2'. In their code, Mossaiby et al. have written the algorithm that checks for broken bonds in 2D, but they have not written the 'CalcBondForce1' algorithm in 2D. This is because they have chosen to do the sum across bond force density contributions in series, without un-rolling the inner horizon loop. It is possible however to do this inner sum in parallel, using a binary parallel reduction algorithm in local memory. Before this is explained, it is necessary to understand the OpenCL memory model.

4.2 The OpenCL memory model

The architecture of a GPU contains a *memory hierarchy*. In general, the lower down and more locally the memory can be used in its architecture, the faster it is. Therefore it is imperative to exploit the memory hierarchy in GPU accelerated programming. In the OpenCL memory model, as shown in Figure 4.1, the concurrent threads are called *work items*. Each *work item* has a small amount ($O(10)$ words) of very fast *private memory* that it uses to do fast floating-point arithmetic, such as the aforementioned additions and subtractions in Appendix A.1. Each *work group* contains many work items that share its *local memory*. This memory is larger ($O(10)$kBytes) than the private memory of a work item, but slower. The largest memory on a GPU is the *global memory* which is on the order of 10Gb and slower still. The work by Mossaiby et al. only uses global memory.

Work groups, like the work items, are run in parallel. There is no way to synchronize the timings of work groups, since in an application there are usually more work groups than hardware to execute them concurrently, so they will be queued to run when there is hardware available to run them in batches, called *wavefronts*.

4.2.1 Parallel reduction of bond forces onto particle forces

Now that the memory model has been explained, the most significant optimisation between this work and existing literature by Mossaiby et al. is demonstrated. In the author's code, the 'CalcBondForce' algorithm is written in 2D and the bond density contributions in

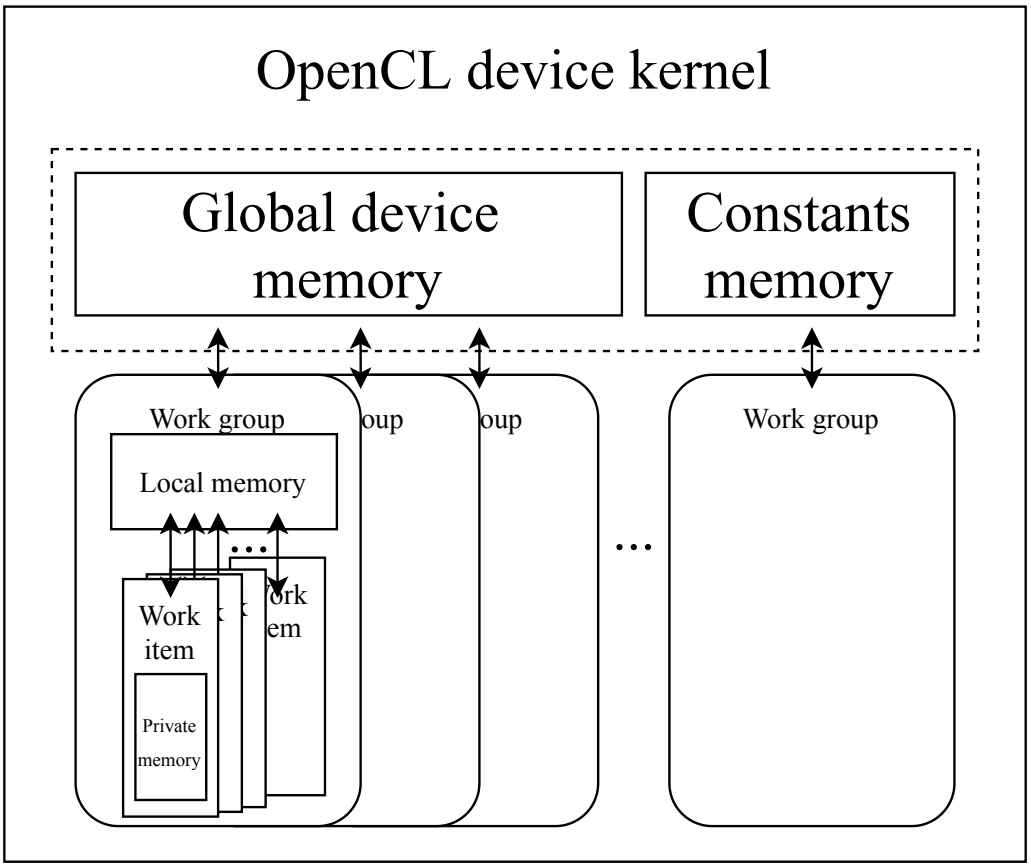


Fig. 4.1 The OpenCL memory model is hierarchical.

$\mathbb{R}^{n\times n}$ are summed by parallel binary reduction down to the particle forces in $\mathbb{R}^{n\times 1}$. The parallel binary reduction is summarised in the pseudo-code in Algorithm 2.

The idea of this compute kernel is that it uses strides to perform an addition on two values and propagates this result downwards so that the result is inserted into the first memory space of that particular local memory group. The innovation here is that the size of the memory group is always equal to the number of particles in a family $\mathcal{H}_i$, so the value which has been reduced to the first element of the local group will also be the particle force in a particular cartesian direction. This of course puts a restraint on the local size (size of the local memory group), since the global size must be a factor of the local size because the global size of the problem are split equally into separate work groups with a local size. This is where we get particularly lucky with peridynamics, and where doing a parallel reduction makes real sense:

The family size is, for an isotropic material, independent of problem size[2]. This is because, typically in literature, a horizon radius δ is selected so that it is the width of 3 of the particle spacings Δx [38]. Therefore the local size need not change between simulations of different problem size and can be selected as an optimal value, 128, 256 or

[2]'problem size' is the number of particles in a simulation, and 'family size' is the maximum number of particles in a family, $\max \text{length}(\mathcal{H}_i)\forall i$

Algorithm 2 Pseudocode of a parallel binary reduction across bond forces in an OpenCL kernel.

```
procedure REDUCE BOND FORCES(bond_force_density_contributions)
    const int i = get_global_id(0)                                  ▷ Get bond
    const int j = get_local_id(0)                                   ▷ Get j ∈ H_i
    int local_size = get_local_size(0)      ▷ The initial size of the row to be reduced
                                        ▷ Copy values to be reduced into local memory
    local_cache[local_id] = bond_force_density_contributions[global_id]
    barrier(CLK_LOCAL_MEM_FENCE)            ▷ Wait for all threads to catch up
    for int i = local_size/2 ; i>0; i/=2 do
        if local_id < i then
            local_cache[local_id] += local_cache[local_id + i]
            barrier(CLK_LOCAL_MEM_FENCE)    ▷ Wait for all threads to catch up
        end if
    end for
    if !local_id then               ▷ First entry in each row, which now contains the sum
        int index = global_id/local_size
        particle_force_densities[index] = local_cache[0]
    end if
end procedure
```

512, depending on the GPU by padding the sparse connectivity matrix, which is a verlet list of connections between each of the particles, with null values [3]. Therefore, the global size is the total number of bonds,

Listing 4.1 Global size of the reduction kernel.

```
global_size = MAX_HORIZON_LENGTH * DEGREES_FREEDOM * n
```

where DEGREES_FREEDOM is 3 for 3D problems, n is the number of particles in the simulation as before, and MAX_HORIZON_LENGTH is the local size is the smallest power of 2 which is larger than the longest length of the verlet lists. This tends to be 64, 128 or 256 for any 3D peridynamics problem, which is a perfect local size for most devices:

> *"To keep the device utilization high with the limited number of workgroups, larger workgroup sizes are required. Use power-of-two workgroup sizes between 64 and 256."*

Fig. 4.2 OpenCL™ Developer Guide for Intel® Processor Graphics https://software.intel.com/en-us/iocl-opg-work-group-size-recommendations-summary . This is in reference to intel on board graphics, but the same applies to GPUs. Usually when testing a kernel, experimentation is used to pick a fastest local size up to a maximum value that depends on the device, which is 1024 for the GeForce RTX 2080 TiGPU used to produce the results in this report.

[3] −1 was used, as the rows of the verlet list are particle indices and as a rule, you can't have a negative particle indices.

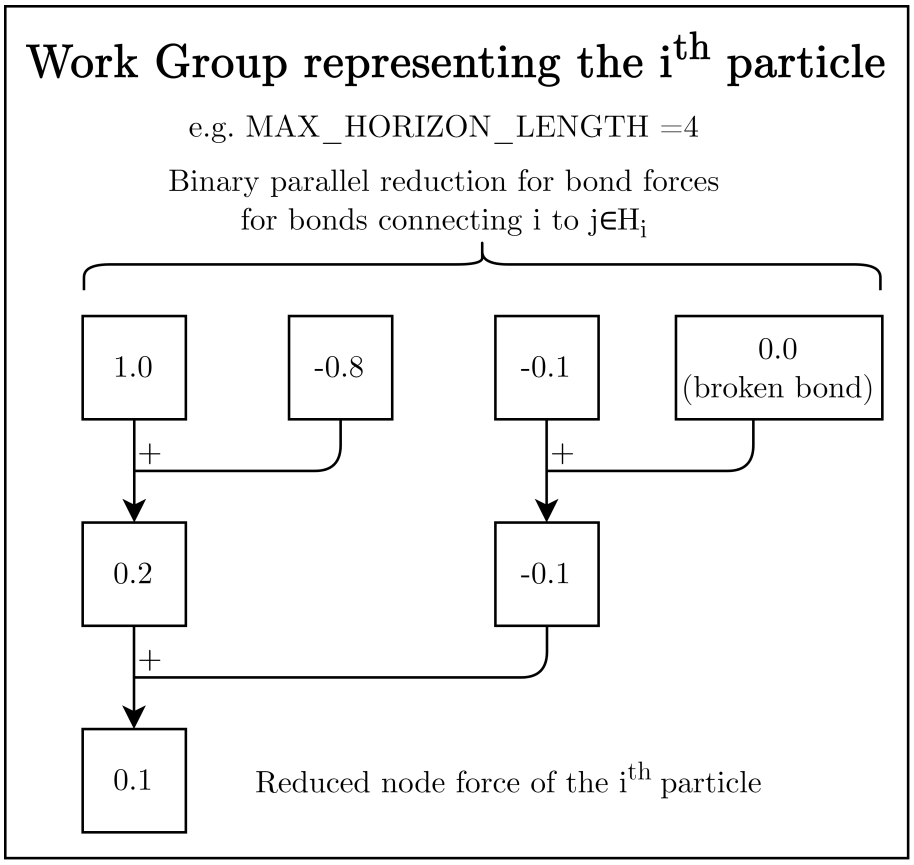


Fig. 4.3 A parallel reduction of bond forces down to particle force.

A graphical representation of the code in Algorithm 2 is shown in Figure 4.3. The reduction is expected to make a significant difference to the speed of the code, however, there is a memory bottleneck associated with storing the bond forces in a global array of size global_size in Algorithm 1 and then accessing them in a separate kernel as is done in Algorithm 2. This bottleneck must be resolved in order for the proposed implementation to scale.

4.2.2 Avoiding the global storage of a bond force matrix

There is more memory efficient way to do the parallel reduction without storing all of the bond force density contributions in an array. By merging the compute kernels of Algorithms 1 and 2 into a single compute kernel, there is no need to store the bond force density contributions in global memory, as they can be directly stored and summed in local memory. With this optimisation, which is denoted 'CalcBondForce3', and displayed in Appendix A.2, the author's implementation will be significantly faster than the existing literature.

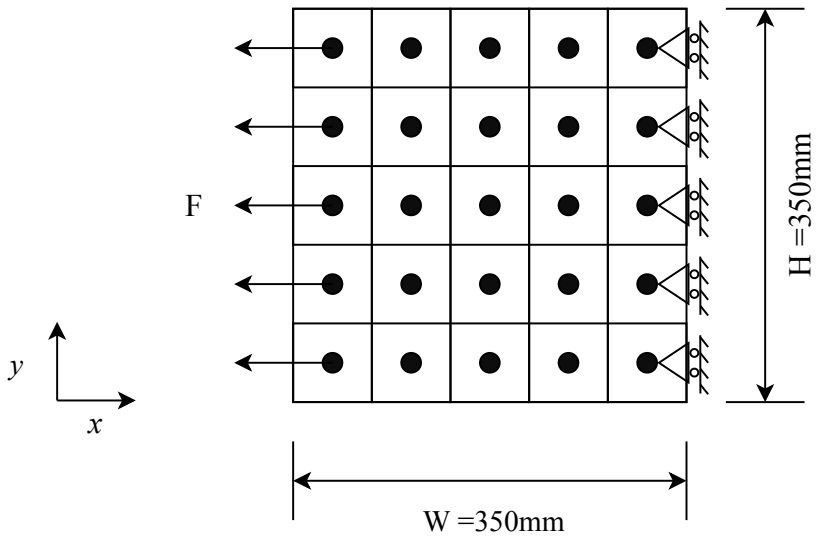


Fig. 4.4 Schematic of the 2D tensile test. The line of particles on the right-hand side of the plate are clamped in the x-direction. A total load of $F = -50$kN in the x-direction was applied, distributed over the particles on the left hand side.

