Deposit & Copying of Dissertation Declaration

UNIVERSITY OF CAMBRIDGE

Board of Graduate Studies

Please note that you will also need to bind a copy of this Declaration into your final, hardbound copy of thesis - this has to be the very first page of the hardbound thesis.

1	Surname (Family Name)	Forenames(s)	Title
	Mantell	Rosemary Genevieve	Miss
2	Title of Dissertation as approved by the Degree Committee		
	Accelerated sampling of energy landscapes		

In accordance with the University Regulations in *Statutes and Ordinances* for the PhD, MSc and MLitt Degrees, I agree to deposit one print copy of my dissertation entitled above and one print copy of the summary with the Secretary of the Board of Graduate Studies who shall deposit the dissertation and summary in the University Library under the following terms and conditions:

1. Dissertation Author Declaration

I am the author of this dissertation and hereby give the University the right to make my dissertation available in print form as described in 2. below.

My dissertation is my original work and a product of my own research endeavours and includes nothing which is the outcome of work done in collaboration with others except as declared in the Preface and specified in the text. I hereby assert my moral right to be identified as the author of the dissertation.

The deposit and dissemination of my dissertation by the University does not constitute a breach of any other agreement, publishing or otherwise, including any confidentiality or publication restriction provisions in sponsorship or collaboration agreements governing my research or work at the University or elsewhere.

2. Access to Dissertation

I understand that one print copy of my dissertation will be deposited in the University Library for archival and preservation purposes, and that, unless upon my application restricted access to my dissertation for a specified period of time has been granted by the Board of Graduate Studies prior to this deposit, the dissertation will be made available by the University Library for consultation by readers in accordance with University Library Regulations and copies of my dissertation may be provided to readers in accordance with applicable legislation.

3	Signature	Date
	RMantell	08.10.17

Corresponding Regulation

Before being admitted to a degree, a student shall deposit with the Secretary of the Board one copy of his or her hard-bound dissertation and one copy of the summary (bearing student's name and thesis title), both the dissertation and the summary in a form approved by the Board. The Secretary shall deposit the copy of the dissertation together with the copy of the summary in the University Library where, subject to restricted access to the dissertation for a specified period of time having been granted by the Board of Graduate Studies, they shall be made available for consultation by readers in accordance with University Library Regulations and copies of the dissertation provided to readers in accordance with applicable legislation.

UNIVERSITY OF CAMBRIDGE

Accelerated sampling of energy landscapes

Rosemary Genevieve Mantell

Department of Chemistry
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Peterhouse October 2017

Accelerated sampling of energy landscapes

Rosemary Genevieve Mantell

In this project, various computational energy landscape methods were accelerated using graphics processing units (GPUs). Basin-hopping global optimisation was treated using a version of the limited-memory BFGS algorithm adapted for CUDA, in combination with GPU-acceleration of the potential calculation. The Lennard-Jones potential was implemented using CUDA, and an interface to the GPU-accelerated AMBER potential was constructed. These results were then extended to form the basis of a GPU-accelerated version of hybrid eigenvector-following. The doubly-nudged elastic band method was also accelerated using an interface to the potential calculation on GPU. Additionally, a local rigid body framework was adapted for GPU hardware. Tests were performed for eight biomolecules represented using the AMBER potential, ranging in size from 81 to 22 811 atoms, and the effects of minimiser history size and local rigidification on the overall efficiency were analysed. Improvements relative to CPU performance of up to two orders of magnitude were obtained for the largest systems. These methods have been successfully applied to both biological systems and atomic clusters.

An existing interface between a code for free energy basin-hopping and the SuiteSparse package for sparse Cholesky factorisation was refined, validated and tested. Tests were performed for both Lennard-Jones clusters and selected biomolecules represented using the AMBER potential. Significant acceleration of the vibrational frequency calculations was achieved, with negligible loss of accuracy, relative to the standard diagonalisation procedure. For the larger systems, exploiting sparsity reduces the computational cost by factors of 10 to 30.

The acceleration of these computational energy landscape methods opens up the possibility of investigating much larger and more complex systems than previously accessible. A wide array of new applications are now computationally feasible.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit of 60 000 words.

Rosemary Genevieve Mantell
October 2017

Acknowledgements

First and foremost, I wish to thank my supervisor, Professor David Wales, for giving me the opportunity to undertake this work. His encouragement, enthusiasm and scientific insight have been invaluable. I am also grateful to him for testing the GPU code through his work on addressable Lennard-Jones clusters.

I am indebted to the many members of the Wales group and the wider Theory Sector, both past and present, for their assistance and useful discussion. In particular, I would like to thank Kyle Sutherland-Cash for his work on the AMBER 12 interface for CPU and the SuiteSparse interface. Thanks are also due to Konstantin Röder for extensive testing of the GPU code, particularly on the coiled-coil peptide described in this work. I also wish to highlight Dr Chris Whittleston for his help in creating input files for GMIN and OPTIM with AMBER. Dr James Farrell is the author of the Python script for generating random potential energy surfaces that was used in preparing Figure 2.5.

The support I have received from the Department of Chemistry computer officers has been most appreciated. Special mention must go to Dr Catherine Pitt for her work on the GPU cluster and help in solving the many technical problems that we have encountered.

Funding for this work has been provided by the EPSRC and Peterhouse. In addition, the EPSRC, Peterhouse and the Department of Chemistry have all contributed to the costs of conference travel.

Finally, I would like to thank my family and friends for helping me to maintain a sense of perspective during these last few years and keeping me motivated throughout the writing process. I am especially grateful to my partner, Dr Stephen Millmore, for useful comments on this document, assistance with using drawing and graphing programs, and his support throughout my time studying in Cambridge.

Publications

The work presented in this thesis has been published in the following papers:

Chapter 3

- R. G. Mantell, C. E. Pitt and D. J. Wales, *J. Chem. Theory Comput.*, 2016, **12**, 6182–6191.
- J. A. Joseph, K. Röder, D. Chakraborty, R. G. Mantell and D. J. Wales, *Chem. Commun.*, 2017, **53**, 6974–6988.
- S. N. Fejer, R. G. Mantell and D. J. Wales, *Phys. Chem. Chem. Phys.*, submitted.

Chapter 4

- K. H. Sutherland-Cash, R. G. Mantell and D. J. Wales, *Chem. Phys. Lett.*, 2017, **685**, 288–293.

Abstract

In this project, various computational energy landscape methods were accelerated using graphics processing units (GPUs). Basin-hopping global optimisation was treated using a version of the limited-memory BFGS algorithm adapted for CUDA, in combination with GPU-acceleration of the potential calculation. The Lennard-Jones potential was implemented using CUDA, and an interface to the GPU-accelerated AMBER potential was constructed. These results were then extended to form the basis of a GPU-accelerated version of hybrid eigenvector-following. The doubly-nudged elastic band method was also accelerated using an interface to the potential calculation on GPU. Additionally, a local rigid body framework was adapted for GPU hardware. Tests were performed for eight biomolecules represented using the AMBER potential, ranging in size from 81 to 22 811 atoms, and the effects of minimiser history size and local rigidification on the overall efficiency were analysed. Improvements relative to CPU performance of up to two orders of magnitude were obtained for the largest systems. These methods have been successfully applied to both biological systems and atomic clusters.

An existing interface between a code for free energy basin-hopping and the SuiteSparse package for sparse Cholesky factorisation was refined, validated and tested. Tests were performed for both Lennard-Jones clusters and selected biomolecules represented using the AMBER potential. Significant acceleration of the vibrational frequency calculations was achieved, with negligible loss of accuracy, relative to the standard diagonalisation procedure. For the larger systems, exploiting sparsity reduces the computational cost by factors of 10 to 30.

The acceleration of these computational energy landscape methods opens up the possibility of investigating much larger and more complex systems than previously accessible. A wide array of new applications are now computationally feasible.

Contents

Abbreviations **xiii**

1 Introduction **1**

2 Methods **3**

2.1 General-purpose computation on graphics processing units 3
2.2 Intermolecular potentials 12
2.2.1 Lennard-Jones 12
2.2.2 AMBER 13
2.3 Basin-hopping global optimisation 21
2.3.1 Limited-memory BFGS 23
2.4 Transition state determination 25
2.4.1 Doubly-nudged elastic band method 26
2.4.2 Hybrid eigenvector-following 27
2.5 Local rigid body framework 28
2.5.1 Root mean square force formulation 30
2.6 Free energy basin-hopping 31
2.7 Cholesky factorisation 33

3 GPU-acceleration of computational energy landscape methods **35**

3.1 Introduction 35
3.2 Methods 37
3.2.1 Basin-hopping global optimisation 38
3.2.2 Local rigid body framework 39
3.2.3 Hybrid eigenvector-following 41
3.2.4 Doubly-nudged elastic band method 42
3.2.5 Potential calculation 42

3.3 Results and discussion 43
3.3.1 Basin-hopping global optimisation 44
3.3.2 Local rigid body framework 50
3.3.3 Hybrid eigenvector-following 54
3.3.4 Doubly-nudged elastic band method 58
3.3.5 Coiled-coil peptide 59
3.3.6 Addressable clusters and aggregates 63
3.4 Conclusions 67

4 Exploiting sparsity in free energy basin-hopping 69
4.1 Introduction 69
4.2 Methods 71
4.3 Results and discussion 72
4.3.1 Atomic clusters 72
4.3.2 Proteins 77
4.4 Conclusions 82

5 Conclusions and future work 83

Bibliography 87

Appendix A Shifting the Hessian eigenvalues 97

Abbreviations

BLAS	Basic Linear Algebra Subprograms
CPU	central processing unit
DNEB	doubly-nudged elastic band
ECC	error-correcting code
EF	eigenvector-following
FEBH	free energy basin-hopping
FSA	factorised superposition approach
GB	generalised Born
GPGPU	general-purpose computation on graphics processing units
GPU	graphics processing unit
HA	haemagglutinin
L-BFGS	limited-memory BFGS
LJ	Lennard-Jones
MD	molecular dynamics
MKL	Math Kernel Library
NAB	Nucleic Acid Builder
$NADP^+$	nicotinamide adenine dinucleotide phosphate
NEB	nudged elastic band

PAC	phenylacetic acid
PB	Poisson-Boltzmann
PME	particle mesh Ewald
PUMA	p53 upregulated modulator of apoptosis
QCI	quasi-continuous interpolation
RMS	root mean square
SIMT	single-instruction, multiple-thread
SM	streaming multiprocessor
SP	streaming processor
vdW	van der Waals

Chapter 1

Introduction

Computational chemistry involves the use of computer modelling and simulation to study the structures and properties of molecules and materials. It is used both as an aid to interpreting experimental work and as a stand-alone, predictive tool. As theoretical methods become ever more sophisticated and computing power increases, its viability as an alternative to experiment in real-world applications is increasing. In the pursuit of new insight, there is a constant push towards the simulation of larger systems on longer timescales, using more accurate methods. This effort leads to increased computational expense, and so drives the development of new approaches to acceleration of computational methods.

The computational techniques that we accelerate in this work all fall into the category of energy landscape methodology. Typically, we are interested in the potential energy, V, as there are many computationally efficient ways to calculate this from the conformation of the system. For a system consisting of N atoms in three dimensions, we often write the atomic coordinates as a $3N$-dimensional vector, $\mathbf{X}$. The corresponding potential energy surface is written as $V(\mathbf{X})$, and describes the variation in potential energy as a function of the coordinates.[1] The concept of the PES rests upon the Born-Oppenheimer approximation,[2] which allows us to separate the nuclear and electronic degrees of freedom. Electrons adjust almost instantaneously to changes in the nuclear coordinates due to their relatively small mass, so the energy of a molecule in its ground electronic state can be considered a function of the nuclear coordinates only.[1] Sometimes we may also be interested in the free energy, which incorporates the effects of entropy at a specified temperature. However, this is generally more expensive to compute.[3]

Geometrical features of a landscape provide important information about the associated system. Primarily, we are interested in minima and the transition states that connect them. Minima are stationary points from which a small displacement in internal coordinates results in an increase in energy. The forces on the system vanish at a stationary point, i.e. $\nabla V = \mathbf{0}$, and the local minima represent stable structures of the system. The minimum that is lowest in energy is known as the global minimum.[1] According to the definition of Murrell and Laidler, transition states are stationary points with one negative Hessian eigenvalue.[4] They correspond to intermediate structures involved in the interconversion of minima. Pathways between pairs of local minima can be found, which may involve just one intervening transition state, or many transition states and intervening minima.[5] Positive and negative displacements along the eigenvector corresponding to the unique negative Hessian eigenvalue associated with a particular transition state lead to the steepest-descent paths, defining the connectivity of the minima.[6] Most emergent thermodynamic and dynamic properties of a system can be calculated from the minima and transition states of an energy landscape. However, as the number of minima on a surface increases exponentially with system size,[7] effective sampling of the minima and transition states becomes much more difficult for larger systems.[1] Techniques to speed up the exploration of energy landscapes are therefore very important.

In this work, we describe the development of some new techniques for accelerated sampling of energy landscapes. Chapter 2 introduces the underlying theory and methodology required to set the results in context. Chapter 3 then presents the acceleration of various computational energy landscape methods using graphics processing units (GPUs). Details of the implementation are discussed and results for a range of biological systems are analysed for methods including basin-hopping global optimisation,[8,9] a local rigid body framework,[10,11] hybrid eigenvector-following[12] (EF) and the doubly-nudged[13] elastic band[14,15] (DNEB) method. Speedups of up to two orders of magnitude are obtained for the largest systems. Applications that demonstrate the validity of these new methods are also presented. In Chapter 4, the refinement of an existing framework for accelerating free energy basin-hopping[16] (FEBH) using sparse Cholesky factorisation is considered. Test results for atomic clusters and biological systems are detailed and a 10 to 30 times increase in speed is demonstrated for the calculation of the vibrational density of states. Finally, the impact of these new developments is summarised in Chapter 5 and possibilities for future work are proposed.

Chapter 2

Methods

This chapter details established techniques and existing methodology relevant to the new developments discussed in Chapters 3 and 4. An overview of each area is presented and the connection to later chapters described.

2.1 General-purpose computation on graphics processing units

The work presented in Chapter 3 focuses on the acceleration of various computational energy landscape methods using graphics processing units (GPUs). This is an example of general-purpose computation on graphics processing units (GPGPU), a term coined by Mark Harris in 2002 to refer to the use of GPUs for non-graphics applications,[17] and is a fairly recent trend in the field of parallel computing.[18] Large increases in computational power can no longer be obtained by increasing central processing unit (CPU) clock speeds, due to heat and power restrictions. Similar reasons, coupled with the physical limits on transistor size, also threaten the potential for speed increases through putting more transistors on a chip.[19] Consequently, there has been a move towards parallel computing to gain more computational power for acceleration of applications.[20] GPUs were originally designed for the purpose of fast graphics rendering, but have become increasingly used in general-purpose computations as massively parallel processors.[21] Relative to a CPU, they have a greater number of transistors devoted to data-processing than to data-caching and flow control. This feature makes them ideal for data-parallel computations with high arithmetic intensity (the ratio of arithmetic operations to memory operations).[22] Throughout this document, we will frequently refer to the CPU as the host and the GPU as the device.

To obtain a speedup using GPUs, the target application must be parallelisable.[20] Compute-intensive regions are offloaded to the GPU, while the rest of the code remains on the CPU.[23] If a computationally expensive region of the code cannot be parallelised, the overall acceleration may be limited, as this will remain on the CPU as a performance bottleneck.[24] GPUs are based on a parallelisation model defined by NVIDIA as 'single-instruction, multiple thread' (SIMT), where multiple independent threads execute concurrently using a single instruction.[25] A thread can be thought of as a single flow of execution through a program. SIMT instructions do allow threads to diverge from other threads through data-dependent branching, although this should be avoided for performance reasons.[22] Most modern processors work on the basis of a cycle where instructions (single operations defined by the processor instruction set) are fetched from memory, decoded and then executed.[20] The advantage of the SIMT model is that it has a reduced overhead associated with the instruction fetch stage relative to other models of parallelism, as only one instruction fetch needs to be performed to enable the execution of many threads.[26] One implication of this model is that the calculations performed by the threads should be independent for effective parallelism. For example, a simple loop that multiplies each element of an array by a constant factor is parallelisable, as the current iteration never depends on the previous iteration. The larger the array to be processed, the greater the available parallelism and resultant speedup. In the case of GPU hardware, thousands to tens of thousands of running threads are required for efficient use of the architecture, so the problem in question must be large. Having a large number of concurrent threads also helps to hide the overhead of fetching instructions and data from memory.[20] The use of too few threads means that the GPU will become idle, waiting on memory transactions. If an application is suitable for GPU-acceleration, it is possible to achieve speedups of several orders of magnitude relative to optimised CPU code.[17] However, it requires much more effort on the part of the programmer with regard to efficient use of the hardware.[18] The suitability of the present applications for parallelisation is discussed in Chapter 3.

NVIDIA dominates the current market in GPUs for general-purpose computing.[27] This company is the creator of the parallel computing platform and programming model CUDA (previously an acronym for 'Compute Unified Device Architecture'), introduced in 2006.[25] In the early days of GPU computing, writing code for GPUs required specialist knowledge of shading languages and computer graphics. Shading languages are adapted for programming graphical effects and cannot easily be used for standard programming tasks. There was little flexibility with regard to how and where data could be input or output, floating-point support was poor or absent, and there was no good way to debug code on the GPU.

The CUDA architecture introduced IEEE compliant floating-point support, an instruction set more suited to general-purpose computation, and a much more flexible memory architecture. NVIDIA's extension to the C language, CUDA C, made writing code for the GPU much simpler and more accessible for programmers.[21] Extensions to other languages, such as Fortran and Python, were developed later.[28] AMD also produce GPUs for general-purpose computing that have similar compute power to NVIDIA GPUs.[29] However, these are less widely used, perhaps due to their later entrance to the market.[20] As CUDA is only supported on NVIDIA hardware,[30] AMD promote the use of their implementation of the open standard OpenCL.[31] NVIDIA GPUs were used in the present work, which allowed a choice between a CUDA-based language extension and NVIDIA's implementation of OpenCL. The decision was made to use CUDA, because of the more mature debugging and profiling tools available, the more extensive community support, and the wider range of highly optimised libraries available at the start of the project. Even though the applications to be accelerated were written in Fortran, CUDA C was chosen in preference to CUDA Fortran, due to the greater number of online resources available for this language.

All NVIDIA GPUs have a compute capability that defines their features and technical specifications.[20,22] The hardware has changed rapidly over the last decade as GPUs have evolved to become more suitable for general-purpose computation.[20] The optimisations required for different compute capabilities, or even different cards with the same compute capability, can be very different.[18] Some CUDA operations were only introduced at a relatively high compute capability, so code initially developed for a modern device might need to be extensively rewritten for compatibility with older devices. For example, atomic operations, which allow threads to perform tasks in memory without interruption,[20] only support certain data types on earlier GPUs.[22] Their performance also varies greatly between different cards, so their use may not be advisable even if they are supported.[20] Commercial applications would typically have many versions of certain functions to ensure optimal performance on all hardware. Compute 1.x (where x refers to the minor revision number) devices were named Tesla,[22,25] compute 2.x encompasses the Fermi architecture,[22,32] compute 3.x are Kepler cards,[22,33] compute 5.x are known as Maxwell devices,[22,34] and compute 6.x are Pascal GPUs.[35,36] The Volta architecture has recently been announced and the first cards in the series will have compute capability 7.0.[37–39] The naming of the first generation of Tesla-class GPUs is potentially confusing, as NVIDIA also use the name Tesla to refer to their server-class cards for scientific computing.[40,41] In general, these differ from consumer-grade cards in terms of their higher memory bandwidth, support for the NVIDIA System Management Interface for remotely querying devices over a network, increased double precision capability (from compute capability 1.3) and ECC (error-correcting code) memory (from compute capability 2.0).[20] In general, double precision performance has increased with respect to each previous generation of cards,[20] with the exception of the Maxwell architecture, which reduced double precision performance to 32 times less than single precision.[42] The work described in Chapter 3 is optimised for devices of the Kepler architecture with compute capability 3.5, specifically Tesla K20 and GTX Titan Black GPUs. The ratio of double precision to single precision performance for Tesla K20 GPUs is 1:3, with a theoretical peak single precision performance of 3.52 TFLOPS.[43,44] The corresponding performance ratio for Titan Black cards is just 1:24 in their default mode of operation, with a peak single precision performance of 5.1 TFLOPS.[45,46] However, they can be switched into a full double precision mode, which reaches one third of the single precision performance at a reduced clock rate.[47]

CUDA facilitates the task of effectively exploiting the specialised GPU architecture. It allows the definition of functions known as 'kernels', which execute n times in parallel on n threads. Each thread is assigned a thread ID that can be used for indexed access into arrays.[21] Small groups of threads execute in lockstep and are known as warps, with a current standard size of 32 on NVIDIA hardware.[22] Threads are grouped together into blocks, many of which can execute concurrently, forming a grid.[20] The maximum number of threads per block is 1024 from the Kepler architecture onward,[33–35] twice the previously allowed maximum.[20,32] This hierarchical structure of threads, blocks and grids is shown in Figure 2.1. A GPU consists of a number of streaming multiprocessors (SMs), each of which has a number of attached streaming processors (SPs), more commonly known as CUDA cores. The GPU's scheduler automatically allocates blocks from kernels to available SMs, and only a subset of the total number of blocks may be running at any one time.[20] Although many blocks can run concurrently on a single SM, the actual number is likely to be less than the theoretical maximum, depending on the memory resources required by each block.[48] In the last decade, the number of CUDA cores per card has increased from hundreds to thousands,[32–35,49] greatly increasing the maximum throughput. Kepler GPUs have 192 single precision CUDA cores and 64 double precision units per SM, which underlies the 3:1 performance ratio for the two data types.[33] The Tesla K20 has 13 SMs[33,43], while the Titan Black has 15, enabling a larger number of thread blocks to run concurrently.

