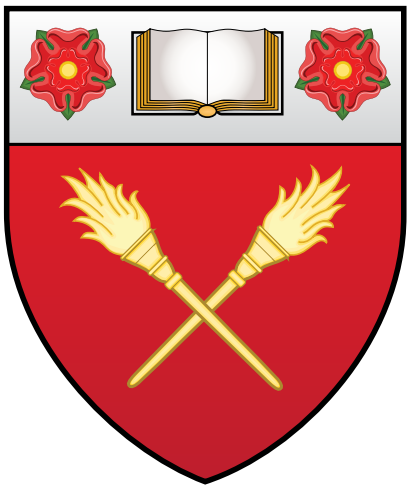
Dislocation Based Modelling of Fusion Relevant Materials



Daniel Celis Garza

University of Oxford

Harris-Manchester College

Department of Materials

Supervisors: Edmund Tarleton & Angus Wilkinson

A thesis submitted for the degree of

Doctor in Philosophy

Hilary Term 2021

Dedication

"Ohana significa familia, y tu familia nunca te abandona, ni te olvida."
"Ohana means family, family means nobody gets left behind, or forgotten."
– Stitch, Lilo & Stitch

Malle santa, cuando te hablé desde el trabajo para darte la noticia me dijiste que siempre lo supiste. Recuerdo que lloraste en el teléfono, se te cerró la garganta y a mí también. A diferencia de otras veces, no pudimos platicar mucho porque nos quedamos sin aliento. Así que le hablaste a abuelita Raquel mientras le hablé a papá. Madre, David y yo te debemos tanto que no podemos repagar en mil vidas, pero ten por seguro que eres nuestro ejemplo a seguir. Nuestra madre chingona y chambeadora, solo hay una y como ella no hay ninguna. Sin tí el mundo sería un lugar más cruel y pobre, no merece un ángel tan grande y puro como tú.

Dad, "Igualito que tu jefe, wey." con tu risa característica fue lo primero que dijiste cuando te avisé "Sí dad, igualito que mi jefe." entre risas respondí. Como siempre, no platicamos mucho pero esta vez porque le querías avisar a abuelita Teté y a mis tíos, y yo le quería avisar a David. Pensé que seguirías aquí para carcajearte al verme en la túnica ridícula, así como lo hicimos nosotros cuando te vimos a tí en la tuya. Escribo esta dedicatoria antes de acabar porque una promesa es una promesa y esta madre la voy a acabar. No te tendré para pedirte consejos y contarte mis avances y tropiezos. Pero a veces pretendo que me escuchas mientras intento comprender o arreglar algo. Te queremos y te extrañamos muchísimo. Buenas noches, dad.

David, "Te mamaste we." me dijiste cuando me abrazaste al llegar a casa después del trabajo el día que me aceptaron. Me has hecho un chingo de falta ojete, te extraño mucho we. Perdón por no hablar tan seguido, pero me duele colgar we. No estoy tan chisqueado como mamá pero siento feo cuando terminamos de platicar. Me gusta mucho ver tus streams porque me recuerda un poco a sentarme a tu lado a verte jugar… o a cuando veíamos Twitch juntos y veíamos a TB "missing legal".

Esto es para mi Ohana por sangre y por elección. Ustedes creyeron en mí cuando yo no lo hacía. Me empujaron a ser mejor. Me extendieron la mano cuando nadie más lo hizo. Me hicieron reir cuando solo sabía llorar. Gracias por hacerme quien soy.

This is for my Ohana by blood and by choice. You believed in me when I did not. You pushed me to be better. Offered me a helping hand when nobody would. Made me laugh when all I knew was sorrow. Thank you for making me who I am.

Acknowledgements

"You can always judge a man by the quality of his enemies."
– Oscar Wilde

I did not get here alone. Truly I don't have much idea what I did to deserve the help and encouragement of such an eclectic mix of incredible people. The list is long, but like the proverbial beating of a butterfly's wings, there is no telling where I'd be without the aid and support of these people.

- Ed and Angus, I have no idea what I did to deserve your guidance and support.
- Fusion CDT
- Harris-Manchester
- Anatoly Kolomeisky, John F. Stanton
- Isla
- Clan del Rano Sentado
- Goosehood
- Kopse Lane Krump v1.0 and v2.0
- Kristmas Karnage: # n
- OUPLC: Perhaps the heaviest things that we lift are not our weights, but our feels.
- Damian Rouson
- Bruce Bromage, Haiyang Yu, Fenxian Liu, Daniel Hortelano-Roig, Junnan Jiang

I guess I'll content myself with being judged unfavourably by the world.

Contents

List of Figures ii

List of Tables iii

List of Algorithms v

0 Preface 1

0.1 Notation and abbreviation . . . 1
0.2 Typesetting . . . 2
0.3 Diagrams . . . 2

1 Introduction 3

1.1 Materials Science and History . . . 3
1.2 Fusion Energy Production . . . 4
1.2.1 Operating environment . . . 4
1.2.2 Effects of Radiation on Reactor Materials . . . 6
1.2.2.1 Radiation Damage . . . 6
1.2.2.2 Transmutation . . . 7
1.3 Parallel Computing . . . 8
1.4 Computation on Graphics Processing Units . . . 10
1.4.1 Hurdles for Parallelisation . . . 12
1.5 3D Dislocation Dynamics Modelling . . . 15
1.5.1 Coupling Dislocation Dynamics to Finite Element Methods 16
1.5.1.1 Superposition Model . . . 16
1.5.1.2 Discrete-Continuum Model . . . 17
1.5.1.3 Level Set Method . . . 19
1.5.2 Non-Singular Continuum Theory of Dislocations . . . 21
1.5.3 Analytical Forces Exerted by a Dislocation Line Segment on Surface Elements . . . 24
1.5.4 Multiphase Simulations . . . 26
1.5.4.1 Polycrystalline Materials . . . 27

1.5.4.2 Inclusions 30
1.5.5 Parallelising Discrete Dislocation Dynamics 32
1.6 Project Outline and New Science 34

2 EasyDD v2.0 **37**
2.1 Bug fixes 37
2.1.1 Correct time-adaptive integrator 37
2.1.2 Matrix conditioning 40
2.2 Research software engineering 44
2.2.1 Organisation 46
2.2.1.1 Source control 46
2.2.1.2 Folder structure 46
2.2.2 Modularisation 47
2.2.2.1 Encapsulation 47
2.2.2.2 Generic functions 48
2.2.3 Optimisation 50
2.2.3.1 Spurious memory allocation 50
2.2.3.2 Finite element optimisation 50
2.2.4 Misc quality of life improvements 51
2.2.5 Summary 52

3 Topology Improvements **53**
3.1 Hinge collision prioritisation 53
3.2 Collision distance prioritisation 53
3.3 Collision-separation Loop 53
3.4 Remeshing 53

4 Dislocation-induced surface tractions **55**

5 Future work **57**

List of Figures

1.1 Hohlraum design for indirect drive in Inertial Confinement Fusion. 6
1.2 Energy expenditure of CPU vs Voltage. 9
1.3 CUDA runtime schematic. 10
1.4 Memory access pattern example. 13
1.5 GPU and CPU asynchronous execution. 14
1.6 Explanation of warp divergence. 14
1.7 Superposition Model for DDD-FEM coupling. 16
1.8 The eigenstrain formalism. 18
1.9 Level set Dislocation Dynamics. 20
1.10 Diagram of the analytical force calculation on linear rectangular surface elements. 26
1.11 Modelling twinned multicrystals with DDD. 29
1.12 Modelling dislocation-inclusion interactions with the Discrete-Continuum Model. 31
1.13 Parallelisation strategies for three problems in 3D DDD. 33

2.1 Node movement along dislocation line should have no drag contribution. 41

List of Tables

1.1 Estimated operating conditions of MCF and ICF fusion reactors. 5

List of Algorithms

2.1 Adaptive Euler-trapezoid predictor-corrector algorithm. 38
2.2 Improved adaptive timestep algorithm. 39
2.3 Avoiding singular matrix by making **B** extremely wrong and hoping the integrator error bounds pick it up and the timestep is decreased. 42
2.4 Improved regularisation of **B** by way of perturbing the diagonal. . 44

Chapter 0

Preface

0.1 Notation and abbreviation

For the sake of clarity the following notation and abbreviation conventions have been used:

- GPU: Graphics Processing Unit.
- CPU: Central Processing Unit.
- Tensors are denoted by sans serif bold italics, $\boldsymbol{T}$.
- Matrices are denoted by serif bold roman, $\mathbf{M}$.
- Vectors are denoted by serif bold italics, $\boldsymbol{V}$.
- Scalars are denoted by serif italics, S.
- Operators and special functions are roman, $\exp(x)$, $\det \mathbf{J}$.
- Angles are in radians.
- Individual items, be they nodes, surface elements, dislocation segments or indices are denoted by a lower-case subscript to the right, a_n.
- Ensembles are denoted by an upper-case subscript to the right, $a_N \coloneqq \sum a_n$.
- Host variables (CPU variables in parallelisation) are denoted by a roman h-superscript on the left, ${}^{\mathrm{h}}a$.
- Global device variables (global variables visible only to the GPU) are denoted by a roman d-superscript on the left, ${}^{\mathrm{d}}a$.
- Thread variables (variables visible only to a GPU thread) are denoted by a roman t-superscript on the left, ${}^{\mathrm{t}}a$.

- Serial indices/counters are the traditional i, j, k….
- Parallel indices are denoted as roman variables, a.
- Pseudo-code is C-style—row-major order, 0-based indexing—because it provides a more direct translation of the C-code.

0.2 Typesetting

The repository https://github.com/dcelisgarza/latex_templates contains the custom document class used to typeset this document. It was compiled with XeLaTeX and BibTeX. We recommend compiling the source with XeLaTeX or LuaLaTeX. Compilation requires `minted`, https://ctan.org/pkg/minted?lang=en, and its dependencies.

0.3 Diagrams

All diagrams were drawn with Inkscape: Open Source Scalable Vector Graphics Editor, https://inkscape.org/, "Draw Freely."

Chapter 1

Introduction

1.1 Materials Science and History

Metals and alloys are of such importance to humankind that entire eras of our history have been defined by and named after discoveries and advances in metallurgy [1, 2]. A civilisation's ability to gain mastery of the properties and usage of materials is often strongly correlated with its success in history [3]. The historical influence of metals and alloys has touched all areas of human existence, from fashion to warfare [4].

It should come as no surprise that a material's use is often linked to its properties. When a civilisation learned to harness the properties of a novel material, its influence often grew as a result [5, 6]. Whether building a woodcutter's axe or a fusion reactor, the mechanical behaviour and deformation of materials—metals and alloys in particular—are still among their most important properties for engineering applications.

Traditionally such properties have been largely studied via empirical methods such as tensile and compressive tests, beam bending, and indentation [7–9]. Such tests have mainly focused on macroscopic scales, thus offering bulk-averaged properties valuable for engineers. Unfortunately such tests only describe observed behaviours without truly elucidating the fundamental mechanisms behind them [10]. This has recently started to change with the advent of micro-scale testing, where tests can be performed on single crystals or individual crystal boundaries, and can even probe specific slip systems to see how they behave under various loading scenarios [11, 12]. The increased resolution and more thorough parameter control goes a long way in providing a window through which more fundamental behaviours can be observed and hopefully explained; thus providing a way of deconvoluting macroscopic observations into their constituent phenomena [13, 14].

However, understanding the underlying mechanisms behind empirical results

through experimental means has proven difficult at every scale [15, 16]. Fortunately modelling and simulation can be powerful allies in this task. In particular, the study of dislocations is experimentally challenging [17–19]. By virtue of being such an important player in the mechanical properties of materials, dislocations demand careful study from both empirical and in-sillico avenues.

The need for a more detailed picture of material properties is especially relevant in highly engineered alloys subjected to extreme conditions, such as those found in fusion reactors. The time-dependent shift in material properties and composition is of the utmost concern such extreme environments; radiation damage, large temperature gradients, gas diffusion and plasma instabilities provide the means for properties to change drastically over a components' lifetime [20–22]. It is therefore imperative that we find ways of predicting and accounting for these changes. This requires a deeper understanding of the process driving the mechanical behaviour. Virtual experiments allow analysis and interpretation of micromechanical tests and can help provide new fundamental insight.

1.2 Fusion Energy Production

There currently exist two major branches of research for fusion energy production [23]: *1*) magnetic confinement fusion (MCF) [24] which confines the plasma using magnetic fields and *2*) inertial confinement fusion (ICF) [25] which uses a frozen fuel pellet which is either directly or indirectly compressed by arrays of powerful lasers. The physics and engineering challenges vary greatly between both approaches but the fundamental materials problems remain largely the same, with few a exceptions such as divertors [26, 27].

1.2.1 Operating environment

One constant feature of nuclear energy production is ionising and non-ionising radiation. In the case of fission, this is mostly in the form of low energy neutrons and residual radiation of fission products; fusion on the other hand, deals with the sparsely explored 14 MeV neutron spectrum [28]. The lack of appropriate sources of suitably energetic neutrons [29] has meant that modelling, and searching for experimental analogues of damage cascades have become a crucial part of materials research [30].

Operating environments change vastly between ICF and MCF as described by table 1.1 [31]. The table should be read with a measure of skepticism as the numbers and conditions are approximate because they are not yet fully known, especially for ICF. However it is still, a fairly reasonable first approximation into

Location	Radiation Type	MCF (ITER)	ICF (LMJ)
1st Wall	Neutron flux	$3 \times 10^{18}\,\mathrm{m^{-2}\,s^{-1}}$	$1.5 \times 10^{25}\,\mathrm{m^{-2}\,s^{-1}}$
	Neutron fluence*	$3 \times 10^{25}\,\mathrm{m^{-2}}$	$3 \times 10^{18}\,\mathrm{m^{-2}}$
	γ-ray dose rate	$2 \times 10^{3}\,\mathrm{Gy\,s^{-1}}$	$\sim 1 \times 10^{10}\,\mathrm{Gy\,s^{-1}}$
	Energetic ion/atom flux	$5 \times 10^{19}\,\mathrm{m^{-2}\,s^{1}}$	…
1st Diagnostic	Neutron flux	$1 \times 10^{17}\,\mathrm{m^{-2}\,s^{-1}}$	$1 \times 10^{26}\,\mathrm{m^{-2}\,s^{-1}}$
	Neutron damage rate	$6 \times 10^{-9}\,\mathrm{dpa\,s^{-1}}$	negligible
	Neutron fluence*	$2 \times 10^{24}\,\mathrm{m^{-2}}$	$\sim 1 \times 10^{19}\,\mathrm{m^{-2}}$
	Neutron damage	$1 \times 10^{-1}\,\mathrm{dpa}$	negligible
	γ-ray dose rate	$\sim 1 \times 10^{2}\,\mathrm{Gy\,s^{-1}}$	$\sim 1 \times 10^{10}\,\mathrm{Gy\,s^{-1}}$
	Energetic ion/atom flux	$\sim 1 \times 10^{18}\,\mathrm{m^{-2}}$	…
	Nuclear heating	$1\,\mathrm{MW\,m^{-3}}$	0
	Operating temperature	520 K	293 K
	Atomosphere	Vacuum	Air
Other	EM pulse	…	10 to $500\,\mathrm{kV\,m^{-1}}$ @ 1 GHz
	Shrapnel	…	1 to $10\,\mathrm{km\,s^{-1}}$ @ $\sim$30 µm

Table 1.1: Estimated operating environment comparison between MCF (ITER) and ICF (LMJ). Reproduced from [31]. Note these numbers are unrepresentative of actual fusion power plants as both ITER and the LMJ are experiments—it is likely power plants will require much harsher operating conditions.
* End of life.

the materials requirements for both "mainstream" types of fusion energy production. That said, the components needed for energy harvesting—a divertor in the case of MCF, and an open problem for ICF—are conspicuous by their absence.

Table 1.1 shows that dosages and dosage rates vary wildly between approaches. For the most part, MCF receives higher doses of neutrons and ions in both the first wall and diagnostic equipment. This seems to indicate that the material requirements for MCF are stricter than for ICF. However, shrapnel production is a strong possibility in ICF, especially in indirect drive reactors; where a specially made container called a "hohlraum" holds the fuel pellet and is subsequently obliterated when the pellet undergoes fusion (see fig. 1.1 [32]). This is a huge challenge as such high energy shrapnel would be capable of destroying diagnostic equipment and damaging the first wall [33–35].

Overall, the table is unrepresentative of potential operating environments in large-scale power stations, but sets lower limits on the demands of the materials involved in both "mainstream" proposals for fusion energy production.

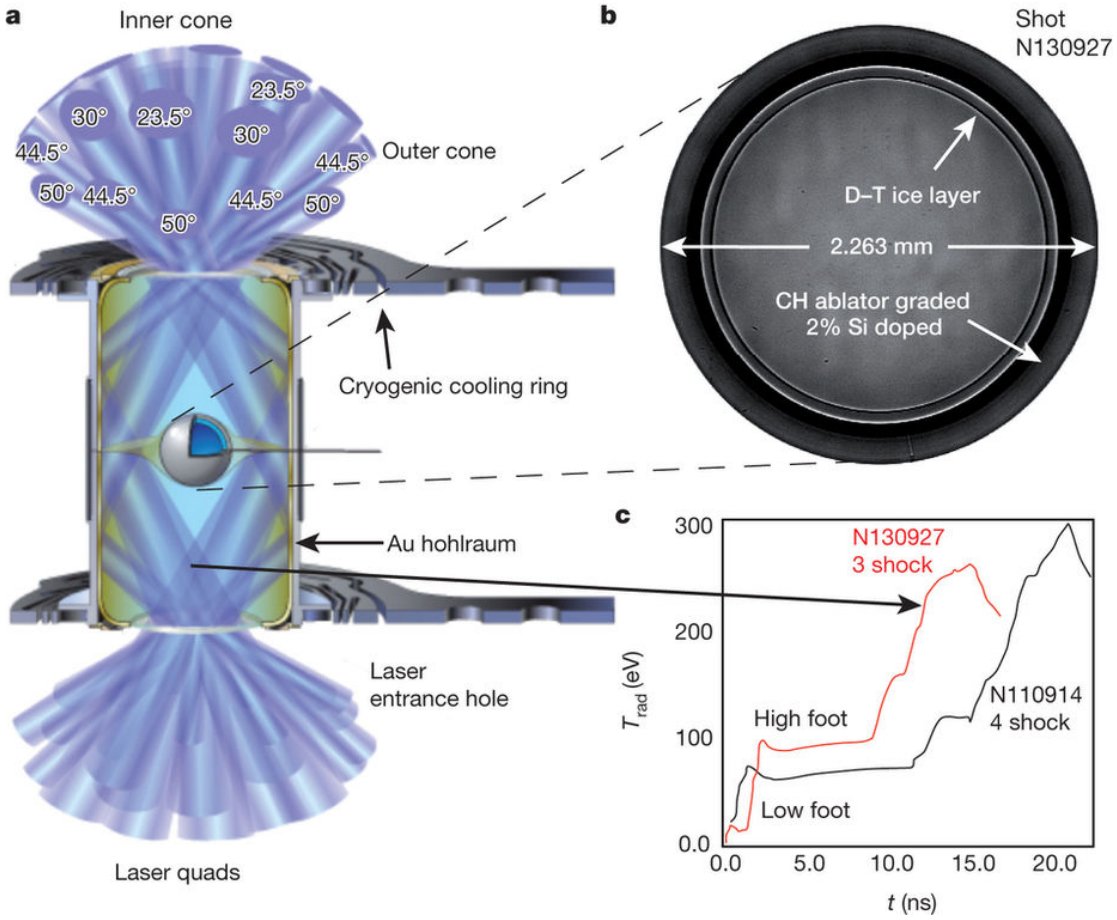


Figure 1.1: **a**, Cross section of the National Ignition Facility's ICF target design showing the gold hohlraum and plastic capsule, fuel pellet, and incident laser bundles. **b**, X-ray image of the actual capsule for N130927 with DT fuel layer and surrounding CH (carbon–hydrogen) plastic ablator. **c**, X-ray radiation drive temperature as a function of time for the National Ignition Campaign (NIC) low-foot implosion and the post-NIC high-foot implosion. Image taken from [32].

1.2.2 Effects of Radiation on Reactor Materials

1.2.2.1 Radiation Damage

One of the bigger problems in radiation damage research is the lack of a truly standardised way of measuring damage on materials [36]. The most common unit is displacements per atom (dpa) [37]. It is defined as the average number of displacements undergone by each atom in a material as a result of being irradiated. The "fundamental" unit of measurement for this is the number of displacements per unit volume per unit time, R

$$R = N \int_{E_{\min}}^{E_{\max}} \int_{T_{\min}}^{T_{\max}} \phi(E)\, \sigma(E,T)\, \upsilon(T)\, \mathrm{d}T\, \mathrm{d}E\,, \tag{1.1}$$

where N is the atom number density (no. of atoms per unit volume); E the incoming particle's energy; T the energy transferred in a collision of a particle of energy E and a lattice atom; $\phi(E)$ the energy dependent particle flux; $\sigma(E,T)$ the cross section for the collision of a particle with energy E resulting in a transfer of energy T to the struck atom; and $\upsilon(T)$ the number of displacements per primary knock on atom as a function of transferred energy T. DPA can be calculated by naïvely multiplying R by the sample volume and total exposure time (or we could use fluence rather than flux). This ignores the fact that $\sigma(E,T)$ and $\upsilon(T)$ will be locally perturbed in the neighbourhood of damage cascades, since the bulk volume is much greater than that of the damage cascades', we assume the functions remain globally unchanged.