4.3 Validation of the code

The convergence of the author's OpenCL implementation to classical continuum mechanics is demonstrated here for a thin 2D plate in plane stress conditions. In 2D plane stress conditions, the peridynamic bond stiffness can be derived as $c = 9E/(\pi t\delta^3)$, and the poisson ratio is $\nu = 1/3$[44].

4.3.1 Two dimensional tensile test

Figure 4.4 shows the two-dimensional tensile benchmark. The displacement of a node $\mathbf{x}_i$ for tensile behaviour can be derived using the Airy stress function as

$$\mathbf{u}_x(\mathbf{x}_i) = \frac{F}{EWT}(\mathbf{x}_{i,x} - W),$$
$$\mathbf{u}_y(\mathbf{x}_i) = -\frac{\nu F}{ET}\left(\frac{\mathbf{x}_{i,y}}{W} - \frac{1}{2}\right),$$

where W, H and T are the plate's width, height and thickness respectively. A thickness $T = 1$mm was used, and the 2D problem was discretised with $n = 96^2$ particles. $\mathbf{x}_{i,x}$ and $\mathbf{x}_{i,y}$ denote the x and y coordinate with the bottom left corner of the plate as origin. Whilst peridynamics correctly predicts the shape of the displacement field when compared to the classical continuum mechanics solution, as shown in Figures 4.5 and 4.6, there is a small error when comparing the classical continuum mechanics solutions to peridynamics solutions, as shown in Figure 4.7. These are discretisation errors and peridynamic surface effects, and not due to errors in the code.

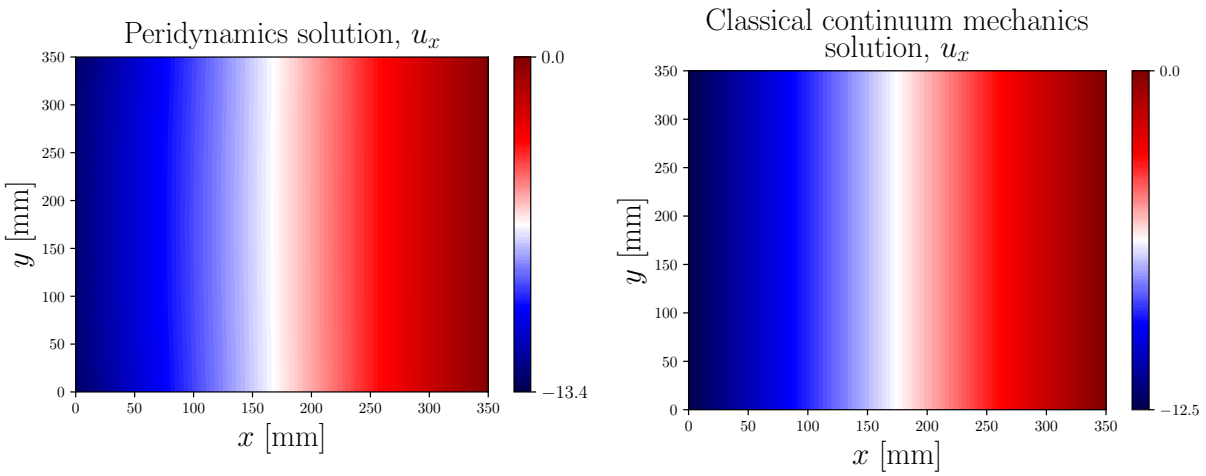


Fig. 4.5 x-direction displacement field solutions.

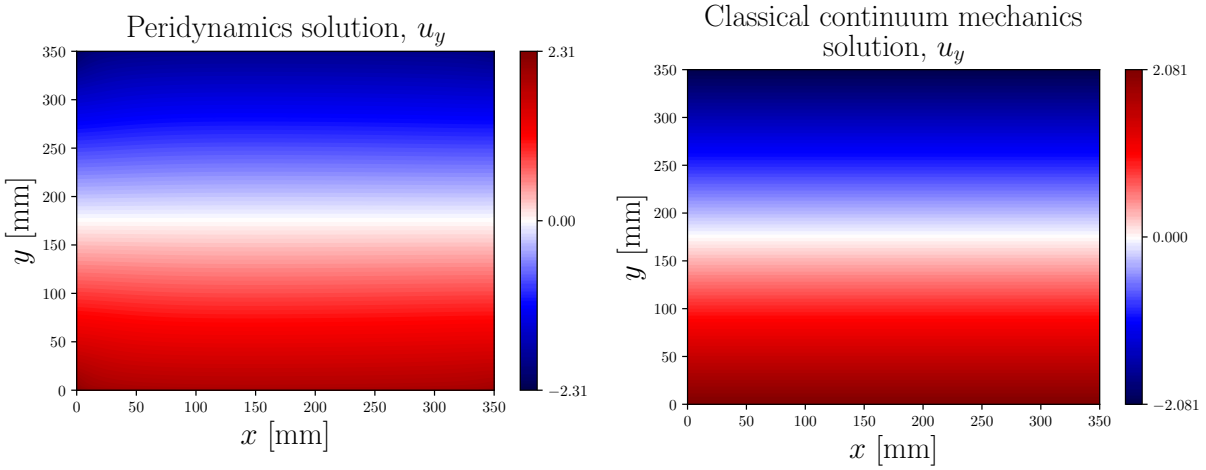


Fig. 4.6 y-direction displacement field solutions.

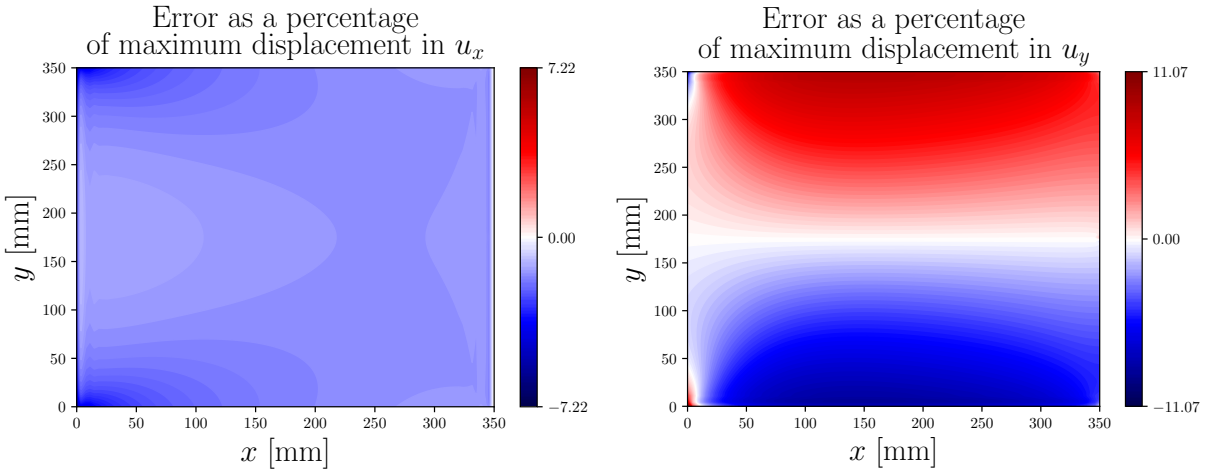


Fig. 4.7 The error as a percentage of the maximum displacement.

Chapter 5

Experimental setup

The author's peridynamics solver was tested using a GeForce RTX 2080 TiGPU from NVIDIA, with a maximum of 68 compute units, a maximum of 1024 work-items per work-group (of which 256 were used), a max work item dimensions of 3 (2 were used), maximum work item sizes of [1024, 1024, 64], a local memory size of 49152 bytes, and 11 GB of GDDR6 VRAM, courtesy of HYDRA and the Data-Centric Engineering Group at the University of Exeter. Due to the heterogeneous nature of OpenCL, the code can be executed on any CPU or inbuilt graphics for small problem sizes.

5.1 GPU implementation profiling versus existing literature

In order to test the stability, speed and scaling of the compute kernels 'CalcBondForce2', 'CalcBondForce3' and 'CalcBondForce3_merged' against the code which was based on the contribution by Mossaiby et al., 'CalcBondForce1', the solver was profiled using the different compute-kernels swapped into place in the determistic Euler-Cromer and Euler time-integrators described in Section 3.1.1 and 3.1.2 respectively. Note that 'CalcBondForce3_merged' is an experimental compute-kernel that merges the displacement and velocity updates of the time integration into the CalcBondForce3. By merging the time integration kernels into a single kernel, there is less memory transfer at the expense of loss of synchronisation between the difference equation updates, UpdateDisplacement (equation 3.6) and UpdateVelocity (equation3.5).

To benchmark the scaling of the GPU implementation using different OpenCL kernels, meshes containing various numbers of particles were tested, ranging from 10^2 to 10^6 particles. The speedup factor of the author's code over the literature, 'CalcBondForce1' for a simulation over 1000 time steps was measured for the Euler quasi-static integrator, and the Euler-Cromer dynamic integrator as were described in sections 3.1.2 and 3.1.1 respectively.

The runtime profiles were then compared by comparing the wall-clock time across the different compute-kernels for 1000 time steps. The CalcBondForce function was also compared to the other compute-kernels within the time integration, such as UpdateDisplacement and UpdateVelocity. Results for this experiment are shown in section 6.1.

5.2 Deterministic concrete experiments and functionality of the implementation

To test the numerical stability and functionality when simulating reinforced concrete, the solver was tested on two concrete beam examples. Results of this experiment give insight into how well the PMB model predicts the failure loads of concrete beams, with the dependency on mesh spacing Δx and meshing method (regular or irregular) on the convergence of a failure load both considered. The peridynamics solver predictions of failure load were compared to lab tests of the beams. All simulations were performed without prior knowledge of the failure loads from the lab tests.

The first lab test was an approximately uniformly loaded cantilever beam of length 3300mm, denoted 'Beam 1', and is shown as a schematic in Figure 5.1. The second lab test, 'Beam 2' was a point loaded cantilever beam, 1650mm in length, as shown in Figure 5.2. These particular tests were picked as comparisons because of the shear dominated failure modes, so that a plastic steel constitutive model need not be defined. Each beam was tested twice in the lab, using concrete of cylinder strengths 31.1 MPa and 33.5 MPa. Beam 1 was tested to failure in the peridynamics simulation using force control, and Beam 2 was tested to failure using both force and displacement control. The ultimate failure load of the beams can be clearly determined from the force displacement graphs produced by the simulation, as would be done in the lab tests. Therefore it is not necessary to define a peridynamic failure criterion. The results for this experiment are shown in section 6.2.

The values of the parameters for the deterministic experiments followed choices in existing literature. The values of material constants were discussed in section 2.3.1. The horizon radius $\delta = \pi\Delta x$ was chosen, which is found extensively in the literature for macroscale analysis[45]. Table 5.1 shows the dynamics parameters that were used.

Table 5.1 Simulation parameters for deterministic concrete experiments.

Parameter	Value	
	Beam 1	Beam 2
Damping coefficient, $\eta \left[\frac{kg}{m^3 s}\right]$)	2.0×10^6	2.0×10^6
Load increment for force control tests $[N]$	1.0	1.0
Maximum load increment per time-step for displacement control test $[m]$	1e-8	1e-8

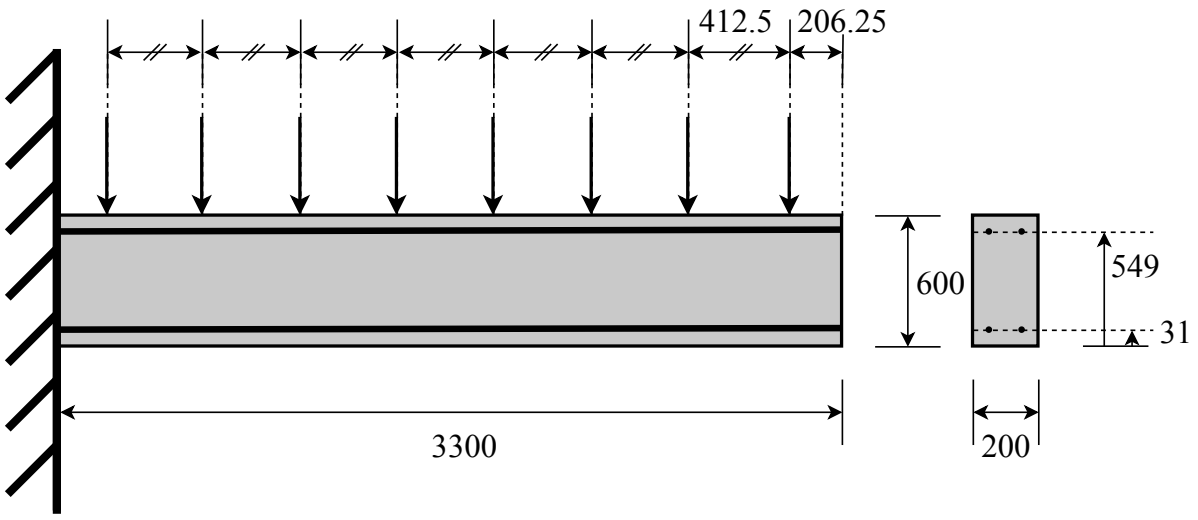


Fig. 5.1 Cantilever Beam 1, loaded approximately uniformly by 8 line loads.