The GPU memory architecture is complex, and understanding this complexity is essential for obtaining good speedups. We must consider both memory bandwidth (the amount of data that can be read from or written to memory in a certain time period) and memory latency (the time taken to complete a fetch request from memory).[20] Although the peak

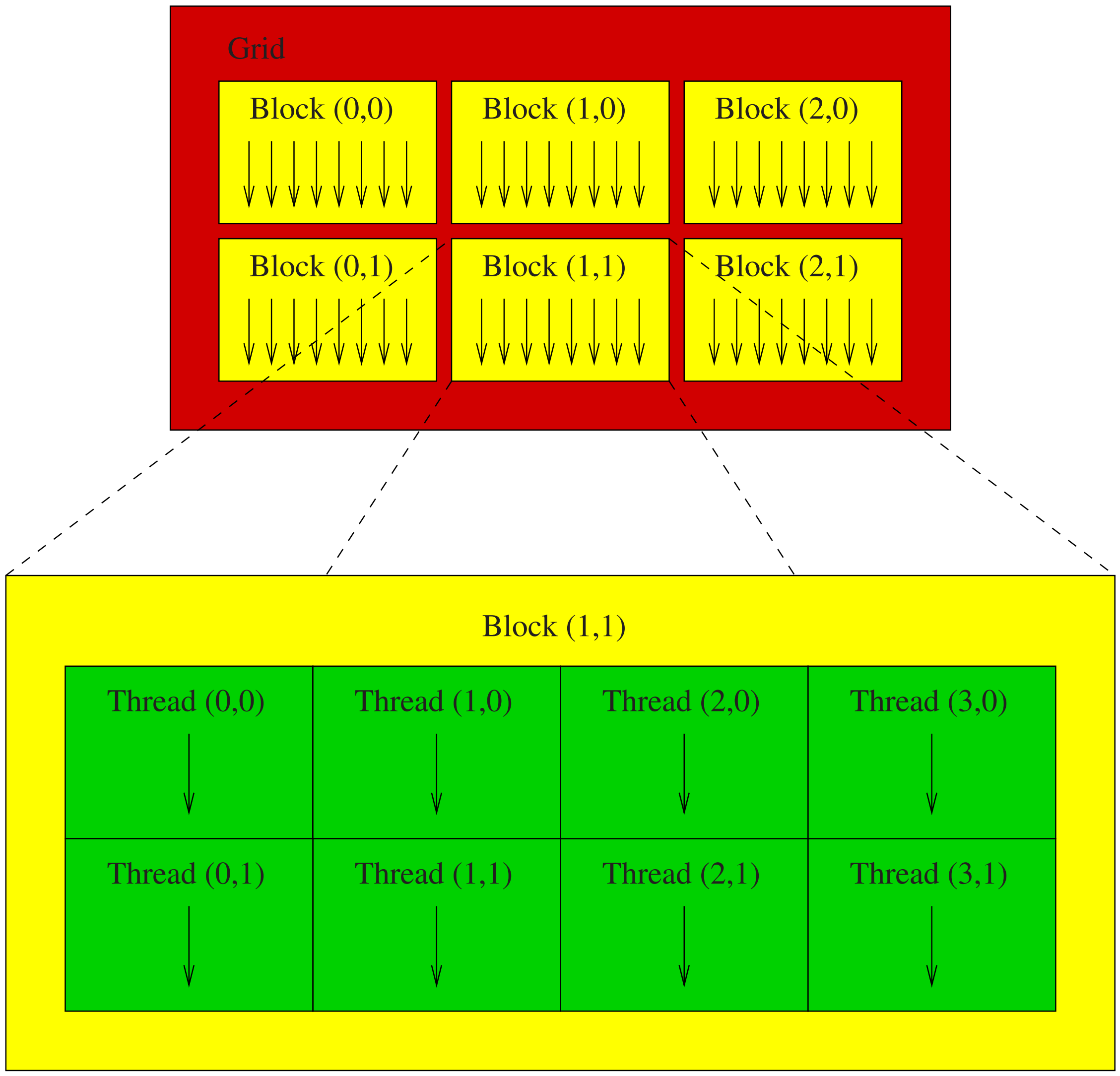


Figure 2.1 The hierarchical structure of threads, blocks and grids used in CUDA. Figure redrawn from reference 22.

compute performance of a GPU is in the teraflop regime, this computation rate is many times in excess of the memory bandwidth capacity, even though the memory bandwidth is itself an order of magnitude greater than that of a CPU.[22] Memory bandwidth is often the limiting factor on the speed of an algorithm, even on a CPU. CPUs typically have several levels of high-speed cache memory close to the processor core that store copies of the data from frequently used memory locations. It is much faster to fetch data from cache memory instead of from main memory, which helps to reduce the limiting effect of memory bandwidth on performance. The first level (L1) cache is the smallest and fastest region of cache. A second level (L2) and third level (L3) cache are often present too, with successive levels being larger and having slower access due to their increased distance from the processor core.[50] Older GPU models have very little cache memory, but devices from compute capability 2.0 and upwards have an L1 cache for each SM and a unified L2 cache.[51] The reduced cache size on a GPU, due to the absence of the L3 cache, makes the consideration of memory bandwidth even more important. Although having a large number of threads can hide memory fetch and instruction execution latency to some extent, through context switching to other warps, poor use of memory can significantly limit the performance of an application. The number of memory requests should be minimised and the amount of work done with the data fetched should be maximised.[20] Effective use must be made of the different memory spaces available, which vary in their bandwidth, latency, size, scope and lifetime. The main types of GPU memory are global memory, shared memory and register memory. Read-only, cached constant memory and texture memory can also be used, though they are simply virtually-addressed areas of the global memory. These memory types are less frequently used on newer devices, which have an L1 and L2 cache.[51] The scope and approximate relative sizes of the three main GPU memory types are shown in Figure 2.2.

The majority of the available GPU memory is global device memory. A Tesla K20 card has around 5 GB of global memory[43,44] and a Titan Black card has approximately 6 GB.[45,46] These values are several times greater than the available memory on earlier cards,[20] though still several times less than on the most recent cards.[35] Global memory is accessible from the host and is therefore used for communication between host and device. However, transfer speeds are slow, with a maximum bandwidth of less than 10 GB/s.[20] Therefore, the number of transfers between host and device should be minimised, which can sometimes mean that it is appropriate to implement certain non-parallel aspects of an algorithm on GPU.[52] The global memory is also available to any thread in any block over the lifetime of the application, so it can be used for inter-block communication. The global memory bandwidth of NVIDIA GPUs has increased by more than two orders of magnitude over the last fifteen

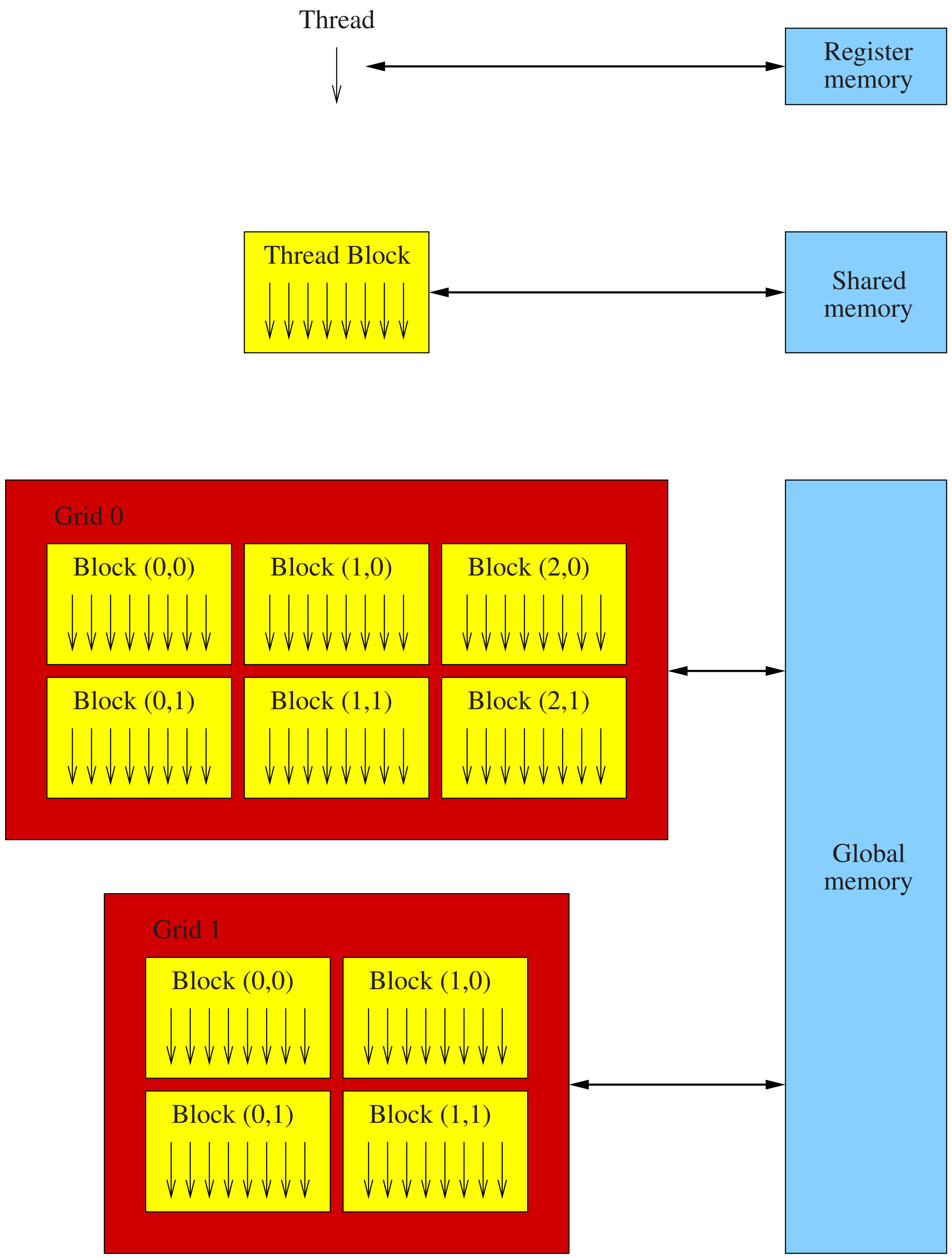


Figure 2.2 The memory hierarchy on a GPU. Figure redrawn from reference 22.

years.[20,34,35] A Tesla K20 card has a memory bandwidth of 208 GB/s,[43,53] while a Titan Black card has a bandwidth of 336 GB/s.[45,46] The global memory is considered to be high latency memory relative to the other memory types available on the GPU, so it is also important to minimise the number of relatively slow accesses to this type of memory from within a kernel.[20] If the thread memory access pattern for a warp is sequential and the start of the memory area is suitably aligned, the accesses are coalesced by the hardware, and all data elements can be served with a single memory transaction.[22] For older devices, the penalty for non-coalesced accesses was severe. However, newer devices have a L1 and L2 cache, so memory accesses are coalesced into as few cache lines as possible. On these devices, the performance penalty for misalignment is negligible and the extra cost for non-sequential access is only significant when accesses are widely separated in address space.[20]

Each SM also has an area of relatively low access latency shared memory, which is mainly used to allow threads within a block to share data, as it has the lifetime of the block.[21] It is effectively a programmer-managed L1 cache, which can be much more effective than a hardware-managed cache, as the programmer has a greater knowledge of the potential for data reuse within a program. This memory has an access latency of a just a few clock cycles, which is low compared to the hundreds of clock cycles required for access to global memory.[20] On both of our Kepler-based GPU models, there is a 64 kB area of memory per SM, 75 % of which is allocated to the L1 cache and the remainder of which constitutes the shared memory. However, this ratio of allocation can be reversed in favour of the shared memory if desired.[33] The size of this memory area contrasts with the much larger size of the L2 cache at 1536 kB. Shared memory is divided into banks, each of which is 64 bits wide on Kepler, and can service only one operation per cycle. Threads should access separate banks to avoid bank conflicts, which result in the operations being serialised. The warp stalls while the serial operations are performed, i.e. context switching to another warp cannot occur. One exception to the requirement for single threads to access single banks is when all the threads of a warp read from the same bank address, which causes the value to be broadcast across the warp in a single cycle.[21] Shared memory can also be used as a staging area for loads from global memory, which is useful when coalesced access cannot be guaranteed. Instead, segments of global memory are copied to shared memory and accessed from there, where coalescing restrictions do not apply.[20] Although shared memory can be useful, compute capability 3.0 introduced the shuffle instruction, which allows threads within a warp to communicate directly to share values. This has a large benefit over the use of shared memory, as there is no associated memory access overhead.[20] Furthermore, the multiple instructions required to use shared memory are replaced by a single shuffle instruction and there is a reduced need for explicit synchronisation of threads.[54]

Each thread can also make use of registers, which have the fastest access. Access to registers is effectively instantaneous, as they have a memory latency of only one clock cycle.[20] Kepler GPUs have 65 536 32-bit registers per SM, which are partitioned between the running blocks.[33] Having a large register file means that the data held in registers does not have to be stored and then restored again when execution switches between different sets of threads, as it would be on a CPU. This arrangement means that context switching between warps has zero overhead, and so is very effective in hiding memory latency, providing there are enough active threads.[20] Despite the large number of registers, it is still possible to request more registers per thread than are available. On older GPUs, this causes the data to spill into global memory, greatly reducing the performance of the application. On newer GPUs, the L1 cache is used instead, although this setup is still not ideal as the L1 cache is then unavailable for other purposes.[20] Furthermore, requesting too many registers per thread reduces the occupancy of the SMs, which can degrade performance.[48] However, higher occupancy does not always equal better performance. If slower types of memory must be used in order to run more blocks per SM, performance may decrease.[55] Moreover, the warp schedulers on Kepler are able to issue two instructions at once, so it is usually beneficial to incorporate instruction level parallelism within a kernel. This effect can be achieved by having multiple independent computations within a kernel that can occur simultaneously. This is sometimes possible through the unrolling of small loops, which then also benefit from the elimination of loop overhead. Instruction level parallelism greatly increases performance, despite the increased register usage and attendant lower occupancy.[20]

The complexity of developing code for GPUs arises not only from the requirement for parallelisation, but also from the necessity for detailed knowledge of the underlying hardware.[18] Many possibilities for optimisation exist, not all of which are covered here, and not all of which will be relevant to every application. Due to this large number of considerations, the best way to approach CUDA development is through an iterative process of applying a single optimisation strategy, profiling the application, and then applying further optimisations only if required based on this analysis. This can be a time-consuming process, as many different options may need to be implemented and compared before the best one is found.[20] Subsequent optimisations also tend to produce diminishing returns in performance for the amount of time required for implementation. Another aspect of CUDA development that contributes to the increased development time required relative to CPU is the increased complexity of debugging any errors that occur. Race conditions are possible on GPU hardware, where multiple threads attempt to access a single value and the output is then dependent on the undefined order of execution.[56] Errors of this kind may not become apparent until some time after they were introduced, making detection more challenging. Furthermore, debugging using breakpoints and single stepping delays the thread being observed, which can cause the problem to disappear under detailed observation.[20] Another type of error that may not be immediately apparent after its introduction is the erroneous indexing of arrays in terms of block size and grid size. Problems may not be seen until an example of a certain size is tried, so extensive testing must be performed after every major change. CUDA performs very few runtime checks, so the programmer must explicitly wrap all of their function calls in code that checks and handles errors that arise. However, the reported location of errors can be misleading if the error checking around even a single function is missed, and the occurrence of an ‘unknown error’ is not infrequent.[20] In summary, impressive speedups of several orders of magnitude may be obtained using GPUs, but the time and effort required for development is significant.

2.2 Intermolecular potentials

To investigate the potential energy landscape of a system, a method of calculating the potential energy is required. We frequently refer to this potential energy function simply as ‘the potential’. Quantum mechanical methods may yield the most accurate energies, but their application is limited due to computational expense. Therefore, we often use empirical potentials or force fields that calculate the energy as a function of nuclear positions. The basic functional forms used for an empirical potential are parameterised to reproduce certain properties, using quantum mechanical or experimental data as a reference. In some cases, empirical potentials are capable of providing results as accurate as quantum mechanical calculations, at a fraction of the computational cost.[3] The two potentials used in this work are described below.

2.2.1 Lennard-Jones

The Lennard-Jones (LJ) potential takes the form

$$V_{\mathrm{LJ}} = 4\varepsilon \sum_{i<j} \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], \tag{2.1}$$

where r_{ij} is the distance between particles i and j, ε is the pair equilibrium well depth and $2^{1/6}\sigma$ is the pair equilibrium separation.[57] It can be used to model van der Waals (vdW) interactions between atoms or molecules and consists of a long-range attractive r^{-6} term and a short-range repulsive r^{-12} term.[58] It can also be parameterised to approximately reproduce the behaviour of specific systems, such as the noble gases.[59] Due to its mathematical and computational simplicity, it is often used as a test system in molecular simulation. In this scenario, ε and σ are commonly set to unity for convenience.[9]

2.2.2 AMBER

AMBER is a set of molecular mechanical force fields for the simulation of biomolecules, and also a package of molecular simulation programs.[60,61] It is so named because the original program was identified as Assisted Model Building with Energy Refinement.[62] The AMBER force fields take the form[63]

$$\begin{aligned}V_{\text{AMBER}} = &\sum_{i}^{n_{\text{bonds}}} b_i(r_i - r_{i,\text{eq}})^2 + \sum_{i}^{n_{\text{angles}}} a_i(\theta_i - \theta_{i,\text{eq}})^2 \\ &+ \sum_{i}^{n_{\text{dihedrals}}} \sum_{n}^{n_{i,\max}} (V_{i,n}/2)[1 + \cos(n\phi_i - \text{ł}_{i,n})] \\ &+ {\sum_{i<j}^{n_{\text{atoms}}}}' \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right) + {\sum_{i<j}^{n_{\text{atoms}}}}' \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}.\end{aligned} \tag{2.2}$$

Simple harmonic expressions are used for the bond and angle terms, with equilibrium bond distance $r_{i,eq}$, equilibrium bond angle $\theta_{i,eq}$ and force constants b_i and a_i. The torsional potentials for the dihedral angles, ϕ_i, are represented by a truncated Fourier expansion in which the individual terms have a potential $V_{i,n}$ with periodicity n and phase shift $\text{ł}_{i,n}$. The vdW interactions are represented by a potential of the LJ form with diatomic parameters A_{ij} and B_{ij}. The last term is the electrostatic interaction between atom-centred point charges q_i and q_j, separated by a distance r_{ij}, where ε_0 is the dielectric constant in vacuum. The nonbonded vdW and electrostatic terms are only included for atoms in different molecules or atoms separated by at least three bonds, as indicated by the prime symbols on the summations. The types of interactions and the distances and angles involved are shown in Figure 2.3. A distance-based cutoff can be applied to the nonbonded interactions to speed up their evaluation,[65] although the validity of this approximation is uncertain.[63] Nonbonded 1-4 interactions (between atoms separated by exactly three bonds, as shown by the numbering in Figure 2.3) are reduced using separate scaling factors for vdW and electrostatic

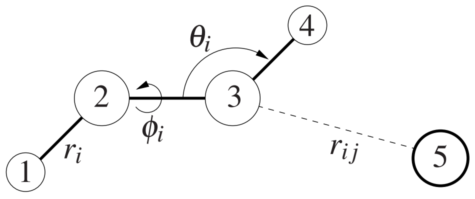


Figure 2.3 A schematic view of the types of interactions in the AMBER force field. Covalent bonds are represented by heavy, solid lines, and nonbonded interactions by light, dashed lines. The bond length is denoted by r_i, the angle between three atoms by θ_i, the dihedral angle between the two planes through atoms 1, 2 & 3 and 2, 3 & 4 by ϕ_i and the nonbonded distance between atoms i and j by r_{ij}. Figure redrawn from reference 64.

interactions, termed SCNB and SCEE, respectively.[66] SCNB reduces the exaggeration of short-range repulsion caused by the 6-12 form of the LJ potential and the nonpolarisable charge model.[67,68] The value of SCEE was chosen to approximately reproduce the conformational energies of some small molecules.[67,69]

The values of the various parameters are derived from experimental, crystallographic and quantum mechanical data and vary depending on the specific version of the force field.[65] The ff99SB force field parameter set[70] was the most recent version available at the start of this project, and it is used throughout this work for consistency. This representation provides improved secondary structure balance and dynamics relative to earlier force fields. However, it has some weaknesses in side chain rotamer and backbone secondary structure preferences. These issues have been addressed in the newer ff14SB parameter set.[71]

A symmetrised version of the AMBER force field is used throughout the present work.[72,73] The potential energy should be invariant to the permutation of identical atoms or groups, but this is not the case for the original AMBER potential. Permutational isomers can differ slightly in energy, which causes problems when constructing kinetic transition networks, as minima that should be identical are categorised as distinct. The observed energy differences originate from the way improper torsions are defined. Improper torsions are a subset of dihedral angles, where instead of forming a chain arrangement, one atom is connected to three others. This arrangement allows more than one possibility for the definition of the two planes defining the angle. In AMBER, the definition of the angle depends on the ordering of the atoms, which is determined by atom type and atom number.[74] A permutation of identical atoms can result in a different definition for the improper torsion, and therefore a different value for the energy. To symmetrise the potential, a script can be used to reorder the atoms in the input file that define the improper torsion angles, such that swaps of identical atoms do not change the definition of the angle. This reordering ensures that permutational isomers always have the same energy.[72,73]

Generalised Born solvent model

AMBER potential calculations can be performed with the system in a vacuum (although this is unsupported on GPU),[75] but the use of a solvent model usually gives more physically relevant results. Explicit solvent models explicitly represent the individual solvent molecules, which allows the modelling of specific, short-range interactions with the solute, and can be more accurate. Explicit solvent calculations are carried out using periodic boundary conditions, where the central box containing the solute and solvent particles is surrounded by identical images of itself to approximate the effect of an infinite bulk fluid.[3] A long-range correction is applied to the truncated vdW interactions[76,77] and a particle-mesh Ewald (PME) approach is used for calculating electrostatic interactions beyond the cutoff.[76,78] However, the addition of the extra complexity associated with the explicit solvent degrees of freedom is undesirable for energy landscapes applications. The use of an implicit solvent model is preferred, where the solvent is represented as a homogeneous polarisable medium, which is much more computationally efficient.[60]

AMBER uses the generalised Born (GB) implicit solvent model, in which the free energy of solvation is decomposed into electrostatic and non-electrostatic parts:[79]

$$\Delta G_{\mathrm{solv}} = \Delta G_{\mathrm{el}} + \Delta G_{\mathrm{surf}}. \tag{2.3}$$

The non-electrostatic component, ΔG_{surf}, is usually approximated by the solvent-accessible surface area, multiplied by a proportionality constant derived from experiment. We introduce the GB equation below, which provides an approximation to the electrostatic component of the energy, and is itself an approximation to the linearised Poisson-Boltzmann (PB) equation.[80] The PB equation is a partial differential equation that must usually be solved using numerical methods.[3] This task is computationally expensive, and the GB approximation provides a much faster analytical alternative. In the GB model, atoms are represented by spheres of radius ρ_i with central charges q_i, filled with a low dielectric material and surrounded by a solvent with a higher dielectric constant. The GB equation, including the electrostatic screening effects of salt,[81] is generally expressed as

$$\Delta G_{\mathrm{el}} = -\frac{1}{2}\sum_{ij} \frac{q_i q_j}{f^{\mathrm{GB}}(r_{ij}, R_i, R_j)} \left(1 - \frac{\exp[-\kappa^{\mathrm{GB}} f_{ij}^{\mathrm{GB}}]}{\varepsilon_\omega}\right), \tag{2.4}$$

where ε_ω is the dielectric constant of the solvent, r_{ij} is the distance between atoms i and j, R_i and R_j are the effective Born radii of atoms i and j respectively, κ^{GB} is the Debye-Hückel screening parameter, and f^{GB} is a smooth function of its arguments.[79] Still et al. incorporated this approximation into molecular mechanics calculations and proposed a commonly used form of f^{GB} where[82]

$$f^{\mathrm{GB}} = [r_{ij}^2 + R_i R_j \exp(-r_{ij}^2/4R_i R_j)]^{1/2}. \tag{2.5}$$

The effective Born radius can be thought of as the degree of burial of an atom within a molecule, and is therefore dependent on the molecular conformation. The accurate determination of the Born radii is very important for the overall accuracy of the calculation. When using highly accurate estimates for the Born radii, the GB approach can achieve very good agreement with the full PB treatment of the electrostatic energy.[83] The popular Hawkins, Cramer and Truhlar model,[84] which we refer to as GB^{HCT}, expresses the effective Born radii as

$$R_i^{-1} = \tilde{\rho}_i^{-1} - \mathscr{I}, \tag{2.6}$$

where

$$\mathscr{I} = \frac{1}{4\pi} \int_{\mathrm{VDW}} \chi(|\mathbf{r}| - \tilde{\rho}_i) \frac{1}{r^4} d^3\mathbf{r}. \tag{2.7}$$

An adjusted value of the atomic radius, $\tilde{\rho}_i = \rho_i - 0.09\,\text{Å}$, is used for better agreement with PB calculations.[82] The quantity $\mathscr{I}$ is a volume integral over all the solute vdW spheres, excluding the atom for which the radius is being determined (using the step-function, χ), and reflects the degree of burial of the atom. To reduce the computational expense, the integration is often approximated as a sum over atom pairs,[84,85] to which a cutoff for the number of pairs considered can be applied to further reduce the calculation time.[86] This model is implemented in AMBER as solvent model `igb` $=$ `1`.