In principle, this is a rather good measure of damage [37]. The catch is that generalised, analytical expressions for $\sigma(E,T)$ and $\upsilon(T)$ depend on a slew of parameters and are therefore incredibly hard if not impossible to derive. This does not mean they cannot be discretised and roughly approximated via Monte Carlo (MC) approaches [36–38]. However, damage cascade modelling falls squarely in the realm of pico- to nanosecond timescales and as such only feasable with Molecular Dynamics (MD) and Kinetic Monte Carlo (KMC) [39, 40] approaches. At the end of such cascades, we are often left with dislocation sources or prismatic dislocation loops [41] which can be used as inputs by a Dislocation Dynamics (DD) simulation to study larger temporal and spatial scales [30].

1.2.2.2 Transmutation

Transmutation products are one of the biggest sources of problems for materials in fusion applications [20, 21]. Not only do they tend to embrittle materials, they often also reduce their thermal conductivity [42]. The former presents significant challenges for structural materials [43]; the latter is especially egregious for energy extraction by limiting the divertor's ability to conduct heat thus lowering the reactor's efficiency and causing thermal stresses due to the generation of hot spots that cannot be easily dissipated. As a result, understanding the mechanical and thermal behaviour of transmutation alloys is crucial for moving forward [44–46] but doing so requires knowledge of the time evolution of a reactor's components. Because the composition changes in a a non-trivial way, so do the mechanical properties. Worse still are the long timescales over which this happens—on the order of years or tens of years [47]. Even if we were able to expermentally irradiate fusion-relevant materials with appropriately energetic neutrons, it would take years before we could characterise their behaviour, and doing so would be problematic due to radioactive decay.

The way we go about addressing the time-dependent compositional change of a material is to model it. Culham Centre for Fusion Energy's (CCFE) FISPACT [48] software takes an MC approach at calculating transmutation and decay products of a sample given certain conditions. The code utilises external data provided by the European Activation File which provides cross sections and decay rates for a wide range of isotopes [49]. Unfortunately there is a lack cross section data for certain neutron energies that contribute in a non-negligible manner to a fusion environment. Interpolating to fill the gaps would not be appropriate as the data is non-smooth and subject to resonance peaks, so the solution is imperfect.

Transmutation products will always be problematic, but they are a fact of working with fusion-relevant materials. Fortunately there are ways in which inclusions may be modelled via DD (section 1.5.4.2). DD may also be used to model dislocation movement and interaction within heterogenous media (section 1.5.4) as is the case for oxide-dispersion-strengthened (ODS) steels; and transmutation alloys of Tungsten divertors, where Osmium, Rhenium and Tantalum clusters which prove highly problematic for its temperature conductivity and structural integrity [50–56].

1.3 Parallel Computing

Processing chips are made up of millions or even billions of transistors acting as switches in logic gates. Each time a logic gate fires, the capacitors inside them charge and discharge at the chip's frequency or clock speed [57]. Consider the power of any electronic component,

$$P(t) = I(t)V(t)\,, \tag{1.2}$$

where I is current and V is voltage both as a function of time. The current, I, of a capacitor and the definition of power, P, as functions of time, t, are given by,

$$I(t) = C\frac{\mathrm{d}V(t)}{\mathrm{d}t}\,, \qquad P(t) = \frac{\mathrm{d}E(t)}{\mathrm{d}t}\,, \tag{1.3}$$

where C is capacitance and E the energy stored in the capacitor. Substituting eq. (1.3) into eq. (1.2) and integrating twice, we obtain the expression for the energy stored in a capacitor,

$$E_\mathrm{c} = \frac{CV_\mathrm{c}^2}{2}\,, \tag{1.4}$$

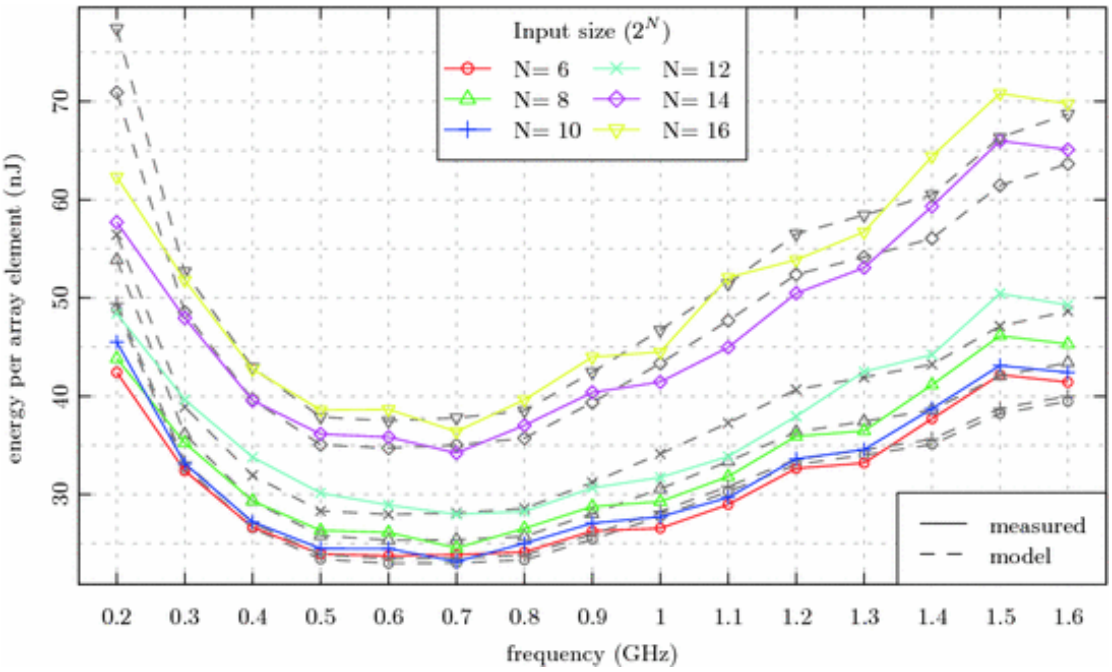


Figure 1.2: Energy required by a Samsung Galaxy S2 CPU at 37 °C to complete the Gold-Rader implementation of the *bit-reverse* algorithm. The dashed lines denote a theoretical model, image and model found in [60].

where V_c and E_c are the voltage and energy stored in the capacitor. Recalling that in a processing chip, the capacitors are charged and discharged at the chip's clock speed f, we arrive at the expression for power consumption, P_c, of a capacitor charging and discharging at frequency f [58],

$$P_c = E_c f \propto C V_c^2 f \,. \tag{1.5}$$

Computer chips are rather more complicated, but their power consumption (also known as power dissipation) is described by a simple addition of terms [59],

$$P = P_{\text{dyn}} + P_{\text{sc}} + P_{\text{leak}} \,, \tag{1.6}$$

where P_{dyn} is the dynamic power dissipation given by eq. (1.5). The two other dissipation mechanisms are: *1*) short-circuit, sc, which depends on frequency and occurs when a direct path from transistor to ground is made as a result of multiple transistors conducting simultaneously; and *2*) leakage, leak, which depends on the voltage and is due to micro-currents between doped parts a transistor. Furthermore, higher voltages and frequencies result in higher temperatures, which in turn mean decreased transistor performance and increased capacitance. The overall result is an energy expenditure curve similar to fig. 1.2 [60] for every processor unit.

This set of optimum conditions is the reason behind the multicore and multi-

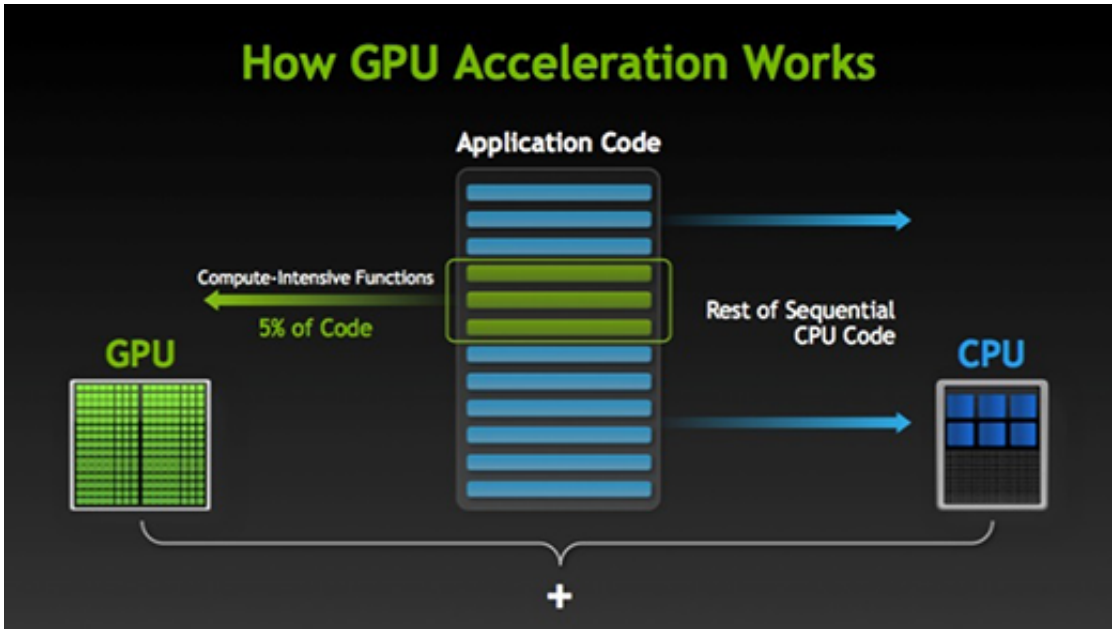


Figure 1.3: CUDA runtime schematic. Repetitive but computationally intensive tasks can be offloaded to the GPU. Both GPU and CPU are completely independent from each other and can work on different parts of the code at the same time, care must be taken to ensure proper syncronisation. Image taken from [61].

threaded design of modern Central Processing Units (CPUs) and Graphics Processing Units (GPUs). It is also why in recent years there has been such a massive push for parallelism in all computing markets. It is simply not feasable to continually increase clock speeds and voltages because cooling solutions would struggle to remove heat fast enough, and power consumption would skyrocket.

1.4 Computation on Graphics Processing Units

Central Processing Units (CPUs) are designed not only to perform mathematical operations but logical ones that control program flow. They are tailored to perform the wide variety of operations required by an operating system. These include program scheduling (load balancing), instantiation (program loading, unloading, loops, recusion instances) & branching (if/case statements, go to's), memory operations (fetching, storing, allocation), input/output (IO), and program monitoring (program counters, recursion counters). Modern CPUs have a degree of parallelism that allows them to increase their total throughput while keeping their operation within near optimum conditions as previously mentioned. They are commonly divided into cores and threads, though certain high-end chips have an additional layer named hyperthreading [62].

Given the limited scope of the first computers, the general purpose of CPUs was enough to cover their needs. With the advent of personal computers, the demands

on CPUs drastically increased. For industrial users these revolved around data acquisition, filtering and preprocessing [63–65]. On the other hand, the domestic market demanded ever increasing levels of abstraction and usability in the form of Graphical User Interfaces (GUIs) such as windows, cursors and their accompanying sound effects. Manufacturers identified this and moved to provide specialised modules that freed CPU resources and improved user experience by accelerating different processes through hardware means [66–69]. These modules include Sound Cards (SCs), Field Programmable Gate Arrays (FPGAs), Graphics Processing Units (GPUs), Application Specific Integrated Circuits (ASICs), Cryptographic Accelerators, Regular Rxpression (RegEx) Accelerators, among others. They are collectively dubbed "hardware accelerators".

Graphics Processing Units were originally intended to offload the very data-intensive but computationally simple operations needed for 3D gaming and rendering [66–68]. Graphics processing was seen as a prime candidate for hardware acceleration because the operations on each pixel are largely the same across the screen. Due to their original purpose as gaming and rendering accelerators, they were never designed to operate on higher than single precision data. In fact, single precision (32-bit precision) is still good enough to encode 32-bit colour depth (8-bit channel per RGB colour + 8-bit alpha channel), which only the most high-end monitors support [70]. It is also worth noting that because the same operations apply to different pieces of (mostly) independent data, they can all be performed at the same time, often trivially reducing the order of polynomial complexity algorithms.

As previously mentioned, GPU parallelism frees up enormous amounts of CPU processing power that can be put to good use running other programs or performing complementary serial processes, while the GPU works concurrently on its dataset. Parallelism has been used by the scientific community for years, but the focus has mainly been on CPU parallelism [71]. Consequently, its scope was lagerly limited to the use of computational clusters. The two main reasons for this were the fact that GPUs lacked support for higher precision arithmetic, and the very limited to non-existent support for scientific computing in languages such as OpenGL [72].

It wasn't until the development of the OpenCL and OpenACC standards that GPUs caught the attanetion of the scientific community as a viable way of exploiting parallelisation without access to a computing cluster.

OpenCL allows one to work with heteragoenous systems and is similar to C in that it's very low level. It works on a wide range of hardware accelerators and is therefore useful for many scientific and engineering applications, but it's also relatively hard to use. One can make use of libraries written in OpenCL to

facilitate development, but it is still a fully fledged C-type language [73].

OpenACC is similar to OpenMP in that they both use pragmas[1] and they both work in shared memory environments—same GPU and same CPU respectively. This is no coincidence as OpenACC was designed as an extension of OpenMP for developing parallel applications on hardware accelerators. Unfortunately, being pragma based, the standard requires significant work by compiler manufacturers, so its adoption has therefore been slow [74]. Furthermore, despite simplifying development and minimising the barrier to entry, the use of pragmas limits the flexibility and adaptability of the framework compared to OpenCL.

Recently however, a third option has become increasingly viable. NVidia's Computer Unified Device Architecture (CUDA) framework provides the best aspects of both OpenCL and OpenACC. The tradeoff is that it only works on NVidia GPUs and is a closed source product. However, the accessibility and flexibility of CUDA provides anyone familiar with C/C++ the means to develop a GPU application with little issue. NVidia is also strongly backing scientific research by adding double precision support on their GPUs. They have also worked to provide parallel equivalents of well known serial libraries—such as cuBLAS & cuFFT—for scientific computing. They have additionally developed a wide range of specialist graphics cards tailor-made for scientific purposes. As such, they are the leaders in GPU computing in scientific communities [61].

1.4.1 Hurdles for Parallelisation

The difficulty in parallelisation varies tremendously from problem to problem. Problems where data is uncorrelated and independent—such as sampling well-behaved probability distributions—are almost trivially parallelisable. Problems where data is correlated or strongly dependent on its neighbours—such as Dislocation Dynamics—require a more careful approach [75].

The largest hurdle when implementing parallel algorithms is often the efficient use of data. In order to obtain good parallel performance, a lot of thought has to be placed on data access patterns, data read/write conflicts, and memory allocation and transfer [61]. For best performance, all this must be analysed on a case by case basis. If done incorrectly, the performance of a parallel application may be

[1] Pragmas are especially formated comments that specific compilers recognise and turn into special instructions (often for moving memory to the appropriate hardware) at compile time. They are not part of the programming language's official standard, but rather compiler-specific extensions. Pragmas are usually programmed in C or Assembly and must be implemented by the compiler manufacturer. Nothing prevents application developers from writing the would-be pragma's code directly into their application, but this can prove a lengthy and difficult process. The rigid nature of pragmas, and the compiler manufacturer's priorities limit their scope and usability.

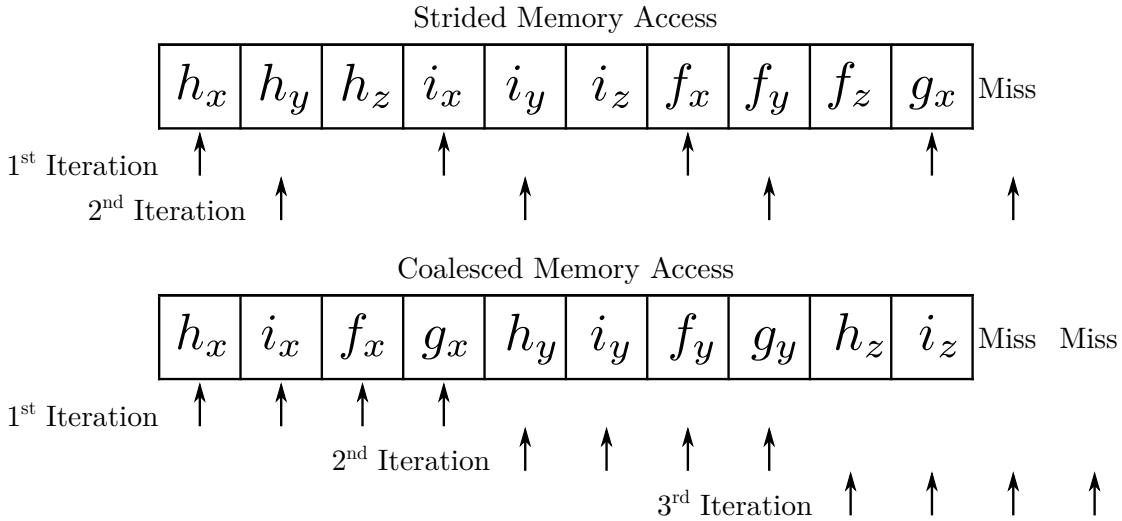


Figure 1.4: Arrows represent fetch requests by single threads in a GPU. Data should be arranged in such a way that all threads in a warp (collections of 32 threads) access contiguous memory locations to reduce time-consuming memory fetch operations. This is often unintuitive but extremely important, especially on scientific computing cards which are optimised for long computational times and low memory fetch frequency.

lower than the serial version. One must consider a wide range of parameters to successfully parallelise a problem. Among these are GPU architecture, problem size, computational and memory complexity, code branching, required arithmetic precision, and error tolerances [61, 76].

Efficient parallelisation of many problems requires "coalesced memory access" as shown in fig. 1.4, which means we have to be extremely careful when mapping CPU memory to global GPU (device) memory. The fact that threads work "simultaneously"[2] means that in order to obtain good performance, data which is to be "simultaneously" loaded into each thread must be contiguous. This maximises cache memory use and therefore reduces slow memory fetch operations to global or shared memory.

Special cases, such as having a parallel dislocation line segment to a surface as discussed in section 1.5.3, must be treated carefully due to the way code branching works in GPUs. There are various ways of doing so: *1*) if the special case is inexpensive, it can be treated within the same GPU function; *2*) if the special case is expensive and always known (certain boundary conditions in FEM), it can be placed in its own GPU function that treats it separately; *3*) if the special case is expesive and found at runtime it may be asynchronously treated by the CPU or buffered into its own GPU function to be executed at a later time.

One of the advantages of GPU-CPU independence is that both can work con-

[2]Not quite but essentially simultaneously. See [61] for details.

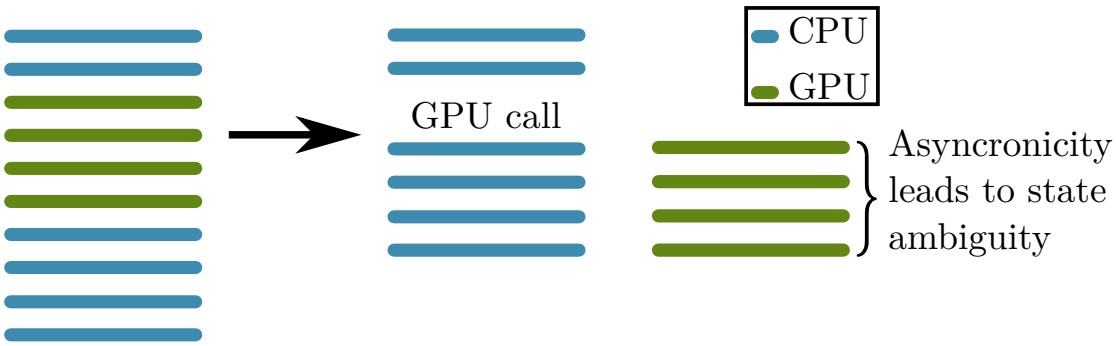


Figure 1.5: GPU and CPU code run independent of each other. Leads to state ambiguities if both sides need to talk to each other [61].

```
if (y == 0){
    x = a+b;
}
else if (y == 1){
    x = a*b;
}
else{
    x = a/b;
}
```

(a) CPU code will only execute if the condition is met. This is called code branching.

```
p = (y == 0);
p: x = a+b;

p = (y == 1);
p: x = a*b;


!p: x = a/b;
```

(b) GPU code executes every line but only stores results if the the flag before the colon is true.