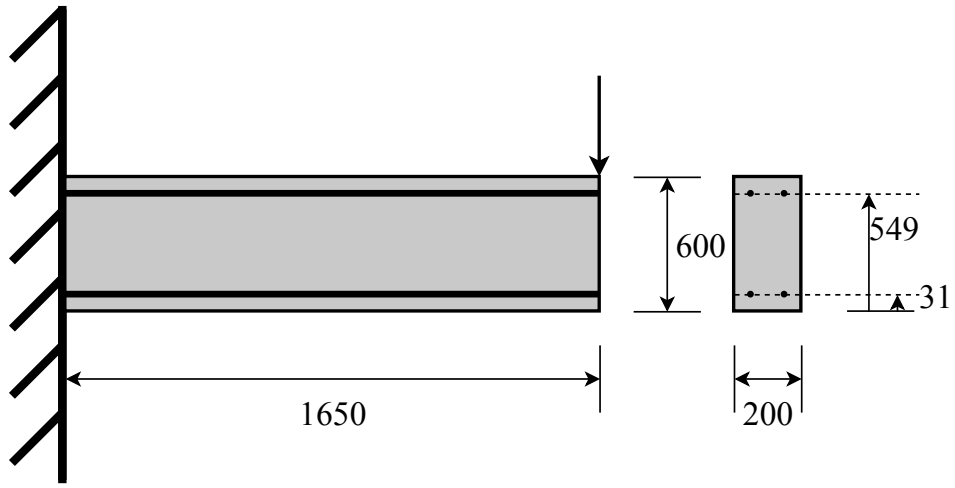


Fig. 5.2 Cantilever Beam 2, point loaded at the tip.

5.3 Stochastic experiments

The stochastic experiments were designed to demonstrate the output of Probabilistic Peridynamics as a forward distribution of crack paths on a small tensile problem shown in Figure 5.3. A number of crack samples are displayed in Figure 5.4. The values of the parameters used to generate this distribution were $log(\ell) = -5.0$ and $log(\sigma) = -4.0$. The parameters $\boldsymbol{\theta} = \{\ell, \sigma\}$ were then recovered via maximum likelihood estimation, using damage data from one of the samples.

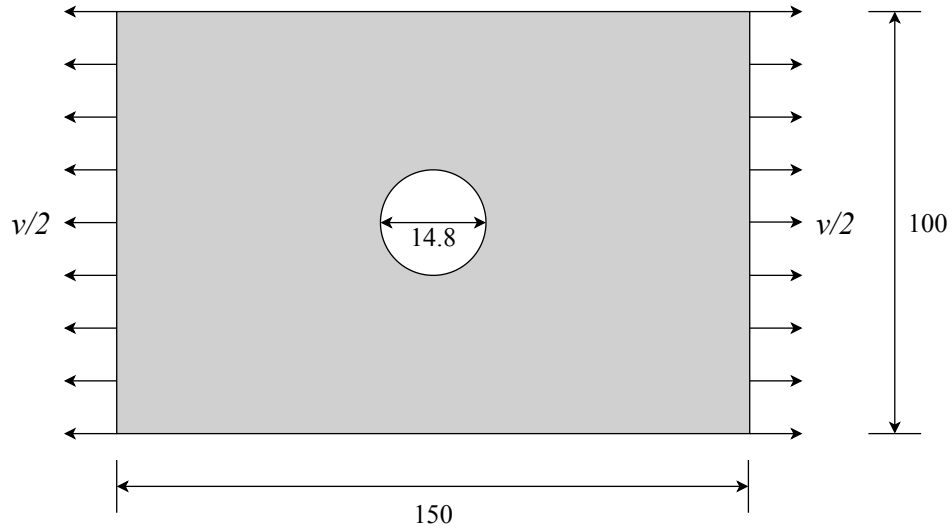


Fig. 5.3 Thin brittle plate, with a thickness 1.7mm. Particular deterministic model parameters were arbitrarily picked for this experiment.

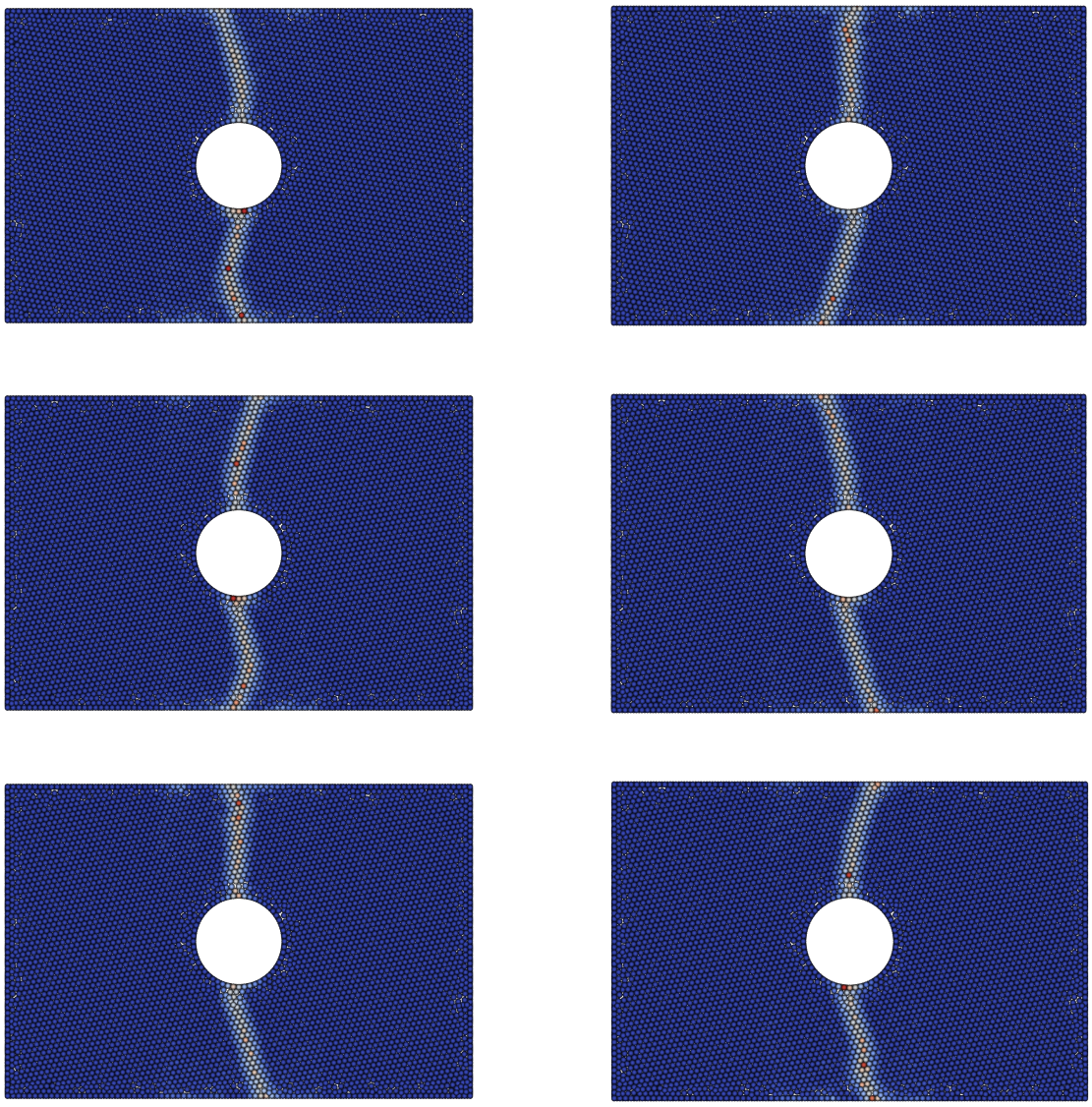


Fig. 5.4 Crack path samples from the distribution over **d** for the 3D tensile problem described in section 5.

5.3.1 Parameter estimation using maximum likelihood

In order to recover the stochastic parameters $\boldsymbol{\theta} = \{\sigma, \ell\}$, it was assumed that the mathematical representation of the process, a random realisation of the peridynamic damage $\boldsymbol{\phi}(\boldsymbol{\theta})|_{t=T}$ at timestep T, is modelled by the parameters

$$\boldsymbol{\theta} = \{\delta, c, s_c, l, \sigma\} \tag{5.1}$$

where for brevity, PD and SDE parameters have been grouped. The values of the deterministic parameters, $\delta = 3\Delta x, c = 280 \times 10^6, s_c = 5 \times 10^{-3}$, were assumed known for this experiment. It was assumed that real-life damage data is normally IID around the mathematical representation of the process, in a scenario where there is the technology to correctly measure and discretise cracking damage data,

$$\mathbf{d} \sim \mathcal{N}(\boldsymbol{\phi}, \sigma_d^2 \mathbf{I}). \tag{5.2}$$

Although peridynamics shows promise in visually matching experimental crack path results more closely than other methods[8], it might be that this is a naive assumption. The caveat is that, for the purposes of this report, synthetic training data was generated using the mathematical representation $\boldsymbol{\phi}$ itself, so that this assumption is a given truth.

The standard deviation of the generative model $\sigma_d = 1$ say, is a constant, and its value controls the tempering of the posterior distribution over parameters $\boldsymbol{\theta}$. Its value doesn't matter for parameter estimation purposes.

By training $\boldsymbol{\theta}$ on data $\mathbf{d}$, the parameters that closest match real cracking data using maximum-likelihood were 'learned'. The likelihood of the data is an integral over the distribution of random damage realisations $\boldsymbol{\phi}$ generated by the stochastic dynamics in 3.10,

$$p(\mathbf{d}|\boldsymbol{\theta}) = \int_{\phi} p(\mathbf{d}, \boldsymbol{\phi}) d\boldsymbol{\phi} \propto \int_{\boldsymbol{\phi}} p(\mathbf{d}|\boldsymbol{\phi}; \boldsymbol{\theta}) d\boldsymbol{\phi} \tag{5.3}$$

is approximated by the sum

$$p(\mathbf{d}|\boldsymbol{\theta}) \propto \frac{1}{M} \sum_{m=1}^{M} p(\mathbf{d}|\boldsymbol{\phi}_m; \boldsymbol{\theta}) = \frac{1}{M} \sum_{m=1}^{M} \prod_{i=1}^{N} p(d_i|\phi_{i,m}; \boldsymbol{\theta}). \tag{5.4}$$

The maximum likelihood estimate is

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \left\{ \log \left(\frac{1}{M} \sum_{m=1}^{M} p(\mathbf{d}|\boldsymbol{\phi}_m; \boldsymbol{\theta}) \right) \right\}, \tag{5.5}$$

Since the peridynamic model does not yield analytical solutions to the damage vector, it is not possible to evaluate analytically the derivatives of the damage data log likelihood surface with respect to either the deterministic peridynamics or probabilistic peridynamics

parameters. In order to find a point estimate of the parameters, it is therefore necessary to do a grid search over parameters, or to use Monte Carlo methods. An alternative method of inference which is performed directly from the SDE (Equation (3.10)) is suggested as further work in Section 7.

A grid search of the negative log likelihood over the stochastic parameters $\boldsymbol{\theta} = \{\sigma, \ell\}$ was performed. The likelihood Equation (5.4) was approximated with $M = 200$ which results in a surface which is not smooth. As the sum closer approximates the integral, the likelihood surface gets smoother, but the number of crack samples taken increases linearly with M. So, to find a global minimum, the negative log likelihood values were used to generate a smooth surrogate model: an optimised Gaussian Process of the negative log likelihood over parameter space, as shown in Figure 6.12. The maximum likelihood estimate of the surrogate model was then compared to the parameters picked a priori to generate the synthetic data.