The GB^{HCT} model for the effective Born radii was developed for small molecules and proved much less accurate for larger molecules with greater interior regions. Onufriev, Bashford and Case[79] proposed the function

$$R_i^{-1} = \tilde{\rho}_i^{-1} - \rho_i^{-1} \tanh(\alpha\psi - \beta\psi^2 + \gamma\psi^3) \tag{2.8}$$

for the effective Born radii, where $\psi = \mathscr{I}\tilde{\rho}_i$ and α, β and γ are adjustable dimensionless parameters that rescale the function proportionally. This procedure corrects for the underestimation of the effective radii for deeply buried atoms, caused by interstitial spaces between vdW spheres behaving as though filled with solvent. The hyperbolic tangent function also places a cap on the value of R_i, preventing it from diverging and becoming negative, as can happen in GB^{HCT}. The following parameter set is referred to as GB^{OBC} and is implemented in AMBER as the solvent model $\mathtt{igb} = 2$:

$$GB^{\mathrm{OBC}}II : \alpha = 1.0, \beta = 0.8, \gamma = 4.85 \tag{2.9}$$

Relative to GB^{HCT}, GB^{OBC} has reduced bias towards the native state and is better able to model large-scale conformational changes.[79] This solvent model is used extensively in the present work.

GPU-acceleration

The AMBER developers accelerated their molecular dynamics (MD) code for both GB implicit solvent[63] and PME explicit solvent[76] using CUDA on NVIDIA GPUs. We focus here on the GB implementation described in the original paper,[63] as this is the implementation that has been interfaced with our energy landscapes programs, as described in Chapter 3. To reduce expensive copying of coordinates between host and device, the entire MD algorithm was implemented on GPU, including the potential calculation, restraints, constraints, thermostats and time step integration. For the single-GPU implementation, copying of data between CPU and GPU was restricted to the initial upload of information at the beginning of a run and some downloads required for writing to the output file and trajectory file. Cutoffs for the long-range nonbonded interactions (vdW and electrostatic) were not implemented on GPU for the GB model, as the validity of these approximations has not been determined, and the resulting acceleration made the use of infinite cutoffs computationally feasible. However, support for cutoffs in calculating the effective Born radii was included.

For the calculation of the nonbonded forces, the pairwise interactions between atoms i and j can be schematically represented as an $N \times N$ interaction matrix, as represented in Figure 2.4. This matrix can be divided into tiles of size $W \times W$, where W is the warp size and N is the number of atoms. To store the results of partial reductions, $(N/W) + 1$ output buffers per atom are allocated, each of which is three values wide to store the x, y and z contributions. Each tile stores two copies of the coordinates and associated parameters for each atom. One set for atoms j to $j+W-1$ is in registers and the second set for atoms i to $i+W-1$ is placed in shared memory.

For the off-diagonal tiles (blue in Figure 2.4), each thread starts at an offset equal to its thread ID (i.e. each thread is offset by one relative to the previous thread). Each thread then loops through the atoms in shared memory, calculating the interactions and accumulating the forces in the appropriate output buffers. The thread offset avoids race conditions in accumulation by ensuring that the threads are not simultaneously updating partial results for the same atom. The symmetry of the interactions is exploited in that once the force for atom i interacting with atom j is obtained, the result for atom j interacting with atom i is simply obtained through negation.

On-diagonal tiles (red in Figure 2.4) include some self-interaction terms and so must be treated differently to off-diagonal tiles. If the same thread offset approach were used as for the off-diagonal tiles, race conditions would occur where more than one thread would simultaneously try to update the partial result for a particular atom. Instead, no offset is used and all of the interactions for atom i with atoms j of the tile are calculated simultaneously, before stepping to the next column. The symmetry of the interactions is not exploited in this approach.

Finally, a kernel using one thread per atom is used to cycle through the output buffers and sum the partial results to obtain the total force on each atom. In AMBER 12, the use of shared memory is replaced by faster operations using shuffle functions, available for devices of compute capability 3.0 or higher. The reductions required for the calculation of the GB terms and the bonded and 1-4 interactions are handled using methods analogous to those used for calculating the nonbonded terms.

The AMBER code supports several precision models for the calculation of the contributions to the nonbonded forces and their accumulation. Their initial implementation in AMBER 11 included SPDP (nonbonded force contributions calculated using single precision floating-point arithmetic, but bonded terms and force accumulation performed using double precision), SPSP (everything calculated using single precision) and DPDP (everything calculated using double precision). Using single precision is faster than using double precision, but single precision rounding errors in the force accumulation lead to unphysical MD trajectories. The developers therefore recommended SPDP for MD in their initial implementation as this provided sufficient accuracy at a reduced computational cost relative to the DPDP model.

AMBER 12 introduced the SPFP precision model,[87] designed to provide increased performance relative to SPDP simulations with comparable numerical accuracy. An IEEE754 double precision number has an approximate relative precision of 10^{-16} and the largest representable number of this type is approximately 1.8×10^{308}. Typical MD simulations do not require the level of precision and dynamic range offered by 64-bit double precision floating-point numbers. The AMBER developers instead use fixed-point 64-bit integer representations $Q\hat{m}.\hat{f}$, where $\hat{m}$ is magnitude bits and $\hat{f}$ is fractional bits. This represent-

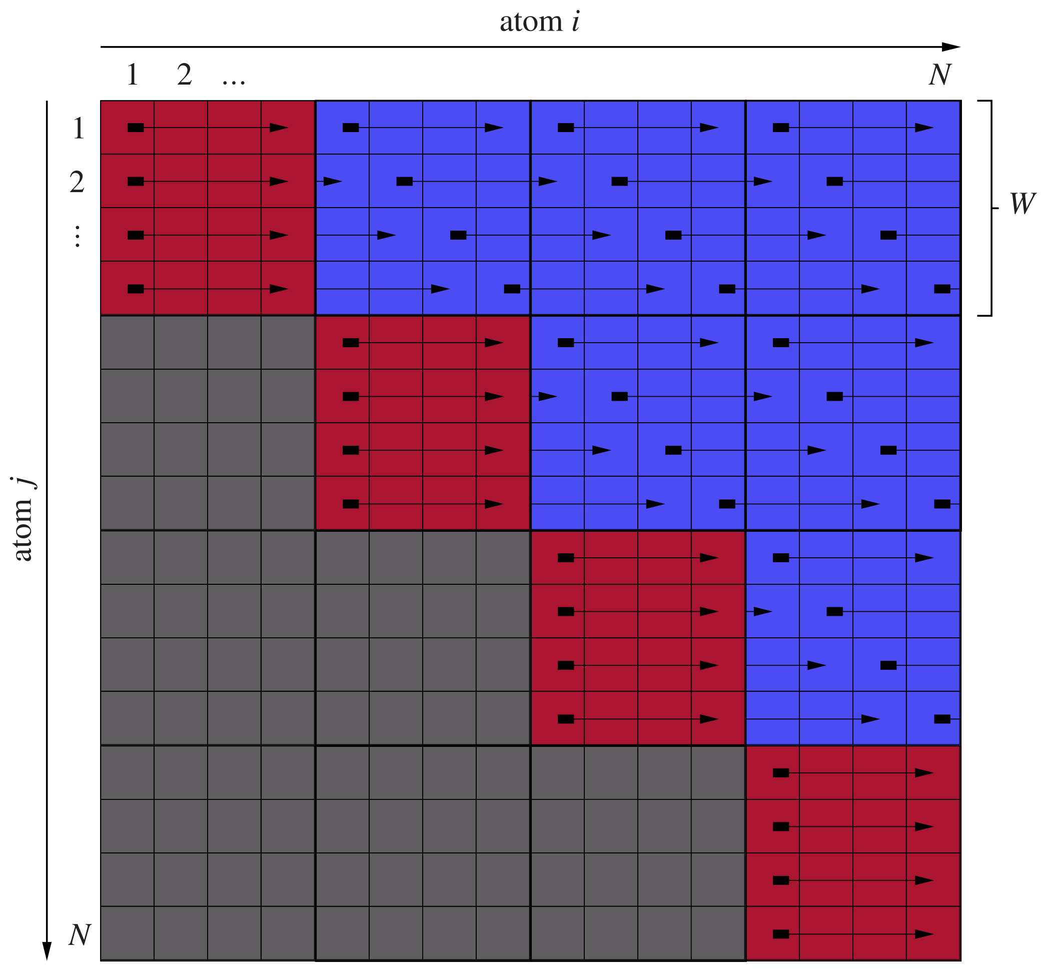


Figure 2.4 A schematic representation of the $N \times N$ interaction matrix used for calculating the nonbonded forces on GPU. A distinction is made between the on-diagonal tiles (red) and the off-diagonal tiles (blue). It should be noted that the actual warp size on NVIDIA hardware is 32. Figure redrawn from reference 63.

ation provides optimal performance on GPUs of the Maxwell architecture and consumer GPUs, where single precision arithmetic usually greatly outperforms double precision. For the force accumulation they use Q24.40 fixed-point integer accumulation. This approach provides seven significant figures to the left of the radix point and 12 to the right, which is more than enough range for a stable MD simulation. Energy accumulation uses Q34.30 fixed-point integers, which provides approximately 10 significant figures to the left of the radix point and nine to the right. The energy does not affect the MD trajectory in any way and is simply written to the output file.

As integer mathematics is associative, the order of summation will not lead to rounding differences, so atomic operations can be used for accumulation in the SPFP model and the force calculation is still deterministic. Using atomic operations provides a significant speed increase for newer GPU architectures, e.g. Kepler atomic operation hardware is three times faster relative to Fermi. Also, the global memory requirements are greatly reduced as the large number of accumulation buffers is no longer needed. This formulation increased the maximum system size for GB simulations from tens of thousands of atoms to millions, depending on the device global memory size. A global memory size of 2 GB allows the simulation of around a million atoms, where meaningful timescales are currently out of reach for proper sampling, due to computational expense. Another possible limit on system size now arises from the possibility of overflowing the fixed precision energy accumulator, as the energy increases linearly with the number of atoms, although this limit would be well past 10 million atoms.

Although the SPFP model provides large speed increases for MD, we have exclusively used the DPDP precision model in this work. This is because our methods require a much greater numerical range for the energies and forces than the SPFP model provides. Step-taking in basin-hopping and the interpolation procedure used in transition state location often produce initial highly strained structures with extremely large forces acting upon the atoms. We wish to retain as many stationary points as possible located from these starting points to maximise the efficiency of the energy landscape exploration. However, use of fixed-point or single precision arithmetic results in frequent overflow of the forces and subsequent failures of attempts to locate minima or transition states. We therefore use AMBER 12, as the DPDP precision model has been deprecated in newer releases of AMBER.

2.3 Basin-hopping global optimisation

The process of finding the lowest minimum of a function is termed global optimisation.[1] The basin-hopping method, based on the 'Monte Carlo-minimization' approach of Li and Scheraga,[8] is a stochastic global optimisation procedure that has been applied with great success in the context of potential energy surfaces to find the lowest energy structures of many atomic, molecular and macromolecular systems.[9] The database of local minima found during the search may be used to calculate other properties of interest using statistical mechanics.[88] The GPU-acceleration of the basin-hopping global optimisation implementation found in the software package GMIN[89] is described in Chapter 3.

The essential aspect of the method is a transformation of the original energy landscape according to

$$\widetilde{V}(\mathbf{X}) = \min\{V(\mathbf{X})\}, \tag{2.10}$$

where min indicates that a local minimisation is carried out from the starting coordinates, $\mathbf{X}$, resulting in the energy of the local minimum at this point, $\widetilde{V}(\mathbf{X})$. The transformed landscape takes the form of a collection of interpenetrating plateaux, or 'basins of attraction', each of which represents the set of points in configuration space that will lead to a particular minimum through geometry optimisation.[1] Figure 2.5 shows a schematic view of this landscape transformation.

The coordinate space is explored stepwise using random structural perturbations. These moves are often highly specific to the system under study. For biomolecules, the perturbations might be Cartesian moves of the backbone atoms, rotations of amino acid side chains, or short MD runs. After each step, a local minimisation is performed. The Metropolis criterion can be employed to decide if the proposed coordinates are to be accepted or rejected. If the new energy is less than the old energy, $\widetilde{V}_{\text{new}} < \widetilde{V}_{\text{old}}$, then the step is accepted. If $\widetilde{V}_{\text{new}} > \widetilde{V}_{\text{old}}$, the step is only accepted if $\exp\{(\widetilde{V}_{\text{old}} - \widetilde{V}_{\text{new}})/k_B T\}$ is greater than a number randomly chosen from between 0 and 1, where k_B is Boltzmann's constant and T is a fictitious temperature.[1] The main advantage of this method is the ease with which areas of the landscape separated by high barriers can be explored, as downhill barriers are effectively removed from the problem through the coordinate transformation.[9]

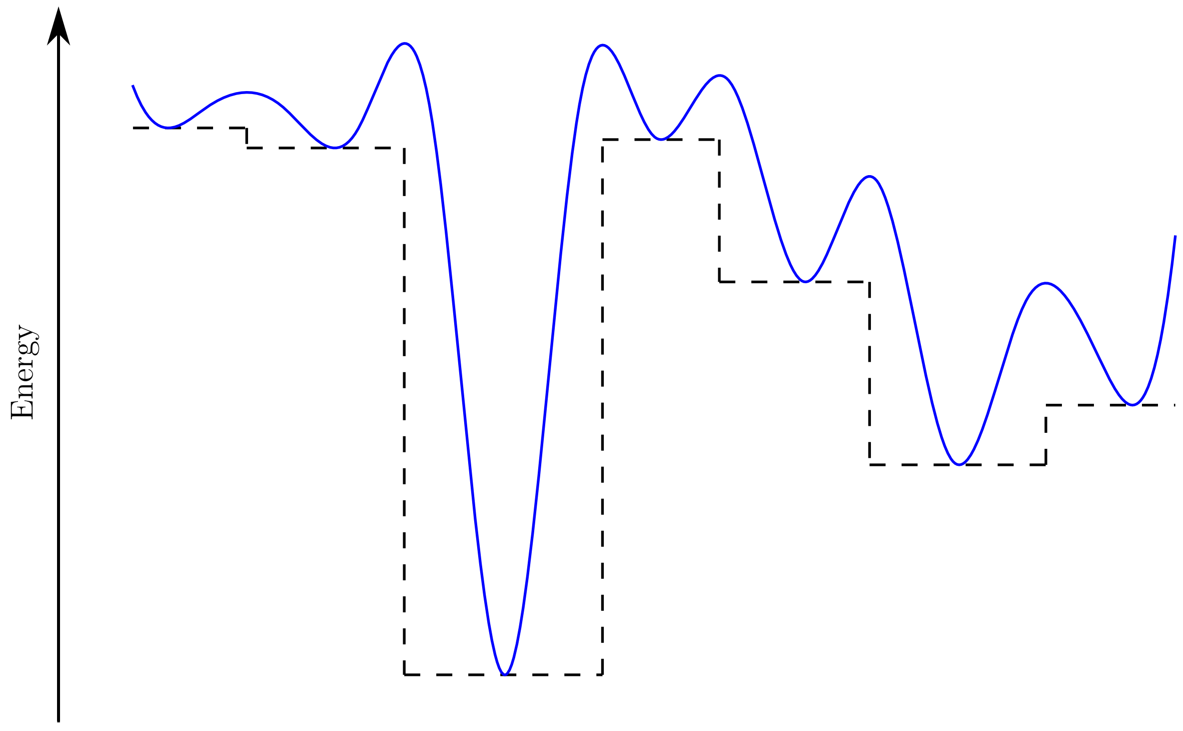


Figure 2.5 An untransformed, one-dimensional potential function is represented by a solid blue line. The function obtained through the coordinate transformation of Equation (2.10) is represented by the dotted line. Figure redrawn from reference 1.

2.3.1 Limited-memory BFGS

The local minimisations required for basin-hopping global optimisation are usually performed using Nocedal's limited-memory BFGS (L-BFGS) algorithm.[90] This is the limited-memory version of the BFGS algorithm, named for Broyden,[91] Fletcher,[92] Goldfarb[93] and Shanno[94]. It belongs to the family of quasi-Newton methods,[95] which have their origins in Newton's method. However, they are less computationally expensive than Newton's method, as they do not require explicit calculation of the Hessian matrix of second derivatives. Instead, the Hessian or its inverse is approximated using a series of matrices determined from the gradient at each iteration. As the name suggests, L-BFGS is based on the BFGS method but requires less storage because the approximation to the full inverse Hessian is not stored in its entirety. Instead, it is represented implicitly through a small number of vector pairs that are updated at each iteration.[90] An overview of the L-BFGS algorithm for the minimisation of a function, f, is shown in Algorithm 1.[96] For our purposes, the function f is usually a potential energy function, $V(\mathbf{X})$, such as the LJ or AMBER potential.