Figure 1.6: The NVidia CUDA compiler replaces `if` and `case` statements with logical flags `p`. Every line is executed, but the data is only stored if the flag prior to the colon is true [61]. This means that having rare but computationally expensive special cases will tank parallel performance and must therefore be dealt with separately.

currently on the problem. If both systems have to talk to each other, then one must tread carefully, ensuring proper syncronisation and data mapping before moving (see fig. 1.5).

The reason why code branching is bad for GPU parallelisation is down to the fact that they work like software customisable vector machines. Where collections of threads all carry out the same operation on different pieces of data at the same time. This means that regardless of whether a condition is true or false, the code will execute. It is only data storage that depends on the condition, as illustrated in fig. 1.6.

Dislocation Dynamics (DD)—in particular 3D Discrete Dislocation Dynamics (3D DDD)—can greatly benefit from parallelisation, especially when coupling to Finite Element Methods (FEM). It is worth noting that there are potential issues arising from the very computationally expensive functions and special cases that

often arise from analytical solutions in 3D DDD. There are also potential issues with data redundancy—which are non-limiting in the short term—that may eventually require a more data-efficient approach as the computational capabilities and data capabilities of GPUs converge. Section 1.5.5 expands on these issues in the context of DD.

1.5 3D Dislocation Dynamics Modelling

The plastic deformation of materials is generally governed by the generation and motion of line defects known as dislocations through the crystal lattice. Microstructural features such as grain boundaries, precipitates and inclusions impede dislocation motion causing strengthening but often limiting ductility [77]. Understanding the behaviour of the dislocation ensemble is highly complicated even when ignoring dislocation-microstructure interactions. However, if we want to truly comprehend their real-world behaviour, we cannot limit ourselves to idealised scenarios.

One of the most often used parametrisations of DD is Discrete Dislocation Dynamics (DDD). Where dislocations are parametrised as a series of nodes linked by straight line segments. This reduces computational requirements and allows for analytic solutions to be obtained. At present, neither DDD or Finite Element (FE) models can truly handle all of the complexities of real-world alloys [78–81]. For one, DDD relies on assuming a linear-elastic isotropic solid domain with periodic boundary conditions, while FE relies on assuming continuum properties inside its *finite* domain.

Crystal Plasticity Finite Element Methods (CPFEM) use constitutive equations to calculate dislocation motion and generation on a set of slip systems. These are given as dislocation densities and thus are still "bulk" models because they don't interact with each other [82]. CPFEM can handle finite strains and anisotropy but not strain localisation/slip bands, etc.

Furthermore, dislocations often accumulate in the vicinity of microstructural features. The interaction of dislocations with microstructure can lead to further increased hardening and local hot spots in stress that can lead to failure initiation [83, 84]. Hardware accleration, i.e. using Graphics Processing Units (GPUs), has been shown to be very effective in DDD [85], and has the potential to enable the simulation of much larger numbers of dislocations for longer timescales. In order to study these phenomena we must find a way of coupling DDD and FEM into a single multiscale model that can simulate micromechanical tests more accurately than CPFEM. With a model such as this we may potentially be able to predict and observe emergent phenomena/properties, explain michromechanical behaviours,

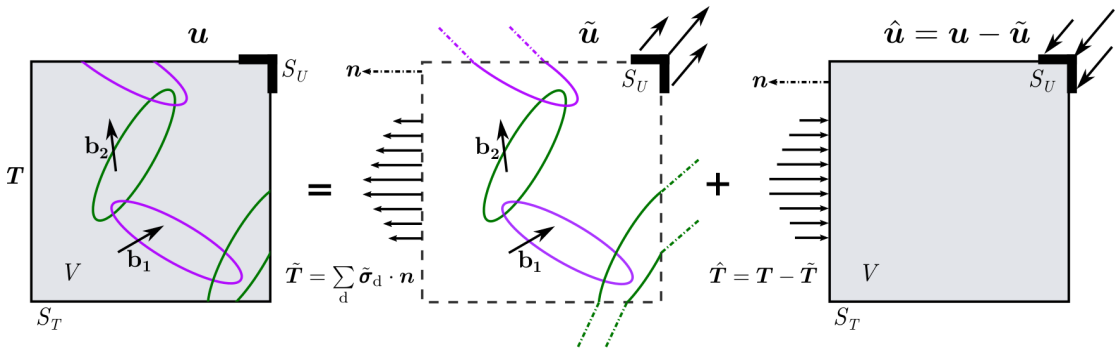


Figure 1.7: Decomposition of the boundary value problem. The volume V is bounded by surface S contains a dislocation ensemble and is subjected to initial traction conditions $\boldsymbol{T}$ on S_T and $\boldsymbol{u}$ on S_u. First, the traction, $\tilde{\boldsymbol{T}}$, and displacement, $\tilde{\boldsymbol{u}}$, fields due to the dislocations in the infinite domain (DDD) are evaluated on the boundaries S_T and S_u respectively. Then a traditional FE method is used to calculate the stress field satisfying the boundary conditions $\hat{\boldsymbol{T}} = \boldsymbol{T} - \tilde{\boldsymbol{T}}$ and $\hat{\boldsymbol{u}} = \boldsymbol{u} - \tilde{\boldsymbol{u}}$. Image edited from [86].

and even predict and explain experimental results based on underlying dislocation mechanisms.

1.5.1 Coupling Dislocation Dynamics to Finite Element Methods

Coupling dislocation dynamics (DD) to FEM is important to properly simulate micromechanical tests because DD provides us with a more precise set of inputs and greater granularity for solving the FE problem. There are at present two methods with which to do so, the Superposition Model and the Discrete-Continuum Model. The most accurate implementation of dislocation dynamics is discrete dislocation dynamics, where the dislocation is broken into nodes and connecting segments. This is compuationally efficient and allows for analytical solutions. However, as discussed in section 1.5.1.3 explicit discretisation is not the only way to tackle the problem.

1.5.1.1 Superposition Model

The Superposition Model (SM) works by decomposing the problem into separate DDD and FE problems (fig. 1.7). It is assumed that a linear-elastic body V bounded by a surface S is subject to traction boundary conditions, $\boldsymbol{T}$, on S_T and displacement boundary conditions, $\boldsymbol{u}$, on S_u. The formulation proposed in [87] to impose traction-displacement boundary conditions on DDD problems in finite domains states that the total displacement and stress fields can be written as a

superposition of displacement and stress fields obtained from DDD and FE,

$$\boldsymbol{u} = \tilde{\boldsymbol{u}} + \hat{\boldsymbol{u}}\,, \tag{1.7a}$$

$$\boldsymbol{\sigma} = \tilde{\boldsymbol{\sigma}} + \hat{\boldsymbol{\sigma}}\,. \tag{1.7b}$$

The ($\tilde{}$) fields are those associated with the dislocation in an infinite medium and are obtained by evaluating analytic fields in a DDD simulation. While the corrective ($\hat{}$) fields are those which must be superimposed to ensure the boundary conditions are met. This means that the image fields can be obtained by running a FE simulation with the "corrected" displacement and traction fields,

$$\hat{\boldsymbol{u}} = \boldsymbol{u} - \tilde{\boldsymbol{u}}\,, \tag{1.8a}$$

$$\hat{\boldsymbol{\sigma}} \cdot \boldsymbol{n} = \hat{\boldsymbol{T}} = \boldsymbol{T} - \underbrace{\tilde{\boldsymbol{T}}}_{\tilde{\boldsymbol{\sigma}} \cdot \boldsymbol{n}}\,, \tag{1.8b}$$

where $\boldsymbol{n}$ is the outer unit normal vector to S. As a dislocation segment moves closer to the surface, its ($\tilde{}$) field diverges and starts causing numerical problems [88]. A further problem with this method is that modelling elastic inclusions not only requires the calculation of forces induced by dislocations on the inclusion's surface, but also demands the calculation of so-called polarisation stresses due to differences in the inclusion's and matrix properties [87–89].

The relative simplicity of the superposition method has made it a popular choice [86, 90, 91] for coupling DDD and FEM because all it requires is the calculation of forces and displacements on the boundaries (see section 1.5.3).

1.5.1.2 Discrete-Continuum Model

The Discrete-Continuum Model (DCM) takes an alternative approach to solving the same problem. The DCM only treats short-range dislocation-dislocation interactions analytically while all other interactions are numerically calculated via FEM [92]. It is based on the regularisation of the atomic displacement jump across the slip plane into a plastic strain inclusion according to eigenstrain theory [93]. Like the Superposition Model, the DCM also assumes the simulated volume to be linear-elastic.

The eigenstrain formalism assumes that material defects can be represented as stress-free strain distributions dubbed eigenstrains [93]. For example, a dislocation loop of any shape may be approximately represented, thin, coherent, plate-like inclusion with the same contour as the loop and a characteristic thickness h as shown in fig. 1.8 [92]. The eigenstrain tensor, ϵ^{p}, can then be defined as

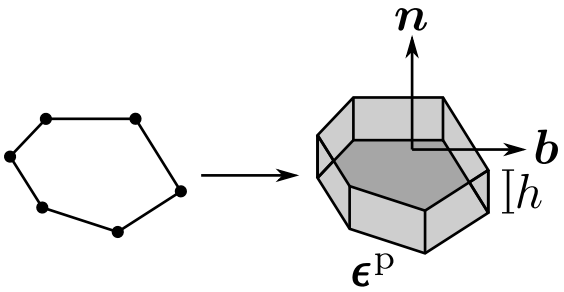


Figure 1.8: The eigenstrain formalism as defined in [93]. The dislocation is being approximated by a thin coherent inclusion of thickness h, whose strain-free stress tensor $\boldsymbol{\sigma}^{\mathrm{p}}$ as defined by eq. (1.9). The vectors $\boldsymbol{n}$ and $\boldsymbol{b}$ are the normal vector to the slip plane and the dislocation line's Burgers vector.

a symmetric dyadic product of the Burgers vector, $\boldsymbol{b}$, and slip plane normal $\boldsymbol{n}$,

$$\boldsymbol{\epsilon}^{\mathrm{p}} \equiv \frac{1}{2h}(\boldsymbol{b} \otimes \boldsymbol{n} + \boldsymbol{n} \otimes \boldsymbol{b})\,. \tag{1.9}$$

eq. (1.9) can be used to calculate approximate elastic fields from the stress-free eigenstrain distribution. The approximation is accurate far from the dislocation core [94]. As we move closer but still outside of the core region, the approximation tends toward the exact discrete dislocation solution as $h \to 0$. Due to linear elasticity, the total plastic strain is the sum of the individual plastic strains due to each dislocation segment.

The formalism then lets us solve the boundary value problem by finding the stress tensor $\boldsymbol{\sigma}$, elastic strain $\boldsymbol{\epsilon}^{\mathrm{e}}$ and displacement $\boldsymbol{u}$ in mechanical equilibrium with the boundary conditions and plastic strain distribution $\boldsymbol{\epsilon}^{\mathrm{p}}$ from the "inclusions".

It is worth noting that in order to calculate the eigenstrains, there must be sufficient FE nodes inside the thin plate. Therefore smaller values of h necessitate finer FE meshes. Furthermore, the original DCM experienced numerical blow up as dislocation-dislocation distances approached h [95], but this has since been addressed. Assuming linear-elasticity, the revised model [92] yields,

$$\nabla \cdot \boldsymbol{\sigma} + \boldsymbol{f} = \boldsymbol{0} \quad \in V \setminus \{A\}\,, \tag{1.10a}$$

$$\boldsymbol{\sigma} = \boldsymbol{E} : \boldsymbol{\epsilon} \quad \in V \setminus \{A\}\,, \tag{1.10b}$$

$$[[\boldsymbol{u}]] \quad \text{across } \{A\}\,, \tag{1.10c}$$

$$\boldsymbol{u} = \boldsymbol{u}_{\boldsymbol{o}} \quad \in S_u\,, \tag{1.10d}$$

$$\boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{T} \quad \in S_T\,. \tag{1.10e}$$

At time t, $\{A\}$ denotes the area swept by the dislocation loops since the start of the simulation. $[[\boldsymbol{u}]]$ denotes displacement jumps tangent to $\{A\}$ due to dislocation glide; its magnitude and direction depend on the Burgers vector $\boldsymbol{b}$. $\boldsymbol{E}$ is the 4^{th}-

order elasticity tensor, $\boldsymbol{\sigma}$ the small strain tensor, $\boldsymbol{f}$ are the body forces and, :, is the double dot product defined as, $(\boldsymbol{E} : \boldsymbol{\epsilon})_{ij} = E_{ijkl}\epsilon_{kl}$, for a rank 4 tensor $\boldsymbol{E}$ and rank 2 tensor $\boldsymbol{\epsilon}$. The operator, $\setminus$, is the set difference defined as, $A \setminus B = \{x \in A | x \notin B\}$. We use the same notation for the volume, surface normal and boundary conditions as in section 1.5.1.1. eq. (1.10) can then be linearly decomposed three parts which are solved via FEM or DDD and coupled.

It is worth noting that the DCM is substantially more complicated than the SM. By requiring the finite elements be small enough for the eigenstate formalism to work, it strongly couples DDD to the FE model and software. It shifts the brunt of the computational workload from DDD to FEM. Essentially trading computationally intensive long range interactions which scale on the order of $\mathcal{O}(N^2)$, where N is the number of dislocation line segments; for computationally intensive tasks scaling on the order of $\mathcal{O}(M^3)$ where M is the number of finite elements in a cubic mesh with N elements per side. Furthermore, it cannot be used with BE methods as the eigenstrain formalism demands the use of internal elements rather than simply requiring a surface mesh.

That said, the DCM allows for reductions in the computational complexity of certain parts of the problem which would otherwise have to be done via DDD [92]; namely long-range dislocation-dislocation or dislocation-surface interactions via an interaction distance cutoff that is only possible with the eigenstrain formulation. Compared to the SM, gains in computational efficiency grow as dislocation density increases with respect to the number of finite elements. Since dislocation-dislocation and dislocation-surface interactions are among the most computationally expensive aspects of DDD, the DCM can reduce the overall computational cost of simulations with large enough dislocation densities.

1.5.1.3 Level Set Method

Even though the level set method has not strictly been used to couple DD to FEM, it has been used to model inclusions [96] as described in section 1.5.4.2. This type of dislocation dynamics is fundamentally distinct from DDD because Level Set DD uses arbitrary functions rather than a discretisation approach to represent dislocations lines. The idea is that in 3D, a dislocation $\gamma(t)$ can be represented by the intersection of two zero levels of two level set functions (see fig. 1.9),

$$\phi(x(t), y(t), z(t), t) = 0\,, \qquad \psi(x(t), y(t), z(t), t) = 0\,, \tag{1.11a}$$

$$\frac{\mathrm{d}\phi}{\mathrm{dt}} = \partial_t\phi + \boldsymbol{v}\cdot\nabla\phi = 0\,, \qquad \frac{\mathrm{d}\psi}{\mathrm{dt}} = \partial_t\psi + \boldsymbol{v}\cdot\nabla\psi = 0\,, \tag{1.11b}$$

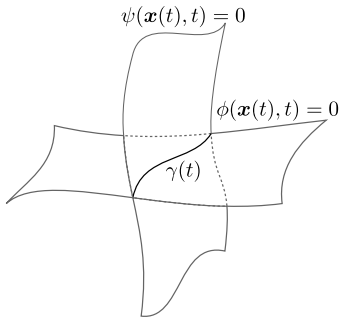


Figure 1.9: Diagram of a continuous dislocation $\gamma(t)$ as the intersection of the zero level of two level set functions $\phi(\boldsymbol{x}(t),t),\ \psi(\boldsymbol{x}(t),t)$ as posed by eq. (1.11). Image edited from [97].

where $\boldsymbol{v}$ is the dislocation velocity. This definition uses the material derivative because it is assumed that the function and its spatial coordinates are all functions of time, t. The material derivative also comes up when deriving the Navier-Stokes equations of fluid dynamics.

The level set method is significantly more computationally more expensive than DDD [97]. It must solve two coupled quasi-linear partial differential equations whose character is in general undefined. It therefore requires the use of very robust numerical solvers such as higher order Total Variation Deminishing (TVD) Runge-Kutta methods for time discretisation (TVD-RK4 or higher) and high order ENO (Essentially Nonoscillatory $\mathcal{O}(N^5)$ or higher) or WENO (Weighted Essentially Non-Oscilatory) interpolation methods for spatial discretisation [98] for the solutions to converge satisfactorily. The method also makes it impossible to obtain many of the useful analytical solutions one can find by discretising dislocations into straight line segments. Another problem is that this method has never been truly coupled to the finite element method. However, if one were to do so there are two naïve ways of going about it: *1*) numerically integrating along the dislocation lines and finding a way to numerically calculate the forces on FE nodes, or *2*) discretising the level set curves and using either the DCM or SM to find forces and displacements. The former would be very computationally expensive. Furthermore, no one has found a way to compute the forces or displacements exerted by generally shaped, non-discretised dislocations, numerical solutions are possible as they are needed to find self-interaction energies and dislocation-dislocation forces [97] but general, closed-form analytical ones are most likely impossible to find. The latter defeats the purpose of the level set method by abandoning the continuum framework. As a result of this, the level set method is not nearly as popular as DDD utilising either the SM or DCM.

1.5.2 Non-Singular Continuum Theory of Dislocations

The Peach-Koehler formula describes the fundamental interactions of dislocations [99] according to the force, $\boldsymbol{f}$, that a local stress $\boldsymbol{\sigma}$ exerts on a dislocation line with Burgers vector, $\boldsymbol{b}$, and line direction, $\boldsymbol{\xi}$,

$$\boldsymbol{f} = (\boldsymbol{\sigma} \cdot \boldsymbol{b}) \times \boldsymbol{\xi} \,. \tag{1.12}$$

According to [100], the internal stress field of a dislocation loop in a homogenous, infinite, linear-elastic medium is given by the contour integral around a loop L,

$$\sigma_{ij}^{\infty}(\boldsymbol{x}) = C_{ijkl} \oint_L \epsilon_{lnh} C_{pqmn} \frac{\partial G_{kp}(\boldsymbol{x} - \boldsymbol{x}')}{\partial x_q} b_m \, \mathrm{d}x'_h \,, \tag{1.13}$$

where C_{ijkl} is the elastic stiffness tensor, ϵ_{lnh} the permutation operator, $\boldsymbol{b}$ the Burgers vector, and $G_{kp}(\boldsymbol{x} - \boldsymbol{x}')$ is Green's function of elasticity [100]. $G_{kp}(\boldsymbol{x} - \boldsymbol{x}')$ is defined as the displacement component in the x_k direction at point $\boldsymbol{x}$ in response to a unit point force applied in the x_p direction at point $\boldsymbol{x}'$ [101]. In an isotropic elastic solid, $G(\boldsymbol{x} - \boldsymbol{x}')$, takes the form,

$$G_{ij}(\boldsymbol{x} - \boldsymbol{x}') = \frac{1}{8\pi\mu} \left[\delta_{ij} \partial_{pp} - \frac{1}{2(1-\nu)} \partial_{ij} \right] R(\boldsymbol{x} - \boldsymbol{x}') \,, \tag{1.14}$$

where μ, ν are the isotropic shear modulus and Poisson's ratio respectively, δ_{ij} is the Kronecker Delta, $\partial_{x_1 \dots x_n} \equiv \dfrac{\partial^n}{\partial x_1 \dots \partial x_n}$, and $R = \|\boldsymbol{x} - \boldsymbol{x}'\|$. However considering that $\boldsymbol{x} \to \boldsymbol{x}' \Rightarrow R \to 0 \Rightarrow \partial_i R \to \infty$, some (or all) components of the stress field diverge. Furthermore, the total elastic energy also diverges,

$$E = \frac{1}{2} \int S_{ijkl} \sigma_{ij}(\boldsymbol{x}) \sigma_{kl}(\boldsymbol{x}) \, \mathrm{d}^3\boldsymbol{x} \,, \tag{1.15}$$

where $S = C^{-1}$ is the elastic compliance tensor. These divergent properties often prove problematic when numerically computing dislocation forces and energies. Equation (1.15) can also be expressed as a double line integral [102, 103],

$$\begin{aligned} E = &- \frac{\mu}{8\pi} \oiint_L \partial_k \partial_k R(\boldsymbol{x} - \boldsymbol{x}') b_i b'_j \, \mathrm{d}x_i \, \mathrm{d}x'_j \\ &- \frac{\mu}{4\pi(1-\nu)} \oiint_L \partial_i \partial_j R(\boldsymbol{x} - \boldsymbol{x}') b_i b'_j \, \mathrm{d}x_k \, \mathrm{d}x'_k \\ &+ \frac{\mu}{4\pi(1-\nu)} \oiint_L \partial_k \partial_k R(\boldsymbol{x} - \boldsymbol{x}') b_i b'_i \, \mathrm{d}x_j \, \mathrm{d}x'_j \\ &- \nu \oiint_L \partial_k \partial_k R(\boldsymbol{x} - \boldsymbol{x}') b_i b'_j \, \mathrm{d}x_j \, \mathrm{d}x'_i \,, \end{aligned} \tag{1.16}$$

which is important when describing how Cai et al. [101] derived their non-singular expression.