5.3.2 Deterministic parameter estimation using MCMC

To investigate the relationship between the deterministic peridynamics parameters and damage for the tensile problem, a posterior distribution over the deterministic model parameters $\boldsymbol{\theta} = \{c, s_c\}$ was estimated using the Metropolis algorithm. The variance of the proposal distribution was tuned such that 165361 of 700000 proposed samples were accepted which took approximately 80 hours.

Chapter 6

Results and discussion

6.1 GPU implementation profiling versus existing literature

Figures 6.1 and 6.2 show the speedup factor that was obtained over the code from existing literature, CalcBondForce1, over 1000 time-steps. The line of best fit is the mean of a fitted Gaussian Process. Note that error bars of the underlying function are not shown, and the mean line should not be used to extrapolate performance. The results show that a speedup of between 2.0 and 5.0 times has been achieved over existing literature by optimising the OpenCL kernels to utilise local memory and perform a parallel reduction.

The speedup factor of the kernel which stores the bond force density contributions in a global array, CalcBondForce2 (blue line), was significantly slower than CalcBondForce1 for problem sizes above $\log(n) = 9$. This is because the implementation was slowed by the memory limitations of the GPU, which had effectively run out of working global memory, becoming memory constrained. For example, a problem with 495000 particles, $\log(n) = 13$ has a global_size $= 256 \times 3 \times 495000 = 0.38016 \times 10^9$. Double precision floating point numbers usually occupy 8 bytes of computer memory, so the space required to store the bond force array is on the order of $8 \times 0.38 = 3.0$Gbytes. This is a huge amount of memory to store every time step, and on the order of the working global memory size of the GPU used to perform these tests.

It has recently been claimed that there is not enough onboard memory in GPUs to support large problem sizes in peridynamics [18]. These results show that this only correct if the bond force density contributions are stored in an array of size global_size, since the CalcBondForce3 compute-kernel, which only stored bond force densities in local memory, was significantly faster than the existing literature for large problem sizes, $\log(n) > 10$.

The UpdateDisplacement and UpdateVelocity kernels are memory bandwidth bound since there are more memory transactions than floating point operations involved in updating the arrays. Therefore, the only way to speed up these kernels is by *fusing*; merging

them with other kernels. It is shown that merging UpdateDisplacement, UpdateVelocity and CalcBondForce3 into a single CalcBondForce3_merged, results in a further marginal speedup. Grouping kernels together in programs allows them to reference and use each-other without their source needing to be duplicated, saving on time and memory, hence the performance increase for large problem sizes. However, since it is not possible to synchronise across the work groups as, for each particle, the displacements used to calculate some of the bond forces may be from an adjacent time step. This introduces a small error into the time integration for elastic problems, and a large error in the time integration for cracking problems. Therefore, this kernel will not be implemented into the solver.

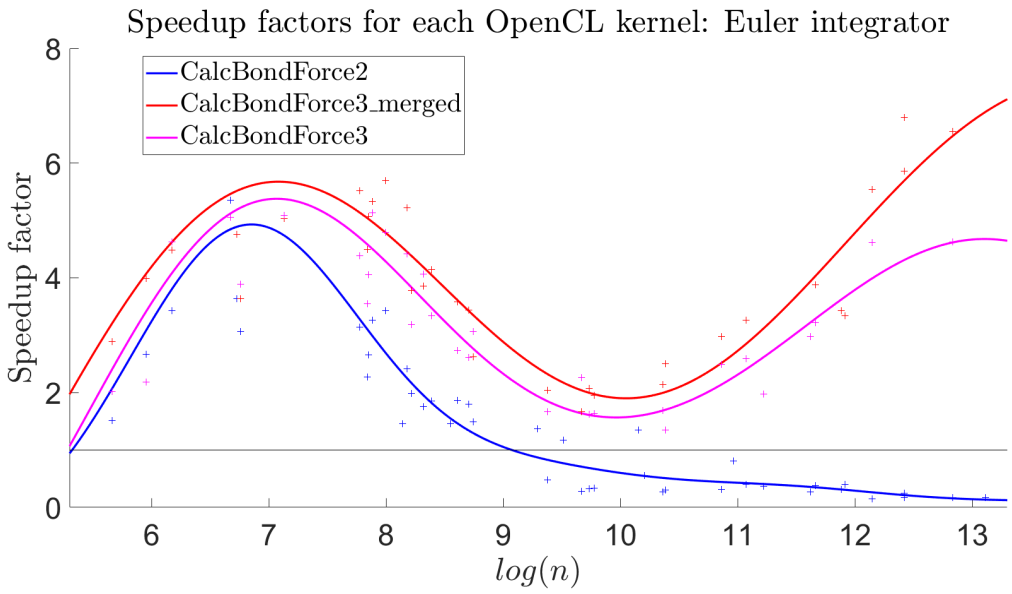


Fig. 6.1 Speedup factor over existing literature for the Euler integrator. The x axis is the *natural* log of the number of particles.

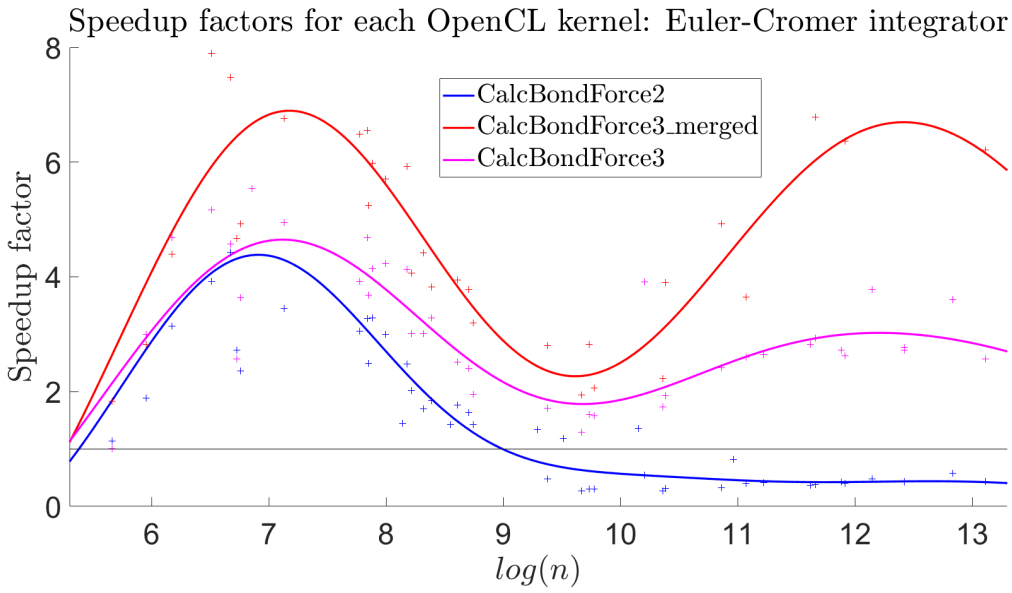
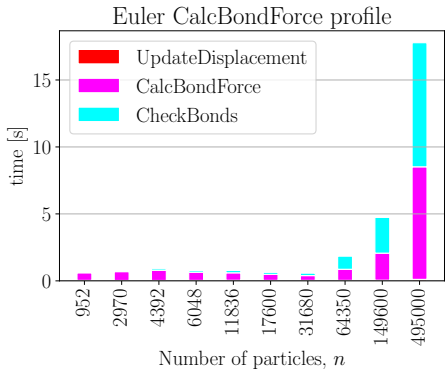


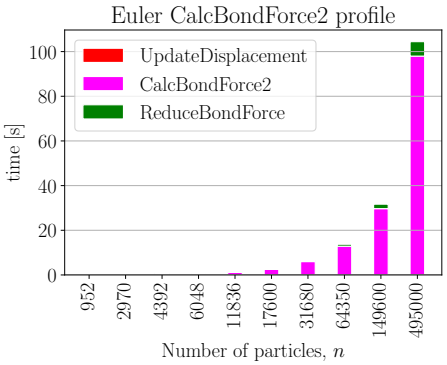
Fig. 6.2 Speedup factor over existing literature for the Euler-Cromer integrator.

Figures 6.3 and 6.4 show the time-integration wall clock time for 1000 time steps across the different compute-kernels for the Euler and Euler-Cromer integrators, respectively. These results show that the compute time for 1000 time steps is quick enough for parameter estimation purposes, with sample times around $0.3s$ for medium sized problems $n = 31680$ using the Euler integrator. The result is that with the author's speed ups, the sample time for cracks is now small enough such that parameter estimation possible with peridynamics.

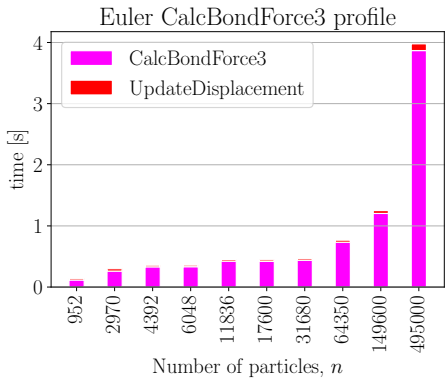
The GPU performance is independent of problem size until the GPU becomes memory constrained, for the Euler integrator at above $n = 31680$. The Euler-Cromer integrator becomes memory constrained at lower problem sizes $n = 17600$ because this integrator requires the additional global storage of a particle velocity array over the Euler integrator.



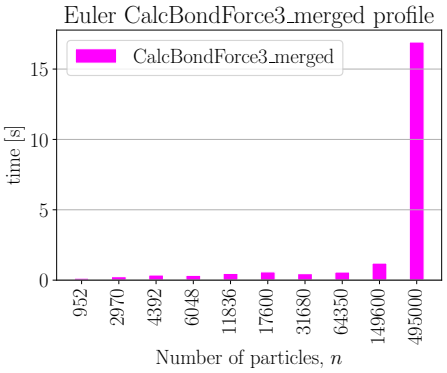
(a) This kernel does not use local memory.



(b) Parallel reduction implementation.

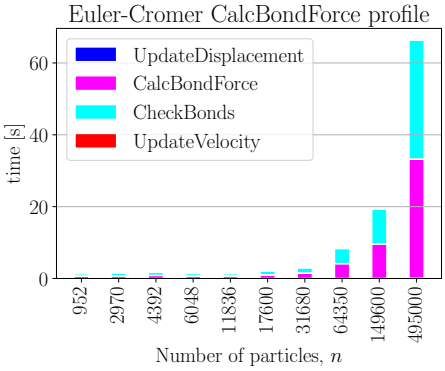


(c) Parallel reduction with memory optimisations.

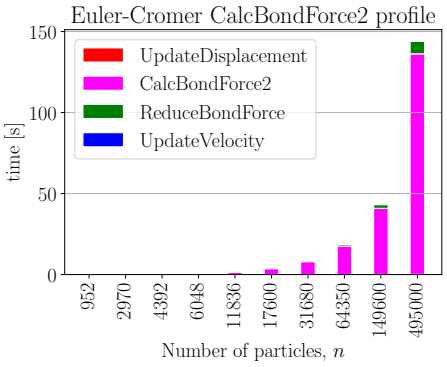


(d) Experimental fused kernel.

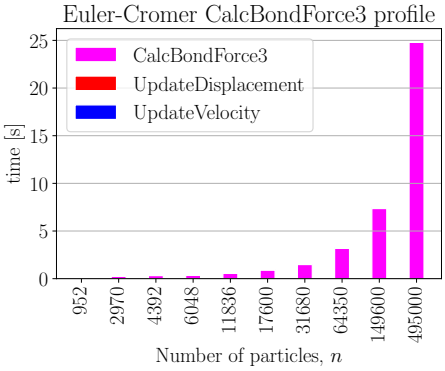
Fig. 6.3 Euler integrator profiling for the different OpenCL kernels. Note how the profile of the integrators differ between implementations, for example, the CheckBonds function was written as a separate kernel in the CalcBondForce implementation, but merged with the CalcBondForce2 and CalcBondForce3.



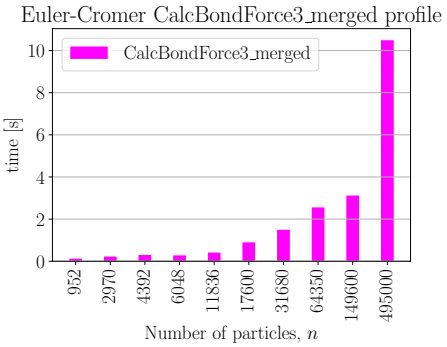
(a) This kernel does not use local memory.