Algorithm 1 L-BFGS

```
1:  Choose a starting point, $\mathbf{X}_0$, and an integer history size, $m > 0$
2:  $k = 0$
3:  **repeat**
4:      Choose an initial Hessian approximation, $H_k^0$, e.g. using Equation (2.11)
5:      Compute the step direction, $\mathbf{P}_k = -H_k \nabla f_k$, using Algorithm 2
6:      Compute the step $\mathbf{X}_{k+1} = \mathbf{X}_k + \hat{\alpha}_k \mathbf{P}_k$, choosing $\hat{\alpha}_k$ as in Algorithm 3
7:      **if** $k > m$ **then**
8:          discard vector pair $\{\mathbf{S}_{k-m}, \mathbf{Y}_{k-m}\}$ from storage
9:      **end if**
10:     Compute and store $\mathbf{S}_k = \mathbf{X}_{k+1} - \mathbf{X}_k, \mathbf{Y}_k = \nabla f_{k+1} - \nabla f_k$
11:     $k = k + 1$
12: **until** convergence.
```

Unlike the BFGS method, the initial approximation to the Hessian, H_k^0 , may vary freely between iterations. One commonly used method is to set $H_k^0 = \hat{\gamma}_k I$, where I is the identity matrix. We calculate

$$\hat{\gamma}_k = \frac{\mathbf{S}_{k-1}^T \mathbf{Y}_{k-1}}{\mathbf{Y}_{k-1}^T \mathbf{Y}_{k-1}}. \tag{2.11}$$

using the change in coordinates, $\mathbf{S}_{k-1}$, and the change in gradient, $\mathbf{Y}_{k-1}$, from the previous iteration. This scaling factor, $\hat{\gamma}_k$, is an estimate of the magnitude of the second derivative along the most recent search direction, ensuring that $\mathbf{P}_k$ is appropriately scaled.[96]

The BFGS update formula is

$$H_{k+1} = \eta_k^T H_k \eta_k + \xi_k \mathbf{S}_k \mathbf{S}_k^T \tag{2.12}$$

where

$$\xi_k = \frac{1}{\mathbf{Y}_k^T \mathbf{S}_k} \tag{2.13}$$

and

$$\eta_k = I - \xi_k \mathbf{Y}_k \mathbf{S}_k^T. \tag{2.14}$$

From this equation, a procedure for calculating $H_k \nabla f_k$ can be derived.[96] This formulation is known as the two-loop recursion algorithm and is shown in Algorithm 2.[96]

Algorithm 2 Two-loop recursion

```
 1: $\hat{\mathbf{D}} = \nabla f_k$
 2: for $i = k-1, k-2, ..., k-m$ do
 3:     $\hat{\alpha}_i = \xi_i \mathbf{S}_i^T \hat{\mathbf{D}}$
 4:     $\hat{\mathbf{D}} = \hat{\mathbf{D}} - \hat{\alpha}_i \mathbf{Y}_i$
 5: end for
 6: $\mathbf{J} = H_k^0 \hat{\mathbf{D}}$
 7: for $i = k-m, k-m+1, ..., k-1$ do
 8:     $\hat{\beta} = \xi_i \mathbf{Y}_i^T \mathbf{J}$
 9:     $\mathbf{J} = \mathbf{J} + \mathbf{S}_i(\hat{\alpha}_i - \hat{\beta})$
10: end for
11: stop with result $H_k \nabla f_k = \mathbf{J}$.
```

We wish to find an appropriate step length, $\hat{\alpha}_k$, for the step in the chosen direction, $\mathbf{P}_k$, according to line 6 of Algorithm 1. Taking steps that are frequently too large or too small will result in the overall minimisation being inefficient. The best reduction in the function, f_k, would be given by a one-dimensional minimisation in the direction of the search, but this would be expensive if performed exactly. Instead, the minimisation is often performed approximately using a line search algorithm. A good line search finds a step length that results in an adequate reduction in the function at minimal computational cost. One commonly used line search requires the proposed step to satisfy the Wolfe conditions, which require both a sufficient decrease in the function and a sufficient change in the curvature (second derivatives) of the function.[96] In practice, the L-BFGS algorithm produces a well-scaled initial step, so a simpler procedure can be used. The approach used in GMIN checks whether the step is a descent direction, and ensures that it does not exceed a maximum step size and maximum energy rise, as specified by the user. This procedure results in fewer overall convergence failures for L-BFGS than a standard line search,[97] and is further described in Algorithm 3.

Algorithm 3 Step scaling in GMIN

```
 1: if $\mathbf{J}_k^T \nabla f_k > 0$ then                 ▷ Check if step, $\mathbf{J}_k$, is a descent direction
 2:     $\mathbf{J}_k = -\mathbf{J}_k$                        ▷ If not, invert the step
 3: end if
 4: $c = 1.0$                                                  ▷ Initialise scaling factor for step size
 5: if $c|\mathbf{J}_k| > |\mathbf{J}|_{\max}$ then   ▷ Check if initial step larger than allowed maximum, $|\mathbf{J}|_{\max}$
 6:     $c = |\mathbf{J}|_{\max}/|\mathbf{J}_k|$           ▷ If it is, find new scaling factor to reduce it
 7: end if
 8: for $i = 1, 2, ..., 10$ do
 9:     $\mathbf{X}_{k+1} = \mathbf{X}_k - c\mathbf{J}_k$      ▷ Calculate coordinates of new, scaled step
10:     if $f_{k+1} - f_k < \Delta f_{k,\max}$ then           ▷ Check energy change for new step
11:         break       ▷ If less than allowed maximum, $\Delta f_{k,\max}$, accept current value of $c$
12:     else                                   ▷ If energy change greater than $\Delta f_{k,\max}$
13:         $c = c/10$                                        ▷ Reduce $c$ by a factor of 10
14:     end if
15: end for
16: stop with result $\hat{\alpha}_k = c$.
```

2.4 Transition state determination

In our software, transition states are usually located using the doubly-nudged[13] elastic band[14,15] (DNEB) method to generate initial guesses between two minima, and then these candidates are refined using hybrid eigenvector-following[12] (EF). Then, small positive and negative displacements along the eigenvector associated with smallest nonzero eigenvalue are made and L-BFGS minimisations are performed from these points to determine the minima directly connected to the transition state.[5] The software package OPTIM[98] contains an implementation of these procedures and Chapter 3 describes the details of acceleration using GPUs. The minima to be connected may be distant from each other, separated by many intervening minima and transition states. In this case, a complete connected pathway cannot be found in a single application of DNEB and hybrid EF, so this procedure for transition state determination must be applied many times in succession. We use Dijkstra's shortest path algorithm[99] on the evolving database of local minima and transition states to choose the endpoints for subsequent searches.[5]

2.4.1 Doubly-nudged elastic band method

The DNEB method is used for double-ended transition state searches, where the objective is to locate transition states that join two end points.[1] For our purposes, these initial endpoints are minima and we aim to find an approximate transition state for further refinement. The DNEB method is derived from Henkelman and Jónsson's nudged elastic band (NEB) method,[14,15] with modifications to increase its stability, efficiency and speed when used in conjunction with minimisation algorithms such as L-BFGS.[13] The path between the two endpoints, $\mathbf{X}_0$ and $\mathbf{X}_{n+1}$, is represented as a series of n images, $\{\mathbf{X}_1, \mathbf{X}_2 \ldots \mathbf{X}_n\}$, where $\mathbf{X}_i$ represents the coordinates of image i. The initial image structures are generated using a simple Cartesian interpolation between the endpoints and they are then refined using DNEB in combination with L-BFGS. Each image is represented by a 'true potential', V_i. In addition, adjacent images are joined by $n+1$ harmonic springs,[12] where the 'spring potential' is

$$V_{\text{spr}} = \frac{1}{2} k_{\text{spr}} \sum_{i=1}^{n+1} |\mathbf{X}_i - \mathbf{X}_{i-1}|^2. \tag{2.15}$$

However, the perpendicular component of the spring gradient, $\mathbf{g}_{\text{spr}}^{\perp}$, can result in corner-cutting where the images are pulled away from the path. Furthermore, the parallel component of the true gradient, $\mathbf{g}^{\|}$, can cause images on the path to slide down towards the end points. These problematic effects mean that poor transition state candidates are produced. In the original NEB formulation, these problems were ameliorated by completely projecting out the corresponding components of the gradient.[14,15] However, when this new gradient is used in combination with the L-BFGS minimiser, it is unstable, as the images diverge from the path. This problem was found to be due to the complete removal of the perpendicular component of the spring gradient, $\mathbf{g}_{\text{spr}}^{\perp}$. A solution is to retain some portion of $\mathbf{g}_{\text{spr}}^{\perp}$, giving a new DNEB gradient of[13]

$$\mathbf{g}_{\text{NEB}} = \mathbf{g}^{\perp} + \mathbf{g}_{\text{spr}}^{\|} + \mathbf{g}_{\text{spr}}^{*}, \tag{2.16}$$

where

$$\mathbf{g}_{\text{spr}}^{*} = \mathbf{g}_{\text{spr}}^{\perp} - (\mathbf{g}_{\text{spr}}^{\perp} \cdot \hat{\mathbf{g}}^{\perp})\hat{\mathbf{g}}^{\perp}. \tag{2.17}$$

In this context, the hat symbol denotes a unit vector. The improved stability of the DNEB method makes it two orders of magnitude faster than the original NEB method.[13] Some corner-cutting still occurs, but the approximate transition states are good enough for further refinement through hybrid EF.

2.4.2 Hybrid eigenvector-following

Hybrid EF is a single-ended transition state search method, meaning that it starts from just one initial point. In this context, that point is an approximate transition state produced using the DNEB method and we wish to use hybrid EF to tightly converge this transition state candidate.[13] In this process, uphill steps with respect to the input structure are taken, along the eigenvector associated with the unique negative Hessian eigenvalue. Minimisation is performed concomitantly in the orthogonal subspace until a transition state is reached.[12] Due to the changing geometry, the uphill eigenvector is recalculated frequently to ensure that we continue in the right direction. For large systems, where we wish to avoid the expense of computing and diagonalising the entire Hessian, or in cases where analytical second derivatives are not available, a variational approach can be used to calculate the smallest eigenvalue and the associated eigenvector. Analytical second derivatives for the AMBER potential have not been implemented for GPU hardware, so we use this variational approach in Chapter 3.

In the variational approach to finding the smallest eigenvalue, we minimise a Rayleigh-Ritz ratio

$$\lambda(\mathbf{x}) = \frac{\mathbf{x}^T H \mathbf{x}}{\mathbf{x}^2} \tag{2.18}$$

with respect to the eigenvector $\mathbf{x}$ associated with the smallest eigenvalue $\lambda(\mathbf{x})$ of the Hessian, H. The eigenvalue, $\lambda(\mathbf{x})$, can be approximated as a numerical second derivative of the energy using the central difference approximation

$$\lambda(\mathbf{x}) \approx \frac{V(\mathbf{X}+\zeta\mathbf{x}) + V(\mathbf{X}-\zeta\mathbf{x}) - 2V(\mathbf{X})}{(\zeta\mathbf{x})^2} \tag{2.19}$$

where $V(\mathbf{X})$ is the energy at point $\mathbf{X}$ in nuclear configuration space and $\zeta \ll 1$. Differentiating Eq. (2.19) gives:

$$\frac{\partial\lambda}{\partial\mathbf{x}} = \frac{\nabla V(\mathbf{X}+\zeta\mathbf{x}) - \nabla V(\mathbf{X}-\zeta\mathbf{x})}{\zeta\mathbf{x}^2} - \frac{2\lambda\mathbf{x}}{\mathbf{x}^2}. \tag{2.20}$$

Roundoff error in Eq. (2.19) can result in loss of precision for systems with large values of $V(\mathbf{X})$. An alternative, mathematically equivalent formulation of $\lambda(\mathbf{x})$ using

$$\lambda(\mathbf{x}) \approx \frac{\{\nabla V(\mathbf{X}+\zeta\mathbf{x}) - \nabla V(\mathbf{X}-\zeta\mathbf{x})\} \cdot \mathbf{x}}{2\zeta\mathbf{x}^2} \tag{2.21}$$

gives a significantly better estimate in these cases, especially when the magnitude of the gradient is small in comparison to the energy.[100] The same estimate for $\partial\lambda/\partial\mathbf{x}$ is used, as shown in Equation (2.20). Components of the eigenvector corresponding to overall translation or rotation of the system are removed by calculating unit vectors for infinitesimal translational and rotational displacements and using these in a projection procedure. As the eigenvector is approximate, it is also normalised at every iteration of the minimisation. After the eigenvector associated with the smallest nonzero eigenvalue has been found, an uphill eigenvector-following step is taken in this direction, with minimisation in all orthogonal directions. To prevent minimisation in the uphill direction, $\mathbf{x}$, the projection

$$\mathscr{P}(\nabla V) = \frac{\nabla V - (\nabla V \cdot \mathbf{x})\mathbf{x}}{|\mathbf{x}|^2} \tag{2.22}$$

is used to remove the component of the gradient along $\mathbf{x}$. This procedure is continued until a converged stationary point is reached.

2.5 Local rigid body framework

The local rigid body framework in our programs, GMIN and OPTIM, provides a way of reducing the number of degrees of freedom of a molecular system, while still retaining full atomistic interactions between the bodies.[10,11] Local rigidification can be applied in cases where groups of atoms move very little with respect to each other, or when their displacements are not of relevance to the problem under investigation. The reduction in the number of degrees of freedom in the transformation from atomistic to rigid body coordinates, $\{\mathbf{r}_i\}_{i\in I} \to \{\mathbf{r}_I, \mathbf{p}_I\}$, greatly increases computational efficiency. Tests on small biological systems with rigid groupings of peptide termini, peptide bonds and side chain rings have shown that mean first encounter times for locating the global minimum using basin-hopping global optimisation are significantly reduced. These tests, performed on CPU, also showed that both the required number of steps in the L-BFGS minimisations and the required number of basin-hopping steps are reduced. Moreover, the topology of the energy landscape may be relatively unaffected if groupings are chosen carefully.[10,101] The groupings are frequently chosen using physical intuition regarding the intrinsic rigidity of the structural elements, although more formal approaches do exist.[102–105]

One great advantage of this method is the ability to combine it with potentials that require atomistic coordinates as input,[11] such as AMBER. For this combination to be possible, a method of converting rigid coordinates to atomistic coordinates prior to the potential call is required, and subsequently a way of projecting the calculated forces on the atoms to obtain the forces and torques on the rigid bodies.[10] As these transformations occur during L-BFGS minimisation, their implementation on GPU hardware is described in Chapter 3. The inverse procedure for converting atomistic coordinates to rigid coordinates is also occasionally required, but it is not the focus of the present work as it does not need to occur during L-BFGS minimisation.

The position of each rigid body is specified using three coordinates for the centre of geometry, $\{\mathbf{r}_I\}$, and three for its orientation, $\{\mathbf{p}_I\}$. These three translational and three rotational degrees of freedom therefore give each rigid body a total of six degrees of freedom.[11] In this discussion, capital letters are used to refer to rigid bodies and lower case letters to atoms. The reference coordinates of the atoms in rigid body I, relative to the centre of geometry, are denoted by $\{\mathbf{r}_i^0\}_{i \in I}$. The rotational degrees of freedom are represented using an angle-axis framework.[106] The rotation vector can be expressed as $\mathbf{p}_I = \theta_I \hat{\mathbf{p}}_I = (p_I^1, p_I^2, p_I^3)$, where $\hat{\mathbf{p}}_I$ is a unit vector defining the rotation axis and θ_I describes the magnitude of the rotation about that axis.[10] The components of the rotation vector in the x, y and z directions are denoted by p_I^1, p_I^2, p_I^3, respectively. The rotation matrix, R_I, can be obtained from Rodrigues' rotation formula[107] as

$$R_I = I + (1 - \cos\theta_I)\tilde{\mathbf{p}}_I \tilde{\mathbf{p}}_I + \sin\theta_I \tilde{\mathbf{p}}_I, \tag{2.23}$$

where I is the identity matrix. The skew-symmetric matrix, $\tilde{\mathbf{p}}_I$, is defined as

$$\tilde{\mathbf{p}}_I = \frac{1}{\theta_I} \begin{pmatrix} 0 & -p_I^3 & p_I^2 \\ p_I^3 & 0 & -p_I^1 \\ -p_I^2 & p_I^1 & 0 \end{pmatrix}, \tag{2.24}$$

which is constructed using the rotation vector $\mathbf{p}_I$. The formula

$$\mathbf{r}_{i \in I} = \mathbf{r}_I + R_I \mathbf{r}^0_{i \in I} \tag{2.25}$$

defines the transformation from rigid body coordinates to atomistic coordinates.[10]

The projection of the atomistic forces onto the translational degrees of freedom of the rigid bodies, $\partial V / \partial r_I^k$ $(k = 1,2,3)$, is given by the sum[10]

$$\frac{\partial V}{\partial r_I^k} = \sum_{i \in I} \frac{\partial V}{\partial r_i^k}. \tag{2.26}$$

The accompanying projection of the forces on the atoms onto the rotational degrees of freedom is obtained through the use of the chain rule as

$$\frac{\partial V}{\partial p_I^k} = \sum_{i \in I} \nabla_i V \cdot (R_I^k \mathbf{r}_i^0), \tag{2.27}$$

where

$$\frac{\partial \mathbf{r}_i}{\partial p_I^k} = R_I^k \mathbf{r}_i^0 \tag{2.28}$$

follows geometrically from Eq. (2.25).[11] The derivative of the rotation matrix, $\partial R_I / \partial p_I^k$, is denoted by R_I^k, and can be expressed as[108]

$$R_I^k = \frac{p_I^k \sin\theta_I}{\theta_I} \tilde{\mathbf{p}}_I^2 + (1 - \cos\theta_I)(\tilde{\mathbf{p}}_I^k \tilde{\mathbf{p}}_I + \tilde{\mathbf{p}}_I \tilde{\mathbf{p}}_I^k) + \frac{p_I^k \cos\theta_I}{\theta_I} \tilde{\mathbf{p}}_I + \sin\theta_I \tilde{\mathbf{p}}_I^k. \tag{2.29}$$

Here, for example,

$$\tilde{\mathbf{p}}_I^1 = \frac{1}{\theta_I} \begin{pmatrix} 0 & p_I^1 p_I^3 / \theta_I^2 & -p_I^1 p_I^2 / \theta_I^2 \\ -p_I^1 p_I^3 / \theta_I^2 & 0 & -(1 - (p_I^1)^2 / \theta_I^2) \\ p_I^1 p_I^2 / \theta_I^2 & (1 - (p_I^1)^2 / \theta_I^2) & 0 \end{pmatrix}. \tag{2.30}$$

2.5.1 Root mean square force formulation

The root mean square (RMS) force is commonly used as a convergence criterion for geometry optimisation procedures and is defined as $\sqrt{\nabla V \cdot \nabla V / 3N}$ for an N atom system. A coordinate independent formulation for the RMS force must be defined for use with rigid systems.[11] As the translational and rotational degrees of freedom have different physical dimensions, the standard Euclidean dot product would not be correct. Instead, we define a weighted metric tensor (or distance measure)

$$g_{I,\alpha\beta} = \sum_i w_i \frac{\partial \mathbf{r}_i}{\partial q_{I,\alpha}} \cdot \frac{\partial \mathbf{r}_i}{\partial q_{I,\beta}} \tag{2.31}$$

with the generalised coordinates $\mathbf{q}_I = \{\mathbf{r}_I, \mathbf{p}_I\}$. For normal-mode analysis, the weights are equivalent to the masses of the atoms. However, for the RMS force calculation we focus on here, $w_i = 1$. By inserting Equation (2.25) into Equation (2.31), we obtain components of the metric tensor that depend on translational and rotational degrees of freedom only as

$$g_{I,\alpha\beta}^{\text{trans}} = \widetilde{W} \delta_{\alpha\beta} \tag{2.32}$$

$$g^{\mathrm{rot}}_{I,\alpha\beta} = \mathrm{Tr}(R_{I,\alpha} S_I R^T_{I,\beta}) \tag{2.33}$$

where Tr is the trace,

$$\widetilde{W} = \sum_i w_i, \tag{2.34}$$

and

$$R_{I,\alpha} = \frac{\partial R_I}{\partial p_{I,\alpha}}. \tag{2.35}$$

S_I is the weighted gyration tensor in the reference frame of the rigid body with

$$S_{I,\alpha\beta} = \sum_i w_i r^0_{i,\alpha} r^0_{i,\beta} \tag{2.36}$$

where $r^0_{i,\alpha}$ is the α component of the vector $\mathbf{r}^0_i$. As $\mathbf{r}_I$ is at the centre of geometry, mixing terms vanish.

We now consider n rigid bodies where the coordinates are $\mathbf{q} = \{q_\alpha\} = \{r_1, \ldots, r_n, p_1, \ldots, p_n\}$ and $g_{\alpha\beta}$ is the metric tensor for the full system, which is block diagonal with each block corresponding to the metric tensor of each individual rigid body. The correct dot product for the gradient of the potential energy is then

$$v^2 = \sum_{\alpha\beta} \frac{\partial V}{\partial q_\alpha} g^{-1}_{\alpha\beta} \frac{\partial V}{\partial q_\beta} \tag{2.37}$$

and the RMS force is $\sqrt{v^2/6n}$ where $6n$ is the total number of degrees of freedom.

2.6 Free energy basin-hopping

Free energy basin-hopping[16] (FEBH) provides a means to locate the potential energy minimum corresponding to the lowest local free energy minimum of a molecular or condensed matter system at a specified temperature. The procedure is analogous to basin-hopping global optimisation,[8,9] with approximate free energies calculated on the fly for each local minimum visited. This procedure allows us to take into account entropic effects that may be important at higher temperatures. FEBH has been implemented in the GMIN package and Chapter 4 discusses the refinement and testing of an accelerated implementation that exploits the sparsity of the Hessian matrix for short-ranged potentials.

The superposition approach[109,110] can be used to express the global partition function, $Z(T)$, as a sum over contributions from the catchment basins of local minima. A catchment basin is formally defined as the region of configuration space from which steepest-descent paths lead to a particular local minimum.[1] In the canonical ensemble,

$$Z(T) = \sum_i Z_i(T), \tag{2.38}$$

where $Z_i(T)$ is the partition function of minimum i at temperature T, and the sum is over all geometrically distinct minima on the surface, including all nonsuperimposable permutation-inversion isomers of each structure. For a system containing N_A atoms of element A, etc., a factor $n_i = 2N_A!N_B!N_C!\dots/o_i$ is used to account for degenerate structures contributing to $Z_i(T)$, where o_i is the order of the point group of minimum i. Employing the harmonic normal mode approximation, the vibrational component of the partition function can therefore be written as

$$Z(T) = \sum_i \frac{n_i \exp[-\beta_T V_i]}{(\beta_T h \overline{\nu}_i)^\kappa} \tag{2.39}$$

where $\beta_T = 1/k_B T$, h is Planck's constant, κ is the number of nonzero eigenvalues for the Hessian matrix, $\overline{\nu}_i = (\prod_{j=1}^{\kappa} \nu_i^j)^{1/\kappa}$ is the geometric mean vibrational frequency of minimum i with ν_i^j the normal mode frequency of the j-th mode in minimum i, and V_i is the potential energy of minimum i.[16] The approximate free energy of minimum i is given by[109]

$$F_i(T) = -k_B T \ln Z_i(T). \tag{2.40}$$

Combining equations (2.39) and (2.40), we find that the harmonic difference in free energy between the current (old) minimum and the new minimum is

$$F_{\text{new}}(T) - F_{\text{old}}(T) = V_{\text{new}} - V_{\text{old}} + k_B T \ln \frac{o_{\text{new}} \overline{\nu}_{\text{new}}^\kappa}{o_{\text{old}} \overline{\nu}_{\text{old}}^\kappa}, \tag{2.41}$$

which is used in the accept/reject criterion for each basin-hopping step. Terms corresponding to rigid rotor degrees of freedom have been shown to have a negligible effect in tests using LJ clusters.[16,111] Initial investigation, through the inclusion of rotational terms in tests on a small number of minima, indicates that this is also the case for the biomolecules under consideration in this work.