The singularity in R is the result of unreasonably and unphysically assuming a dislocation loop's Burgers vector distribution is a Delta function. The assumption was made to allow for closed-form and relatively simple expressions. As noted in [104], other distributions may be used but they either result in significantly more complicated expressions or destroy the analytical nature of the classical formulation.

There have been many attempts at removing this singularity, including finite-strain elasticity [105], non-local and gradient elasticity [106, 107], interaction cut-off radius [99], average stress at two points on opposite sides of the dislocation line [108, 109], and spreading the Burgers vector distribution out over a finite width [104, 110, 111]. Unfortunately all of these approaches failed in one way or another [101]. Depending on the approach, the following undesirable qualities may present themselves include inconsistencies, impractical implementation, lack of closed-form solutions for non-straight finite dislocations, lack of self-consistency, and the possibility for multiple expressions and solutions for the line integral of the dislocation line energy.

Cai et al. [101] took it upon themselves to define and justify a different Burgers vector distribution that maintains the mathematical convenience of the classical formulation that also eliminates such an unphysical assumption. They did so by introducing a Burgers vector density function,

$$\boldsymbol{b} = \int \boldsymbol{g}(\boldsymbol{x}) \, \mathrm{d}^3\boldsymbol{x} \,, \tag{1.17a}$$

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{b}\tilde{w}(\boldsymbol{x}) = \boldsymbol{b}\tilde{w}(r) \,, \tag{1.17b}$$

where $r \equiv \|\boldsymbol{x}\|$. When substituting eq. (1.17a) into eqs. (1.13) and (1.16) the result of multiplying R with components of $\boldsymbol{b}$ results in the following integrals,

$$R(\boldsymbol{x}-\boldsymbol{x}')b_m = \int R(\boldsymbol{x}-\boldsymbol{x}'')g_m(\boldsymbol{x}''-\boldsymbol{x}') \, \mathrm{d}^3\boldsymbol{x}'' \,, \tag{1.18a}$$

$$R(\boldsymbol{x}-\boldsymbol{x}')b_m b'_n = \iint R(\boldsymbol{x}''-\boldsymbol{x}''')g_m(\boldsymbol{x}-\boldsymbol{x}'')g_n(\boldsymbol{x}'''-\boldsymbol{x}') \, \mathrm{d}^3\boldsymbol{x}'' \, \mathrm{d}^3\boldsymbol{x}''' \,. \tag{1.18b}$$

Using eqs. (1.17) and (1.18) as guidelines, they defined a the following convolutions,

$$w(\boldsymbol{x}) \equiv \tilde{w}(\boldsymbol{x}) * \tilde{w}(\boldsymbol{x}) = \int \tilde{w}(\boldsymbol{x}-\boldsymbol{x}')\tilde{w}(\boldsymbol{x}') \, \mathrm{d}^3\boldsymbol{x}' \,, \tag{1.19a}$$

$$R_a \equiv R(\boldsymbol{x}) * w(\boldsymbol{x}) = \int R(\boldsymbol{x}-\boldsymbol{x}')w(\boldsymbol{x}') \, \mathrm{d}^3\boldsymbol{x}' \,. \tag{1.19b}$$

At which point they assumed there exists an integrable function $w(\boldsymbol{x})$ such that

$$R_a = \sqrt{R(\boldsymbol{x})^2 + a^2} = \sqrt{x^2 + y^2 + z^2 + a^2}\,, \tag{1.20}$$

where a is an arbitrary constant meant to represent the dislocation core radius whose value may be estimated from atomistic simulations. This is essentially a definition that can be used to replace R in the classical equations and eliminate the singularity by including a free parameter. However, in order to ensure this is mathematically sound, there must indeed exist a function which yields R_a as Cai et al. [101] defined it. This can be done by making use of the following property for the convolution of two suitably differentiable functions, $\partial_i(f*g) = \partial_i f * g = f * \partial_i g$. We may take the Laplacian twice,

$$\nabla^2[\nabla^2\{R(\boldsymbol{x}) * w(\boldsymbol{x})\}] = \nabla^2[\nabla^2 R_a(\boldsymbol{x})]\,, \tag{1.21a}$$

$$\nabla^2[\nabla^2\{R(\boldsymbol{x})\}] * w(\boldsymbol{x}) = \nabla^2[\nabla^2 R_a(\boldsymbol{x})]\,, \tag{1.21b}$$

$$\nabla^2[\nabla^2\{R(\boldsymbol{x})\}] = \nabla^2\left[\frac{2}{R}\right] = -8\pi\delta^3(\boldsymbol{x})\,, \tag{1.21c}$$

$$\nabla^2[\nabla^2 R_a(\boldsymbol{x})] = \nabla^2\left[\frac{2}{R_a} + \frac{a^2}{R_a^3}\right] = -\frac{15a^4}{R_a^7}\,, \tag{1.21d}$$

$$w(\boldsymbol{x}) = \frac{15a^4}{8\pi R_a^7}\,. \tag{1.21e}$$

Equation (1.21c) is a very brave, handwavy statement given that,

$$\nabla^2\left[R^{-1}\right] = \nabla^2\left[(x^2 + y^2 + z^2)^{-1/2}\right] = 0\,, \tag{1.22}$$

but may be physically "justified" by pretending $\nabla^2\left[R^{-1}\right]$ defines a spherical surface of radius 0. And eq. (1.21e) is a similarly handwavy statement that can be "justified" by noting that,

$$\lim_{a\to 0} w(\boldsymbol{x}) = \delta^3(\boldsymbol{x})\,. \tag{1.23}$$

Nevertheless, the non-singular formulation by Cai et al. [101] fixes the physically and mathematically problematic assumption that Burgers vectors follow 3D Dirac delta distributions, and proves useful in producing analytical expressions (see section 1.5.3) that are not much more complex than the singular case.

1.5.3 Analytical Forces Exerted by a Dislocation Line Segment on Surface Elements

Whether using the SM or DCM, coupling DDD to FEM requires the traction field $\boldsymbol{\sigma}^{\infty}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x})$ to be distributed among the set of relevant discrete nodes of a FE or BE model. In the DCM model this applies to dislocations that are sufficiently close to the boundary; while in the SM, it applies to all dislocations. Regardless of the coupling model, the force exerted by a dislocation ensemble on a node n on element e is given by,

$$\boldsymbol{F}^{(n)} = \int_{S_e} \left[\boldsymbol{\sigma}^{\infty}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x})\right] N_n(\boldsymbol{x}) \, \mathrm{d}S_e \,, \tag{1.24}$$

where $\mathrm{d}S_e$ is the infinitesimal surface element with surface area S_e. $N_n(\boldsymbol{x})$ are so-called shape functions (interpolation functions) that distribute the traction field among the surface element's nodes.

The problematic singularity associated with the classical Volterra dislocation is avoided by using the non-singular formulation of Cai et al. [101] discussed in section 1.5.2, which changes eq. (1.14) into eq. (1.25),

$$G_{ij}(\boldsymbol{x} - \boldsymbol{x}') = \frac{1}{8\pi\mu} \left[\delta_{ij}\partial_{pp} - \frac{1}{2(1-\nu)}\partial_{ij}\right] R_a(\boldsymbol{x} - \boldsymbol{x}') \,. \tag{1.25}$$

Using the non-singular definition of $G(\boldsymbol{x} - \boldsymbol{x}')$ in eq. (1.13) we obtain the expression for the stress field of a single straight dislocation line segment bounded by two dislocation nodes (1, 2) [101],

$$\begin{aligned}
\boldsymbol{\sigma}^{(12)}(\boldsymbol{x}) = &- \frac{\mu}{8\pi} \int_{\boldsymbol{x_1}}^{\boldsymbol{x_2}} \left(\frac{2}{R_a^3} + \frac{3a^2}{R_a^5}\right) \left[(\boldsymbol{R} \times \boldsymbol{b}) \otimes \mathrm{d}\boldsymbol{x}' + \mathrm{d}\boldsymbol{x}' \otimes (\boldsymbol{R} \times \boldsymbol{b})\right] \\
&+ \frac{\mu}{4\pi(1-\nu)} \int_{\boldsymbol{x_1}}^{\boldsymbol{x_2}} \left(\frac{1}{R_a^3} + \frac{3a^2}{R_a^5}\right) \left[(\boldsymbol{R} \times \boldsymbol{b}) \cdot \mathrm{d}\boldsymbol{x}'\right] \mathbf{I}_2 \\
&- \frac{\mu}{4\pi(1-\nu)} \int_{\boldsymbol{x_1}}^{\boldsymbol{x_2}} \frac{1}{R_a^3} \left[(\boldsymbol{b} \times \mathrm{d}\boldsymbol{x}') \otimes \boldsymbol{R} + \boldsymbol{R} \otimes (\boldsymbol{b} \times \mathrm{d}\boldsymbol{x}')\right] \\
&+ \frac{\mu}{4\pi(1-\nu)} \int_{\boldsymbol{x_1}}^{\boldsymbol{x_2}} \frac{3}{R_a^5} \left[(\boldsymbol{R} \times \boldsymbol{b}) \cdot \mathrm{d}\boldsymbol{x}'\right] \boldsymbol{R} \otimes \boldsymbol{R} \,,
\end{aligned} \tag{1.26}$$

Substituting eq. (1.26) into eq. (1.24) yields as many integrals as there are surface nodes. Their exact form depends on the shape functions used. The specifics are out of the scope of this work. For a rectangular surface element, the vector formulation can be converted into a set of local scalar coordinates via a series of

scalar projections of the set of basis vectors via eq. (1.27),

$$\begin{aligned} \boldsymbol{R} &= \boldsymbol{x} - \boldsymbol{x}' = y\boldsymbol{t} + r\boldsymbol{p} + s\boldsymbol{q}\,, & R_a &= \sqrt{\boldsymbol{R}\cdot\boldsymbol{R} + a^2}\,, \\ \boldsymbol{n} &= \boldsymbol{p}\times\boldsymbol{q}\,, & y &= \frac{\boldsymbol{R}\cdot\boldsymbol{n}}{\boldsymbol{t}\cdot\boldsymbol{n}}\,, \\ r &= \frac{\boldsymbol{R}\cdot(\boldsymbol{q}\times\boldsymbol{t})}{\boldsymbol{p}\cdot(\boldsymbol{q}\times\boldsymbol{t})}\,, & s &= \frac{\boldsymbol{R}\cdot(\boldsymbol{p}\times\boldsymbol{t})}{\boldsymbol{q}\cdot(\boldsymbol{p}\times\boldsymbol{t})}\,, \\ \mathrm{d}\boldsymbol{x}' &= -\boldsymbol{t}\,\mathrm{d}y\,, \end{aligned} \tag{1.27}$$

where $\boldsymbol{n},\ \boldsymbol{p},\ \boldsymbol{q},\ \boldsymbol{t}$ are basis vectors[3] that respectively correspond to the outside surface normal of the surface element, the two orthogonal vectors that define the sides of the rectangular surface element, and the dislocation line direction. The integral bounds can then be set up accordingly.

Noting that $\boldsymbol{n} \equiv \boldsymbol{p}\times\boldsymbol{q}$ is normal to the surface element, if $\boldsymbol{t}$ is parallel to the surface element i.e. a linear combination of $\boldsymbol{p},\ \boldsymbol{q}$ then the product $\boldsymbol{t}\cdot\boldsymbol{n} = 0$, so this formulation is problematic when a dislocation line segment is parallel to a surface. An easy workaround is to slightly around the axis defined by $\boldsymbol{t}\times\boldsymbol{n}$ using the line's midpoint as a fulcrum. This works well in serial code but is problematic for parallelisation due to warp divergence (see section 1.5.5).

Figure 1.10 diagramatically shows the definition of the four basis vectors and summarises the line integrals needed to find an analytical expression of the nodal force. It is worth noting that over 40 triple integrals arise from eqs. (1.26) and (1.27). These were solved via three families of recurrence relationships found via integration by parts and six seed functions found by direct integration. A more detailed description can be found in [86].

Quadratic triangular surface elements pose a much more challenging problem due to the fact that $\boldsymbol{p} \not\perp \boldsymbol{q}$. This property is not the only addition that significantly complicates the integrals, the interpolation functions are quadratic which increase the number of integrals. The non-orthogonality of $\boldsymbol{q},\ \boldsymbol{p}$ means the integration limits in the $r,\ s$ coordinates must be parametrised appropriately and the integrations should be carried out in an order suitable for the chosen parametrisation. Aside from these technical considerations, the techniques for solving the problem for quadratic rectangular surface elements are the same as the linear rectangular case [86]. It is worth noting however, that the implementation of such a procedure may require quadruple precision arithmetic.

[3]Unitary vectors that define a space. This space is not orthogonal because in general only $\boldsymbol{p}\perp\boldsymbol{q}$, $\boldsymbol{n}\perp\boldsymbol{p}$ and $\boldsymbol{n}\perp\boldsymbol{q}$.

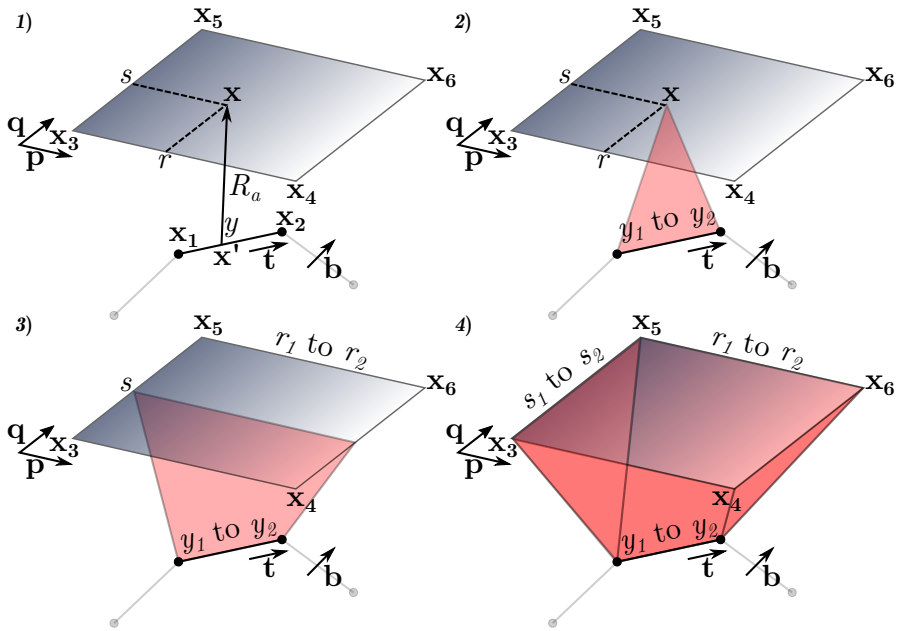


Figure 1.10: Diagram of the line integral method used to find analytical expressions for the forces exerted by dislocation on linear rectangular surface elements [86]. *1)* For any given point $\boldsymbol{x}$ on the surface element and any given point $\boldsymbol{x}'$ on the dislocation line segment, define distance R_a. *2)* Integrate from $x_1 \to x_2$ along line direction $\boldsymbol{t}$. *3)* Integrate from $r_1 \to r_2$ along vector $\boldsymbol{p}$. *4)* Integrate from $s_1 \to s_2$ along vector $\boldsymbol{q}$.

1.5.4 Multiphase Simulations

Modelling heterogeneity in DDD remains challenging. The core assumption of DDD is that we have a homogenous linear elastic space with invariant properties. However, modelling multiphase systems such as multicrystals and inclusions necessitates a weakening of such assumptions, particularly the space's homogeneity. Multiphase models are also necessary to apply DDD to more realistic scenarios.

Fusion in particular is riddled with examples of highly heterogenous materials. As previously mentioned, neutron bombardment often causes clustering of transmutation products particularly on divertor and first wall materials; such clusters often substantially change many of their mechanical and thermal properties. The whole point of Oxide-Dispersion Strengthened (ODS) steels [112] is to hamper radiation damage by distributing oxide inclusions within their matrix. If we hope to model such alloys with DDD we must find a way to relax our homogeneity criteria.

Furthermore, many potential candidates for structural and even first wall materials of fusion reactors are designed with complex microstructures in order to resist thermal creep and radiation damage. Understanding dislocation behaviours near the crystal boundaries of these microstructurally complex materials is another niche where DDD simulations can shed light into the mechanisms that give

rise to these materials' desirable properties.

1.5.4.1 Polycrystalline Materials

Polycrystalline materials have been studied using DDD, however most of these are in 2D. One of the easiest phenomena to investigate with 2D DDD are grain size effects [113, 114].

It is well known that much of the grain size strengthening effects arise from dislocation-grain boundary (GB) interactions including transmission, reflection, emission and absorption of dislocations [115, 116]; with transmission being the most typically observed.

The model depends on the grain boundary energy density, $E_{\mathrm{GB}}(\delta\theta)$, where $\delta\theta$ is the crystallographic misorientation between neighbouring crystals; resolved shear stress, τ, on the incoming dislocation with Burgers vector $\boldsymbol{b_1}$; critical penetration stress for the GB, τ_{GB}; and Burgers vector of the dislocation debris, $\boldsymbol{\Delta b} = \boldsymbol{b_1} - \boldsymbol{b_2}$, left behind when the incoming dislocation, $\boldsymbol{b_1}$, transmits throught the grain boundary to become a dislocation with Burgers vector, $\boldsymbol{b_2}$. The relationship may be approximated by,

$$\tau|\boldsymbol{b_1}|^2 \geq \tau_{\mathrm{pass}}|\boldsymbol{b_1}|^2 = E_{\mathrm{GB}}(\theta)|\boldsymbol{b_1}| + \alpha G|\boldsymbol{\Delta b}|^2\,, \tag{1.28}$$

where α is the material constant and G the shear modulus. The grain boundary energy density was proposed by [117] to have the simple form,

$$E_{\mathrm{GB}} = \begin{cases} k\dfrac{\delta\theta}{\theta_1} & 0 \leq \delta\theta < \theta_1\,, \\ k & \theta_1 \leq \delta\theta < \theta_2\,, \\ k\dfrac{\pi/2 - \delta\theta}{\pi/2 - \theta_2} & \theta_2 \leq \delta\theta < \pi/2\,, \end{cases} \tag{1.29}$$

where $k,\ \theta_1,\ \theta_2$, are material specific.

This is of course a gross simplification of the real 3D problem, but this approach was used by Li et al. [113] to investigate the Hall-Petch effect, which correlates grain size with flow stress of a polycrystal,

$$\sigma = \sigma_0 + \kappa\left(\frac{d_0}{d}\right)^n\,, \tag{1.30}$$

where σ_0 is the yield stress, d_0 a reference crystal size, κ the Hall-Petch slope and n the crystal size sensitivity parameter. Li et al. [113] showed that the model reproduces the Hall-Petch effect quite successfully. As a consequence, they showed that even what might appear as an overly simplistic approach can describe a

complex emergent phenomenon such as the Hall-Petch effect.

Aside from the Hall-Petch effect, the model has also been utilised by [114] to study the thickness effects of three different types of polycrystalline thin films: *1*) no surface treatment, *2*) surface passivation layer and *3*) surface grain refinement zones . In this study, Frank-Reed sources were seeded across the domain, and the superposition principle was utilised to calculate displacement, strain and stresses in the thin films. Their results qualitatively reproduced experimental observations from expected dislocation patterns, stress distributions, and the eventual disappearance of the size effect as the films got thicker [114].

Two-dimensional models might seem useless but they have their place, particularly when thin films are concerned. However, they may also be of use when investigating the effects of dislocations on the mechanical behaviour of superconducting tape—whose applications range from medical imaging to fusion energy production. The exotic composition and crystallography of many superconductors (perovskites) would make 3D models very complex and computationally expensive, so 2D models may offer viable alternatives. On top of this, a superconducting tape's operating environment would often have it under strains which might not strictly lie in-plane, but given a small enough segment of tape, strains orthogonal to the its plane may be neglected. Thus making 2D models an acceptable first attempt at tacking the problem, at leas before computational resources and developments in 3D models make more realistic studies of such complex systems possible.

The 3D case is substantially more complicated than the 2D case. A recent study by Fan et al. [118] looked at the role of twinning on the hardening response of polycrystalline Mg using 3D DDD.

The article is a perfect example of why 3D DDD is so much more complex than 2D DDD. Admitedly, it uses a material with a HCP crystal structure which complicates matters compared to FCC or BCC.