(b) Parallel reduction implementation.



(c) Parallel reduction with memory optimisations.



(d) Experimental fused kernel.

Fig. 6.4 Euler-Cromer integrator profiling for the different OpenCL kernels. In comparison, the OpenMP Euler-Cromer solver currently in use by the Peridynamics Group at Cambridge University Engineering Department takes 150s for 1000 time steps for a problem size of $n = 424500$ on the CSD3 Peta4 supercomputer.

6.2 Force displacement curves

The force displacement curves for the peridynamic simulations of load tests are shown in Figures 6.6 - 6.9. The results generated from the regular mesh were mesh size Δx dependent, and did not converge to a failure load, whereas the results generated from the irregular mesh did converge on failure loads. An explanation to the non-convergence of the regular, Cartesian cuboidal mesh is explained by the ratio of steel particles to concrete particles. Figure 6.5 shows that the steel percentage (ratio of steel particles to total particles) converges quickly as the particle count increases for the irregular mesh (blue line), where particle positions are pseudo-random, when compared to the regular cuboidal mesh (red line). This is because in the regular cuboidal mesh, the ratio of steel to concrete particles is heavily biased if one of the grid lines passed through the cross-section of the steel reinforcement. For problem sizes below $\log(n) = 9$, there were no steel particles in the discretisation of the cuboidal mesh. Figure 6.5 compares the steel percentage convergence to the convergence to a failure load of 70.0kN for Beam 1. Both metrics follow a similar

pattern, and it makes sense that the failure load and mode of a simulation would be dependent on the ratio of steel to concrete particles. Furthermore, if the Cartesian grid lines of the regular mesh fall differently on each side of the rebar, then there will be less steel on one half of the cross-section when compared to the other. This nodal antisymmetry will cause unequal end moments at the two ends of the specimen, resulting in rigid body rotation and development of additional internal stresses. It is therefore recommended that irregular meshes are used to study concrete behaviour.

As a result of the large variation in steel percentage in the regular mesh, Beam 2 exibits two different types of failure, depending on the steel percentage. As shown in figure 6.8 For $n = 144900$ and above, the failure is characterised by almost no cracking (resulting in the constant stiffness) and a sudden shear failure. Contrastingly, for lower values of n, and thus lower steel percentages, the failure was flexural, with cracks developing well before the ultimate load, resulting in softening behaviour.

Beam 1 with the regular mesh exhibited a transition between these two failure modes.

The failure loads for Beam 1 and Beam 2 are shown in Tables 6.1, and 6.2 respectively. The failure loads for force control and displacement control tests are similar, given the same mesh type. When the failure mode was a sudden shear failure, the failure load predicted by peridynamics overestimated the true values. When the failure mode was flexural or shear-flexural, the failure load was underestimated.

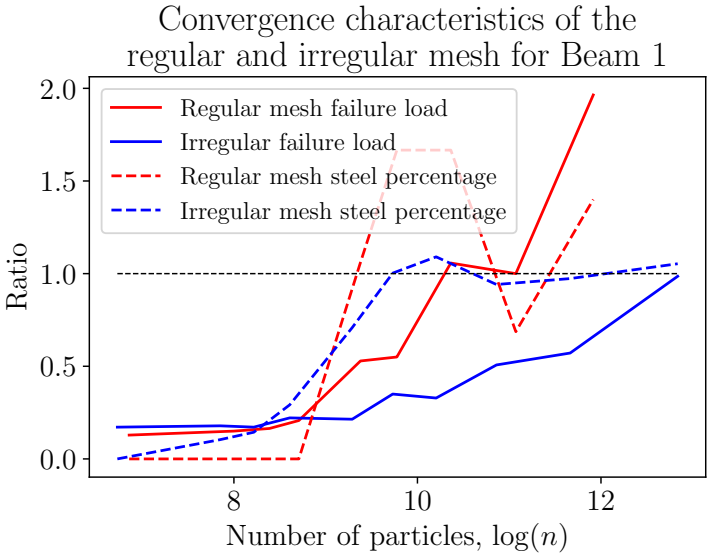


Fig. 6.5 The y axis is a ratio between the steel percentage of the discretised geometry versus the true steel percentage of the continuous geometry (solid line) and the ratio of the failure load to a converged failure load of 70.0kN (dashed line). With increasing problem size, the discretised geometry of the mesh gets becomes more refined, and as a result, the steel percentage, measured in the ratio of steel volume (particles) to total volume (particles) tends towards the true steel percentage of the geometry. Comparing this ratio to the convergence of the failure load, suggests that the steel percentage convergence is correlated with failure load convergence.

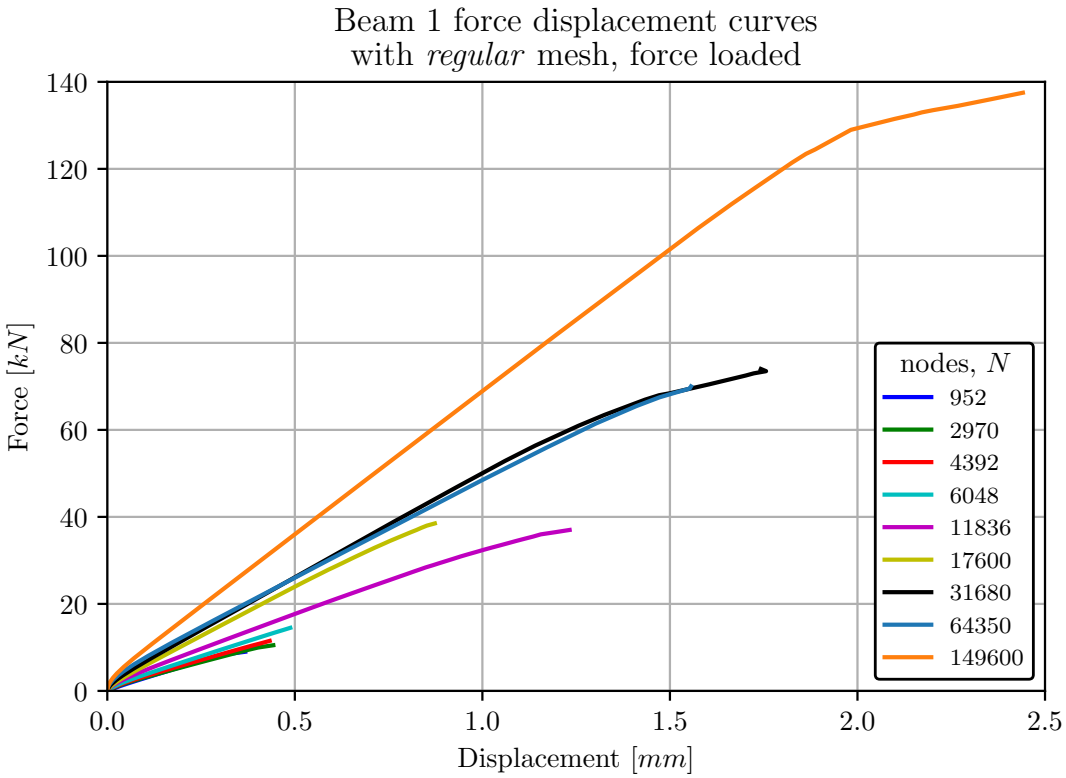


Fig. 6.6 Force controlled test force displacement curve for beam 1, regular mesh.

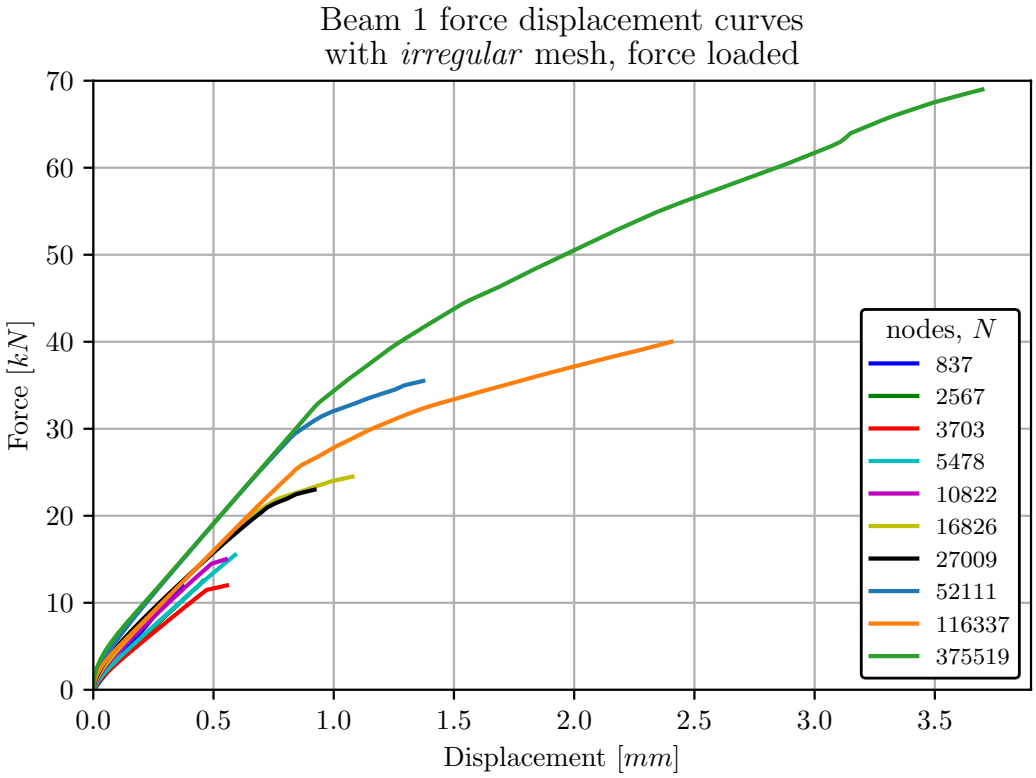


Fig. 6.7 Force controlled test force displacement curve for beam 1, irregular mesh.

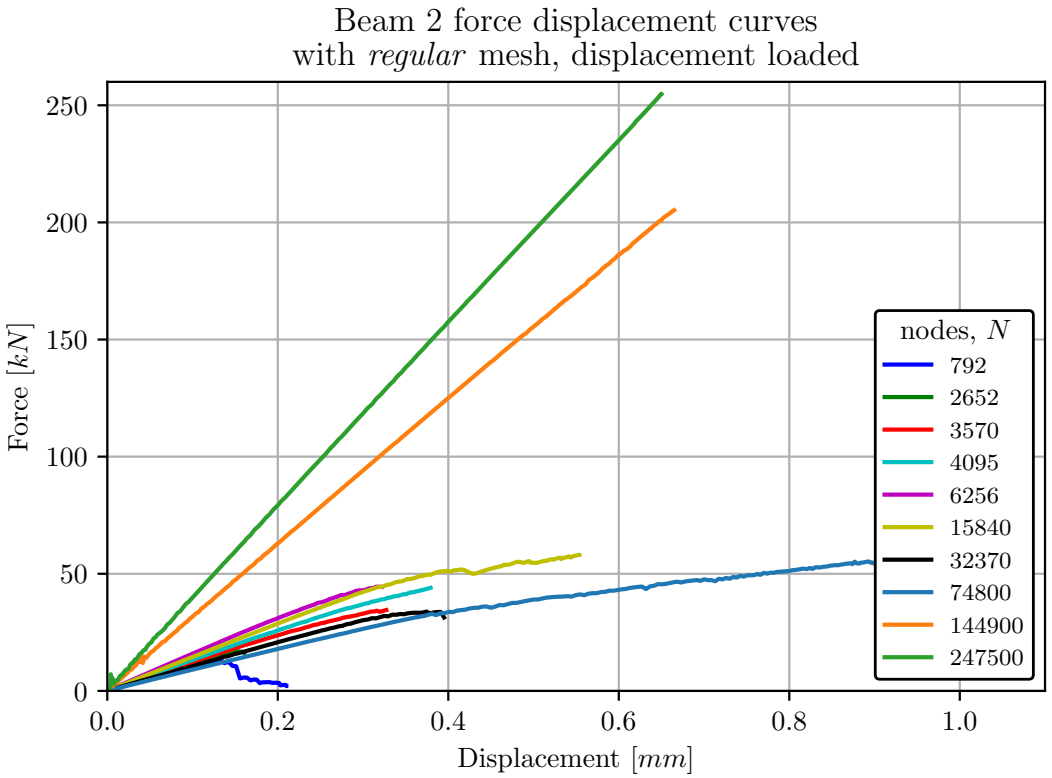


Fig. 6.8 Displacement controlled test force displacement curve for beam 2, regular mesh.