2.7 Cholesky factorisation

A Cholesky factorisation can be used to uniquely decompose a symmetric, positive definite matrix into a product of a lower triangular matrix and its conjugate transpose:[112]

$$M = LL^T. \tag{2.42}$$

The determinant of the matrix can then be calculated from the product of diagonals of L and L^T.[113] This forms the basis of the alternative method for calculating the log product of positive eigenvalues of the Hessian required for FEBH, described in Chapter 4. There are many possible ways of computing the factorisation, which all scale as $\mathscr{O}(N^3)$, though they vary in efficiency.[114] A commonly used method can be represented in the following form, where lower case letters refer to individual elements of the matrices denoted by the equivalent capital letters:[115]

$$\begin{aligned}
l_{11} &= \sqrt{m}_{11}, \\
l_{j1} &= \frac{m_{j1}}{l_{11}}, \quad j \in [2,n], \\
l_{ii} &= \sqrt{m_{ii} - \sum_{p=1}^{i-1} l_{ip}^2}, \quad i \in [2,n], \\
l_{ji} &= \left(m_{ji} - \sum_{p=1}^{i-1} l_{ip} l_{jp} \right) / l_{ii}, \quad i \in [2,n-1], j \in [i+1,n].
\end{aligned}$$

Cholesky factorisation is closely related to the LDL factorisation, $M = LDL^T$, where D is a diagonal matrix. The relationship can be expressed as:[114]

$$M = LDL^T = LD^{\frac{1}{2}} D^{\frac{1}{2}T} L^T = LD^{\frac{1}{2}} (LD^{\frac{1}{2}})^T \tag{2.43}$$

The computational complexity of the LDL factorisation is the same as for the Cholesky factorisation when both are implemented efficiently.[116] The package SuiteSparse[117] contains implementations of both Cholesky and LDL factorisation, but the developers have applied greater optimisation effort to their Cholesky implementation. The method described in Chapter 4 therefore uses the Cholesky factorisation, although a conversion of the final factor to LDL^T form is applied for more straightforward array access to the required diagonal elements.

The scaling of these algorithms can be improved if the matrix to be factorised is sparse, i.e. if only a small number of the matrix elements are nonzero. Operations on zeros can then be skipped, saving both computational time and storage.[112] This property applies to the Hessian matrix for short-ranged potentials, allowing the acceleration of FEBH using an algorithm for sparse Cholesky factorisation.

Chapter 3

GPU-acceleration of computational energy landscape methods

3.1 Introduction

Basin-hopping global optimisation,[8,9] the doubly-nudged[13] elastic band[14,15] (DNEB) method, and hybrid eigenvector-following[12] (EF) are well-established tools for the location of minima and transition states in the characterisation of potential energy landscapes. Their application to biomolecules has so far focused mainly on smaller systems, due to computational expense. We would like to open up the study of larger biomolecules within the computational potential energy landscape approach, through the use of graphics processing units (GPUs).

GPUs have become popular for the acceleration of scientific applications, and many GPU-accelerated programs exist in the area of computational chemistry. The NVIDIA GPU Applications Catalog[118] contains many examples of GPU-accelerated programs, from areas as diverse as fluid dynamics and finance, along with the speedups they have achieved. These speedups range from a factor of two or three, up to two or three orders of magnitude. The improvement depends very much upon the degree of data parallelism inherent in the application, and also upon the nature of the hardware used in benchmarking.

The work described in this chapter makes use of the GPU-accelerated AMBER potential, described in Section 2.2.2. In the benchmarks for the original publication, AMBER 11 molecular dynamics (MD) simulations using generalised Born (GB) implicit solvent and the SPDP precision model for GPU were around 10 to 20 times faster than a 12 core parallel CPU run. In general, the DPDP model was around four to seven times slower in performance than the SPDP model.[63] However, it is important to note that significant technological advances have been made in the design of GPU hardware since these numbers were recorded. In particular, many professional cards now have greatly improved double precision performance. AMBER 12 introduced the SPFP precision model, designed to increase performance without sacrificing accuracy.[87] This implementation is particularly suited to cards for which the single precision performance is very high relative to the double precision performance, such as consumer cards and cards based on the Maxwell architecture. On the latest hardware, the AMBER 16 code using the SPFP model provides two orders of magnitude speedup relative to a 20 core CPU MPI run for a system of 25 000 atoms, and MPI runs using four GPUs provide around three times this single-GPU performance. We make use of the AMBER GPU code in our applications, but we require the somewhat slower DPDP model (deprecated beyond AMBER 12) for full double precision calculation of the forces.

The limited-memory BFGS (L-BFGS) algorithm[90–94] is a key component of the computational energy landscape methods considered here and has been implemented on GPUs by various other authors.[52,119–128] In adapting any application to make use of GPU hardware, a decision must be made as to which computations should take place on the GPU and which would better be left on the CPU. In ‘GPU Computing Gems’, Haque and Pande consider this problem with reference to the L-BFGS algorithm. For their application, the function and gradient evaluation are well-suited to efficient parallelisation on the GPU because of high data parallelism and arithmetic intensity. Conversely, they find that the L-BFGS direction update is ill-suited to GPU parallelisation, as it consists of a large number of sequential, low-dimensional vector operations, requiring frequent thread synchronisation and leaving many idle threads. However, leaving the update operations on the CPU necessitates copying large amounts of data between the host and device, which is expensive. In their case, implementing the entire algorithm on the GPU turned out to be the most efficient approach.[52] D’Amore et al.[119] also point out the difficulty of parallelising the update operations in the L-BFGS routine. Although a dot product computation has a high degree of parallelism, each multiplication requires two read operations from the input vectors and one write operation to the output vector. The calculation is therefore strongly memory bandwidth bound. The upper limit for GPU-acceleration of the dot product is given by the ratio of GPU to CPU memory bandwidth, which is usually less than ten. D’Amore et al. made use of the cuBLAS library in their work, a GPU-accelerated version of the complete standard Basic Linear Algebra Subprograms (BLAS) library.

Several other authors have also implemented versions of L-BFGS with both the function and gradient evaluation and vector update operations taking place on the GPU, namely Fei et al.,[120] Zhang et al.[121] and Wetzl et al.[122] In contrast, others have opted to put only the function and gradient evaluation on the GPU.[123–128] In cases where the problem size has low dimensionality, the cost of data transfer between host and device is lowered, and the poor parallelisation of the update steps becomes more significant.[125] However, several of these authors speculate that costly memory transfer negatively impacts the performance of their applications.[126–128] The most efficient approach likely depends on the hardware, the nature of the cost function, and the L-BFGS parameters selected.

The remainder of this chapter describes the implementation of basin-hopping global optimisation, a local rigid body framework, hybrid EF and the DNEB method to provide GPU-accelerated calculations. Speedups of up to two orders of magnitude are observed for tests performed on selected biomolecules represented using the AMBER force field, and an analysis of the effects of L-BFGS history size and local rigidification on the overall efficiency is presented. A GPU implementation of L-BFGS, with an interface to the GPU-accelerated AMBER potential, forms the basis of the accelerated versions of basin-hopping global optimisation and hybrid eigenvector-following. Comparison of an implementation of L-BFGS with the update operations on GPU, with an implementation with the update operations on CPU, shows that the benefit of the full implementation of L-BFGS on GPU with the AMBER potential is only seen for very large and complex systems. However, preliminary tests on a smaller system represented using a GPU-accelerated Lennard-Jones (LJ) potential show an order of magnitude speedup relative to having the update operations on CPU. The study of much larger biomolecular systems is now possible, and the GPU implementation of the energy landscape framework provides a good basis for acceleration using other potentials in future work. Applications to a coiled-coil peptide represented using the AMBER potential[129,130] and addressable clusters and aggregates represented using the LJ potential[131] are also described.

3.2 Methods

To identify the most time-consuming components of basin-hopping global optimisation in GMIN and transition state location in OPTIM, some short profiles were performed using Callgrind from the Valgrind Tool Suite[132] for an 81-atom model alpha helix[133] using the AMBER 12 potential and for a 100-atom LJ cluster. The calculation of the potential energy and gradient was found to take 80 to 99 % of the total time, depending on the L-BFGS history size used. Smaller values for the history size resulted in the potential calculation forming a greater percentage of the overall run time. The remainder of the compute time was found to consist mostly of calls to the vector operations of L-BFGS. For basin-hopping global optimisation, the calls to the potential were found to originate almost exclusively from within the main L-BFGS routine. For transition state location, the calls to the potential were found to originate partly from the two modified versions of L-BFGS in hybrid EF, and partly from the modified version of L-BFGS in the DNEB method. The proportions of time taken by these different routines depended greatly on the parameters used for the overall run.

3.2.1 Basin-hopping global optimisation

First, we discuss the CUDA implementation of the L-BFGS routine for the acceleration of basin-hopping global optimisation. It is not immediately obvious whether implementing L-BFGS on GPU would be beneficial. Although many of the individual vector calculations in the routine are amenable to parallellisation, the loops through the history size, m in Algorithm 2 (Section 2.3.1), are necessarily serial. The larger the history size, the larger the number of vector calculations that must therefore be performed sequentially. This scaling gives a more accurate step direction, so fewer steps (and hence fewer costly potential calls) would be required for convergence. Although simpler potentials only require a small history size, a large history size is usually optimal for the AMBER potential on CPU. As mentioned in Section 3.1, the maximum speedup for the update operations is not large, but having these operations on GPU may be worthwhile to avoid costly memory copying between host and device.

We chose to port the whole L-BFGS algorithm to GPU so that we could compare it to having just the potential energy function on GPU. Our implementation is a modified version of 'CudaLBFGS'[122,134] (Creative Commons Attribution 3.0 Unported License), an existing GPU-accelerated version of L-BFGS, authored by Wetzl and Taubmann. They programmed all the vector operations of Algorithms 1 and 2 (Section 2.3.1) on GPU using the NVIDIA cuBLAS library, which delivers 6 to 17 times faster performance than the latest Intel Math Kernel Library (MKL) implementation of the BLAS routines.[135] Single-threaded kernels are used to perform operations in device memory not involving vectors or matrices. Various modifications were made to the code to enable it to run correctly, including the correction of some deprecated CUDA syntax and removal of memory leaks. We also implemented the step scaling procedure described in Algorithm 3 (Section 2.3.1) using cuBLAS, and made various other modifications to make the L-BFGS routine as similar as possible to its CPU equivalent in GMIN. The most serious problem with the minimiser occurred when the starting structure used was near to a minimum already, which meant that the change in coordinates, **S**, and the change in gradient, **Y**, were very small. Minimisations would not converge due to components $\mathbf{S}_{k-1}^T\mathbf{Y}_{k-1}$ and $\mathbf{Y}_{k-1}^T\mathbf{Y}_{k-1}$ becoming too small for the calculation of the Hessian approximation of Equation (2.11) to give sensible values. This problem was fixed by resetting these values to $\pm 10^{-20}$ if they fell inside this range.

3.2.2 Local rigid body framework

If the entirety of the L-BFGS algorithm is to be on the GPU, the coordinate transformation and gradient projection from our local rigid body framework should also take place there to avoid unnecessary memory copying. The rotation matrix of Eq. (2.23) (Section 2.5) is calculated independently for each rigid body, presenting an obvious opportunity for parallelisation. A kernel to calculate the rotation matrices was implemented, launching a number of threads equal to the number of rigid bodies, each of which also constructed the intermediate skew-symmetric matrix of Eq. (2.24) (Section 2.5) in registers. The calculations involving loops through small, nine-element matrices were unrolled for performance. The resulting rotation matrices, stored in global memory, were passed as an argument to the subsequent transformation kernel. This kernel encoded the main transformation of Eq. (2.25) (Section 2.5) and utilised a greater degree of parallellisation than the previous kernel, with a number of active threads equal to the number of atoms. Single atoms not in rigid bodies were also copied using a one thread per atom kernel design.

For the sum of Eq. (2.26) (Section 2.5), providing the translational forces on the rigid bodies, an approach based on the ‘Shuffle On Down’ algorithm was used to perform a parallel reduction.[54] This procedure makes use of the shuffle instruction, which allows threads to read the registers of other threads in the same warp. There are four shuffle intrinsics, but only the shuffle down instruction is used in this case, specifically the double precision implementation detailed on the NVIDIA Developer blog.[54] The final result of the shuffle down instruction is that values held in registers for each thread are shifted to lower thread indices. Reduction within a warp can be achieved by summing pairs of values obtained through a shuffle down operation, where the shift is initially half the warp size, and then repeatedly bisected until thread 0 holds the final value, i.e. constructing a reduction tree. A schematic representation of this process can be seen in Figure 3.1. Parallel reduction using shuffle instructions in this way is much faster than using shared memory to exchange data between threads. This function was implemented for summations involving rigid bodies of 32 atoms or fewer. Using a kernel launch with a number of threads equal to 32 times the number of rigid bodies, the register values to be summed are either the appropriate value read from global memory, or zero where the atom does not exist. The first thread of each warp was designated to write the final reduced values to global memory.

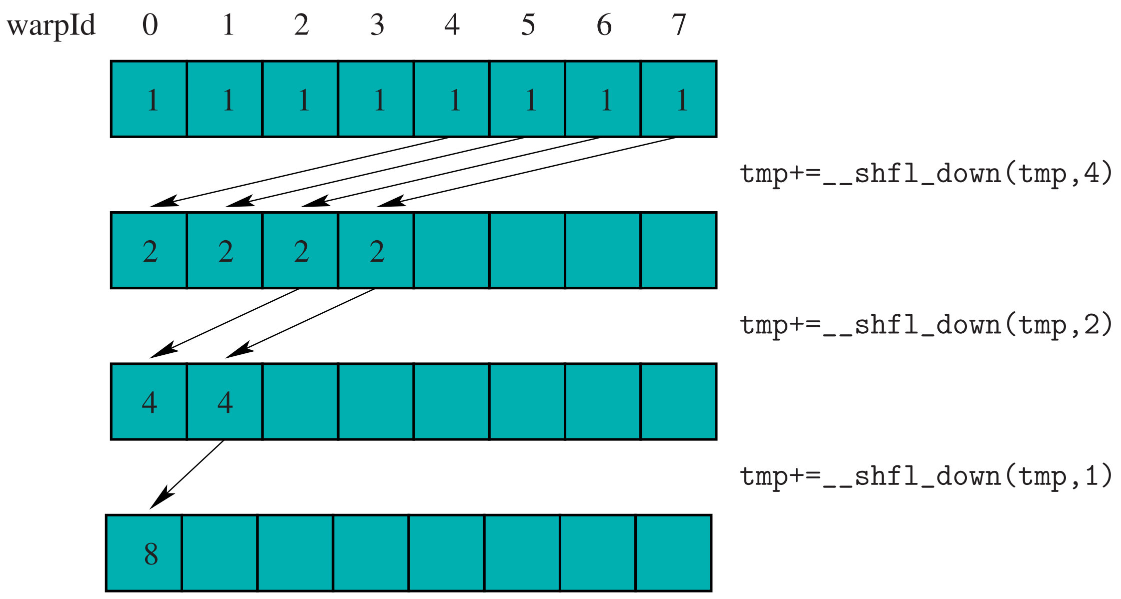


Figure 3.1 A reduction algorithm using shuffle down. It should be noted that the full warp size is 32 and that all threads of the warp shift values simultaneously, although arrows are only included here for threads contributing to the final result. Figure redrawn from reference 54.

The most frequently employed local rigid groupings do not involve more than 32 atoms and so the above approach is sufficient in these cases. However, sometimes larger rigid bodies are needed, so the reduction must be extended to summing more than 32 values. To reduce across a whole block of threads, reductions must first be performed within each warp. The first thread of each warp can then write its partial sum to an array in shared memory. After thread synchronisation, the first warp can read these values from shared memory and perform another warp reduction to give the final value.

To perform a reduction across the whole grid, more than one kernel launch is needed for global communication. A kernel was designed such that the number of values to be reduced for each rigid body was equal to the number of atoms in the largest rigid body in the system, rounded up to the nearest multiple of the block size, so that no block contained a mixture of values from different rigid bodies. The kernel launched a number of threads equal to this number multiplied by the number of rigid bodies. A block reduction was performed for all blocks, with the results written to a temporary array in global memory. Another similar kernel was then used to reduce this array in sections (one section per rigid body). This kernel was configured to launch repeatedly until the number of blocks became equal to the number of large rigid bodies, at which point the reduced value for each block was written to the global memory gradient array.

In a similar manner to the coordinate transformation, a kernel using one thread per rigid body was used to calculate the derivatives of the rotation matrices following Eq. (2.29) (Section 2.5). The nine-element loops were unrolled as before and the final values were written to global memory. Due to higher register use, optimal performance required a smaller block size than that for the coordinate transformation kernel.

The derivatives of the rotation matrices were passed in global memory to another kernel, which performed the matrix multiplication and dot product of Eq. (2.27) (Section 2.5) in parallel for each atom. This result was again saved in global memory and passed to another pair of parallel reduction kernels, similar to the previous implementation of the 'Shuffle On Down' algorithm, to obtain the rotational forces on the rigid bodies. Single atoms not in rigid bodies were copied via a kernel using one thread per atom.

In our tests, we used the root mean square (RMS) force formulation described in Section 2.5.1, based on a distance measure for angle-axis coordinates that is invariant to global translation and rotation. This approach gives a more accurate result than the standard procedure, and fewer minimisation steps are required for convergence. This method was adapted for GPUs using techniques similar to those already discussed, such as incorporating parallelism equivalent to either the number of atoms or the number of rigid bodies, unrolling fixed-size loops, and performing reductions using the 'Shuffle On Down' algorithm.

3.2.3 Hybrid eigenvector-following

As the AMBER developers have not implemented second derivatives on GPU for their potential, a variational approach must be used to calculate the smallest nonzero eigenvalue and associated eigenvector. L-BFGS has proved faster than conjugate gradient algorithms for minimisation of the Rayleigh-Ritz ratio and also for the subspace minimisation,[136] meaning that much of the CUDA code written for basin-hopping global optimisation can be reused. The approximate eigenvalue and its derivative, $\lambda(\mathbf{x})$ and $\partial\lambda/\partial\mathbf{x}$ of Eq. (2.19) and Eq. (2.20) (Section 2.4.2), were calculated using the cuBLAS library. Orthogonalisation and normalisation of the uphill eigenvector required a mixture of cuBLAS operations and custom kernels with vector length parallelisation for more complex vector operations. The determination of the length of the EF step took place on CPU, with the actual step itself performed using a cuBLAS call. The projection of gradient components in the subspace minimisation, defined in Eq. (2.22) (Section 2.4.2), also used cuBLAS.

3.2.4 Doubly-nudged elastic band method

As the putative transition states located by the DNEB method are only intended to be approximate, they are not converged to high accuracy, since this task is performed using hybrid EF. A small history size of four is usually used for the L-BFGS minimisation, as there is little benefit to using more accurate steps in this situation. Having seen initial results for the other methods, it was not thought likely that implementing the whole algorithm on GPU for the DNEB method would bring much benefit with the AMBER potential. Calls to the GPU-accelerated AMBER potential from the CPU algorithm were inserted to provide acceleration.

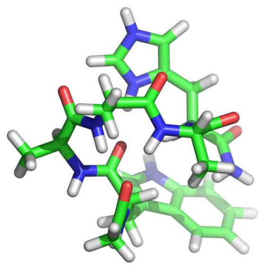
3.2.5 Potential calculation

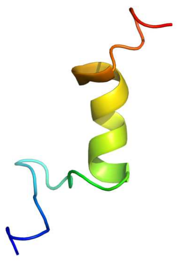
The calculation of the potential energy and gradient is the most time-consuming part of all of these energy landscape methods. For testing purposes during the development of the CUDA implementation of L-BFGS, the LJ potential was implemented using CUDA. The kernel was designed to launch $N \times N$ threads, where N is the number of atoms. Each thread handled the interaction of one atom with one other atom, with positions stored in registers, and calculated the contribution to the energy and gradient for that atom. The ‘Shuffle On Down’ reduction was used to perform the summations of the appropriate partial results to obtain the total energy and final $3N$-dimensional gradient. Although a respectable speedup was obtained, this potential was initially intended only as a test system, so there is possibly scope for further optimisation with regard to efficient global memory access and register usage. Detailed comparison to CPU results has not been performed for this potential, as our main aim was to achieve acceleration for biomolecular systems, so it was judged a better use of development time and computational resources to focus on the GPU-accelerated AMBER potential.

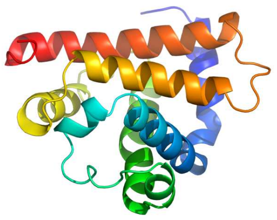
Much of the previous work employing basin-hopping to investigate biomolecular systems has used the CPU implementation of the AMBER potential interfaced with our GMIN code.[89] The interface with the CPU code of AMBER 12, implemented by Kyle Sutherland-Cash, is henceforth referred to as ‘A12GMIN’. To allow the use of the existing AMBER 12 setup routines and the existing basin-hopping procedure, the GPU implementation of L-BFGS was interfaced with GMIN. The ISO_C_BINDING module was used to ensure the safe interaction of the C/C++ and Fortran code. It was programmed such that the starting coordinates and other relevant parameters were copied from host to device at the start of a calculation, and the energy, gradient and other useful output values were copied from device to host at the end. This newly interfaced code will be referred to from this point onwards as 'CUDAGMIN'. The GB implicit solvent GPU potential from AMBER 12[63] was then interfaced with CUDAGMIN. The DPDP precision model was used, in which contributions to forces and their accumulation are both performed in double precision. A few functions were written to copy the coordinates from the device memory being used by L-BFGS to the expected structures in the AMBER code and to then copy back the energies and gradients for further use in L-BFGS. The AMBER energy and gradient function was modified to prevent automatic copying of the energies back to the CPU and a double precision reduction kernel was implemented to obtain a total energy on the GPU. Another ISO_C_BINDING interface was implemented to allow the GPU-accelerated potential to be called directly from CPU for purposes of comparison and debugging. In addition, the GPU-accelerated L-BFGS routines have been interfaced with OPTIM, in a similar manner to the interface with GMIN, to form 'CUDAOPTIM'. The GPU-accelerated AMBER potential has also been interfaced with CUDAOPTIM. The existing AMBER 12 CPU interface for OPTIM, implemented by Kyle Sutherland-Cash, is referred to as 'A12OPTIM'.