In 3D DDD one must account for dislocation-twin boundary (TB) interactions. These may be obtained via geometric considerations, MD, and experimental evidence [119–122]. We must have information regarding how dislocations are transmitted through a TB such as *1*) whether a dislocation leaves a residual dislocation on the TB, *2*) the possible pre-TB and post-TB slip planes a dislocation can be transmitted to, *3*) which loading conditions are conducive to which transmission behaviour and *4*) which types of dislocation can be transmitted or reflected and in what ways [118]. All this obviously depends on the twinning plane, so in order to study more realistic scenarios one must have all the necessary knowledge to properly account for all dislocation-TB interactions. Furthermore, it is possible that certain dislocations may leave behind twinning dislocations that end up as

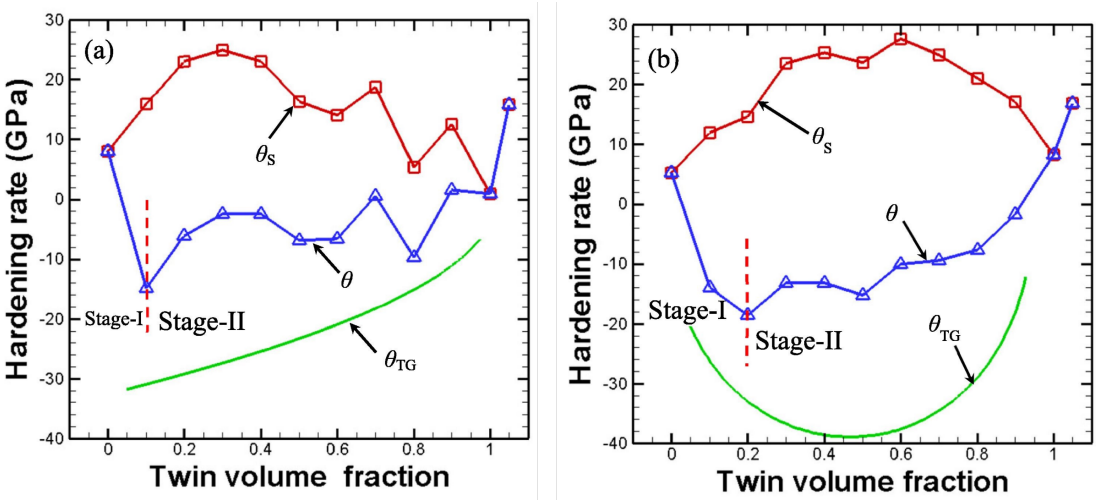


Figure 1.11: Simulated hardening rate as a function of twin volume fraction for (a) yz compressive loading and (b) tensile loading. Image edited from [118].

TB steps, whose movement can lead to TB migration and twin growth [123, 124].

Fan et al. [118] modelled four scenarios in order to deconvolute the effects that twins and grain boundaries have on the mechanical behaviour of a cubic crystal. They modelled *1*) a cubic twinned polycrustal with two TBs and GBs on the edges of the cube, *2*) a single crystal with no TBs or GBs, *3*) a polycrystal with no TBs but GBs, *4*) a twinned crystal with TBs but not GBs. It is worth noting that the effects of twin growth in plasticity were not accounted for in their DDD simulation and instead had to be numerically computed via the hardening rate θ,

$$\theta = \frac{\mathrm{d}\sigma}{\mathrm{d}\epsilon} = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\left[E\left(\epsilon - \epsilon^{\mathrm{p}}_{\mathrm{slip}} - \epsilon^{\mathrm{p}}_{\mathrm{twin}}\right)\right] = \theta_{\mathrm{s}} + \theta_{\mathrm{TG}}\,, \tag{1.31a}$$

$$\theta_{\mathrm{TG}} = -\frac{\mathrm{d}}{\mathrm{d}\epsilon}\left[E\epsilon^{\mathrm{p}}_{\mathrm{twin}}\right] = -E\bar{m}\gamma_{\mathrm{twin}}\frac{\mathrm{d}}{\mathrm{d}\epsilon}\left[f_{\mathrm{twin}}\right]\,, \tag{1.31b}$$

where θ_{TG} is the hardening rate due to twin growth, $\bar{m}$ the mean Schmid factor of the participating twins γ_{twin} the characteristic twinning shear and $\frac{\mathrm{d}}{\mathrm{d}\epsilon}\left[f_{\mathrm{twin}}\right]$ is the change in the change in the twin volume fraction with respect to applied strain. These parameters must be empirically obtained.

With their model, Fan et al. [118] managed to qualitatively reproduce experimental observations, including the concave shape of strain-stress curves that arises from the competing hardening effect of TBs restricting dislocation motion (fig. 1.11), and the softening produced by twin growth as obtained from eq. (1.31b).

Twin boundaries in 3D are analogous to grain boundaries in 2D, given that they are coherent, narrow boundaries rather than messy, often incoherent grain boundaries. Consequently, grain boundaries are much harder to treat in 3D; instead, the community focuses on multiphase models where the details of dis-

location transmission between phases are ignored or simplified as mentioned in section 1.5.4.2.

1.5.4.2 Inclusions

Inclusions have been modelled using the three methods described in section 1.5.1. However, they have gone about it differently. The SM and DCM apply their formulations heirarchically to allow for dislocation motion into and out of the inclusion. Furthermore, being coupled DDD-FEM problems, the inclusions must also be appropriately meshed, oftentimes this means inclusion geometries are limited to the mesh type. However, FEM allows for the calculation of coherency stresses which are seen by the dislocations and cause dislocation localisation and forresting in the vicinity of inclusions, just as experimentally observed [125, 126].

The level set method, having never been coupled to FEM, takes a different approach. Where the particles can be modelled as a region where dislocations experience a user-defined force. Coherency stresses have so far been neglected, but their effects may be modelled with the use of functions akin to diffuse orbital functions used in theoretical and computational quantum chemistry. Within this framework the inclusions may be arbitrarily defined to have any smooth shape, especially if one were to use piecewise parametric fourier curves.

Dislocation-permeable inclusions have been simulated via the SM [125]. As mentioned in section 1.5.1.1, there is a need to calculate polarisation stresses due to differences in the elasticity of both phases. Using the same notation as section 1.5.1.1 this can be done by breaking up the image stress into,

$$\hat{\boldsymbol{\sigma}} = {}^{\mathrm{m}}\boldsymbol{C}\hat{\boldsymbol{\epsilon}} \quad \in V_{\mathrm{m}} \tag{1.32a}$$

$$\hat{\boldsymbol{\sigma}} = {}^{\mathrm{p}}\boldsymbol{C}\hat{\boldsymbol{\epsilon}} + [{}^{\mathrm{p}}\boldsymbol{C} - {}^{\mathrm{m}}\boldsymbol{C}]\,\tilde{\boldsymbol{\epsilon}} \quad \in V_{\mathrm{p}}\,, \tag{1.32b}$$

where m, p denote whether a variable belongs to the matrix or precipitate respectively and $\boldsymbol{C}$ is the elastic stiffness tensor.

Yashiro et al. [125] not only modelled a cuboid inclusion, but bimetallic interfaces in 3D. They found that under the right conditions, dislocations can pass from one phase to the other. When a dislocation passes into a different phase, it leaves an antiphase boundary on the slip plane. Therefore, in order for a dislocation to move from one phase to another, it needs excess energy equal to the antiphase (APB) energy of the area it sweeps within the precipitate. This means that as a node passes from one phase to another, it feels a repulsive force F_b. The next dislocation moving along the same APB will then feel an attractive force F_b to dissolve the APB. Once both dislocations move into the new matrix, they form a superdislocation bound by the APB energy [127]. However, as the disloca-

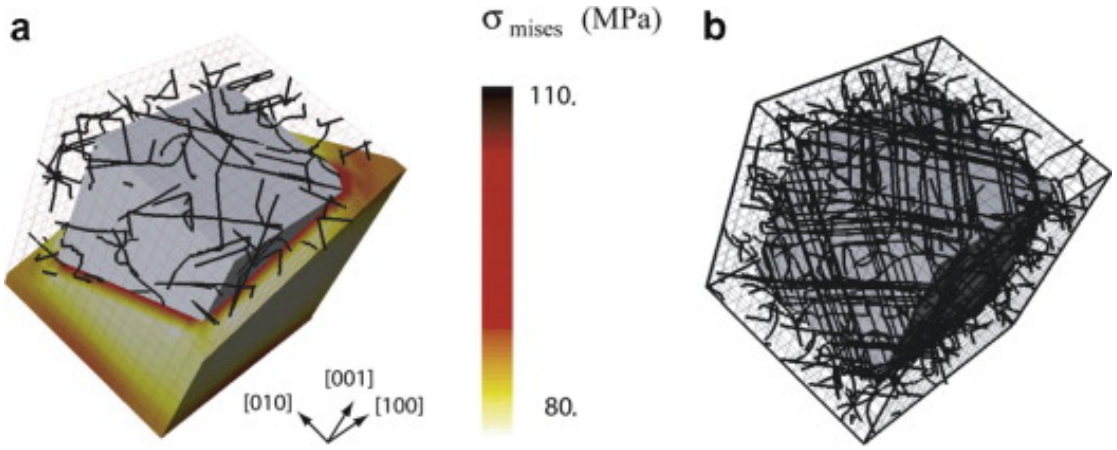


Figure 1.12: (a) Shows the initial dislocation configuration after relaxation in the presence of the calculated von Mises coherency stresses $\boldsymbol{\sigma}_{\text{mises}}$. (b) Shows the dislocation forresting around the inclusion as a result of 0.2% plastic strain for the [001] case. Image obtained from [126].

tion length increases with respect to the precipitate's volume, the production of Orowan loops around it becomes more energetically favourable than moving into the new phase [125, 127]. Both of these behaviours have been observed when using the superposition method.

The eigenstrain method [128] as described in section 1.5.1.2 has been shown to be a viable solution to this problem. One can compute the stresses and displacements produced by the dislocation ensemble on the inclusions' surfaces via DCM or SM. The SM method however needs to calculate polarisation stresses [88], an operation that significantly increases the number of calculations necessary to solve the problem.

The DCM method has been used to simulate multiphase materials where the simulated dislocations reproduced experimentally observed behaviours such as the zig-zag patterns and dislocation forests produced by dislocations around precipitates in nickel superalloys [95, 126]. Reproducing such behaviours is not as trivial as simply adding inclusions to the simulation domain. In order to produce accurate results, one requires knowledge of the coherency stress between the different phases [126]. Such stresses are often due to lattice mismatches and differences in the thermal expansion coefficients. These coherency stresses localise the dislocations around the inclusions and are responsible for the observed localisation and aggregation of dislocations around precipitates as seen in fig. 1.12 [126].

The level set method has also been successfully used to model inclusions. In contrast to the aforementioned methods, they are assumed to be regions in space where dislocations experience a repelling force, instead of a region in space with distinct characteristics to the matrix. The force profile can be very simply defined to depend on the radial distance from the dislocation to a point in the "inclusion's"

volume [96]. It can also be modulated depending on the nature of the inclusion by making the force a radial function, thus avoiding many of the numerical issues facing DDD-FEM couplings (see sections 1.5.1 and 1.5.3). It is also possible to make the inclusions impermeable or semi-permeable depending on the magnitude and steepness of the force gradient as presented in [96]. Arbitrarily shaped may also be possible if one were to also use angular components to the force function, or even through the use of spherical harmonics.

Though mathematically appealing, the level set method is far from representing the reality of dislocation-inclusion interactions. More realistic scenarios require use of more commonplace DDD-FEM coupling methods. Implementing arbitrarily shaped particles is not impossible in DDD—particularly if they can be discretised. Aside from the significantly lower computational requirements, one of the biggest advantages of DDD is that one can find tractions and displacements natively through either the DCM or SM, whereas the level set method requires the force to be user-defined and displacements on the inclusion are completely ignored. Furthermore, if the inclusion is a different crystal or a twin of the same material as in [118], it is possible to use the DCM or SM to study the transmission of dislocations from one phase to the other; something that is not yet possible to do with the level set method.

1.5.5 Parallelising Discrete Dislocation Dynamics

It has been shown that DDD lends itself well to parallelisation. Ferroni et al. [85] investigated some of the more computationally intensive parts of DDD in DDLab [129], a freely available DD code for Matlab. The algorithms whose parallel performance was studied in [85] were 1. segment-segment interaction, 2. surface tractions and 3. image stresses.

The segment-segment interaction has $\mathcal{O}(N^2)$ computational complexity, where N is the number of dislocation segments. The quadratic scaling is due to the r^{-1} dependence of the dislocation stress fields, so long range interactions cannot be ignored.

Both the surface traction and image stress calculations have $\mathcal{O}(kN)$ computational where k is the total number of surface nodes and N the number of dislocation line segments.

As is usually the case in parallel computing, there is often more than one way to parallelise a problem. The optimal strategy depends on the problem itself. It was no different in [85] as shown in fig. 1.13. Ferroni et al. [85] realised that the segment-segment interaction calculation can be done via $\mathcal{O}(N^2)$ parallelism, where single pairs of dislocation segments are sent to a single thread, and then globally

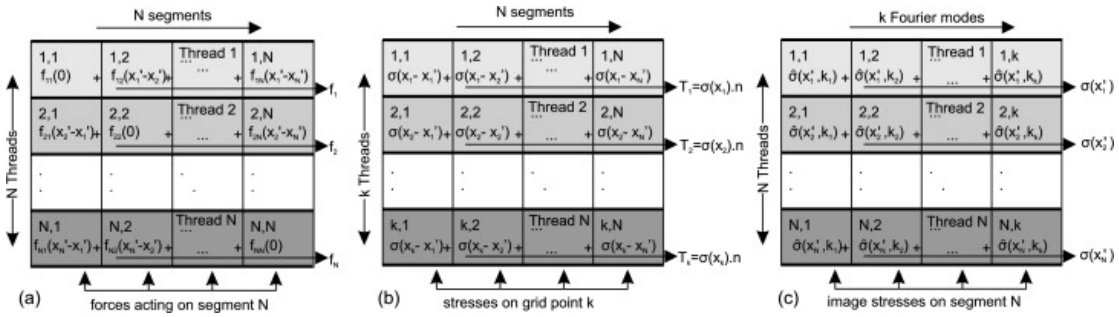


Figure 1.13: Parallelisation strategies for: (a) the N^2 segment interactions; (b) surface traction at k grid points and; (c) and image stresses on N segments. Image taken from [85].

reduced. However this approach requires $\mathcal{O}(N^2)$ memory and can lead to inefficient data reuse. So they opted instead for $\mathcal{O}(N)$ parallelisation and serialisation. In this scheme each thread is assigned a unique segment, and each thread computes the forces between its assigned segment and all the others while serially adding the force contributions. The memory access pattern they found best was that of a serial procedure, the reason is that in-thread calculations are serialised so having coalesced memory access patterns for threads would cause problems when serially accessing data for the calculation.

However, there is another approach Ferroni et al. [85] did not consider. It combines both of the aforementioned strategies, $\mathcal{O}(N^2)$ memory and $\mathcal{O}(N)$ parallelisation and serialisation. Such an approach requires two copies of the input data (one per access pattern). One pattern would allow each thread to obtain its uniquely assigned segment in a coalesced manner. The second would be the serial access pattern already present. Thus obtaining the best performance possible performance at the cost of doubling the required memory. This is a better strategy provided there is enough device memory[4], if that is not so, the problem must be split into different GPU calls.

New NVidia architectures have made varying degrees of nested parallelisation possible, NVidia calls this "dynamic parallism". When applied to the computationally optimal solution, the $\mathcal{O}(N^2)$ memory requirement is relaxed to $\mathcal{O}(N)$ (with the same parallel-specific data structure) and $\mathcal{O}(N^2)$ computational parallelism.

The parallisation strategies for the other two problems, (b) and (c) in fig. 1.13,

[4]Assuming the line direction is calculated inside the program, each dislocation line segment represents 3, 3-element vectors of type double (8-bytes per entry), totalling 72 bytes in global memory per dislocation line segment (two nodes + Burgers vector per segment). There must also be enough global memory to output the forces, which represent 1, 3-element vector of type double per dislocation node, totalling 24 bytes per node. Assuming a single dislocation loop with N nodes (therefore N segments), this brings the total global memory requirement to $96N$ bytes. Using two access patterns for the dislocation line segments and Burgers vector means an extra $72N$ bytes, giving a grand total of $168N$ bytes. Assuming the material parameters are stored in cosntant memory.

involved similar decompositions as the segment-segment interaction. For surface tractions Ferroni et al. [85] parallelised over k surface nodes with sequential calculation and addition of the stress from each dislocation line segment. For the image stresses, they chose the reverse strategy since it allows for the serialised addition of the image stresses on dislocation line segments.

Their findings showed very promising results for parallelising DDD simulations. Compared to C, their findings showed that depending on the specific problem and parameters used, the parallel implementation reduced computational time by a factor of ~ 110 at best and ~ 3 at worst for sufficiently large problems. This is of course, provided one uses a card designed for high performance computing, as these are tailored towards double precision arithmetic and computationally intensive procedures. However, gaming graphics cards are increasingly capable of double precision arithmetic and though not as drastic, speed ups are still substantial.

1.6 Project Outline and New Science

If anything should be clear from this work is that materials science is far from being solved. Even within the small niche of dislocation dynamics modelling, there are a huge number of open problems and an equally large number of solutions; each with its particular sets of advantages, disadvantages, ideosyncrasies and challenges.

The rapid advancement in computer technology; pushed by the ever-increasing complexity of the problems that occupy humankind; but simultaneously restricted by the physical constraints of classical computers, have opened up new avenues for research. Research that blurs the boundaries between fields and produces something new. Undoubtedly materials science sprang up this way, when metallurgists decided to take elements from the fields of chemistry, physics and mathematics to create a new discipline.

With increasing frequency, one finds people across vastly different fields who can understand each other via a common language. The language of scientific computing, where fields intertwine and produce new research methods; where techniques developed accross disciplines are taken, improved and adapted to work on the individual researcher's problem of interest.

This project applies such techniques to accelerate and improve the ease with which outstanding problems in materials science can be tackled; making use of more accurate models, hardware acceleration and the techniques of high performance computing to enable larger and more complex simulations than ever before—striving for increasingly faithful and insightful recreations of reality with a user-friendly code on a desktop PC.

Some of the work presented herein has already proven useful in the simulation

of complex simulations such as nano-indentation [130] and several other research projects within the Tarleton group. Here we showcase the new capabilities, findings and methods; improvements, fixes and redesign of the codebase; as well as a showcase of relatively simple, yet formerly inaccessible simulations resulting from this work. Finally, we present an ambitious proposal for future work resulting from the insights gained during the project.

Chapter 2

EasyDD v2.0

EasyDD can be found in https://github.com/TarletonGroup/EasyDD.

As mentioned in section 1.6, this project aims to integrate multidisciplinary skills for creating increasingly faithful recreations of reality in a user friendly way. This chapter details the software engineering that moved us closer to this goal and yielded a much improved version of EasyDD.

2.1 Bug fixes

2.1.1 Correct time-adaptive integrator

The first obvious hurdle to overcome when making more complex simulations possible was the tremendously long computational time taken for a dislocation plasticity simulation to sufficiently advance past the elastic regime. A simple microcantilever bending simulation with a few frank reed sources would take a whole night to reach the plastic regime, some early simulations by Bruce Bromage took days to get there.

The unacceptable slowness prompted a closer examination of the adaptive-time integration method found in algorithm 10.2 and described by equations (10.42, 10.45 and 10.46) found in [129, p. 214–216], an explicit Euler-trapezoid predictor-corrector solver. These are algorithm 2.1 and eq. (2.1),

$$\boldsymbol{r}_i^{\mathrm{P}}(t+\Delta t) = \boldsymbol{r}_i(t) + \boldsymbol{g}_i\left(\{\boldsymbol{r}_j(t)\}\right)\Delta t\,, \tag{2.1}$$

$$\boldsymbol{r}_i(t+\Delta t) = \boldsymbol{r}_i + \frac{\boldsymbol{g}_i(\{\boldsymbol{r}_j(t)\}) + \boldsymbol{g}_i\left(\left\{\boldsymbol{r}_i^{\mathrm{P}}(t+\Delta t)\right\}\right)}{2}\Delta t\,, \tag{2.2}$$

$$\boldsymbol{v}_i \coloneqq \frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t} = \boldsymbol{g}_i\left(\{\boldsymbol{r}_j\}\right)\,, \tag{2.3}$$

where $\boldsymbol{r}_i$ are nodal coordinates, $\boldsymbol{v}_i$ are nodal velocities, $\boldsymbol{g}_i$ is the function that uses the mobility model and nodal forces to compute the nodal velocities, and the

superscript P denotes the predictor. This solver is fast and accurate for a small enough timestep Δt. Two free parameters $\Delta t_{\max}$ and ϵ denote the maximum allowed timestep and accuracy.

Algorithm 2.1 Adaptive Euler-trapezoid predictor-corrector algorithm.