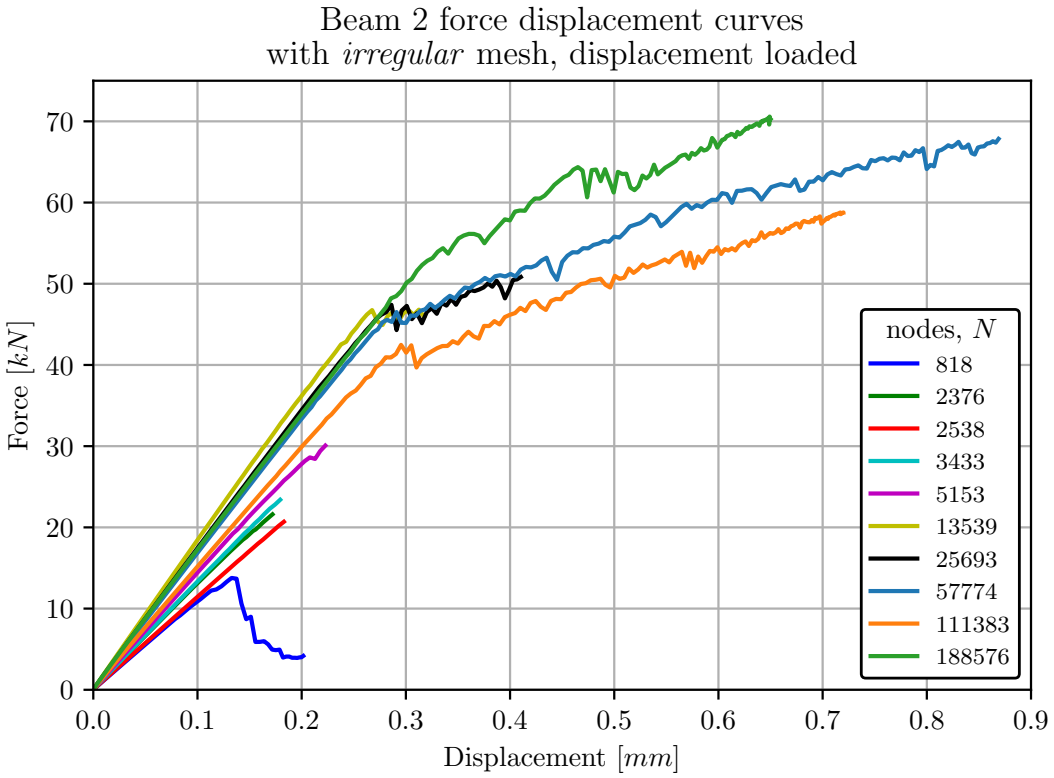


Fig. 6.9 Displacement controlled test force displacement curve for beam 2, irregular mesh.

Table 6.1 Failure loads for Beam 1. *Instabilities caused no convergence on a failure load.

Number of particles	Beam 1, regular mesh	Beam 1, irregular mesh
regular mesh (irregular mesh)	Force controlled [kN]	Force controlled [kN]
952 (837)	9.0	12.0
2970 (2567)	10.5	12.5
4392 (3703)	11.5	12.0
6048 (5478)	14.5	15.5
11836 (10822)	37.0	15.0
17600 (16826)	38.5	24.5
31680 (27009)	74.0	23.0
64350 (52111)	70.0	35.5
149600 (116337)	137.5	40.0
495000 (375519)	*	69.0
Actual failure load $f_{ck} = 31.1MPa$: 174kN		Actual failure load $f_{ck} = 33.5MPa$: 190kN

Table 6.2 Failure loads for Beam 2

Number of particles	Beam 2, regular mesh		Beam 2, irregular mesh	
regular mesh (irregular mesh)	Force loaded [kN]	Displacement loaded [kN]	Force loaded [kN]	Displacement loaded [kN]
792 (818)	14.0	13.4	15.0	13.8
2652 (2376)	18.0	16.1	26.0	21.6
3570 (2538)	35.5	34.5	21.5	20.7
4095 (3433)	44.5	44.0	24.5	23.4
6256 (5153)	46.0	44.4	30.5	30.1
15840 (13539)	59.5	58.1	49.0	45.9
32370 (25693)	36.5	31.4	55.5	50.8
74800 (57774)	60.0	55.3	70.5	67.8
144900 (11383)	206.5	205.2	81.0	58.7
247500 (188576)	246.5	254.9	99.5	70.3
Actual failure load $f_{ck} = 31.1MPa$: 132kN			Actual failure load $f_{ck} = 33.5MPa$: 154kN	

6.2.1 Fracture patterns

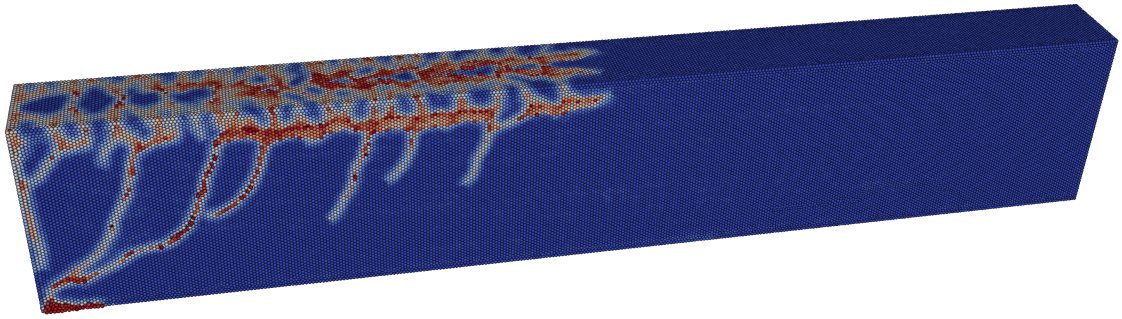
The fracture pattern results confirm that there were two different types of failure mode exhibited, causing either the under estimation or over estimation of failure loads. Figure 6.10 shows the fracture patterns after failure for both the regular and irregular mesh, for the smallest mesh spacing Δx tested on each mesh. The irregular meshes in Figure 6.10a and 6.10c produced shear-flexural failure that lies closer to the boundary of flexural failure than the regular meshes in Figure 6.10b and 6.10d, as can be seen from the longer flexural cracks and the early cracking causing softening shown in the force displacement curve in Figures 6.7 and 6.9.

Contrastingly, in the regular mesh, as the loading approaches the failure load, a critical diagonal shear crack has initiated from a flexural crack and travelled towards the compression zone. The failure mode was closer to 'true' shear in the case of Figure 6.10b, as aforementioned, there is very limited flexural cracking before ultimate shear failure. The crack has also travelled along the steel reinforcement, where it has met resistance, and caused splitting between the concrete and reinforcement, which is typical of shear failure. In this sense the regular mesh has done a better job at predicting the failure mode of the beams.

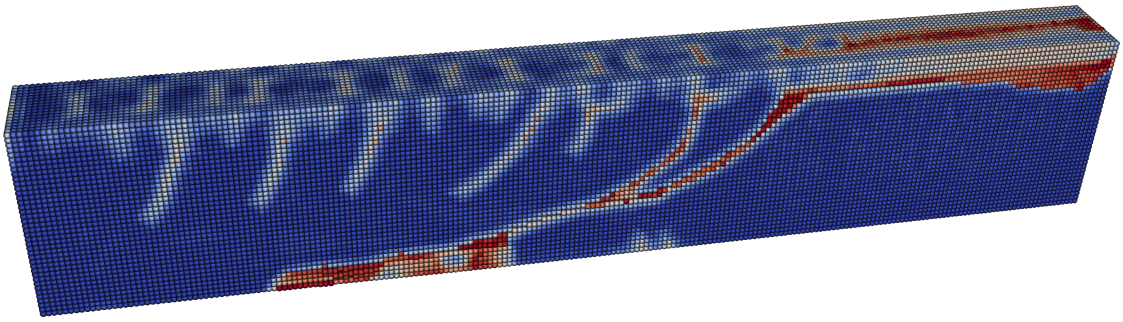
6.3 Stochastic experiments

Figure 6.11a shows the joint and marginal posterior distributions over the deterministic parameters, as described in Section 5.3.2. The parameters are expressed as a ratio of the values used to generate the cracking data, with the true values represented by a red dot. This plot shows the values of bond stiffness and critical stretch that produce similar looking deterministic cracks to the data, those ones with a high probability density. The rejected samples have little to do with physical explanations, but are due to computational instability as the bond stiffness gets large, or boundary effects when bond stiffness or bond critical stretch are low. As a result the distribution does not have a positive tail in the bond stiffness axis, and is not centred around (1.0, 1.0). There is a negative correlation between bond stiffness and critical stretch that produces near identical cracks in the deterministic case, which appears to be non-linear. Further work should be done to investigate the non-linear relationship between parameters that produce similar looking cracks, and to extend this analysis to the Probabilistic Peridynamics, which would require taking a factor of M more samples to marginalise over the crack paths.

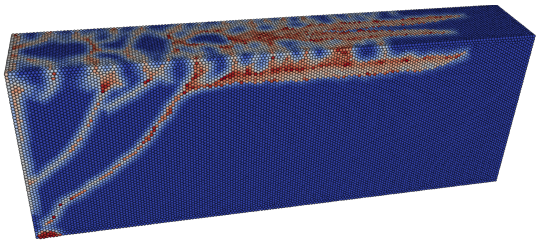
Figure 6.12 shows the negative log likelihood (NLL) surface described in Section 5.3.1. Maximum likelihood estimations of the parameters given the cracking sample data lie at the minimum of this surface, as marked by the red dot, and were $[log(\hat{\sigma}), log(\hat{\ell})] = [-3.9, -5.6]$. The 'true' values used to generate the data were $[log(\sigma), log(\ell)] = [-4.0, -5.0]$. The



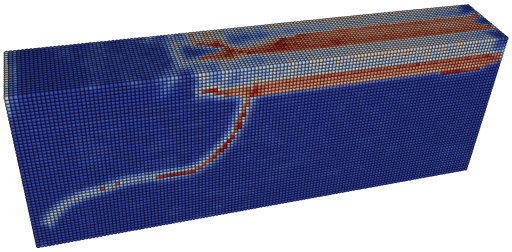
(a) Force controlled test Beam 1, using an irregular mesh with $n = 375519$.



(b) Force controlled test on Beam 1, using a regular mesh with $n = 149500$.

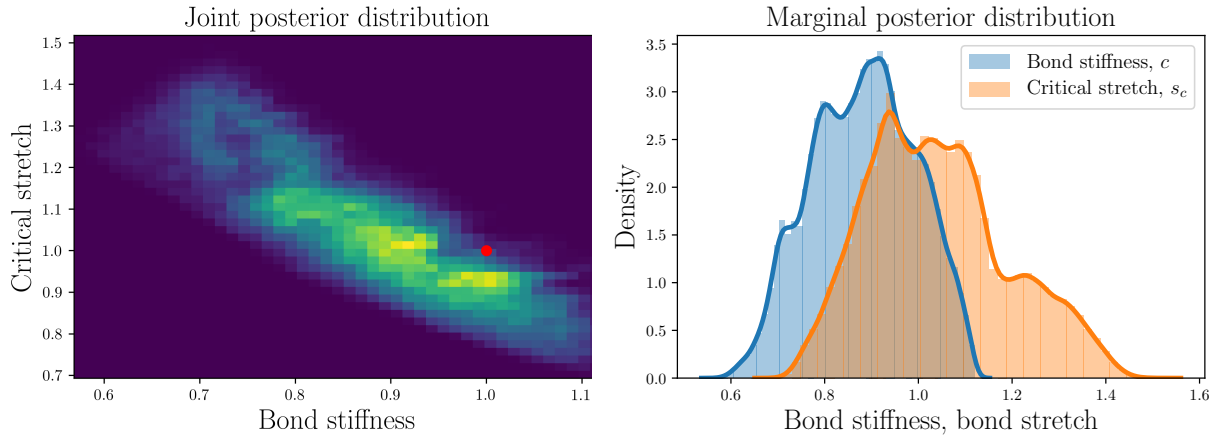


(c) Displacement controlled test on Beam 2, using an irregular mesh with $n = 188576$.



(d) Displacement controlled test on Beam 2, using a regular mesh with $n = 144900$.

Fig. 6.10 Fracture patterns for the concrete beam tests.