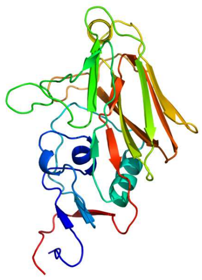
3.3 Results and discussion

Here we present results for CUDAGMIN and CUDAOPTIM with the AMBER potential. Eight different systems of varying sizes (shown in Figure 3.2) were used to test these methods: the pentapeptide Ac-WAAAH$^+$-NH$_2$ (W$_1$H$_5$)[133] (81 atoms), the p53 upregulated modulator of apoptosis (PUMA) protein[137,138] (581 atoms), the myoglobin structure from the AMBER GPU Benchmark Suite[139] (2492 atoms), human aldose reductase with its nicotinamide adenine dinucleotide phosphate (NADP$^+$) cofactor[140] (5113 atoms), an epoxide hydrolase from *Acinetobacter nosocomialis*[141] (10 053 atoms), the trimeric haemagglutinin (HA) glycoprotein of the influenza A(H1N1) virus[142] (22 811 atoms), a monomeric version of HA (7585 atoms), and finally a truncated version of this monomer (3522 atoms). HA has previously been used to investigate receptor binding.[143,144] Due to its large size, a truncated monomer structure was employed to make the simulations computationally feasible on CPU. All systems used the AMBER ff99SB force field with an effectively infinite nonbonded cutoff (999.99 Å). Although a finite cutoff would be more usual for CPU calculations, this option is not supported on GPU, so an infinite cutoff was used in both cases for a proper comparison. The modified GB solvent model[79,80] (AMBER input flag `igb` $= 2$) was used at a salt concentration $0.2\,M$ with a cutoff of 12 Å for the calculation of the effective Born radii. CUDAGMIN and CUDAOPTIM were compiled on the x86_64 architecture running the Ubuntu 14.04.4 operating system, using Intel compiler version 14.0.4 and NVIDIA CUDA Toolkit 6.5.[145] GeForce GTX TITAN Black GPUs with the NVIDIA Linux driver version 346.59 and 2.10 GHz Intel Xeon E5-2620 v2 CPUs were employed. A12GMIN and A12OPTIM were compiled on the x86_64 architecture running the Scientific Linux release 6.2 operating system, using Intel compiler version 14.0.4, and run on 2.67 GHz Intel Xeon X5650 CPUs.

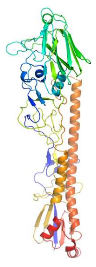

(a) W_1H_5 (81 atoms)

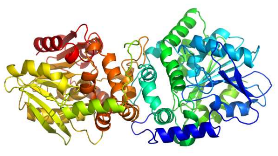

(b) PUMA (581 atoms)


(c) Myoglobin (2492 atoms)


(d) HA truncated monomer (3522 atoms)

(e) Aldose reductase (5113 atoms)


(f) HA monomer (7585 atoms)


(g) Epoxide hydrolase (10 053 atoms)

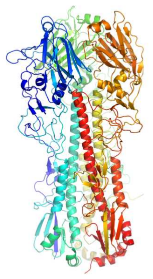

(h) HA trimer (22 811 atoms)

Figure 3.2 The structures of the biomolecules used in the present tests.

3.3.1 Basin-hopping global optimisation

Basin-hopping global optimisation can be viewed in terms of sequential L-BFGS minimisations. Profiling of the CPU code shows that routines other than L-BFGS account for a negligible amount of the overall run time, so it is reasonable to benchmark L-BFGS in isolation so that test minimisations can run concurrently. The calculation of the energy and gradient accounts for the majority of the time spent in L-BFGS, with the percentage of time taken up by linear algebra operations increasing with the size of the history of updates, *m*. As the history size increases, a larger number of vector operations are performed in the sequential loop through the history and fewer L-BFGS steps (and hence fewer potential calls) are needed for convergence, as the directions of the steps taken are more accurate. It is not immediately clear whether a larger history size would result in reduced time for convergence on GPU hardware, although this is the case for the AMBER potential on CPU. Results are presented here for both a small history size of four and a relatively large history size of 1000. For each of the eight systems, 100 configurations were extracted from high temperature MD runs and used as starting structures for minimisation. Table 3.1 shows the average L-BFGS minimisation times for these sets of structures for a history size of four. Timings are shown for three different implementations: both the L-BFGS vector operations and the potential function on GPU, L-BFGS on CPU with the potential on GPU, and both L-BFGS and the potential on CPU. The equivalent results for a history size of 1000 are shown in Table 3.2. All minimisations used a maximum L-BFGS step size of 0.4 Å, with the maximum rise in energy allowed per step capped at $10^{-4}\,\mathrm{kcal\,mol^{-1}}$, and a convergence condition for the RMS force of $10^{-5}\,\mathrm{kcal\,mol^{-1}\,\mathring{A}^{-1}}$. These are parameters that have been successfully used in basin-hopping global optimisation previously for similar systems, although the RMS force convergence criterion is rather stricter than is usual.

Table 3.1 L-BFGS benchmarking ($m = 4$) using two GPU implementations and a CPU implementation[a]

System	**Number of atoms**	**Average minimisation time for GPU Implementation 1 ($m = 4$) / s**	**Average minimisation time for GPU Implementation 2 ($m = 4$) / s**	**Average minimisation time for CPU Implementation ($m = 4$) / s**	**Time for CPU Implementation / GPU Implementation 1 ($m = 4$)**	**Time for CPU Implementation / GPU Implementation 2 ($m = 4$)**
A	81	2.6	1.2	1.9	0.7	1.6
B	581	5.9	3.4	131.6	22.3	39.2
C	2492	32.8	29.0	4192.1	127.8	144.4
D	3522	40.1	39.4	5937.7	148.1	150.8
E	5113	69.2	71.7	11768.0	169.9	164.2
F	7585	489.9	492.1	86552.8	176.7	175.9
G	10053	1333.4	1374.4	248394.9	186.3	180.7
H	22811	2546.9	2527.1	517567.7	203.2	204.8

[a]GPU Implementation 1 has the entire L-BFGS routine on GPU, including the calculation of the potential energy and gradient. GPU Implementation 2 has just the potential calculation on GPU. Averages are taken from the minimisation of 100 different starting structures from high temperature MD trajectories. The systems used are are labelled as follows: 'A' is W_1H_5, 'B' is PUMA, 'C' is myoglobin, 'D' is the truncated monomer of HA, 'E' is aldose reductase and NADP$^+$, 'F' is the full monomer of HA, 'G' is epoxide hydrolase and 'H' is the full trimeric HA structure. The ff99SB force field was used for all systems with the modified GB solvent model[79,80] (AMBER input flag `igb` = 2).

Table 3.2 L-BFGS benchmarking ($m = 1000$) using two GPU implementations and a CPU implementation[b]

System	**Number of atoms**	**Average minimisation time for GPU Implementation 1** ($m = 1000$) **/ s**	**Average minimisation time for GPU Implementation 2** ($m = 1000$) **/ s**	**Average minimisation time for CPU Implementation** ($m = 1000$) **/ s**	**Time for CPU Implementation / GPU Implementation 1** ($m = 1000$)	**Time for CPU Implementation / GPU Implementation 2** ($m = 1000$)
A	81	11.3	0.4	0.6	0.1	1.4
B	581	144.5	14.1	92.9	0.6	6.6
C	2492	475.9	201.9	3091.1	6.5	15.3
D	3522	382.1	239.6	4561.2	11.9	19.0
E	5113	421.8	364.4	9123.3	21.6	25.0
F	7585	1249.5	1609.0	48754.5	39.0	30.3
G	10053	2478.7	4217.0	142404.4	57.5	33.8
H	22811	2392.5	4747.1	337077.0	140.9	71.0

[b]Implementation and parameters are identical to those in Table 3.1.

Comparing the times shown in Tables 3.1 and 3.2 for all three implementations, the CPU-only minimisations are all faster for the larger history size of 1000. However, the calculations where the GPU is used are mostly faster for the smaller history size of four. These CPU results can be explained by the fact that fewer expensive potential calls need to be made for a larger history size. This result is also true for the GPU, but the large acceleration of the potential calculation means that it is fastest to employ a smaller history size that uses more potential calls and fewer vector operations (which are relatively slower and must be performed sequentially), even if this requires a greater number of steps. Comparing times for the two GPU implementations within the same history size, very little difference can be seen in average times for history size $m = 4$, as vector operations are only a tiny percentage of the overall calculation. However, for history size 1000, where more vector operations are performed, we see that L-BFGS with both the potential and vector calculations on the GPU is slower than the implementation with just the potential calculation on the GPU for small systems, but faster for larger systems. These results occur because BLAS calculations, such as dot products, are actually slower on the GPU relative to CPU for small vectors. Speed improvements start to be seen for cuBLAS in the three largest systems. Profiling runs show that memory copying between host and device does not become a significant bottleneck for either GPU implementation, even at these large history sizes, presumably because the potential calculation is sufficiently time-consuming in comparison. Overall, excellent speedups are obtained for a range of history sizes. The only system for which a significant improvement is not seen is W_1H_5, which is too small for the GPU arithmetic hardware to be fully utilised.[63]

It is sensible to optimise the history size to find the fastest balance between the number of potential calls and cuBLAS calls, particularly for larger systems where the speedup for cuBLAS operations becomes more significant. Referring to Tables 3.1 and 3.2, the 22 811-atom full trimeric HA system actually minimises faster with the larger history size of 1000, than for the smaller history size of four. Results are collected in Table 3.3, showing average minimisation times for both GPU implementations for a range of history sizes. Timings for the whole L-BFGS algorithm are faster than those with just the potential on the GPU for all history sizes, except the very smallest. The fastest average minimisation time is for the all-GPU implementation with a history size of 75.

Table 3.3 Variation of HA minimisation time with history size using two GPU implementations[c]

History size (m)	**Average HA minimisation time for GPU Implementation 1 / s**	**Average HA minimisation time for GPU Implementation 2 / s**
4	2546.9	2527.1
10	2035.3	2047.2
50	1817.9	1917.8
75	1788.3	1951.3
100	1836.3	1999.6
250	1880.8	2253.0
500	2010.1	3229.4
750	2223.8	4067.4
1000	2392.5	4747.1

[c]GPU Implementations 1 and 2 are described in Table 3.1. HA refers to the full trimeric structure (‘H’ in Tables 3.1 and 3.2).

3.3.2 Local rigid body framework

The local rigid body framework[10] is available on GPU for both CUDAGMIN and CUDAOPTIM. Benchmarking was performed for just the L-BFGS algorithm, as for basin-hopping global optimisation. Locally rigid systems were compared to their atomistic equivalents using the implementation of L-BFGS where the whole algorithm is on the GPU. A comparison to CPU timings was not performed, because the aim was not to show an acceleration for the rigid body calculations on GPU, but to demonstrate that they do not add much extra overhead at each step and to analyse their use relative to atomistic GPU calculations. However, the average number of L-BFGS steps required was recorded, which will be very similar to the number of steps required on CPU. The same parameters were used for L-BFGS as for the atomistic benchmarking and two systems with different patterns of rigidification were considered. The results in Table 3.4 are for a system containing a large number of small local rigid bodies, namely the HA monomer, which was used in the previous benchmarking set, with aromatic rings, peptide bonds and sp^2 centres rigidified (a total of 671 rigid bodies, none exceeding a size of 11 atoms). The results in Table 3.5 are for a system with one large rigid body and a few small rigid bodies, as might be used in a factorised superposition approach[140] (FSA) calculation of binding free energy (the extensive rigidification substantially reduces the number of minima on the energy landscape). The system in question is aldose reductase with aromatic rings, peptide bonds and sp^2 centres rigidified between 16 Å and 17 Å distant from the binding site of the phenylacetic acid (PAC) ligand (a total of 22 bodies, none exceeding a size of 11 atoms), and everything beyond 17 Å grouped as one large rigid body of size 3678 atoms. Starting structures for the test minimisations were taken from a basin-hopping run for the locally rigidified structure, with random perturbations at each step consisting of side chain group rotations and backbone Cartesian moves. The first 100 starting structures to successfully converge formed the benchmarking set. The average minimisation times, the average number of L-BFGS steps taken, and the time per step averaged over the whole set are shown in Tables 3.4 and 3.5 for a range of history sizes for both rigid and atomistic systems. Different sets of history sizes are used for the two cases, which were chosen to best illustrate the variations observed.

Table 3.4 GPU minimisation of the HA monomer for locally rigid and atomistic representations[d]

History size (m)	**Average minimisation time (GPU Implementation 1) / s**		**Average number of L-BFGS steps**		**Average time per step / s**	
	Rigid	Atomistic	Rigid	Atomistic	Rigid	Atomistic
4	360.6	296.2	19639	16514	0.018	0.018
10	281.6	236.0	15102	13006	0.019	0.018
25	249.2	219.4	12799	11563	0.019	0.019
50	252.4	230.4	12013	11296	0.021	0.020
75	256.9	245.6	11279	11249	0.023	0.022
100	265.5	265.9	10837	11238	0.024	0.024
250	348.8	385.3	9885	11023	0.035	0.035
500	496.9	581.3	9450	10979	0.053	0.053
750	635.7	719.9	9153	10757	0.069	0.067
1000	757.4	890.1	8858	10561	0.085	0.084

[d]GPU Implementation 1 has the entire L-BFGS routine on GPU, including the calculation of the potential energy and gradient. Averages were taken for the minimisation of 100 different starting structures obtained from a basin-hopping run. The atomistic HA monomer is labelled ‘F’ in Tables 3.1 and 3.2. The rigid representation has aromatic rings, peptide bonds and sp^2 centres grouped as local rigid bodies.

Table 3.5 GPU minimisation of aldose reductase for locally rigid and atomistic representations[e]

History size (m)	**Average minimisation time (GPU Implementation 1) / s**		**Average number of L-BFGS steps**		**Average time per step / s**	
	Rigid	Atomistic	Rigid	Atomistic	Rigid	Atomistic
4	335.8	49.2	32431	5224	0.010	0.009
250	110.9	102.9	4155	3999	0.027	0.026
500	92.8	162.1	2200	3890	0.042	0.041
750	87.9	215.5	1739	3827	0.050	0.056
1000	94.7	265.7	1571	3798	0.059	0.069
1500	82.7	347.5	1391	3713	0.057	0.092
2000	78.6	414.6	1350	3691	0.057	0.109

[e]GPU Implementation 1 has the entire L-BFGS routine on GPU, including the calculation of the potential energy and gradient. Averages were taken for the minimisation of 100 different starting structures obtained from a basin-hopping run. Aldose reductase is is complex with with the PAC ligand, and the rigidified version is considered for an FSA simulation with an atomistic layer of radius 16 Å, a local rigid layer of 1 Å with rigid aromatic rings, peptide bonds and sp^2 centres, and the rest of the protein (3678 atoms) rigidifed as a single large rigid body.[140]

The average time per step for both systems shows that the coordinate and gradient transformations add very little overhead to each step, as they are well optimised for the GPU. As for basin-hopping global optimisation, a large history size gives optimal minimisation time on CPU. It was found in the original work for rigid body systems on CPU, that minimisation takes less time than for atomistic representations, as fewer steps are taken due to the reduction in the number of degrees of freedom.[10] Looking at the average number of L-BFGS steps taken for rigid and atomistic systems in Tables 3.4 and 3.5, we see that this result holds only for larger history sizes. At small history sizes, rigid body systems actually take more steps than their atomistic equivalents, an effect that is particularly pronounced for the system with FSA rigidification in Table 3.5. This result implies that rigid body systems require a more accurate step direction in minimisation. Rigid body timings only become lower than atomistic timings for history sizes where fewer steps are taken. On GPU, due to the competing effects of history size, it follows that the optimal history size for fast minimisation may be larger for locally rigid systems than for atomistic. This result is certainly true for the highly rigidified system in Table 3.5, but the average number of L-BFGS steps does not decrease as steeply with increasing history size for the system with minimal rigidification, so the optimal history size is the same as for atomistic. We note that for both systems the fastest minimisation is still for the atomistic system with a fairly small history size. It is likely that individual L-BFGS minimisations will only be faster for locally rigid systems on GPU when the system is sufficiently large and complex that a large history size is required for the atomistic representation. However, excellent speedups compared to CPU are indeed obtained for locally rigid systems, and the advantage of the significant reduction in minima on the potential energy surface is still present.

3.3.3 Hybrid eigenvector-following

Hybrid EF[12] and DNEB[13] optimisations account for the majority of the time spent in locating transition states with the OPTIM program. As explained in Section 2.4.2, gradient-only hybrid EF consists of two different modified versions of L-BFGS with an uphill EF step in between. This overall process of finding a transition state starting from coordinates proposed by the DNEB method was analysed as a whole, rather than for the separate components, as the Rayleigh-Ritz minimisation and the orthogonal subspace minimisation (with uphill gradient components projected out) are never performed separately in real applications. The implementation of L-BFGS with the entire routine on GPU was used in these tests. The same systems were used as for basin-hopping global optimisation (shown in Figure 3.2), excluding the smallest, where too few unique transition states were found. Short basin-hopping runs were performed to generate a distinct second endpoint for each system. Connection attempts using OPTIM were then performed to try and locate some transition states between the two endpoints. The first 100 starting coordinates generated by the DNEB procedure, which went on to successfully converge to transition states, were saved as the reference coordinates.

The approximation of Eq. (2.21) (Section 2.4.2) was used to calculate the eigenvector in all cases. A maximum L-BFGS step size of 0.4 Å with a maximum allowed rise in energy of $10^{-4}\,\mathrm{kcal\,mol^{-1}}$ was used in both versions of L-BFGS. The maximum number of iterations in the Rayleigh-Ritz ratio minimisation for calculating the smallest nonzero Hessian eigenvalue[12] was set to 1000 and a convergence criterion of $0.01\,\mathrm{kcal\,mol^{-1}\,Å^{-1}}$ was adopted for the RMS gradient, with the additional constraint that the percentage change of the eigenvalue must have fallen below 1 % for the final two steps. The subspace minimisation was limited to a maximum of 20 iterations before the eigenvalue converged, as judged by modulus overlap with the previous vector better than 0.9999, and then increased to a maximum of 200 iterations when this condition was achieved. A trust radius of 0.5 was used for adjusting the size of the maximum uphill step along the eigenvector,[146] where the trust ratio is defined as $|1 - \lambda(\mathbf{x})_{\text{predicted}}/\lambda(\mathbf{x})_{\text{actual}}|$, and the maximum step is increased or decreased by 10 % depending on whether the trust ratio is less than or greater than the trust radius, respectively. The predicted value of the eigenvalue, $\lambda(\mathbf{x})_{\text{predicted}}$, is calculated through the finite difference of the present and previous components of the gradient in the given eigendirection. The maximum step size was constrained between the bounds of 0.01 Å and 0.5 Å and an overall maximum of 1000 iterations of hybrid EF was allowed. Transition state convergence was deemed to have occurred when the RMS force fell below $10^{-5}\,\mathrm{kcal\,mol^{-1}\,Å^{-1}}$ and the magnitude of the hybrid EF step fell below 0.02 Å. Again, these parameters were chosen as they had been used successfully for similar applications using the AMBER potential, although the RMS force convergence criterion is stricter than would usually be applied. The same initial guess for the eigenvector was used for all 100 starting DNEB structures, and the time taken to converge to a transition state was recorded for both CPU and GPU tests.

Table 3.6 shows the average time taken for hybrid EF with both GPU and CPU, and the corresponding speedups using a small history size of four. The equivalent results for large history sizes of 1000 are shown in Table 3.7. Again, the GPU implementation is fastest with a history size of four and the CPU implementation is fastest with a history size of 1000. Excellent speedups are obtained on GPU hardware, though slightly below those found for basin-hopping global optimisation, due to the greater number of dot products and other vector operations performed in projecting out components of the gradient in the subspace minimisation.

Table 3.6 Hybrid EF benchmarking ($m = 4$) for GPU and CPU[f]

System	**Number of atoms**	**Average time for GPU Implementation** ($m = 4$) **/ s**	**Average time for CPU Implementation** ($m = 4$) **/ s**	**Time for CPU Implementation / GPU Implementation** ($m = 4$)
B	581	14.8	192.1	13.0
C	2492	20.2	2102.4	104.2
D	3522	25.7	3529.1	137.4
E	5113	51.4	7399.4	143.8
F	7585	197.1	36779.8	186.6
G	10053	195.9	33549.2	171.3
H	22811	1047.6	176550.7	168.5

[f]The GPU implementation has the entire Rayleigh-Ritz L-BFGS and L-BFGS routines with gradient projection on GPU, including the calculation of the potential energy and gradient. 100 DNEB structures that successfully converged to transition states during a connection attempt between two random minima were used as starting points for the searches. The systems used are labelled as follows: ‘B’ is PUMA, ‘C’ is myoglobin, ‘D’ is the truncated monomer of HA, ‘E’ is aldose reductase and NADP$^+$, ‘F’ is the full monomer of HA, ‘G’ is epoxide hydrolase and ‘H’ is the full trimeric HA structure. The ff99SB force field was used for all systems with the modified GB solvent model[79,80] (AMBER input flag `igb` = 2).

Table 3.7 Hybrid EF benchmarking ($m = 1000$) for GPU and CPU[g]

System	**Number of atoms**	**Average time for GPU Implementation** ($m = 1000$) **/ s**	**Average time for CPU Implementation** ($m = 1000$) **/ s**	**Time for CPU Implementation / GPU Implementation** ($m = 1000$)
B	581	113.3	142.7	1.3
C	2492	215.1	1650.3	7.7
D	3522	186.3	3160.9	17.0
E	5113	238.3	6576.0	27.6
F	7585	669.6	32046.2	47.9
G	10053	449.9	30073.3	66.8
H	22811	1225.4	169234.0	138.1

[g]Implementation and parameters are identical to those in Table 3.6.