1. Initialise time step $\Delta t \coloneqq \Delta t_{\max}$.
2. $\Delta t_0 \coloneqq \Delta t$.
3. Compute $\boldsymbol{r}_i^{\mathrm{P}}(t + \Delta t)$ and corrector $\boldsymbol{r}_i(t + \Delta t)$ from eq. (2.1).
4. If $\max_i \left(\| \boldsymbol{r}_i^{\mathrm{P}}(t + \Delta t) - \boldsymbol{r}_i(t + \Delta t) \| \right) > \epsilon$, reduce time step $\Delta t \coloneqq \Delta t/2$ and go to 3.
5. $t \coloneqq t + \Delta t$.
6. If $\Delta t = \Delta t_0$, increase time step to $\Delta t \coloneqq \min(1.2\Delta t, \Delta t_{\max})$.
7. Return to 2, unless total number of cycles is reached.

The algorithm being used was not exactly this algorithm 2.1, as it used a different way of calculating the error but other than that they were the same. Unfortunately, this only increases the time step after converging to a satisfactory answer and advancing the time. So if anything caused the timestep to decrease, such as a collision or dislocation reaction, the time step would only increase very little every subsequent step, thus eading to unecessarily small time steps in most cases. The original implementation of this is found in commit `65907b0` under the name `int_trapezoid.m`. An improved algorithm is described in algorithm 2.2. This algorithm takes close to the maximum allowed time step for the maximum allowable accuracy, ($\epsilon_{\max}$, $\upsilon_{\max}$), maximum and minimum timestep, ($\Delta t_{\max}$, $\Delta t_{\min}$) and number of iterations, $\mathrm{iter}_{\max}$. It also uses a better error heuristic and more conservative timestep increase. It can be found in the most current version of `int_trapezoid.m`.

Algorithm 2.2 is more computationally expensive per iteration, but can increase the timestep much more rapidly than algorithm 2.1 without going through the rest of the simulation, which involves computationally expensive procedures. This improvement by itself let the early version of the software move through the elastic regime of our simulations orders of magnitude times faster than before. Simulations that took hours to days to reach the plastic regime, started doing so in minutes to hours. It also narrowed the gap between using different loading conditions, as the simulations could now effectively adjust the timestep to match the error tolerances much more easily than before. Both of which were a great

Algorithm 2.2 Improved adaptive timestep algorithm.

Convergent $\coloneqq$ false, $\Delta t_{\text{valid}} \coloneqq 0$, iter $\coloneqq 0$, flag $\coloneqq$ false
while Convergent **do**
  Compute $\boldsymbol{r}_i^{\text{P}}(t+\Delta t)$ and corrector $\boldsymbol{r}_i(t+\Delta t)$ from eq. (2.1).
  $\Delta \boldsymbol{r}_i \coloneqq \boldsymbol{r}_i(t+\Delta t) - \boldsymbol{r}_i^{\text{P}}(t+\Delta t)$
  $\overline{\boldsymbol{r}}_i \coloneqq \dfrac{\boldsymbol{g}_i(\{\boldsymbol{r}_j(t)\}) + \boldsymbol{g}_i\left(\left\{\boldsymbol{r}_i^{\text{P}}(t+\Delta t)\right\}\right)}{2}\Delta t$
  $\epsilon \coloneqq \max_i\left(\|\Delta \boldsymbol{r}_i\|\right)$
  $\upsilon \coloneqq \max_i\left(\|\Delta \boldsymbol{r}_i - \overline{\boldsymbol{r}}_i\|\right)$
  if $\epsilon > \epsilon_{\max}$ and $\upsilon > \upsilon_{\max}$ **then**
    $\Delta t_{\text{valid}} = \Delta t$
    $\gamma \coloneqq 1.2\left(\dfrac{1}{[1+(1.2^{20}-1)(\epsilon/\epsilon_{\max})]}\right)^{1/20}$
    flag $\coloneqq$ true
    iter $\coloneqq$ iter $+ 1$
    $\Delta t \coloneqq \max\left(\gamma \Delta t,\ \Delta t_{\max}\right)$
  else
    if flag == true **then**
      $\Delta t \coloneqq \Delta t_{\text{valid}}$
      iter $\coloneqq \text{iter}_{\max}$
    else
      $\Delta t \coloneqq \Delta t/2$
    end if
  end if
  if iter $> \text{iter}_{\max}$ or $\Delta t == \Delta t_{\max}$ **then**
    Convergent = true
  end if
  if $\Delta t < \Delta t_{\min}$ **then**
    $\Delta t \coloneqq \Delta t_{min}$
    iter $\coloneqq \text{iter}_{\max} + 1$
  end if
end while
$t \coloneqq t + \Delta t$
Proceed with the rest of the simulation.

boon to the usability of the code and to the research group's output capacity. This was the first major bottleneck to be fixed, which allowed us to identify and resolve previously unknown issues downstream.

2.1.2 Matrix conditioning

A common problem with dislocation dynamics is the tendancy for some networks to have a large degree of variation in the speed of dislocation nodes [129, 131, 132]. This is a consequence of the fact that in discrete dislocation dynamics, nodes are simply the discretised representation of a dislocation line. As such, nodal motion must be constrained to the motion of dislocations, i.e. nodal movement must be consistent with how a dislocation glides, climbs and cross lips accounting for the dislocation character and orientation in the crystal. Moreover, dislocation motion can be assumed to follow a linear, anisotropic viscous drag model [129],

$$\boldsymbol{F}_{\text{drag}}(\boldsymbol{x}) = -\mathcal{B}(\boldsymbol{\xi}(\boldsymbol{x})) \cdot \boldsymbol{v}(\boldsymbol{x})\,, \tag{2.4}$$

where $\mathcal{B}$ is a tensor constructed from the anisotropic drag that a dislocation node, $\boldsymbol{x}$, experiences as a result of its constrained velocity $\boldsymbol{v}(\boldsymbol{x})$ due to being part of a discritesed segment $\boldsymbol{\xi}(\boldsymbol{x})$. In simple terms, this means a node experiences very different resitances to moving along its allowed directions.

Different mobility laws have different parametrisations for these directions. A problem that often plagues traditional mobility laws[1] is the movement of a node along a line segment itself. Having a node move along a single line segment has no effect on the energy of a network, as seen in section 2.1.2. Both structures are equivalent because the line—and thus the dislocation—does not change from one to the other, merely the limits of the line integral change. Therefore nodal movement along a line should not contribute to the total drag, and therefore have a zero line drag coefficient. On the other end of the spectrum we have climb, which is orders of magnitude less favourable than glide. But this can make eq. (2.7) singular, so its value is the careful balancing act of keeping the matrix invertible whilst keeping the associated energy cost of moving along a line low. This also causes nodal speeds along line directions to be much higher than other directions, which often limits the step size an integrator can take.

In the overdamped regime, $\dfrac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}(\boldsymbol{x}) \to \boldsymbol{0}$, known driving forces (Peach-Köhler

[1] Bruce Bromage, of [133] has developed a new BCC mobility law, `mobbcc1_bb.m` which can be found in the most current version of EasyDD. It solves many of the issues with traditional laws and is our new go-to for BCC materials. Among the things it fixes are: unreasonable cross-slip, pencil glide, and the problematic line-direction parametrisation. The drag matrix, $\mathbf{B}$, can still be singular however, so this law also includes the scale-averaged Marquardt-Levenberg regularisation.

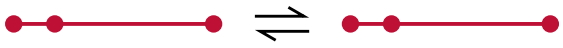


Figure 2.1: Node movement along dislocation line should have no drag contribution.

force + segment-segment forces + self-force) and the unkown drag force sum to give,

$$\boldsymbol{F}_{\text{drag}}(\boldsymbol{x}) + \boldsymbol{F}_{\text{drive}}(\boldsymbol{x}) = \boldsymbol{0}\,. \tag{2.5}$$

Substituting eq. (2.4) into eq. (2.5) and rearranging gives,

$$\mathcal{B}(\boldsymbol{\xi}(\boldsymbol{x})) \cdot \boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{F}_{\text{drive}}(\boldsymbol{x})\,. \tag{2.6}$$

Strictly speaking, solving eq. (2.4) requires global knowledge of the network. The discretisation makes it possible to distribute $\mathcal{B}$ along a line segment $i - j$,

$$\mathbf{B}_{ij} \coloneqq \oint_C N_i(\boldsymbol{x})\mathcal{B}(\boldsymbol{\xi}(\boldsymbol{x}))N_j(\boldsymbol{x})\mathrm{d}L\,, \tag{2.7}$$

where C is the whole network, N_i and N_j are simply shape functions that interpolate quantities given the relative position of a node to a segment, and $\mathrm{d}L$ is the infinitesimal line segment. Thus giving individual expressions for node i connected to node j. Using this discretisation and assuming the nodes connected to i are subject to similar conditions, eq. (2.6) can be broken up into local segments,

$$\boldsymbol{v}_i \approx \left(\sum_j \mathbf{B}_{ij}\right)^{-1} \cdot \boldsymbol{f}_i, \tag{2.8}$$

where i is the node in question, j are the nodes it shares a segment with, and $\boldsymbol{f}_i$ is the local force on the node. From now on, we will drop the Einstein notation and refer to the local frame as $\boldsymbol{v} = \mathbf{B}^{-1}\boldsymbol{f}$.

The immediate implication of the orders of magnitude difference between the drag coefficients used to construct $\mathcal{B}$—and therefore $\mathbf{B}$—means the condition number of $\sum_j \mathbf{B}$ can be very large. As $\mathbf{B}$ is symmetric and therefore normal, its condition number is given by,

$$\kappa(\mathbf{B}) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}\,, \tag{2.9}$$

where λ_i is the i^{th} eigenvalue.

The condition number of a matrix dictates how much collinearity exists in its basis. The larger the condition number, the larger the collinearity and the

more sensitive the sytem is to small perturbations. Such systems are said to be ill-conditioned. As a general rule, a condition number $\kappa(\mathbf{A}) = 10^k$ represents a loss of up to k digits of precision on top of what is already lost from arithmetic methods under limited precision [134].

True to Murphy's law, $\mathbf{B}$ is frequently ill-conditioned—even in the simplest simulations. The way these cases were handled relied on the integrator reducing the timestep enough to sidestep the issue. We call this "strongly coupled" behaviour. Strong coupling between functionally independent functions is a *very* bad practice in software engineering.

A function $\mathcal{X}$ with a set of $\mathbb{X}$ bugs with phase space $\chi(x)$ that is, $\forall x \in \mathbb{X}\ \chi(x)$. Bugs in production code are often either low impact with broad phase spaces, or high impact with narrow phase spaces. Both of which can be avoided with a combination of proper software design and testing protocols. All this goes out the window when functions are not properly decoupled. The multiplicative product of the corresponding phase spaces is often non-trivial, and can sometimes cancel out in some cases and blow up in others. Complicating the diagnosis and solution of errors.

The way the mobility functions dealt with ill-conditioned $\mathbf{B}$, relied on the integrator picking up the slack. There are several flaws with the approach found in algorithm 2.3, the actual mobility function can also be found under commit `65907b0` under the name `mobbcc1.m`. Every mobility law in that commit has this issue.

Algorithm 2.3 Avoiding singular matrix by making $\mathbf{B}$ extremely wrong and hoping the integrator error bounds pick it up and the timestep is decreased.

Compute eigenvalue matrix, $\mathbf{D} \coloneqq \mathbf{P}^{-1}\mathbf{B}\mathbf{P}$
$\kappa(\mathbf{B}) \coloneqq |\lambda_{\max}|/|\lambda_{\min}|$
if $\kappa(\mathbf{B}) < \kappa_{\min}$ **then**
    $\mathbf{D}_{\text{norm}} \coloneqq \mathbf{D}/\lambda_{\max}$
    $\boldsymbol{f} \coloneqq \boldsymbol{f}/\lambda_{\max}$
                                ▷ Invert $\mathbf{D}_{\text{norm}}$.
    for all $(\lambda_k \in \mathbf{D}_{\text{norm}}) \neq \lambda_{\max}$ **do**
        if $\lambda_k > \lambda_{\text{crit}}$ **then**
            $\lambda_k \coloneqq 1/\lambda_k$
        else
                                ▷ The inverse of 0 is not 0.
            $\lambda_k \coloneqq 0$
        end if
    end for
    $\boldsymbol{v} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1}\boldsymbol{f}$
else
    $\boldsymbol{v} = \mathbf{B}^{-1}\boldsymbol{f}$
end if

The first is assuming that an ill-conditioned matrix can be fixed by diagonalising. This is simply not true, perturbing the smallest eigenvalue in a direction such that κ improves is a good way of doing so without adding much error. However, perturbations of this type are often not enough when the condition number is too large. Furthermore, one wants to keep the overall behaviour of the system, i.e. maintain the relative sizes of the eigenvalues with respect to one another. If the multiple eigenvalues are close in magnitude to the smallest eigenvalue, one can change the dynamics of the system if the perturbation is too large. Not to mention the fact that κ may still be too large. In a system as stiff and dynamic as discrete dislocation dynamics, this approach is not enough for some cases.

The second flaw is the assumption that the case where $\lambda_k < \lambda_{\text{crit}}$ for non-maximal eigenvalues, is so rare, it won't have much of an effect. Unfortunately, it's common in junctions where nodal mobility is limited, as well as in volumes with large stress gradients. Although to be fair, these are also consequence of the assumption of locality, yielding eq. (2.8). Moreover, when every non-maximal eigenvalue is smaller than λ_{crit}, the error much greater. For a every eigenvalue under the threshold of of $\lambda_{\text{crit}} = 10^{-k}$, the upper bound of the error is k digits in the corresponding mobility component.

The third is the assumption that the integrator will not allow obviously wrong solutions because it will fail the error check and reduce the time step. This assumption breaks down whenever dislocation velocities are calculated outside the context of time integration, such as every topological operation where a new node is spawned.

The regularisation of ill-conditioned systems in an active field of research [135–137]. However, they require some knowledge of the matrix structure or follow a well-known statistical distribution to thier values/eigenvalues. Regularlisation is most often applied to high rank matrices because poorly conditioned low rank matrices ones are often very sensitive to perturbations, thus requiring specialised methods. In this case, $\mathbf{B}$ is of rank equal to the number of orthogonal crystallographic directions, so it is always of low rank.

One of the most effective regularisations for low rank matrices is dampening the inversion by adding a small value, c—called the Marquardt-Levenberg coefficient—to the diagonal of the matrix. This is roughly equivalent to perturbing the smallest eigenvalue as previously mentioned. Luckily, if this factor is added the whole diagonal, the inversion can be dampened sufficiently whilst keeping the main contributions from each basis roughly equal in relation to each other. This amounts to adding a multiple of the identity matrix to an ill-conditioned matrix. For a more accurate solution, one can iteratively refine c to find the smallest value that keeps the matrix inversion from blowing up. A good heuristic for the value

of the Marquardt-Levenberg coefficient is,

$$c = \max\left(|\mathbf{A}|\right)\sqrt(\epsilon)\,, \tag{2.10}$$

where ϵ is machine precision. This ensures the perturbation is scaled to a number that will not underflow during arithmetic operations. Simply put, it ensures the perturbation is small but does not disappear during matrix inversion.

It is worth noting that doing this and inverting is still a source of error, adding noise even if small and along the diagonal, does move the system away from its true behaviour. However, the degree to which it is wrong is many orders of magnitude smaller than what is done in algorithm 2.3. Indeed, algorithm 2.4 uses $\pm c$ and both perturbed solutions are averaged to give a much more accurate and simpler solution even under the traditionally problematic scenarios discussed above. This new algorithm is now present in the mobility laws we now use, such as the most current version of `mobbcc_bb.m`.

Algorithm 2.4 Improved regularisation of $\mathbf{B}$ by way of perturbing the diagonal.

if $\|\mathbf{B}\| < \epsilon$ **then**
  $\boldsymbol{v} \coloneqq \mathbf{0}$
else if $|\lambda_{\max}|/|\lambda_{\min}| < \epsilon$ **then**
  $\mathbf{B}_+ \coloneqq \mathbf{B} + \sqrt{\epsilon}\max(\mathbf{B})$
  $\mathbf{B}_- \coloneqq \mathbf{B} - \sqrt{\epsilon}\max(\mathbf{B})$
  $\boldsymbol{v}_+ \coloneqq \mathbf{B}_+^{-1}\boldsymbol{f}$
  $\boldsymbol{v}_- \coloneqq \mathbf{B}_-^{-1}\boldsymbol{f}$
  $\boldsymbol{v} \coloneqq (\boldsymbol{v}_- + \boldsymbol{v}_+)/2$
else
  $\boldsymbol{v} \coloneqq \mathbf{B}^{-1}\boldsymbol{f}$
end if

This change enabled other team members, namely Haiyang Yu and Fengxian Liu to get further with their simulations. Haiyang's nanoindentation simulations have particularly complex dislocation structures with a substantial amount of junction formation; while Fengxian's have large localised stress gradients as a result of inclusions. Both require nodal velocieties to be accurately calculated, else the dislocations move erratically, readily cross slip, and cause massive slowdowns as simulations advance.

2.2 Research software engineering

Many scientists and researchers spare no thought for good software. However, with the presently on-going SARS-CoV 2 pandemic, the importance of research software has been made clear. Modelling has played a large role in informing

government policy in the UK and the world [138–140]. With modelling thrust into the spotlight and the public release of the Imperial College modelling code used to inform so much policy [140], `https://github.com/mrc-ide/covid-sim`, criticism was levied at the lack of care taken to ensure the correctness of the code [141]. In experimental research, a lot of care is take to ensure a methodoogy used is reproducible, robust and sound. There are standards, both external and internal, which ensure the quality of the results. But even with these measures in place, there is already a common issue with irreproducibility in science [142–144]. In obtaining results and analysing them, software is ubiquitous. It should be good, *especially* if the science is modelling-based.

Along this vein, one of the primary goals of this project was to develop a competent codebase that non-experts can take advantage of with minimal preparation. There were multiple sub-objectives that help us achieve this.

1. Provide a generic framework upon which the software can be modularly expanded with minimal change to the source: prevents increasingly divergent and incompatible source code between researchers.

2. Provide an easy way to autocomplete inputs with sensible default parameters: prevents the software from crashing when—inevitably—one of the required arguments for a function is undefined.

3. Improve readability and organisation: makes it easy to identify, fix bugs and know what the code does.

4. Compartmentalise variables: makes the code more generic and increases usability and readability by reducing the number of function arguments.

5. Optimise memory and computation: improves computational speed and reduces resource use.

Akin to the Ship of Theseus, it is hard to tell whether the new version of the code is the same code as when this project began. While `EasyDD v2.0` is "complete", the process of improving the code is on-going. The amount of initial technical debt since its inception as the infinite-domain discrete dislocation dynamics code, `DDLab` [129], plus what it accrued as the Tarleton Group expanded its functionality and turned it into `EasyDD`, is quite large and needs to be carefully chipped away so as not to introduce regressions. However, while the quality and capabilities of the code can be greatly improved, there is only so much that can be done without fundamentally redesigning it completely. This is in part the subject of chapter 5.

Other team members have made significant contributions to the release of `EasyDD v2.0`. Some of which remain to be added, but should slowly make their way in. Given the code is the amalgam of the research group's work, we will credit other members' contributions where relevant but will leave the details for them to explain.

This is actively evolving on research software. Therefore the work described herein will continue past the end of this project, for there are still many improvements to be made to the old codebase on top of whatever functionality is added by future researchers.

2.2.1 Organisation

An oft-overlooked aspect of code useability is the organisational aspect. However as a codebase grows, it becomes more important for both a code be properly organised and source controlled. This makes it easier to browse, understand, is less overwhelming for users and developers and crucially provides a safey blanket against things going wrong.

2.2.1.1 Source control

Source control is a crucial aspect of industrial software development. It is an industry standard and the first thing to set up when starting a software development project. Academia is very far behind in this regard. Not only does it provide a history of restore points in case things break; but keeps a history of changes, who and when they made them; can perform automated checks such as testing or conflict detection; lets developers and users (if the repository is open-sourced) make changes independently and request their changes be added to the main code; among many other tasks.

In regards to reproducibility, source control is a crucial tool. In both sections 2.1.1 and 2.1.2 we took advantage of this to provide a specific commit and a file name, which lets anyone with access to a repository, the ability to see the entire history of the code. In the digital version of this document, those link commits are links directly to the file—and where relevant—specific lines. The advantages of such a system are obvious. For example, when submitting a manuscript for publication one can also submit a link to the specific version of the software used to obtain the results. This is a gigantic stride towards open, reproducible science.

2.2.1.2 Folder structure

An on-going task is improving the folder structure. Which in the past was almost non-existant. Having a default place to place inputs, outputs and source code is

of great import if a code is to be used by non-experts. It is trivial to do and a boon to the quality of life of developers and users.

2.2.2 Modularisation

Modern software practices usually make heavy use of some form of variable encapsulation and code modulirisation to make code safer, simpler to expand and easier to use. This enables users and developers to take advantage of a set of toolboxes that can be chosen according to their needs without having to directly modify the source code, this concept is called "moduliration".