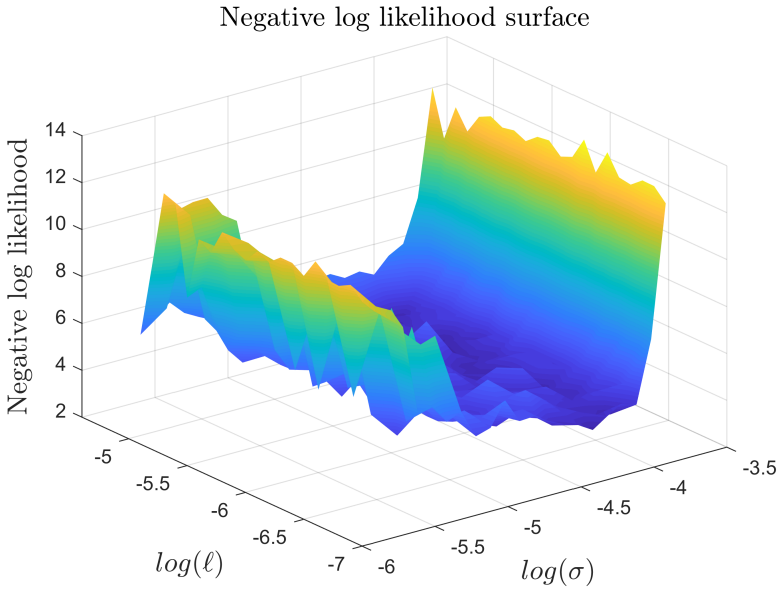


(a) Joint posterior.

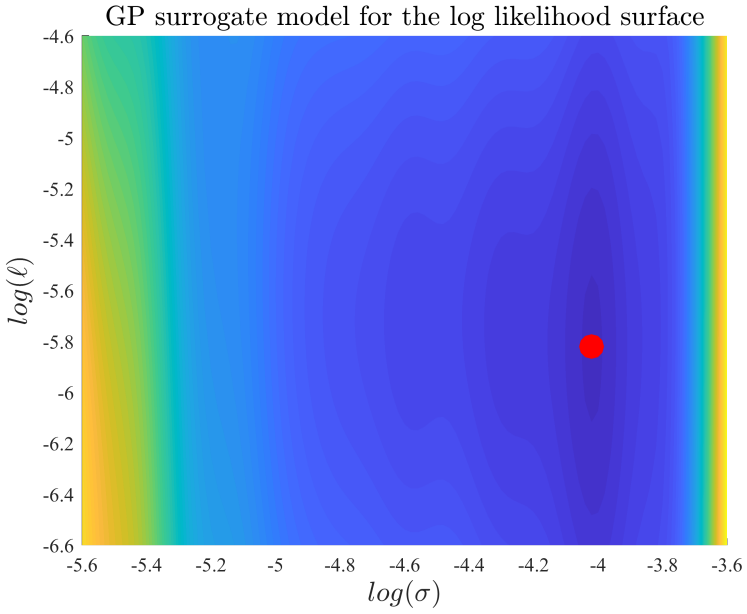
(b) Marginal posterior.

Fig. 6.11 Metropolis was used take 165361 samples of the posterior distribution over deterministic parameters, 'true' values are marked by the red dot.

vertical noise perturbation σ was correctly recovered from the data, whereas values of $-6.6 < log(\ell) < -4.6$ produced similar looking cracks: the cracking damage was independent with respect to the noise length scale, ℓ. It is not shown in Figure 6.12 that the likelihood surface is bounded in the positive $log(\ell)$ direction with low likelihood once ℓ reaches a threshold value, resulting in highly correlated noise over the length of the plate which causes macroscale in vibrations of the plate, and numerical instability. The surface is not bounded in the negative $log(\ell)$ direction which represents uncorrelated brownian motion for large negative values of $log(\ell)$. Note that this stochastic formulation, described in Section 3.1.2 allows for a much more general covariance structure. This intuition behind the the covariance matrix $\mathcal{C}$ is that it is a corrector applied to the forcing $\nabla V(\mathbf{U})$ on the particles. Each value of the covariance matrix represents a correction applied to the force in each bond. It therefore makes sense to apply a covariance structure that is similar to the bond connectivity matrix, with non-zero values only for the bonds which exist. This analysis has been limited to a covariance structure which is parameterised by just one length scale, ℓ.



(a) Negative log likelihood surface with $M = 200$.



(b) Smoothed negative log-likelihood surface using a Gaussian Process. The red dot displays the global minimum.

Fig. 6.12 Negative log likelihood surface for the damage data over parameter space.

Chapter 7

Conclusions and Future work

7.1 Conclusions

The main achievement of this project was producing an easy to install and research friendly peridynamics solver in python, that is capable of modelling composite materials. The following conclusions can be made from this report.

- This work proposed an as yet unexplored way of decreasing the compute time for peridynamics simulations, and has measured a 2-5x increase in speed over similar codes of that type in existing OpenCL GPU literature as a result of utilisation of local GPU memory. This allowed some basic parameter estimation to be performed.
- It was shown that the failure mode of reinforced concrete beams as predicted by peridynamics is dependent on the ratio of steel percentage in the mesh. Since irregular meshes converge to a true steel particle density faster, it is preferred to model reinforced concrete beams with irregular meshes.
- It was shown that by adding Brownian forcing to each particle, that a instead of a deterministic crack path that is reproduced given the same geometry, boundary conditions and simulation parameters, a distribution of crack paths can be sampled from.

7.2 Further work

It is hoped by the author that this versatile, open source solver can be used by the wider peridynamics community for further research. The roadmap for the continued development of the solver, in collaboration with the University of Exeter Data-Centric Engineering group and the Alan Turing Institute, includes a refactoring of the author's GPU code to bring it up to software engineering standards, and implementation of different peridynamic

models, such as the bi-linear damage model discussed in section 2.2. The author plans to continue work on parameter estimation for Probabilistic Peridynamics.

7.2.1 Parameter estimation directly from the displacement states

It has been shown that by reformulating peridynamics as a Stochastic Differential Equation, the stochastic dynamics, an overdamped Langevin diffusion, generates a distribution of crack paths. To fully realise the potential of peridynamics, work should now be done to formally estimate peridynamics parameters from both cracking damage data and displacement data of real life structures. By using a Bayesian approach, it would be possible to obtain a confidence interval in selection of these parameters.

However, the vision of a fully data-centric approach is not complete. While this work has demonstrated that the author's optimised OpenCL peridynamics implementation is fast enough to do parameter estimation, with a sample time of 0.3s for medium sized peridynamics problems ($n = 10^3$), it leaves parameter estimation relatively unexplored. The reason for this is the formulation of Probabilistic Peridynamics which was presented in section 3.1.2 could only be used for direct inference for elastic problems, as shall be explained in the next paragraph. To get around this and to do basic parameter estimation, the damage data was assumed normally distributed around the peridynamics solution. Also, only the stochastic parameters $\boldsymbol{\theta} = \{\sigma, \ell\}$ were estimated using maximum likelihood. This was because an expensive grid search over the parameters had to be performed, since the gradient of the log-likelihood surface couldn't be calculated analytically. Adding parameters to optimise over increases the dimensions of the search space, and compounds the search time.

For elastic problems, the potential energy function, $V(\mathbf{U})$ is reversible. This means that there is a unique mapping from $\mathbf{U} \rightarrow V(\mathbf{U})$, and given a smooth approximation to equation 2.10 and the additional constraining term in the potential energy Equation 3.9[1] [40], the forward crack distribution is well defined. However, once the material starts to break, the point of interest, then the potential energy $V(\mathbf{U})$ becomes dependent on its history. In this case the the analysis becomes much more complex: the formulation is no longer a Markov process.

By introducing a new continuous variable that describes the damage, which is parametrised by new variables that would also have to be inferred, it can be shown that there is a unique mapping between the state variables and energy. Therefore, the cracking process is Markov, as the next state depends only on the current state. The update probability follows on from equation 3.10 as

[1] If the reader is interested about the mathematical formulation of this model, they should refer to L. Ellam, M. Girolami et al. [40] and further reading [46, 47].

$$p(\mathbf{U}_t|\mathbf{U}_{1:t-1}) = p(\mathbf{U}|\mathbf{U}_{t-1}) \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}) \tag{7.1}$$

where the mean vector is

$$\boldsymbol{\mu}_t = \mathbf{U}_{t-1} - \eta^{-1}\mathcal{C}\nabla V(\mathbf{U}_{t-1})\Delta t \tag{7.2}$$

and the covariance matrix is

$$\Sigma = \sigma^2\sqrt{2\mathcal{C}}. \tag{7.3}$$

Then, estimations of Probabilistic and deterministic Peridynamics parameters could be made using the log-likelihood of the displacement states $\mathbf{U}_{1:T}$ over time. The prediction error decomposition used to calculate the log-likelihood is written out below.

$$\log p_{\boldsymbol{\theta}}(\mathbf{U}_t|\mathbf{U}_{t-1}) = -\frac{n}{2}log(2\pi) - \frac{1}{2}\log|\Sigma^{-1}| - \frac{1}{2}(\mathbf{U}_t - \boldsymbol{\mu}_t)^T\Sigma^{-1}(\mathbf{U}_t - \boldsymbol{\mu}_t) \tag{7.4}$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{U}_{1:T}) = \sum_{t=1}^{T}\log p_{\boldsymbol{\theta}}(\mathbf{U}_t|\mathbf{U}_{t-1}) \tag{7.5}$$

The result would be extraordinary: given enough computing power, and the technology to measure the displacement states over time of discretised points in a concrete beam over the progress of its cracking and failure, a maximum likelihood estimate the Probabilistic Peridynamics parameters that best describe the displacement states for continuous and discontinuous displacement fields could be made.

References

[1] J. Orr, A. Bras, and T. Ibell. Effectiveness of design codes for life cycle energy optimisation. *Energy and Buildings*, pages 61–67, 2017.

[2] J. Orr, A. Darby, T. Ibell, M. Evernden, and M. Otlet. Concrete structures using fabric formwork. *The Structural Engineer*, pages 20–26, 2011.

[3] C.Diyaroglu, E.Oterkus, S.Oterkus, and E.Madenci. Peridynamics for bending of beams and plates with transverse shear deformation. *International Journal of Solids and Structures*, 69-70:152–168, 2015.

[4] T. Belytschko and T. Black. Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45:601–620, 1999.

[5] N. Moës, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46:131–150, 1999.

[6] J.V. Cox. An extended finite element method with analytical enrichment for cohesive crack modeling. *International Journal for Numerical Methods in Engineering*, 78:48–83, 2009.

[7] S. A. Silling. Reformulation of elasticity theory for discontinuities and long-range forces. *Journal of the Mechanics and Physics of Solids*, 48:175–209, 2000.

[8] A. Agwai, I. Guven, and E. Madenci. Predicting crack propagation with peridynamics: a comparative study. *International Journal of Fracture*, 171:65–78, 2011.

[9] J. Lee, S.E. Oh, and J.W. Hong. Parallel programming of a peridynamics code coupled with finite element method. *International Journal of Fracture*, 203:99–114, 2016.

[10] R.A. Wildman and G.A. Gazonas. A finite difference-augmented peridynamics method for reducing wave dispersion. *International Journal of Fracture*, 190:39–52, 2014.

[11] F. Bobaru, J.T.Foster, P.H.Geubelle, and S.A.Silling. *Handbook of peridynamic modeling*. CRC press, 2016.

[12] F. Mossaiby, R. Rossi, P. Dadvand, and S. Idelsohn. OpenCL-based implementation of an unstructured edge-based finite element convection-diffusion solver on graphics hardware. *Internat. J. Numer. Methods Engrg.*, 89:1635–1651, 2012.

[13] P. M. Ness. Modelling the Failure Behaviour of Concrete Structures Using Peridynamics. Master's thesis, University of Cambridge, 2019.

[14] H. David Miranda, J. Orr, and C. Williams. Fast interaction functions for bond-based peridynamics. *European Journal of Computational Mechanics*, 2018.

[15] M.L. Parks, D.J. Littlewood, J.A. Mitchell, and S.A. Silling. *Peridigm Users' Guide, Tech. Report SAND2012-7800*. Sandia National Laboratories, 2012.

[16] M.L. Parks, P. Seleson, S.J. Plimpton, R.B. Lehoucq, and Stewart A. Silling. *Peridynamics with LAMMPS: A User Guide*. Dept. of Scientific Computing, 400 Dirac Science Library, Florida State University, Tallahassee, FL 32306-4120, 2008.

[17] Martin Rädel. *Peridigm Installation Guide*. DLR German Aerospace Center, Composite Structures and Adaptive Systems, Structural Mechanics, Braunschweig, 2019.

[18] Patrick Diehl, Prashant K. Jha, Hartmut Kaiser, Robert Lipton, and Martin Levesque. An asynchronous and task-based implementation of Peridynamics utilizing HPX – the C++ standard library for parallelism and concurrency. 2020.