3.3.4 Doubly-nudged elastic band method

The DNEB method generally uses a fixed maximum number of steps, rather than continuing to convergence, as we only aim to find approximate guesses for transition states. Therefore, the time taken to apply the method to a particular system with a fixed set of parameters varies little between different pairs of starting minima. This consistency makes the analysis of the DNEB method straightforward, as it is only necessary to test a small number of pairs for each system. In these tests, 10 images were employed in the DNEB and the maximum number of DNEB iterations was set to 300. A small history size of $m = 4$ was employed. The convergence tolerance for the RMS DNEB force on all the images was set to 0.01 $\mathrm{kcal\,mol^{-1}\,\AA^{-1}}$.

The speedups are shown in Table 3.8 along with the average times taken for both CPU and GPU (only the potential is implemented on GPU). These times are an average of those for the first three individual DNEB runs performed in the connection attempts used for generating the DNEB structures for benchmarking hybrid EF. Again, speedups of more than two orders of magnitude have been obtained with the AMBER potential, providing a significant acceleration for the complete process of transition state location.

Table 3.8 DNEB benchmarking ($m = 4$) for GPU and CPU[h]

System	Number of atoms	Average time for GPU Implementation ($m = 4$) / s	Average time for CPU Implementation ($m = 4$) / s	Time for CPU Implementation / GPU Implementation ($m = 4$)
B	581	2.8	90.1	32.5
C	2492	10.5	1203.8	114.2
D	3522	18.6	2275.3	122.2
E	5113	30.9	4593.6	148.5
F	7585	59.5	9569.0	160.8
G	10053	100.1	16535.6	165.2
H	22811	452.9	80680.2	178.1

[h]The GPU implementation has just the potential calculation on GPU. The average of three DNEB runs of 300 iterations each was taken from a connection attempt between two random minima. The systems used are the same as those in Tables 3.6 and 3.7.

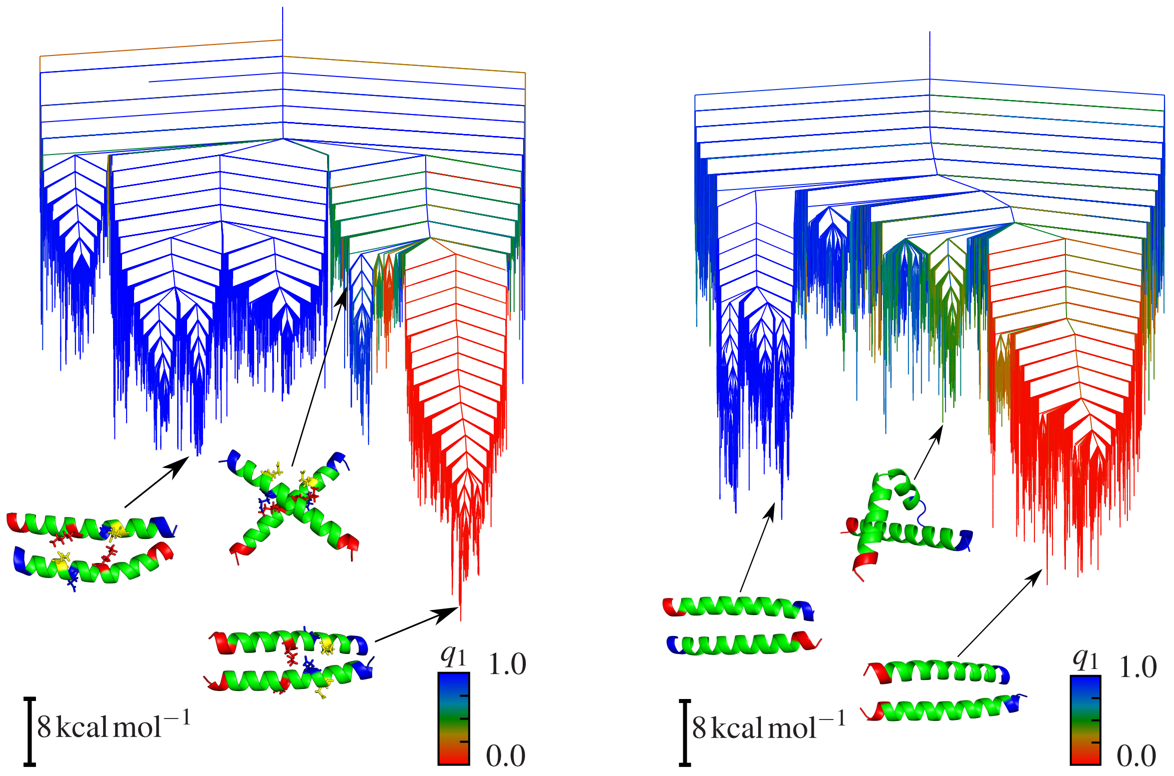
3.3.5 Coiled-coil peptide

The results described in this chapter have been successfully applied to investigating the effects of point mutations on the energy landscape of a coiled-coil peptide,[129,130] through a collaboration with Konstantin Röder. The coiled-coil system is known as GCN4-pLI, and is derived from the leucine zipper of the yeast transcription factor, GCN4.[147] In the native state, two identical helical dimers form a parallel tetrameric configuration. However, competing parallel and antiparallel structures are seen experimentally for a single mutation, E20S,[148] suggesting the existence of a multifunnel energy landscape. The sequence of the monomer of the parent structure is MKQIED-KLEEILS-KLYHI$\underline{\text{E}}$N-ELARIKK-LLG.

The GPU-accelerated versions of the DNEB method, hybrid EF and L-BFGS interfaced with the AMBER potential were used to explore the potential energy landscapes of the dimer and the E20S mutant. A parallel and an antiparallel configuration were used as the starting structures for each system, and kinetic transition networks were constructed. The procedure described in Section 2.4 was used to construct initial connected paths, and these were further refined to remove artificially high barriers and kinetic traps using the SHORTCUT[149,150] and UNTRAP[150] schemes. Two new techniques were used in this work to improve the efficiency of finding an initial path for large systems. After each Dijkstra analysis, only minima and transition states on the current best path were included in the database, to prevent the expense of the required distance calculations between minima from increasing too quickly. Also, connection attempts between minima were initially restricted to distances larger than 20 Å.

The results of the GPU-accelerated sampling of the energy landscapes were presented using disconnectivity graphs.[151,152] In this method of visualisation, branch termini represent the energies of minima on a vertical scale, and separate branches join at energies where the associated minima can interconvert via connected transition states. The horizontal axis has no physical meaning, although branches can be arranged horizontally in space to best highlight specific features. This representation allows a broad overview of the features of the energy landscape. For this application, two structural order parameters were defined for colouring the graphs. The first, q_1, is a continuous measure between zero and one, where the limiting values correspond to a parallel structure and an antiparallel structure, respectively. The second, q_2, defines the degree of kinking observed in one helix of the dimer, and ranges from zero to one, from a straight helix to the maximum degree of kinking observed.

Figure 3.3 is coloured using order parameter q_1 and shows that the energy landscapes for the parent sequence and the E20S mutant both contain parallel and antiparallel regions. The parallel configuration is the lowest energy region in both cases, but a much smaller energy difference is observed for the mutant landscape. The increased stability of the parallel regions relative to the antiparallel regions may result in part from the principle of maximum symmetry, which implies that structures with a higher symmetry content may result in deeper funnels on the energy landscape.[152] GPU-accelerated minimisation was also performed on tetrameric structures formed from the combination of pairs of dimers. Analysis of these structures revealed that the increased stability of the parallel configurations that is observed experimentally also results from an increased number of interhelical polar side chain interactions and the formation of a solvent-excluded hydrophobic core. The smaller energy difference between the parallel and antiparallel regions on the mutant landscape can be accounted for by the absence of the Lys15-Glu20 salt bridges that occur in the parent structure, which enhance the relative stability of the parent parallel configuration.



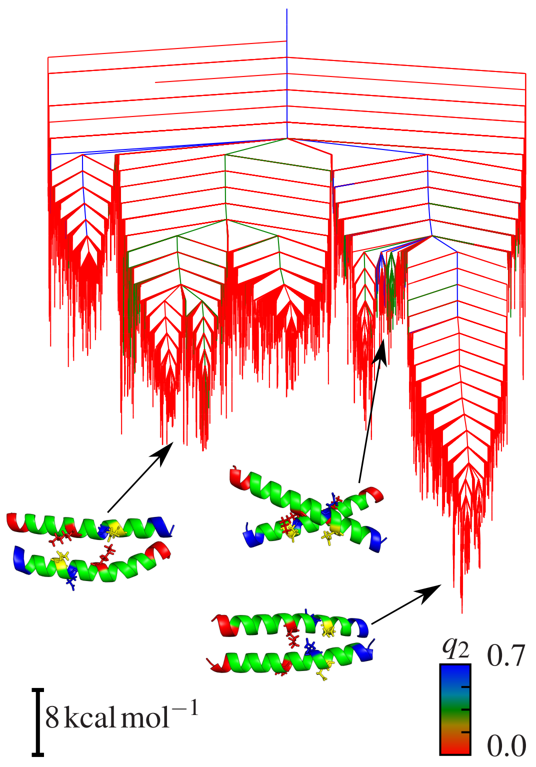
(a) Parent sequence

(b) E20S mutant

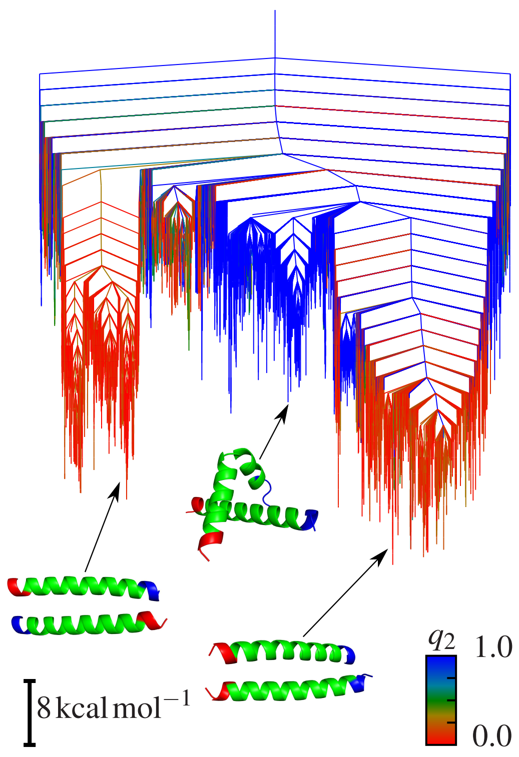
Figure 3.3 Disconnectivity graphs are shown for the energy landscapes of the parent sequence and the E20S mutant of the coiled-coil peptide. They are coloured using order parameter, q_1, which differentiates parallel structures (red) from antiparallel structures (blue). Figure adapted from references 129 and 130.

Two main subfunnels appear in the antiparallel region of the parent energy landscape. The funnel most distant from the parallel region (in terms of rearrangements via single transition states) corresponds to structures that can form a hydrophobic core in the tetrameric conformation, but have a reduced number of favourable polar side chain interactions relative to the parallel configuration. This funnel is characterised by the presence of a Lys15-Glu22 salt bridge. The subfunnel closer to the parallel region corresponds to structures that also form this salt bridge, in addition to a Lys15-Glu20 salt bridge (also observed in the parallel structures). The formation of both of these salt bridges causes a rotation of one of the helices, which prevents the formation of a hydrophobic core in tetrameric structures. This effect means that tetrameric antiparallel configurations are destabilised relative to parallel structures and therefore have a very low equilibrium occupation probability. These salt bridge interactions do not occur on the antiparallel mutant landscape. This difference results in relative stabilisation of the antiparallel configuration for mutant structures, further explaining the smaller energy difference observed between the parallel and antiparallel regions relative to the parent landscape.

Figure 3.4 is coloured using order parameter q_2 and highlights a third region of the mutant landscape corresponding to structures with a kink near to the point mutation. These structures are separated from the parallel and antiparallel regions by relatively low energy barriers and could therefore act as intermediates in structural interconversion between the two regions. This possibility accounts for the experimental observation of both of the structures in equilibrium.[148] However, no such kinked structures exist on the parent landscape, making the antiparallel region kinetically inaccessible. Although formation of the kinked structures incurs an energy cost for helix breaking, reduced solvent exposed area and increased contact between the strands encourages their formation for the mutant sequence. In tests using GPU-accelerated geometry optimisation, kinked structures were not found to be stable for the parent sequence. This result was ascribed to increased exposure of hydrophobic residues to the solvent.



(a) Parent sequence



(b) E20S mutant

Figure 3.4 Disconnectivity graphs are shown for the energy landscapes of the parent sequence and the E20S mutant of the coiled-coil peptide. They are coloured using order parameter, q_2, which differentiates kinked structures (blue) from structures that are not kinked (red). Figure adapted from references 129 and 130.

The databases of local minima and transition states obtained were also used in calculating the heat capacity and the landscape entropy for both systems. This analysis supported the observed difference between the multifunnel energy landscape of the E20S mutant and the more dominant single funnel of the parent landscape. The effects of the E20S mutation are in agreement with experimental observations,[148] so the study was extended to predictions of the behaviour of systems with E20A, E20P and E20G mutations.

In summary, this application shows that the novel GPU-accelerated energy landscape methodology presented in this chapter can be successfully used to explore complex energy landscapes, and the agreement with experiment strongly supports the validity of the results obtained. Although the system size for this application was relatively small compared to those benchmarked in this chapter, a useful acceleration was obtained for this work. Larger systems can now be accessed in future work.

3.3.6 Addressable clusters and aggregates

The GPU-accelerated methods described in this chapter have also been applied to addressable clusters, in collaboration with Prof. Wales and Dr Fejer. Addressable systems have the specific relative positions, or ‘addresses’, of their components encoded in their interactions, so that they can self-assemble into desired target structures. Doubly-addressable systems consist of addressable components that have a choice of two target morphologies. Hierarchical self-assembly behaviour is possible when interactions within the target monomer are significantly stronger than those between monomers.

An addressable form of the LJ potential incorporating a bias towards a particular local minimum was previously formulated, where the magnitude of non-nearest-neighbour interactions was reduced with reference to the target structure.[153] The latest application uses a more flexible formulation,[131] which enables us to design greater specificity into the potential energy landscape of target structures (target monomers), and tune the interactions so that aggregates of multiple target monomers retain recognisable monomer components. This formulation uses separate coefficients for the repulsive and attractive terms and can be expressed as

$$V = 4\sum_{i<j}\left[\varepsilon_{ij}^{\text{rep}}\left(\frac{\sigma}{r_{ij}}\right)^{12} - \varepsilon_{ij}^{\text{att}}\left(\frac{\sigma}{r_{ij}}\right)^{6}\right], \tag{3.1}$$

where r_{ij} is the distance between atoms i and j. Each atom is then programmed with knowledge of the local environment for a reference structure in which the distances are r_{ij}^0:

$$\varepsilon_{ij}^{\text{rep}} = \begin{cases} \varepsilon_{\text{NN}}^{\text{rep}}, & r_{ij}^0 \le r_c, \\ \varepsilon_{\text{NNN}}^{\text{rep}}, & r_{ij}^0 > r_c, \end{cases} \qquad \varepsilon_{ij}^{\text{att}} = \begin{cases} \varepsilon_{\text{NN}}^{\text{att}}, & r_{ij}^0 \le r_c, \\ \varepsilon_{\text{NNN}}^{\text{att}}, & r_{ij}^0 > r_c. \end{cases} \tag{3.2}$$

The cutoff, $r_c = 1.2$, distinguishes nearest-neighbour contacts (NN) and non-nearest-neighbours (NNN). To design a potential with multiple addresses we use the minimum value of r_{ij}^R over all the reference structures, R, in place of r_{ij}^0.

For multiple copies of a particular target monomer containing N particles, we use $\mathscr{D}(i,N) = \text{Mod}(i-1,N)+1$, where $\text{Mod}(i-1,N)$ is the particle index modulo N, ranging from 0 to $N-1$. The function $\mathscr{D}(i,N)$ therefore produces an address label in the range 1 to N, and for a single reference structure with interparticle distances r_{ij}^0 we choose

$$\varepsilon_{ij}^{\mathrm{rep}} = \begin{cases} \varepsilon_{\mathrm{NN}}^{\mathrm{rep}}, & r^0_{\mathscr{D}(i,N)\mathscr{D}(j,N)} \le r_c, \\ \varepsilon_{\mathrm{NNN}}^{\mathrm{rep}}, & r^0_{\mathscr{D}(i,N)\mathscr{D}(j,N)} > r_c, \\ \varepsilon_{\mathrm{self}}^{\mathrm{rep}}, & \mathscr{D}(i,N) = \mathscr{D}(j,N), \end{cases} \qquad \varepsilon_{ij}^{\mathrm{att}} = \begin{cases} \varepsilon_{\mathrm{NN}}^{\mathrm{att}}, & r^0_{\mathscr{D}(i,N)\mathscr{D}(j,N)} \le r_c, \\ \varepsilon_{\mathrm{NNN}}^{\mathrm{att}}, & r^0_{\mathscr{D}(i,N)\mathscr{D}(j,N)} > r_c, \\ \varepsilon_{\mathrm{self}}^{\mathrm{att}}, & \mathscr{D}(i,N) = \mathscr{D}(j,N), \end{cases} \tag{3.3}$$

where separate factors are introduced for particles that share the same address.

Multiple copies of the target cluster were considered for 13 LJ atoms, setting the reference structure to the global minimum Mackay icosahedron.[154] This addressable cluster is denoted $\mathrm{LJ}_{13}^{I_h}$, and the aggregate containing n copies of the target monomer is written $\left[\mathrm{LJ}_{13}^{I_h}\right]_n$. The repulsive term that acts between particles with the same address is needed to obtain aggregates of the addressable $\mathrm{LJ}_{13}^{I_h}$ cluster where the target monomers can be distinguished. For $\varepsilon_{\mathrm{self}}^{\mathrm{att}} = 0$ a value of $\varepsilon_{\mathrm{self}}^{\mathrm{rep}} = 10^4$ produces a clear separation of target monomers for $\mathrm{LJ}_{13}^{I_h}$. The large value arises because the repulsion needs to have a significant effect at relatively large separations.

The energy landscapes for both single and multiple copies of the target monomers were explored. Structure prediction employed basin-hopping global optimisation, starting from a set of random configurations and from candidates obtained by explicit construction. Kinetic transition networks were sampled by systematic searches for transition states and pathways connecting local minima using the DNEB approach and hybrid EF. The largest systems used the GPU-accelerated implementations of these methods with the CUDA implementation of the LJ potential, modified for addressability.

Studies of the energy landscapes for $\mathrm{LJ}_{13}^{I_h}$ for a range of values of the next-nearest-neighbour attraction confirmed that maximal bias towards the target structure is obtained for $\varepsilon_{\mathrm{NNN}}^{\mathrm{att}} = 0$. This value was also used for investigation of the aggregate structures. Since the global minima for the aggregates will be composed of correctly addressed target monomers, we can explore configuration space efficiently by moving the monomers as local rigid bodies. For the largest aggregates, the CUDA implementation of the local rigid body framework was employed.

The global minima for $\left[\mathrm{LJ}_{13}^{I_h}\right]_n$ with $2 \le n \le 13$ were characterised through basin-hopping global optimisation on CPU. Larger aggregates of 55, 147, 309, and 561 target monomers were also studied, which correspond to the sizes where complete Mackay icosahedra[154] exist. The basin-hopping runs for the aggregates of 309 and 561 monomers, containing 4017 and 7293 atoms in total, respectively, were performed using GPU-acceleration. The predicted global minima for the larger aggregate structures are shown in Figure 3.5.

In each case, global optimisation runs were initiated from structures with the target monomers placed on a suitably scaled icosahedral template. Steps were taken using a local rigid body scheme for the target monomers, with full relaxation of all atomic degrees of freedom for the lowest minima. At each size, the predicted global minimum is based on underlying icosahedral packing. To check whether the same putative global minimum was located, additional basin-hopping runs were performed from randomly positioned monomers for $\left[\mathrm{LJ}_{13}^{I_h}\right]_{55}$ and $\left[\mathrm{LJ}_{13}^{I_h}\right]_{147}$. The same lowest minimum was located for 11 out of 17 runs of 20 000 steps for $\left[\mathrm{LJ}_{13}^{I_h}\right]_{55}$, and four out of 17 runs of 15 000 steps for $\left[\mathrm{LJ}_{13}^{I_h}\right]_{147}$. The remaining runs succeeded in finding a low-lying minimum with one of the vertex $\mathrm{LJ}_{13}^{I_h}$ clusters displaced to a surface site. None of the runs initiated from randomised starting geometries produced a lower energy, providing strong evidence that the structures based on hierarchical icosahedral ordering are indeed the global minima. Only minor local reorientations were observed when the lowest minima identified with rigid target monomers were fully relaxed. The energetic ordering was unchanged on relaxation for all the aggregates.

This study further illustrates the validity of the GPU-accelerated energy landscape framework, using a different potential. The acceleration has facilitated access to system sizes beyond the reach of the standard implementations on CPU.