2.2.2.1 Encapsulation

The act of keeping variables neatly stored in a way that fits a purpose is called encapsulation. By encapsulating variableswe greatly improve the usability of software by increasing readability, reducingthe risk of human error by reducing the number of things to keep track of, and if done properly can even improve performance. This concept is tightly bound to the design philosophy of software as described in the following list.

- Object-Oriented Programming (OOP): where variables are viewed as objects upon which the logic acts. Functions can take these objects and operate on them. This design pattern keeps related variables as part of a single entity and reduces the number of things users and developers have to keep track of, thus reducing the chance of implementation and user error. There are many advanced concepts and pitfalls in OOP that unecessarily increase complexity, so it is best to apply the KISS (keep it simple, stupid) approach for best results.

- Data-Oriven Programming (DDP): where the data dictates the structure of the programme. It is essentially a modified version of OOP where the object is made to fit the data, not the data made to fit the object. It often leads to more performant code and avoids many of the issues that arise from OOP because it forces developers to really think about how to manage the data.

- Functional programming: where everything is a function. There is no state, but also no side effects. This is the strongest form of encapsulation but also the most inflexible.

- Procedural programming: this is the most ancient form of programming where there are only functions and variables. It is the form most scientific software packages take and it is the most difficult to work with both, as

a developer, user. Even modern computers which can make use of branch prediction and deletion, cache pre-fetching, and compiler optimisations can be hindered by this approach.

Procedural programming is what gives us so-called spagghetti code[2]. Unfortunately, education in programming or software engineering in science is sorely lacking. As a result, a large amount of scientific software tends to be written in this way (particularly legacy software). Thus we strive to separate ourselves from it as much as possible.

Functional programming is impractical for most use cases so it is seldom used outside of computer science and very specific industrial and academic applications [145, 146].

Object-oriented programming can be very useful if used properly. But it has the potential to unecessarily increase complexity and reduce performance. Fortunately, `MATLAB` does not truly have object-oriented capabilities, so it forces either a procedural or data-driven approach. Unfortunately once more, it leads to software being written in a procedural manner.

This can be remedied by using `struct` and being sensible about, i.e. by taking a data-driven approach, in building them. While they are more of an ordered dictionary (Hash table), than a true object, they are sufficient for our use-case. They have allowed us to make use of generic functions for different mobility functions, loading conditions, finite element meshes, the dislocation network itself, etc.All of which have a massively simplifying effect on the code without sacrifing performance. While this project pioneered this task, Daniel Hortelano-Roig has been doing much in the way of increasing its scope. His changes greatly simplify the code and will slowly be ported over to the main branch.

2.2.2.2 Generic functions

Generic functions are implementation-agnostic functions that can be given by the user as any other variable. Only the variable in this case is a function. `MATLAB` has an old method of doing so using `feval()`, where the first argument is the name of the file whose function is being called, and subsequent arguments are passed on to the function. However this has some limitations, mainly their performance, and function handles are now preferred, e.g.

```
% myFunction is defined in 'myFunction.m' and is called
% like so myFunction(foo, bar)
```

[2]Code with many branches and disparate procedures without a clear purpose. It is difficult to work with as a user and developer. It also tends to interfere with branch prediction and deletion as well as compiler optimisations.

```
3     genericFunction = 'myFunction.m';
4     foo = 1;
5     bar = 2;
6     feval('genericFunction', foo, bar);
```

as opposed to,

```
1     % myFunction is defined in 'myFunction.m' and is called
2     % like so myFunction(foo, bar)
3     genericFunction = @myFunction;
4     foo = 1;
5     bar = 2;
6     genericFunction(foo, bar);
```

which are not dissimilar, but the latter is more performant and easier to follow. Generic functions were already in use for dislocation mobility but there are now ones for calculating numeric and analytic tractions (see chapter 4), loading conditions (multi-stage loading, constant loading, cyclic loading), various post-processing functions, boundary conditions and various miscelaneous functions that support different simulation types.

The main advantage of generic functions is reducing the need for direct source code modification. Changing source code, especially when there are no automated tests to speak of, is a dangerous proposition. It greatly increases the probability of introducing regressions (new bugs) and code divergeance between researchers. Regressions are always problematic, but code divergeance is a big issue that still affects the Tarleton Group. If experts within the same research group find it difficult to incorporate each others' additions to our work, there is little hope for the general user.

The introduction of regressions makes it so code must be thoroughly and exhaustively tested before merging two people's work. The use of generic functions makes it so if there is a problem with a new addition, the problem is localised and can be quickly identified and fixed because it is isolated. We can be sure the changes are only found in the new piece of code. And since it is generic, we can chose whether or not to use it by changing an input file rather than messing with the source code.

If on the other hand, one were to add a new branch to an `if` statement (or a brand new `if` statement) with new computations, likely the function needs new arguments to cover this new case, we may or may not need to make up- or downstream modifications so the function can accommodate the additions. Then very

quickly we will find ourselves procedurally programming and fixing the innevitable problems that come with hard to understand spaghetti code—which is most likely also less performant now than it used to be. The problem worsens as we add more functionality. Which can all be avoided by intelligently designing a generic approach, that can be easily expanded if the need arises.

2.2.3 Optimisation

> We *should* forget about small efficiencies, say about 97% of the time: pre-mature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.
>
> — *Donald E. Knuth [147, p. 268]*

2.2.3.1 Spurious memory allocation

`MATLAB` is infamous for the way it will happily dynamically grow an array when going out of bounds. This *terrible* practice is even fundmental to the operation of `EasyDD`. However, changing this fundamental part of the codebase requires a complete rewrite and not deemed worthwhile, there are worse bottlenecks. However, there were many instances of memory reallocated or arrays grown every cycle that were completely unecessary.

For example, the amount of memory unecessarily reallocated in the function that couples DDD and FEM was $12(N_x + 1)(N_y + 1)(N_z + 1)$ double precision floating point numbers, where N_i is the number of elements in dimension i of the finite domain. The cubic scaling is a performance killer at larger mesh sizes. In a simulation with hundreds or thousands of steps and a mesh with a few thousand FE nodes, the cost added up.

It is not only the cost of allocating, but also the cost of garbage collection [148] i.e. automatically freeing memory based on a set of criteria can be significantly higher. Timing the equivalent of allocating a $20 \times 20 \times 20$ mesh in `MATLAB` 2020b takes on the order of 70 µsec (for x86 CPU architectures), which is in-line to what it costs in `C`. Deallocating memory takes about 5× longer. In a function that otherwise takes a few m sec, adding another 400 µsec of spurious allocation is a significant overhead that can be easily remedied.

2.2.3.2 Finite element optimisation

The use of sparse arrays is a must when working with finite elements, where matrices often take structured sparse forms will significantly reduce memory footprint

and computational expense. The code already used these special data structures and Cholesky factorisation for faster solution of linear systems. However, `MATLAB` has a special form of it that exploits sparsity. This reduced the time required for the factorisation by around 50×, reduced the memory footprint of the final sparse arrays by a similar amount, and reduced the memory footprint of the actual factorisation procedure by approximately 10×. The whole point of this factorisation is to make solving linear systems faster, in this case the calculation of the corrective displacements, $\hat{\boldsymbol{u}}$ on the free boundaries, S_U of freedom as a result of the forces acting upon them,

$$\hat{\boldsymbol{u}} = \mathbf{K}^{-1}\boldsymbol{f} \quad \text{on } S_U \tag{2.11}$$

It improved solution speed by a factor of 3, thus reducing the cost of coupling DDD to FEM to be dominated by the computation of dislocation-induced tractions and displacements.

The final two optimisations also deal with memory. The first, removes unecessary allocations in the calculation of the global stiffness matrix and the introduction of a reduced stiffness matrix such that it only includes relevant degrees of freedom.

In aggregate these changes have given us the ability to use 15× more nodes as well as letting us build meshes over two orders of magnitude faster, mostly down to improved memory access patterns inside the mesh building loops.

2.2.4 Misc quality of life improvements

Formerly, initialising a simulation would involve a series of non-obvious steps, particularly when initially compiling the C and CUDA code used for accelerating simulations. This is now done automatically on a need-to-compile basis and has a fallback in case no `CUDA` compiler is found.

There was also a recurring problem with simulations suddenly failing when a function was called with an undefined argument. This extremely common scenario is unbecoming of good software, especially if the failure happened a few minutes after starting a simulation. This has been remedied with the automatic calculation of a set of reasonable defaults for each and every undefined variable, aside from the arrays defining a dislocation network. Any new developments get added to this very simple, yet much needed script.

2.2.5 Summary

Scientific modelling is inherently an interdisciplinary skill. EasyDD is now faster, more efficient and more capable than ever before, in large part thanks to the work—ongoing or otherwise—described within this chapter. Subsequent chapters explore aspects more specific to discrete dislocation dynamics. However, it is important not to overlook the work that enables the obtention of results. Modern academia deems the final result to be paramount. But what is a result if it cannot be scrutinised because nobody can understand how it came about? Moreover, by developing good software, we allow others to partake in the obtention of results. We deem this to be one of the main contributions of the present project. We feel safe our ability to claim that we now have the capacity for providing more faithful recreations of reality in a less user-unfriendly way. And without so much as a glancing touch on dislocation dynamics.

Chapter 3

Topology Improvements

3.1 Hinge collision prioritisation

3.2 Collision distance prioritisation

3.3 Collision-separation Loop

3.4 Remeshing

Chapter 4

Dislocation-induced surface tractions

Chapter 5

Future work

It is clear that there are some fundamental problems preventing us from exploiting the capabilities of discrete dislocation dynamics. The code DDLab, the code that EasyDD is based on made some decisions that would ultimately limit its potential. Namely, the lack of attention paid to the cost of dynamic memory allocation, procedural nature of the code, and its choice of programming language. These choices were reasonable upon inception, but as the Tarleton group has increased in size and maturity it has become evident that it is not enough. We are reaching the upper limits of what can be done with `MATLAB`. There are more things we can do, more things we are doing. But a constant obstacle in obtaining results is how long they take to get and how difficult it is to add new functionality, even changing boundary conditions.

The work described in chapters 2 to 4 has done much in not only producing more accurate simulations, but doing so faster. Fengxian Liu's work on modelling climb, diffusion and inclusions would greatly benefit from better designed code. Haiyang Yu's work on modelling hydrogen [130] and nanoindentation would also stand to benefit a great deal from a parallelised and faster dislocation separation algorithm. Potential work relevant to modelling dislocation dynamics in nuclear reactors such as post-damage cascade dislocation dynamics [149], necessitates spontaneous generation of dislocations as a network evolves. Including stochasticity in the form of Langevin dynamics [150] is also highly relevant, especially at higher temperatures.

Many of these require a complete overhaul to some of the most fundamental parts of EasyDD. Even so it would be hindered by the fact that `MATLAB` is a scripting language, not designed to be a scientific computing one. It lacks many modern features that make life easy when dealing with large codebases. Furthermore, the need for licences can represent a prohibitative barrier for users and researchers. Not only this, but the cost of distributed licences that allow `MATLAB` to run on clusters severely limits scalability and potential for collaboration.

But there is a better way. Prompted by the public release of the Imperial College COVID-19 source code `https://github.com/mrc-ide/covid-sim`, and the recurring issues with computational efficiency, compiler compatibility and availability, and fundamental technical debt inherited from `DDLab` [129]. We started a weekend project that has turned out to be *extremely* promising.

The language of choice is `Julia` [151]. An open-source JIT (just in time) compiled language designed for scientific computing. It has many features that make it an excellent candidate for a new generation of open, scientific software.

1. Multiple dispatch: methods are dispatched and compiled based on the types of all their arguments, the types propagate their way through child functions, creating optimal code without having to worry about data types.

2. Type system: its type system is based on set theory, where types are arranged in a genealogical tree with broader types giving way to narrower ones, e.g. `Float64 <: AbstractFloat <: Real <: Number <: Any == true`. Structures can be parametric on type. These features let developers create generic code without bothering with type annotations. This type of code greatly improves modularity and allows for "magic" like automatic differnetiation without the programme knowing the rules of calculus.

3. CPU and GPU Parallelisation: parallelising loops is trivial.

4. Metaprogramming: lets code write code. This can be used to autoparallelise functions without rewriting. It also lets developers define custom syntax and precompute values like an extremely powerful C-preprocessor.

5. Modern features: `Julia` has all the features of a modern programming language. From a booming and quickly growing package ecosystem already with a few best in class libraries; an integrated testing system and documentation generation; extremely powerful code introspection tools; to interctivity.

The code can be found here https://github.com/dcelisgarza/DDD.jl. As it has been a didactic side project, it is nowhere near feature complete. But so far, comparisons with C-accelerated portions of EasyDD peg it at around 10% faster. Direct comparisons to pure `MATLAB` show between 5 to $> 1000\times$ faster depending on the function. It is the result of the lessons learned while working on EasyDD. Its design avoids the same pitfalls and has a clear vision for usability, modularity, and extensibility, without sacrificing performance. We think it to be a more than worthwhile endeavour to bite the bullet now rather than wait until there is no more juice to squeeze out of `MATLAB`. We are looking to continue this as a research software engineering project. Not only so that the shackles of proprietary software can be broken; but also so the full, ambitious vision of EasyDD can be realised.

Bibliography

[1] Ellen G Howe and Ulrich Petersen. Silver and lead in the late prehistory of the mantaro valley, peru. *Archaeometry of pre-Columbian sites and artifacts*, pages 183–198, 1994.

[2] John Chapman. *Fragmentation in archaeology: people, places and broken objects in the prehistory of south eastern Europe*. Routledge, 2013.

[3] Kris MY Law and Marko Kesti. *Yin Yang and Organizational Performance: Five elements for improvement and success*. Springer, 2014.

[4] Bryan K Hanks and Katheryn M Linduff. *Social complexity in prehistoric Eurasia: Monuments, metals and mobility*. Cambridge University Press, 2009.

[5] Jared M Diamond. *Guns, germs and steel: a short history of everybody for the last 13,000 years*. Random House, 1998.

[6] Arthur Wilson. *The living rock: the story of metals since earliest times and their impact on developing civilization*. Woodhead Publishing, 1994.

[7] EW Hart. Theory of the tensile test. *Acta metallurgica*, 15(2):351–355, 1967.

[8] SU Benscoter. A theory of torsion bending for multicell beams. *Journal of Applied Mechanics*, 21(1):25–34, 1954.

[9] RF Bishop, Rodney Hill, and NF Mott. The theory of indentation and hardness tests. *Proceedings of the Physical Society*, 57(3):147, 1945.

[10] S Zhandarov, E Pisanova, and B Lauke. Is there any contradiction between the stress and energy failure criteria in micromechanical tests? part i. crack initiation: stress-controlled or energy-controlled? *Composite Interfaces*, 5 (5):387–404, 1997.

[11] Srinivasa D Thoppul and Ronald F Gibson. Mechanical characterization of spot friction stir welded joints in aluminum alloys by combined experi-

mental/numerical approaches: part i: micromechanical studies. *Materials characterization*, 60(11):1342–1351, 2009.

[12] Dierk Raabe, M Sachtleber, Zisu Zhao, Franz Roters, and Stefan Zaefferer. Micromechanical and macromechanical effects in grain scale polycrystal plasticity experimentation and simulation. *Acta Materialia*, 49(17): 3433–3441, 2001.

[13] YW Kwon and JM Berner. Micromechanics model for damage and failure analyses of laminated fibrous composites. *Engineering Fracture Mechanics*, 52(2):231–242, 1995.

[14] Dierk Raabe, M Sachtleber, Zisu Zhao, Franz Roters, and Stefan Zaefferer. Micromechanical and macromechanical effects in grain scale polycrystal plasticity experimentation and simulation. *Acta Materialia*, 49(17): 3433–3441, 2001.

[15] Risto M Nieminen. From atomistic simulation towards multiscale modelling of materials. *Journal of Physics: Condensed Matter*, 14(11):2859, 2002.

[16] JA Elliott. Novel approaches to multiscale modelling in materials science. *International Materials Reviews*, 56(4):207–225, 2011.

[17] Lihua Wang, Xiaodong Han, Pan Liu, Yonghai Yue, Ze Zhang, and En Ma. In situ observation of dislocation behavior in nanometer grains. *Physical review letters*, 105(13):135501, 2010.

[18] T Tabata and HK Birnbaum. Direct observations of the effect of hydrogen on the behavior of dislocations in iron. *Scripta Metallurgica*, 17(7):947–950, 1983.

[19] M Dao, L Lu, RJ Asaro, J Th M De Hosson, and E Ma. Toward a quantitative understanding of mechanical behavior of nanocrystalline metals. *Acta Materialia*, 55(12):4041–4065, 2007.

[20] MR Gilbert, SL Dudarev, S Zheng, LW Packer, and J-Ch Sublet. An integrated model for materials in a fusion power plant: transmutation, gas production, and helium embrittlement under neutron irradiation. *Nuclear Fusion*, 52(8):083019, 2012.

[21] Steven J Zinkle and GS Was. Materials challenges in nuclear energy. *Acta Materialia*, 61(3):735–758, 2013.

[22] Ian Cook. Materials research for fusion energy. *Nature Materials*, 5(2):77, 2006.

[23] Heinrich Hora. Developments in inertial fusion energy and beam fusion at magnetic confinement. *Laser and Particle Beams*, 22(04):439–449, 2004.

[24] Weston M Stacey. *Fusion: an introduction to the physics and technology of magnetic confinement fusion*. John Wiley & Sons, 2010.

[25] ROBERT L McCrory and CHARLES P Verdon. Inertial confinement fusion. *Computer Applications in Plasma Science and Engineering (Springer Science & Business Media, 2012)*, page 291, 2012.

[26] Mohamed E Sawan. Geometrical, spectral and temporal differences between icf and mcf reactors and their impact on blanket nuclear parameters. *Fusion Technology*, 10(3P2B):1483–1488, 1986.

[27] GL Kulcinski and ME Sawan. Differences between neutron damage in inertial and magnetic confinement fusion materials test facilities. *Journal of nuclear materials*, 133:52–57, 1985.

[28] Steven J Zinkle and Jeremy T Busby. Structural materials for fission & fusion energy. *Materials Today*, 12(11):12–19, 2009.

[29] Alban Mosnier, PY Beauvais, B Branas, M Comunian, A Facco, P Garin, R Gobin, JF Gournay, R Heidinger, A Ibarra, et al. The accelerator prototype of the ifmif/eveda project. *IPAC*, 10:588, 2010.

[30] T Muroga, M Gasparotto, and SJ Zinkle. Overview of materials research for fusion reactors. *Fusion engineering and design*, 61:13–25, 2002.

[31] JL Bourgade, AE Costley, R Reichle, ER Hodgson, W Hsing, V Glebov, M Decreton, R Leeper, JL Leray, M Dentan, et al. Diagnostic components in harsh radiation environments: Possible overlap in r&d requirements of inertial confinement and magnetic fusion systemsa). *Review of Scientific Instruments*, 79(10):10F304, 2008.

[32] OA Hurricane, DA Callahan, DT Casey, PM Celliers, Charles Cerjan, EL Dewald, TR Dittrich, T Döppner, DE Hinkel, LF Berzak Hopkins, et al. Fuel gain exceeding unity in an inertially confined fusion implosion. *Nature*, 506(7488):343, 2014.

[33] D Böhne, I Hofmann, G Kessler, GL Kulcinski, J Meyer-ter Vehn, U von Möllendorff, GA Moses, RW Müller, IN Sviatoslavsky, DK Sze, et al.

Hiball—a conceptual design study of a heavy-ion driven inertial confinement fusion power plant. *Nuclear Engineering and Design*, 73(2):195–200, 1982.

[34] John D Lindl, Robert L McCrory, and E Michael Campbell. Progress toward ignition and burn propagation in inertial confinement fusion. *Phys. Today*, 45(9):32–40, 1992.

[35] RW Moir, RL Bieri, XM Chen, TJ Dolan, MA Hoffman, PA House, RL Leber, JD Lee, YT Lee, JC Liu, et al. Hylife-ii: A molten-salt inertial fusion energy power plant design—final report. *Fusion Science and Technology*, 25(1):5–25, 1994.

[36] Roger E Stoller, Mychailo B Toloczko, Gary S Was, Alicia G Certain, Shyam Dwaraknath, and Frank A Garner. On the use of SRIM for computing radiation damage exposure. *Nuclear instruments and methods in physics research section B: beam interactions with materials and atoms*, 310:75–80, 2013.

[37] GH Kinchin and RS Pease. The displacement of atoms in solids by radiation. *Reports on progress in physics*, 18(1):1, 1955.

[38] James F Ziegler, Matthias D Ziegler, and Jochen P Biersack. SRIM–the stopping and range of ions in matter (2010). *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 268(11):1818–1823, 2010.

[39] Daniel Mason. Incorporating non-adiabatic effects in embedded atom potentials for radiation damage cascade simulations. *Journal of Physics: Condensed Matter*, 27(14):145401, 2015.