[19] P. Diehl. Implementierung eines peridynamik –verfahrens auf GPU. Master's thesis, University of Stuttgart, 2012.

[20] P. Diehl and M.A. Schweitzer. *Efficient neighbor search for particle methods on GPUs in: M. Griebel, M.A. Schweitzer (Eds.), Meshfree Methods for Partial Differential Equations VII*. Master's thesis, Springer International Publishing, 2015.

[21] Q. Le, W. Chan, and J. Schwartz. A two-dimensional ordinary, state-based peridynamic model for linearly elastic solids. *Internat. J. Numer. Methods Engrg.*, 98:547–561, 2014.

[22] W. Liu and J.-W. Hong. Discretized peridynamics for brittle and ductile solids. *Internat. J. Numer. Methods Engrg.*, 89(8):1028–1046, 2012.

[23] G. Zhang and F. Bobaru. Modeling the evolution of fatigue failure with peridynamics. *Rom. J. Tech. Sci. Appl. Mech.*, 61(1):20–39, 2016.

[24] F. Mossaiby, A. Shojaei, M. Zaccariotto, and U. Galvanetto. OpenCL implementation of a high performance 3D Peridynamic model on graphics accelerators. *Computers and Mathematics with Applications*, 1(74):1856–1870, 2017.

[25] S. A. Silling and E. Askar. A meshfree method based on the peridynamic model of solid mechanics. *Computers & Structures*, 83:1526–1535, 2005.

[26] W. Liu and J.W.Hong. Discretized peridynamics for linear elastic solids. *Computational Mechanics*, 50:579–590, 2012.

[27] S. A. Silling, M. Epton, O. Weckner, J. Xu, and E. Askari. Peridynamic States and Constitutive Modeling. *Journal of Elasticity*, 88:151–184, 2007.

[28] W. Gerstle and N. Sau. Peridynamic modelling of concrete structures. *Proceedings of the Fifth International Conference on Fracture Mechanics of Concrete Structures, Ia-FRAMCOS*, 162:949–956, 2010.

[29] Y. Ha and F. Bobaru. Studies of dynamic crack propagation and crack branching with peridynamics. *International Journal of Fracture*, 162:949–956, 2010.

[30] Y. Ha and F. Bobaru. Characteristics of dynamic brittle fracture captured with peridynamics. *Engineering Fracture Mechanics*, 78:1156–1168, 2011.

[31] A. M. Neville. *Properties of concrete*, volume 5th edition ed. Pearson Education Limited, London, 2011.

[32] D. Yang, W. Dong, X. Liu, S. Yi, and X. He. Investigation on mode-I crack propagation in concrete using bond-based peridynamics with a new damage model. *Engineering Fracture Mechanics*, 199:567–581, 2018.

[33] Cambridge University Engineering Department. *Materials Data Book*. 2017.

[34] W. Gerstle, N. Sau, and S. Silling. Peridynamic Modeling of Plain and Reinforced Concrete Structures. *New Methods in Computational Mechanics, IASMiRT, Beijing*, 2005.

[35] W. Liu and J.-W. Hong. A coupling approach of discretized peridynamics with finite element method. *Computer Methods in Applied Mechanics and Engineering*, 245-246:163–175, 2012.

[36] Z. Bažant and M. T. Kazemi. Size Effect in Fracture of Ceramics and Its Use To Determine Fracture Energy and Effective Process Zone Length. *Journal of the American Ceramic Society*, 73:1841–1853, 1990.

[37] Q. V. Le and F. Bobaru. Surface corrections for peridynamic models in elasticity and fracture. *Computational Mechanics*, page 499–518, 2018.

[38] W. Gerstle, N. Sakhavand, and S. Chapman. Peridynamic and continuum models of reinforced concrete lap splice compared, in: Recent Advances. *Fracture Mechanics of Concrete, Korea Concrete Institute, Seoul*, 2010.

[39] D.J.Littlewood, T. Shelton, and J. D. Thomas. *Estimation of the critical time step for peridynamic models, Technical report*. Sandia National Lab.(SNL-NM), Albuquerque, NM, United States, 2013.

[40] L. Ellam, M. Girolami, G. A. Pavliotis, and A. Wilson. Stochastic modelling of urban structure. *Proc. R. Soc. A*, 474, 2018.

[41] S. F. Henke and S. Shanbhag. Mesh sensitivity in peridynamic simulations. *Compute Physics Communications*, 2013.

[42] P. Seleson. Improved one-point quadrature algorithms for two-dimensional peridynamic models based on analytical calculations. *Computer Methods in Applied Mechanics and Engineering*, 282:184–217, 2014.

[43] M. Parks, R.B. Lehoucq, S.J. Plimpton, and S.A. Silling. Implementing peridynamics within a molecular dynamics code. *Computer Physics Communications*, 179(11):777–783, 2008.

[44] J. Trageser and P Seleson. Bond-Based Peridynamics: a Tale of Two Poisson's Ratios. *Journal of Peridynamics and Nonlocal Modeling*, 2020.

[45] E. Madenci and E. Oterkus. *Peridynamic Theory and Its Applications*. Springer, 2014.

[46] A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC Texts, 1995.

[47] J. Kaipio and E. Somersalo. *Statistical and computational inverse problems*, volume 160. Springer Science & Business Media., Berlin, Germany, 2006.

Appendix A

Appendix

A.1 Vectorisation of neighbour distance

Define the column vector containing the x coordinates of the particles in the body $\mathcal{R}$ as $\mathbf{y}_x = [y_{1,x}, y_{2,x}...y_{N,x}]^T \in \mathbb{R}^{n\times 1}$, similarly, define the column vector containing the y coordinates and z coordinates as $\mathbf{y}_y$ and $\mathbf{y}_z$ respectively. The peridynamics solver can be vectorised by defining the sparse connectivity matrix,

$$\begin{aligned} A_{ij} &= 1 \quad \text{for} \quad i > j, i \in \mathcal{H}_j \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{A.1}$$

which is lower triangular, equivalent to a verlet list representation of the bond connectivity. Calculate $\mathbf{Y}_x$, the nodal coordinate column vectors tiled N times

$$\mathbf{Y}_x = [\mathbf{y}_x, \mathbf{y}_x, ..., \mathbf{y}_x] \in \mathbb{R}^{n\times n}. \tag{A.2}$$

$\mathbf{Y}_y$ and $\mathbf{Y}_z$ are calculated in a similar way.

The Cartesian components of bond lengths in the x direction $\mathbf{\Lambda}_x$ are calculated by element-wise multiplication with the sparse bond connectivity matrix

$$\mathbf{\Lambda}_x = \mathbf{Y}_x \odot \boldsymbol{A} - \mathbf{Y}_x^T \odot \boldsymbol{A} \tag{A.3}$$

$\mathbf{\Lambda}y$ and $\mathbf{\Lambda}_z$ are calculated in a similar way. The Euclidean norms of all the bonds in the body $\mathcal{R}$ are then found from a simple elementwise operation,

$$\boldsymbol{L} \odot \boldsymbol{L} = \mathbf{\Lambda}_x \odot \mathbf{\Lambda}_x + \mathbf{\Lambda}_y \odot \mathbf{\Lambda}_y + \mathbf{\Lambda}_z \odot \mathbf{\Lambda}_z \tag{A.4}$$

giving $\ell_{ij} = \ell_{ji} = (L_{ij})_{j>i\in N}$. $\ell_{0,ij}$ is calculated in a similar way. The calculation of bond stretch and force density contributions of the bonds is then a simple element-wise addition, division and multiplication. The final particle force density is a reduction (sum) across the rows of the bond force density contribution matrix from $\mathbb{R}^{n\times n}$ to $\mathbb{R}^{n\times 1}$.

A.2 OpenCL kernels

```
// Calculate bond forces using the displacments, Un
__kernel void
CalcBondForce1(Udn, Un, Coords, Vols, Stiffnesses, Connectivity, BCValues){
const int i = get_global_id(0); // Get particle index
double f_x, f_y, f_z = 0.00; // Initialise the node force sum
if (i < NUMBER_OF_NODES){
  for (int j = 0; j < MAX_HORIZON_LENGTH; j++){ // Inner loop over the
    bonds
      const int n = Connectivity[MAX_HORIZON_LENGTH * i + j];
      if (n != -1){ // If the bond hasn't been broken
        // Original bond lengths, xi_y and xi_z calculated similarly
        const double xi_x = Nodes[3 * n + 0] - Nodes[3 * i + 0];
        // Current bond lengths, xi_eta_y and xi_eta_z calculated similarly
        const double xi_eta_x = Un[3 * n + 0] - Un[3 * i + 0] + xi_x;
        // Calculate bond strain
        const double xi = sqrt(xi_x * xi_x + xi_y * xi_y + xi_z * xi_z);
        const double y = sqrt(
        xi_eta_x * xi_eta_x + xi_eta_y * xi_eta_y + xi_eta_z * xi_eta_z);
        const double y_xi = (y - xi);
        // Projections of bond on to coordinate axis
        const double cx = xi_eta_x / y; // similarly for cy and cz
        // Calculate the scalar valued pairwise force function
        const double _E = Stiffnesses[MAX_HORIZON_LENGTH * i + j];
        const double _A = Vols[n];
        const double _L = xi;
        const double _EAL = _E * _A / _L;
        // Add the bond force contribution vector to the nodal force vector
        f_x += _EAL * cx * y_xi; // similarly for f_y and f_z
      }
    }
  // Update particle velocities, add boundary conditions
  Udn[3 * i + 0] = f_x + BCValues[3 * i + 0]; // similarly for y and z
}
}
```

Listing A.1 CalcBondForce1: Extended code based on OpenCL kernels by Mossaiby et al. (2017)

```
__kernel void
CalcBondForce3(Un, Udn, Vols, Horizons, Coords, Stiffnesses, FailStretches,
    BCValues,
__local double * local_cache_x,
__local double * local_cache_y,
__local double * local_cache_z){
  // Get bond number (1D index space over bonds)
```

```
7   const int global_id = get_global_id(0);
8   // local_id is the LOCAL node id in range [0, local_size] in the family
9   const int local_id = get_local_id(0);
10  // local_size is the MAX_HORIZONS_LENGTHS, usually 128 or 256
11  const int local_size = get_local_size(0);
12  if ((global_id < (PD_NODE_NO * local_size)) && (local_id >= 0) && (
     local_id < local_size)){
13    // Find corresponding node id
14    const double temp = global_id / local_size;
15    const int node_id_i = floor(temp);
16    // Access local node within node_id_i's horizon with corresponding
     node_id_j,
17    const int node_id_j = Connectivity[global_id];
18    if (node_id_j != -1){ // If bond is not broken
19      return const double xi_x, xi_y, xi_z, xi_eta_x, xi_eta_y, xi_eta_y,
     xi, y, y_xi, cx, cy, cz, _EAL // as in 'CalcBondForce1'
20      // Copy bond forces into local memory
21      local_cache_x[local_id] = _EAL * cx * y_xi; // similarly for y and z
22      // Check for state of bonds here, and break it if necessary
23      return const double PD_S0 , s; // as before
24      if (s > PD_S0){Connectivity[global_id] = -1;}  // Break the bond
25    }
26    else{ // bond is broken
27      local_cache_x[local_id] = 0.00; // similarly for y and z
28    }
29  // Wait for all threads to catch up
30  barrier(CLK_LOCAL_MEM_FENCE);
31  // Parallel reduction of the bond force onto node force
32  for (int i = local_size/2; i > 0; i /= 2){
33    if(local_id < i){
34      local_cache_x[local_id] += local_cache_x[local_id + i]; // s.f. y, z
35    }
36    //Wait for all threads to catch up
37    barrier(CLK_LOCAL_MEM_FENCE);
38  }
39  if (!local_id) {
40    //Get the reduced forces
41    int node_no = global_id/local_size;
42    // Update Velocties in each direction
43    Udn[3 * node_no + 0] = local_cache_x[0] + FORCE_LOAD_SCALE * BCValues[3
     * node_no + 0]; // similarly for y and z
44    }
45  }
46 }
```

Listing A.2 CalcBondForce3: optimised using local memory.

Appendix B

Risk assessment retrospective

The risks identifed at the start of this project were related to extensive computer use: repetitive strain injury, eye strain and sitting for long periods of time. In retrospect this was an accurate assessment as the project consisted of writing and running software on a computer. No lab work was performed by the author during the completion of this project. There exists ergonomic mice and keyboards should the risks need to be mitigating further. The COVID-19 pandemic did not significantly affect the outcome of this project or the risks exposed to the author during the completion of this project.