[40] AE Sand, SL Dudarev, and K Nordlund. High-energy collision cascades in tungsten: Dislocation loops structure and clustering scaling laws. *EPL (Europhysics Letters)*, 103(4):46003, 2013.

[41] T Diaz De La Rubia and MW Guinan. New mechanism of defect production in metals: A molecular-dynamics study of interstitial-dislocation-loop formation in high-energy displacement cascades. *Physical review letters*, 66 (21):2766, 1991.

[42] V Barabash, G Federici, J Linke, and CH Wu. Material/plasma surface interaction issues following neutron damage. *Journal of Nuclear Materials*, 313:42–51, 2003.

[43] Luke L Hsiung, Michael J Fluss, Scott J Tumey, B William Choi, Yves Serruys, Francois Willaime, and Akihiko Kimura. Formation mechanism and the role of nanoparticles in fe-cr ods steels developed for radiation tolerance. *Physical Review B*, 82(18):184103, 2010.

[44] Xunxiang Hu, Takaaki Koyanagi, Makoto Fukuda, NAP Kiran Kumar, Lance L Snead, Brian D Wirth, and Yutai Katoh. Irradiation hardening of pure tungsten exposed to neutron irradiation. *Journal of Nuclear Materials*, 480:235–243, 2016.

[45] AE Sand, SL Dudarev, and K Nordlund. High-energy collision cascades in tungsten: Dislocation loops structure and clustering scaling laws. *EPL (Europhysics Letters)*, 103(4):46003, 2013.

[46] DEJ Armstrong, X Yi, EA Marquis, and SG Roberts. Hardening of self ion implanted tungsten and tungsten 5-wt% rhenium. *Journal of Nuclear Materials*, 432(1):428–436, 2013.

[47] MR Gilbert and J-Ch Sublet. Neutron-induced transmutation effects in w and w-alloys in a fusion environment. *Nuclear Fusion*, 51(4):043005, 2011.

[48] Robin A Forrest et al. *FISPACT-2007: User manual*. EURATOM/UKAEA fusion association, 2007.

[49] Robin A Forrest, Michael G Sowerby, Bryan H Patrick, and David AJ Endacott. The data library ukact1 and the inventory code fispact. In *Nuclear data for science and technology*, 1988.

[50] Makoto Fukuda, Kiyohiro Yabuuchi, Shuhei Nogami, Akira Hasegawa, and Teruya Tanaka. Microstructural development of tungsten and tungsten–rhenium alloys due to neutron irradiation in hfir. *Journal of Nuclear Materials*, 455(1):460–463, 2014.

[51] Akira Hasegawa, Takashi Tanno, Shuhei Nogami, and Manabu Satou. Property change mechanism in tungsten under neutron irradiation in various reactors. *Journal of Nuclear Materials*, 417(1):491–494, 2011.

[52] Akira Hasegawa, Makoto Fukuda, Kiyohiro Yabuuchi, and Shuhei Nogami. Neutron irradiation effects on the microstructural development of tungsten and tungsten alloys. *Journal of Nuclear Materials*, 471:175–183, 2016.

[53] M Klimenkov, U Jäntsch, M Rieth, HC Schneider, DEJ Armstrong, J Gibson, and SG Roberts. Effect of neutron irradiation on the microstructure of tungsten. *Nuclear Materials and Energy*, 2016.

[54] Xiaoou Yi, Michael L Jenkins, Marquis A Kirk, Zhongfu Zhou, and Steven G Roberts. In-situ tem studies of 150 kev w+ ion irradiated w and w-alloys: Damage production and microstructural evolution. *Acta Materialia*, 112: 105–120, 2016.

[55] Alan Xu, Christian Beck, David EJ Armstrong, Krishna Rajan, George DW Smith, Paul AJ Bagot, and Steve G Roberts. Ion-irradiation-induced clustering in w–re and w–re–os alloys: A comparative study using atom probe tomography and nanoindentation measurements. *Acta Materialia*, 87:121–127, 2015.

[56] Alan Xu, David EJ Armstrong, Christian Beck, Michael P Moody, George DW Smith, Paul AJ Bagot, and Steve G Roberts. Ion-irradiation induced clustering in w-re-ta, w-re and w-ta alloys: An atom probe tomography and nanoindentation study. *Acta Materialia*, 124:71–78, 2017.

[57] Robert N Noyce. Microelectronics. *Scientific American*, 237(3):62–69, 1977.

[58] Dermot Roddy. *Introduction to microelectronics*. Elsevier, 2013.

[59] Kihwan Choi, Wonbok Lee, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 29–34. IEEE Computer Society, 2004.

[60] Karel De Vogeleer, Gerard Memmi, Pierre Jouvelot, and Fabien Coelho. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. In *International Conference on Parallel Processing and Applied Mathematics*, pages 793–803. Springer, 2013.

[61] NVidia, CUDA. NVidia CUDA C programming guide. *NVidia Corporation*, 120(18):8, 2011.

[62] John Goodacre and Andrew N Sloss. Parallelism and the arm instruction set architecture. *Computer*, 38(7):42–50, 2005.

[63] Stephen M Trimberger. *Field-programmable gate array technology*. Springer Science & Business Media, 2012.

[64] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques 2000*, pages 71–82. Springer, 2000.

[65] John A Flanders, David C Ready, Steven Van Seters, Leonard Schwartz, and William D Townsend. Hardware filtering method and apparatus, March 21 2000. US Patent 6,041,058.

[66] Brian Kelleher and Thomas C Furlong. Software configurable memory architecture for data processing system having graphics capability, August 28 1990. US Patent 4,953,101.

[67] Masayuki Sumi. Image processing devices and methods, March 31 1998. US Patent 5,734,807.

[68] Joseph Celi Jr, Jonathan M Wagner, and Roger Louie. Advanced graphics driver architecture, February 3 1998. US Patent 5,715,459.

[69] Thomas J Bensky and SE Frey. Computer sound card assisted measurements of the acoustic doppler effect for accelerated and unaccelerated sound sources. *American Journal of Physics*, 69(12):1231–1236, 2001.

[70] Daniel E Evanicky. Enhanced viewing experience of a display through localised dynamic control of background lighting level, April 9 2013. US Patent 8,416,149.

[71] Rolf Rabenseifner, Georg Hager, and Gabriele Jost. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 427–436. IEEE, 2009.

[72] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2), 2010.

[73] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66–73, 2010.

[74] Ruymán Reyes, Iván López, Juan J Fumero, and Francisco de Sande. A preliminary evaluation of openacc implementations. *The Journal of Supercomputing*, 65(3):1063–1075, 2013.

[75] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms*, volume 400. Benjamin/Cummings Redwood City, 1994.

[76] Cristobal A Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu

architectures. *Communications in Computational Physics*, 15(2):285–329, 2014.

[77] YJ Lee and LB Freund. Fracture initiation due to asymmetric impact loading of an edge cracked plate. *Journal of Applied Mechanics*, 57(1):104–111, 1990.

[78] Stefan Sandfeld, Thomas Hochrainer, Peter Gumbsch, and Michael Zaiser. Numerical implementation of a 3d continuum theory of dislocation dynamics and application to micro-bending. *Philosophical Magazine*, 90(27-28):3697–3728, 2010.

[79] Vasily Bulatov, Wei Cai, Jeff Fier, Masato Hiratani, Gregg Hommes, Tim Pierce, Meijie Tang, Moono Rhee, Kim Yates, and Tom Arsenlis. Scalable line dynamics in paradis. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 19. IEEE Computer Society, 2004.

[80] MP O'day and William A Curtin. A superposition framework for discrete dislocation plasticity. *Transactions of the ASME-E-Journal of Applied Mechanics*, 71(6):805–815, 2004.

[81] Robert Gracie, Giulio Ventura, and Ted Belytschko. A new fast finite element method for dislocations based on interior discontinuities. *International Journal for Numerical Methods in Engineering*, 69(2):423–441, 2007.

[82] Franz Roters, Philip Eisenlohr, Luc Hantcherli, Denny Dharmawan Tjahjanto, Thomas R Bieler, and Dierk Raabe. Overview of constitutive laws, kinematics, homogenization and multiscale methods in crystal plasticity finite-element modeling: Theory, experiments, applications. *Acta Materialia*, 58(4):1152–1211, 2010.

[83] El Arzt. Size effects in materials due to microstructural and dimensional constraints: a comparative review. *Acta materialia*, 46(16):5611–5626, 1998.

[84] R Madec, B Devincre, L Kubin, T Hoc, and D Rodney. The role of collinear interaction in dislocation-induced hardening. *Science*, 301(5641):1879–1882, 2003.

[85] Francesco Ferroni, Edmund Tarleton, and Steven Fitzgerald. Gpu accelerated dislocation dynamics. *Journal of Computational Physics*, 272:619–628, 2014.

[86] S Queyreau, J Marian, BD Wirth, and A Arsenlis. Analytical integration of the forces induced by dislocations on a surface element. *Modelling and Simulation in Materials Science and Engineering*, 22(3):035004, 2014.

[87] Erik Van der Giessen and Alan Needleman. Discrete dislocation plasticity: a simple planar model. *Modelling and Simulation in Materials Science and Engineering*, 3(5):689, 1995.

[88] B Devincre, A Roos, and S Groh. Boundary problems in dd simulations. In *In: Thermodynamics, Microstructures and Plasticity, A. Finel et al., Nato Sciences Series II: Mathematics, Physics and Chemistry, 108, p. 275, Eds (Kluwer, NL-Dordrecht*. Citeseer, 2003.

[89] CS Shin, MC Fivel, and KH Oh. Nucleation and propagation of dislocations near a precipitate using 3d discrete dislocation dynamics simulations. *Le Journal de Physique IV*, 11(PR5):Pr5–27, 2001.

[90] MC Fivel and GR Canova. Developing rigorous boundary conditions to simulations of discrete dislocation dynamics. *Modelling and Simulation in Materials Science and Engineering*, 7(5):753, 1999.

[91] Akiyuki Takahashi and Nasr M Ghoniem. A computational method for dislocation–precipitate interaction. *Journal of the Mechanics and Physics of Solids*, 56(4):1534–1553, 2008.

[92] O Jamond, R Gatti, A Roos, and B Devincre. Consistent formulation for the discrete-continuous model: Improving complex dislocation dynamics simulations. *International Journal of Plasticity*, 80:19–37, 2016.

[93] Toshio Mura. *Micromechanics of defects in solids*. Springer Science & Business Media, 2013.

[94] G Saada and XL Shi. Description of dislocation cores. *Czechoslovak Journal of Physics*, 45(11):979–989, 1995.

[95] A Vattré, B Devincre, F Feyel, R Gatti, S Groh, O Jamond, and A Roos. Modelling crystal plasticity by 3d dislocation dynamics and the finite element method: the discrete-continuous model revisited. *Journal of the Mechanics and Physics of Solids*, 63:491–505, 2014.

[96] Yang Xiang and DJ Srolovitz. Dislocation climb effects on particle bypass mechanisms. *Philosophical magazine*, 86(25-26):3937–3957, 2006.

[97] Yang Xiang, Li-Tien Cheng, David J Srolovitz, and E Weinan. A level set method for dislocation dynamics. *Acta Materialia*, 51(18):5499–5518, 2003.

[98] Siu Sin Quek, Yang Xiang, and David J Srolovitz. Loss of interface coherency around a misfitting spherical inclusion. *Acta Materialia*, 59(14):5398–5410, 2011.

[99] John P Hirth and Jens Lothe. Theory of dislocations. 1982.

[100] Toshio Mura. *Micromechanics of defects in solids*. Springer Science & Business Media, 2013.

[101] Wei Cai, Athanasios Arsenlis, Christopher R Weinberger, and Vasily V Bulatov. A non-singular continuum theory of dislocations. *Journal of the Mechanics and Physics of Solids*, 54(3):561–587, 2006.

[102] R De Wit. Some relations for straight dislocations. *physica status solidi (b)*, 20(2):567–573, 1967.

[103] R De Wit. The self-energy of dislocation configurations made up of straight segments. *physica status solidi (b)*, 20(2):575–580, 1967.

[104] Vladimir L Indenbom and Jens Lothe. *Elastic strain fields and dislocation mobility*. Elsevier, 2012.

[105] Bernard Fedelich. The glide force on a dislocation in finite elasticity. *Journal of the Mechanics and Physics of Solids*, 52(1):215–247, 2004.

[106] M Yu Gutkin and EC Aifantis. Screw dislocation in gradient elasticity. *Scripta Materialia*, 35(11):1353–1358, 1996.

[107] M Yu Gutkin and EC Aifantis. Edge dislocation in gradient elasticity. *Scripta Materialia*, 36(1):129–135, 1997.

[108] LM Brown. The self-stress of dislocations and the shape of extended nodes. *Philosophical Magazine*, 10(105):441–466, 1964.

[109] SD Gavazza and DM Barnett. The self-force on a planar dislocation loop in an anisotropic linear-elastic medium. *Journal of the Mechanics and Physics of Solids*, 24(4):171–185, 1976.

[110] R Peierls. The size of a dislocation. *Proceedings of the Physical Society*, 52 (1):34, 1940.

[111] FRN Nabarro. Dislocations in a simple cubic lattice. *Proceedings of the Physical Society*, 59(2):256, 1947.

[112] K Gururaj and C Robertson. Plastic deformation in ods ferritic alloys: A 3d dislocation dynamics investigation. *Energy Procedia*, 7:279–285, 2011.

[113] Zhenhuan Li, Chuantao Hou, Minsheng Huang, and Chaojun Ouyang. Strengthening mechanism in micro-polycrystals with penetrable grain

boundaries by discrete dislocation dynamics simulation and hall–petch effect. *Computational Materials Science*, 46(4):1124–1134, 2009.

[114] Haidong Fan, Zhenhuan Li, Minsheng Huang, and Xiong Zhang. Thickness effects in polycrystalline thin films: Surface constraint versus interior constraint. *International Journal of Solids and Structures*, 48(11-12):1754–1766, 2011.

[115] Z Shen, RH Wagoner, and WAT Clark. Dislocation pile-up and grain boundary interactions in 304 stainless steel. *Scripta metallurgica*, 20(6):921–926, 1986.

[116] Z Shen, RH Wagoner, and WAT Clark. Dislocation and grain boundary interactions in metals. *Acta metallurgica*, 36(12):3231–3242, 1988.

[117] GC Hasson and C Goux. Interfacial energies of tilt boundaries in aluminium. experimental and theoretical determination. *Scripta metallurgica*, 5(10): 889–894, 1971.

[118] Haidong Fan, Sylvie Aubry, Athanasios Arsenlis, and Jaafar A El-Awady. The role of twinning deformation on the hardening response of polycrystalline magnesium from discrete dislocation dynamics simulations. *Acta Materialia*, 92:126–139, 2015.

[119] S Lay and G Nouet. Interaction of slip dislocations with the (01 1 2) twin interface in zinc. *Philosophical Magazine A*, 70(6):1027–1044, 1994.

[120] JI Dickson and C Robin. The incorporation of slip dislocations in {1102} twins in zirconium. *Materials Science and Engineering*, 11(5):299–302, 1973.

[121] DI Tomsett and M Bevis. The incorporation of basal slip dislocations in {10 1 2} twins in zinc crystals. *Philosophical Magazine*, 19(157):129–140, 1969.

[122] MH Yoo and CT Wei. Slip modes of hexagonal-close-packed metals. *Journal of Applied Physics*, 38(11):4317–4322, 1967.

[123] A Serra and DJ Bacon. A new model for {10 1 2} twin growth in hcp metals. *Philosophical Magazine A*, 73(2):333–343, 1996.

[124] Jindong Wang, Gen He, Jinwen Qin, Lidong Li, and Xuefeng Guo. Preparation of silicon nanowires by in situ doping and their electrical properties. *Colloids and Surfaces A: Physicochemical and Engineering Aspects*, 450: 156–160, 2014.

[125] Kisaragi Yashiro, F Kurose, Y Nakashima, K Kubo, Yoshihiro Tomita, and HM Zbib. Discrete dislocation dynamics simulation of cutting of γ precipitate and interfacial dislocation network in ni-based superalloys. *International Journal of Plasticity*, 22(4):713–723, 2006.

[126] A Vattré, B Devincre, and A Roos. Orientation dependence of plastic deformation in nickel-based single crystal superalloys: Discrete–continuous model simulations. *Acta Materialia*, 58(6):1938–1951, 2010.

[127] SI Rao S, TA Parthasarathy, DM Dimiduk, and PM Hazzledine. Discrete dislocation simulations of precipitation hardening in superalloys. *Philosophical Magazine*, 84(30):3195–3215, 2004.

[128] Hesam Askari, Hussein M Zbib, and Xin Sun. Multiscale modeling of inclusions and precipitation hardening in metal matrix composites: Application to advanced high-strength steels. *Journal of Nanomechanics and Micromechanics*, 3(2):24–33, 2012.

[129] Vasily Bulatov and Wei Cai. *Computer simulations of dislocations*, volume 3. Oxford University Press on Demand, 2006.

[130] Haiyang Yu, Alan Cocks, and Edmund Tarleton. Discrete dislocation plasticity helps understand hydrogen effects in bcc materials. *Journal of the Mechanics and Physics of Solids*, 2018. ISSN 0022-5096. doi: https://doi.org/10.1016/j.jmps.2018.08.020.

[131] N Bertin, S Aubry, A Arsenlis, and W Cai. Gpu-accelerated dislocation dynamics using subcycling time-integration. *Modelling and Simulation in Materials Science and Engineering*, 27(7):075014, 2019.

[132] Athanasios Arsenlis, Wei Cai, Meijie Tang, Moono Rhee, Tomas Oppelstrup, Gregg Hommes, Tom G Pierce, and Vasily V Bulatov. Enabling strain hardening simulations with dislocation dynamics. *Modelling and Simulation in Materials Science and Engineering*, 15(6):553, 2007.

[133] B Bromage and E Tarleton. Calculating dislocation displacements on the surface of a volume. *Modelling and Simulation in Materials Science and Engineering*, 26(8):085007, 2018.

[134] E Ward Cheney and David R Kincaid. *Numerical mathematics and computing*. Cengage Learning, 2012.

[135] Paul Rodríguez. Total variation regularization algorithms for images corrupted with different noise models: a review. *Journal of Electrical and Computer Engineering*, 2013, 2013.

[136] Peter J Bickel, Bo Li, Alexandre B Tsybakov, Sara A van de Geer, Bin Yu, Teófilo Valdés, Carlos Rivero, Jianqing Fan, and Aad van der Vaart. Regularization in statistics. *Test*, 15(2):271–344, 2006.

[137] Zhanxuan Hu, Feiping Nie, Rong Wang, and Xuelong Li. Low rank regularization: A review. *Neural Networks*, 2020.

[138] URL https://www.gov.scot/collections/coronavirus-covid-19-modelling-the-epidemic/.

[139] Grant Hill-Cawthorne. Models of covid-19: Part 1, Dec 2020. URL https://post.parliament.uk/models-of-covid-19-part-1/.

[140] Grant Hill-Cawthorne. Models of covid-19: Part 2, Dec 2020. URL https://post.parliament.uk/models-of-covid-19-part-2/.

[141] Dalmeet Singh Chawla. Critiqued coronavirus simulation gets thumbs up from code-checking efforts, Jun 2020. URL https://www.nature.com/articles/d41586-020-01685-y.

[142] Niels G Mede, Mike S Schäfer, Ricarda Ziegler, and Markus Weißkopf. The "replication crisis" in the public eye: Germans' awareness and perceptions of the (ir) reproducibility of scientific research. *Public Understanding of Science*, page 0963662520954370, 2020.

[143] David Randall and Christopher Welser. *The Irreproducibility Crisis of Modern Science: Causes, Consequences, and the Road to Reform.* ERIC, 2018.

[144] Roberto Bolli. Reflections on the irreproducibility of scientific papers. *Circulation research*, 117(8):665–666, 2015.

[145] Simon Thompson. *Haskell: the craft of functional programming*, volume 2. Addison-Wesley, 2011.

[146] Konrad Hinsen. The promises of functional programming. *Computing in Science & Engineering*, 11(4):86–90, 2009.

[147] Donald E Knuth. Structured programming with go to statements. *ACM Computing Surveys (CSUR)*, 6(4):261–301, 1974.

[148] David R Hanson. Fast allocation and deallocation of memory based on object lifetimes. *Software: Practice and Experience*, 20(1):5–12, 1990.

[149] Andrea Elisabet Sand, Kai Nordlund, and SL Dudarev. Radiation damage production in massive cascades initiated by fusion neutrons in tungsten. *Journal of Nuclear Materials*, 455(1-3):207–211, 2014.

[150] Yang Li, Max Boleininger, Christian Robertson, Laurent Dupuy, and Sergei L Dudarev. Diffusion and interaction of prismatic dislocation loops simulated by stochastic discrete dislocation dynamics. *Physical Review Materials*, 3(7):073805, 2019.

[151] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. URL https://doi.org/10.1137/141000671.