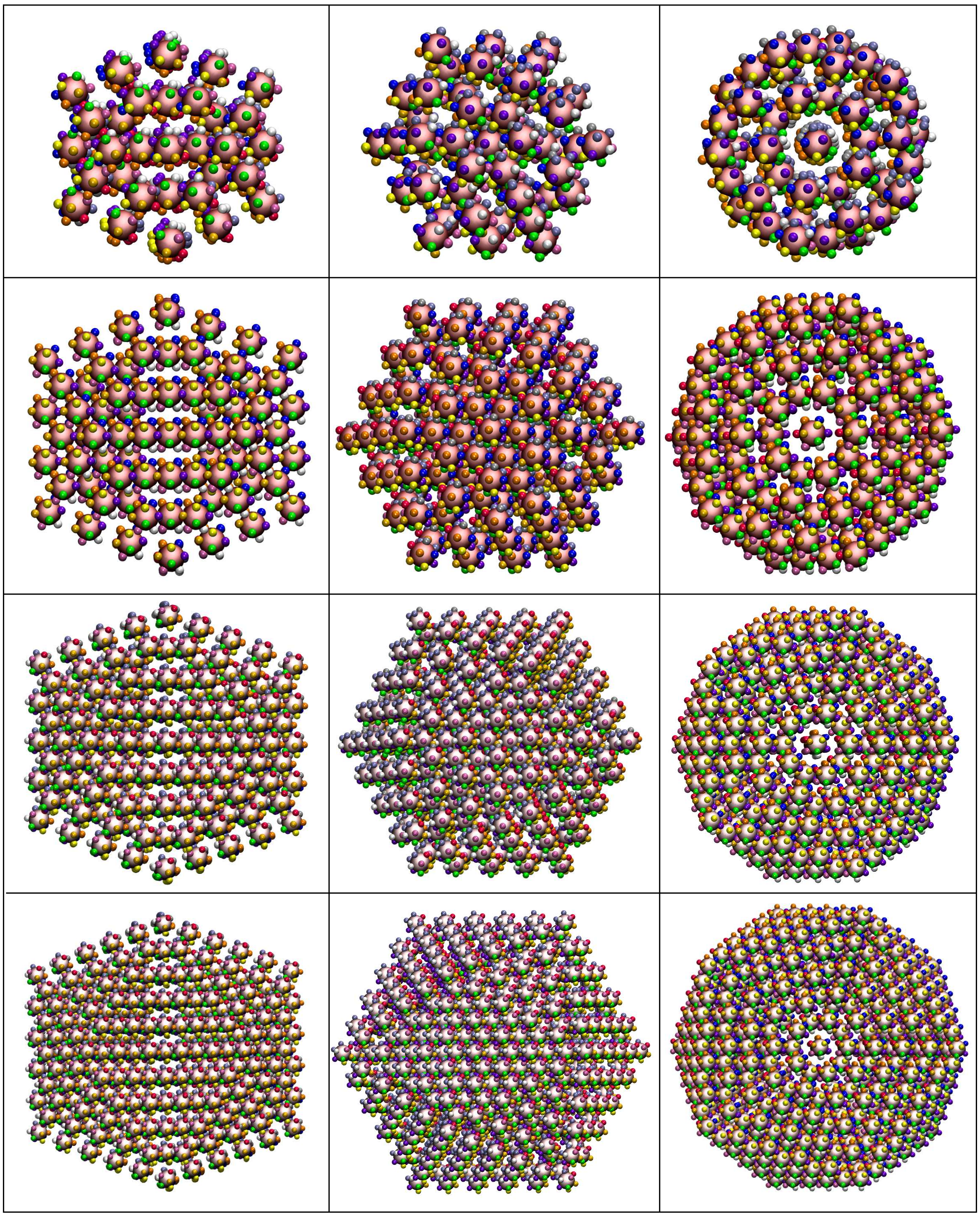


Figure 3.5 Lowest minima located for $\left[\mathrm{LJ}_{13}^{I_h}\right]_{55}$, $\left[\mathrm{LJ}_{13}^{I_h}\right]_{147}$, $\left[\mathrm{LJ}_{13}^{I_h}\right]_{309}$, $\left[\mathrm{LJ}_{13}^{I_h}\right]_{561}$ (top to bottom) with $\varepsilon_{\mathrm{NN}}^{\mathrm{rep}} = \varepsilon_{\mathrm{NN}}^{\mathrm{att}} = \varepsilon_{\mathrm{NNN}}^{\mathrm{rep}} = 1$, $\varepsilon_{\mathrm{NNN}}^{\mathrm{att}} = 0$, $\varepsilon_{\mathrm{self}}^{\mathrm{rep}} = 10^4$, $\varepsilon_{\mathrm{self}}^{\mathrm{att}} = 0$. The 13 different particles with distinct addresses are coloured consistently, with the central particle magnified to highlight the overall structure. The three views correspond to approximate 2-fold, 3-fold and 5-fold axes (left to right).

3.4 Conclusions

This chapter has detailed the implementation of various key components of computational energy landscape theory on GPU hardware. First, basin-hopping global optimisation was accelerated in the GMIN code, using a version of L-BFGS adapted for CUDA and an interface to the GPU-accelerated AMBER potential.[63] The LJ potential was also implemented using CUDA. These results were then extended to form the basis of a GPU-accelerated version of hybrid EF, one component of transition state location in the OPTIM code. The DNEB method, the other component employed in OPTIM for transition state characterisation, was also accelerated using the interfaced potential. Additionally, the local rigid body framework was adapted for GPU hardware. Testing performed for AMBER system sizes in the range of 81 to 22811 atoms gave a speedup relative to CPU of up to two orders of magnitude on Titan Black GPUs. Hence it will now be feasible to explore the energy landscapes of much larger biological systems than previously accessible, opening up a wide array of new applications. Two applications of CUDAGMIN and CUDAOPTIM to the energy landscapes of various mutations of a coiled-coil peptide and the hierarchical self-assembly of icosahedral LJ clusters are summarised in Sections 3.3.5 and 3.3.6, demonstrating the validity of these new, GPU-accelerated methods.

Although significant benefit is only seen for the implementation of L-BFGS with the whole algorithm on GPU for very large systems, this analysis is likely to be potential dependent. In the case of AMBER, the cost of evaluating the potential is very much greater than the cost of copying arrays between the host and device, so the importance of minimising the number of these copy operations is not high. Preliminary GPU benchmarking of L-BFGS with the LJ potential has been performed, using a history size of four and a test system of 1024 atoms with 100 random starting configurations. An order of magnitude speedup was obtained for the version of L-BFGS entirely on GPU, relative to the version with only the LJ potential on GPU. This result may be due to the cost of memory transfer between the host and device being more significant relative to the cost of the potential. Brief tests also indicate that this performance differential between the two implementations becomes even more pronounced for larger LJ clusters, highlighting the importance of having the entire L-BFGS algorithm on GPU. A full analysis on a range of LJ cluster sizes would be an interesting avenue for future work. The advances described in this chapter will provide an excellent foundation for the addition of other GPU-accelerated potentials to GMIN and OPTIM.

Chapter 4

Exploiting sparsity in free energy basin-hopping

4.1 Introduction

So far, we have mainly considered the potential energy landscape, which does not incorporate the effects of entropy at nonzero temperatures. The global potential energy minimum is not necessarily the same as the global free energy minimum at a specified temperature, and extensive reordering of the relative free energies of local minima can occur as the temperature is changed.[16] Various methods exist for constructing entire free energy landscapes, particularly through projection onto selected order parameters,[155–161] but these are computationally expensive. Free energy basin-hopping[16] (FEBH) presents a more computationally affordable alternative for applications where we are interested in the free energies of individual minima, particularly the global free energy minimum of a system at a relevant temperature.

FEBH retains all the advantages of standard basin-hopping global optimisation. Furthermore, the accept/reject test for step-taking based on free energies results in lower mean first encounter times for the global minimum using FEBH than for simply performing basin-hopping using the potential energy and calculating the free energies of the minima a posteriori.[16] In FEBH, the mass-weighted Hessian and the log product of its nonzero eigenvalues must be calculated for each minimum in the chain of proposed moves in order to calculate the local free energies. These additional calculations significantly increase the computational expense for each step involving a new minimum.

We routinely use the LAPACK DSYEV routine[162] for matrix diagonalisation and vibrational normal mode analysis, which provides the information required to calculate equilibrium thermodynamics and rates within the harmonic approximation[1]. However, the individual normal mode frequencies are not required in FEBH. Only the log product of positive Hessian eigenvalues is needed within the harmonic approximation to the vibrational density of states, which is identical to the log of the determinant of the Hessian with zero eigenvalues shifted to unity (a procedure described in Appendix A). Not only does the DSYEV routine generate unnecessary information, but the algorithm also scales as $\mathscr{O}(N^3)$, making it expensive for larger system sizes.

Georgescu and Mandelshtam have tackled a similar problem in their method for estimating free energies,[163] and they exploited two properties of the Hessian matrix to speed up their calculations. Firstly, the Hessian is symmetric positive definite if the zero eigenvalues corresponding to translation and rotation are excluded. This property means that a Cholesky factorisation can be used to calculate the determinant of the Hessian. Secondly, the Hessian is expected to be sparse, with many negligible components, for reasonably short-ranged interatomic potentials. The two potentials we focus on here are the Lennard-Jones (LJ) form[57] and the AMBER force field.[70] The second derivative of the LJ potential tends to zero as r^{-8}, where r is the interatomic distance, so components of the Hessian corresponding to interactions between distant atoms are negligible. The AMBER potential consists of bonded terms, which are short-ranged by definition, and nonbonded terms.[65] The van der Waals (vdW) interactions are modelled using the LJ form and are hence relatively short-ranged. The electrostatic interactions are modelled as point charges and are longer-ranged. These interactions are often truncated through the use of a cutoff.

In this chapter, we describe the refinement of an implementation of sparse Cholesky factorisation for acceleration of FEBH and present tests that demonstrate the acceleration obtained relative to the standard, non-sparse diagonalisation procedure. The graphics processing unit (GPU) implementation of the AMBER 12 potential was used,[63,164] which does not support the use of a cutoff for the electrostatic interactions. For this reason, we took an alternative approach and simply set all values of the Hessian below a certain threshold to zero after the calculation of the second derivatives to ensure sparsity. For the results shown here, this approach was also applied to LJ systems. Then, we employed a sparse Cholesky factorisation on CPU (the GPU implementation provides little benefit for the system sizes we consider), instead of DSYEV, to find the log product of eigenvalues. In the best case, the cost of the Cholesky algorithm can be reduced from $\mathscr{O}(N^3)$ to $\mathscr{O}(N)$ for sparse matrices.[163] Our results indicate that run times can be decreased by factors of 10 to 30 for the larger examples considered, providing faster structure prediction, and access to more complex systems.

4.2 Methods

For the accelerated implementation of FEBH, the usual call to DSYEV after the calculation of the mass-weighted Hessian was replaced with a call to routines for sparse Cholesky factorisation. At a local minimum, the Hessian for an isolated molecule has six zero eigenvalues corresponding to overall translation and rotation, so the determinant will be zero. It is necessary to calculate the log product of the nonzero eigenvalues corresponding to vibrational modes, so we implemented a procedure that shifts the zero eigenvalues by $\lambda_{\text{shift}} = 1$. After ensuring that the Cartesian coordinates are in the centre-of-mass frame and coincident with the principal axes of rotation, this procedure constructs normalised eigenvectors, $\boldsymbol{\nu}$, corresponding to infinitesimal translations and rotations of the system (see Appendix A).[165] The corresponding Hessian eigenvalues are then shifted according to the following equation:

$$H^{\dagger}_{\alpha\beta} = H_{\alpha\beta} + \lambda_{\text{shift}} \nu_\alpha \nu_\beta. \tag{4.1}$$

Before the Cholesky factorisation, all values of the Hessian below a specified cutoff magnitude are set to zero to ensure the sparsity of the matrix.

The initial interface to the Cholesky factorisation procedure in SuiteSparse (a package of sparse matrix solvers) was implemented by Kyle Sutherland-Cash.[117] This interface was then further refined through the correction of some errors, and then validated and tested. The SuiteSparse library is mostly written in C, so an interface was constructed using the Fortran ISO_C_BINDING module to ensure interoperability between C and Fortran types, and to allow safe management of pointers and their associated memory. For our purposes, we required only functionality from the SuiteSparse CHOLMOD package.[166] The nonzero lower triangular elements of the Hessian were copied into a cholmod_triplet matrix, as triplets of nonzero values with their row and column indices. The `cholmod_triplet_to_sparse` function was used to convert this representation into a cholmod_sparse object in compressed sparse column form, which was then passed to `cholmod_analyze`. This function selects a matrix ordering option that gives the best reduction in matrix fill-in and performs a symbolic factorisation. The most efficient algorithm to use for factorisation (either supernodal or simplicial) is automatically selected. The numeric factorisation was then performed using `cholmod_factorize`. Finally, the `cholmod_change_factor` function was used to convert the returned factor to an LDL^T form, where D is diagonal. This procedure allowed the required diagonal entries to be accessed more easily, so that the determinant could then be calculated through accumulation of the logarithms of these values. The integration of SuiteSparse into our existing CMake build was greatly facilitated by the CMake scripts available on GitHub written by Jose Luis Blanco and Jerome Esnault.[167]

4.3 Results and discussion

Tests were run to compare the calculation of the Hessian determinant using DSYEV and the new sparse Cholesky factorisation method as a function of system size for both LJ clusters and selected biomolecules represented by the AMBER force field.[70] Our objective was to achieve a significant speedup with minimal loss of accuracy. To this end, we wish to find the largest magnitude for the Hessian cutoff that still provides sufficient accuracy. If the Hessian cutoff is too large, not only do we lose accuracy, but the Cholesky factorisation can fail outright because the matrix is no longer positive definite. Factorisation failures due to non-positive definite matrices can also occur if the minima are not converged tightly enough, so the root mean square (RMS) force convergence criterion for the local minima is also an important parameter. For a given set of parameters, even if the calculations are still reasonably accurate, too many factorisation failures will reduce the efficiency of the FEBH procedure. We regard a proportion of factorisation failures below 1 % as ideal.

The GMIN FEBH code[89] was compiled on the x86_64 architecture running the Ubuntu 14.04.4 operating system, using Intel compiler version 16.0.4 with the MKL library version 11.3.4. The tests were run on 2.40 GHz Intel Xeon E5-2620 v3 CPUs and employed version 4.5.3 of SuiteSparse.

4.3.1 Atomic clusters

Tables 4.1a and 4.1b show the percentage of factorisation failures for LJ_{1000} and LJ_{5000} clusters respectively, for various combinations of the RMS force convergence criterion for local minimisation and the Hessian cutoff. These percentages are calculated from 100 FEBH runs of nine steps each, starting from randomly generated atomic configurations, giving 1000 local minima. The FEBH step size was set to a maximum of $0.43\,\sigma$ for any individual Cartesian coordinate, and angular steps were taken for individual LJ atoms above a pair binding energy tolerance of $0.52\,\varepsilon$. For both cluster sizes, the values for the RMS force were chosen to span a range showing the effects of both 'under-convergence' and 'over-convergence'. The effect of an RMS force that is too large can be seen in Table 4.1a for a value of $10^{-3}\,\varepsilon\sigma^{-1}$ and in Table 4.1b for $10^{-4}\,\varepsilon\sigma^{-1}$, both of which exhibit factorisation failure frequencies greater than 1 %. Similarly, the Hessian cutoff spans a range of values that demonstrate the effects of a cutoff that is both too large and too small. An increased number of factorisation failures is observed for a Hessian cutoff value that is too large, as in Table 4.1a for $10^{-2}\,\varepsilon\sigma^{-1}$ and in Table 4.1b for $2\times10^{-3}\,\varepsilon\sigma^{-1}$. A visual representation of the effect of varying the Hessian cutoff on overall sparsity for LJ_{1000} is shown in Figure 4.1.

Table 4.1 Percentage of factorisation failures for two LJ cluster sizes as a function of RMS force convergence threshold and the cutoff for neglect of Hessian matrix elements[i]

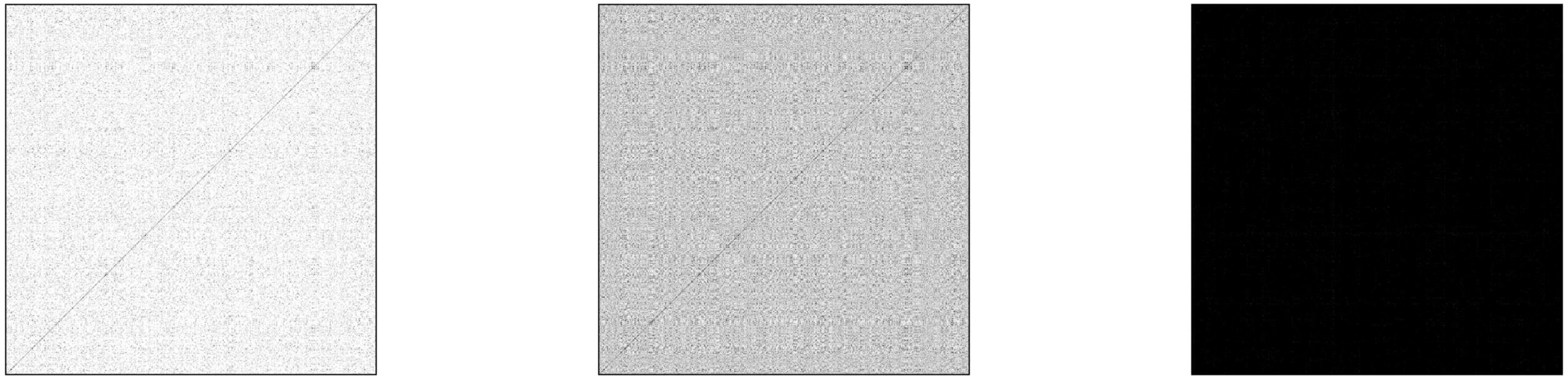
(a) LJ_{1000}

RMS force / $\varepsilon\sigma^{-1}$	**Hessian cutoff** / $\varepsilon\sigma^{-2}$		
	10^{-6}	10^{-3}	10^{-2}
10^{-3}	4.1	4.1	18.2
10^{-5}	0.7	0.7	15.3
10^{-7}	0.9	0.9	13.3

(b) LJ_{5000}

RMS force / $\varepsilon\sigma^{-1}$	**Hessian cutoff** / $\varepsilon\sigma^{-2}$		
	10^{-6}	10^{-3}	2×10^{-3}
10^{-4}	3.2	3.1	11.4
10^{-6}	0.4	0.4	6.5
10^{-8}	0.0	0.0	4.3

[i]Factorisation failures result from a non-positive definite Hessian being used for the sparse Cholesky factorisation. Percentages are calculated from a total of 1000 structures generated from 100 FEBH runs of nine steps each, starting from randomly generated atomic configurations.



(a) $10^{-2}\,\varepsilon\sigma^{-2}$ (b) $10^{-3}\,\varepsilon\sigma^{-2}$ (c) $10^{-6}\,\varepsilon\sigma^{-2}$

Figure 4.1 A visualisation of the effect of cutoff variation for neglect of Hessian matrix elements on the sparsity of the resulting matrix. The plots are derived from a single LJ_{1000} minimum converged to an RMS force tolerance of $10^{-5}\,\varepsilon\sigma^{-1}$. Nonzero matrix elements are coloured black and matrix elements set to zero are coloured white. Each subfigure is labelled with the appropriate value for the cutoff.

We aim for the calculation of the log product of Hessian eigenvalues using this sparse matrix method to produce results of comparable accuracy to full diagonalisation. Tables 4.2a and 4.2b show the percentage errors in the sparse calculations for the same range of values of the RMS force convergence threshold and Hessian cutoff as Tables 4.1a and 4.1b. We see that the results are accurate, even for values of the cutoff and RMS force threshold that produced relatively large numbers of failures due to non-positive definite matrices.

Table 4.2 Percentage errors in the log product of positive Hessian eigenvalues for two LJ cluster sizes as a function of RMS force convergence threshold and the cutoff for neglect of Hessian matrix elements[j]

(a) LJ_{1000}

RMS force / $\varepsilon\sigma^{-1}$	**Hessian cutoff** / $\varepsilon\sigma^{-2}$		
	10^{-6}	10^{-3}	10^{-2}
10^{-3}	0.01	0.01	0.15
10^{-5}	0.01	0.01	0.15
10^{-7}	0.01	0.01	0.15

(b) LJ_{5000}

RMS force / $\varepsilon\sigma^{-1}$	**Hessian cutoff** / $\varepsilon\sigma^{-2}$		
	10^{-6}	10^{-3}	2×10^{-3}
10^{-4}	0.01	0.02	0.04
10^{-6}	0.01	0.02	0.04
10^{-8}	0.01	0.02	0.04

[j]These results are generated from the same structures used in Table 4.1. Percentage errors are calculated for sparse Cholesky factorisation relative to a non-sparse diagonalisation procedure.

The principal objective of this study was to speed up the calculation of the log product of eigenvalues of the Hessian, and Tables 4.3a and 4.3b summarise the results. The values are reported as the average time for a DSYEV calculation divided by the average time for a sparse Cholesky calculation from the same basin-hopping runs used to produce all the preceding results. Timings for the calculations that failed were not included in the averages. As expected, the speedups are greater for larger values of the Hessian cutoff, due to the greater sparsity of the resulting matrices. The RMS force threshold does not seem to affect the speedups obtained for either system to a large extent. Greater efficiency gains were obtained for the large Hessian matrices associated with the larger LJ_{5000} cluster.

Table 4.3 Average speedups for finding the log product of positive eigenvalues for two LJ cluster sizes as a function of RMS force convergence threshold and the cutoff for neglect of Hessian matrix elements[k]

(a) LJ_{1000}

RMS force / $\varepsilon\sigma^{-1}$	**Hessian cutoff /** $\varepsilon\sigma^{-2}$		
	10^{-6}	10^{-3}	10^{-2}
10^{-3}	2.9	4.6	7.7
10^{-5}	2.8	4.5	7.8
10^{-7}	2.8	4.5	7.7

(b) LJ_{5000}

RMS force / $\varepsilon\sigma^{-1}$	**Hessian cutoff /** $\varepsilon\sigma^{-2}$		
	10^{-6}	10^{-3}	2×10^{-3}
10^{-4}	7.7	34.0	37.7
10^{-6}	9.2	35.2	36.8
10^{-8}	10.1	36.2	38.1

[k]These results are generated from the same structures used in Table 4.1. Speedups are calculated for sparse Cholesky factorisation relative to a non-sparse diagonalisation procedure.

4.3.2 Proteins

The activity of biological systems can be strongly dependent on temperature, so these molecules constitute an important application for structure prediction using FEBH.[16] The systems investigated here are a truncated monomeric version (3522 atoms) of the trimeric haemagglutinin (HA) glycoprotein of the influenza A(H1N1) virus and a full monomeric structure (7585 atoms),[142] as seen previously in Figures 3.2d and 3.2f. The AMBER ff99SB forcefield[70] was used in both cases with an effectively infinite nonbonded cutoff (999.99 Å). All tests used the modified GB solvent model[79,80] (AMBER input flag `igb` $= 2$) at a salt concentration $0.2M$ with a cutoff of 12 Å for the calculation of the effective Born radii. Each test employed a set of 100 starting structures, taken from high temperature molecular dynamics (MD) runs. For the truncated monomer, nine basin-hopping steps were taken for each structure to generate 1000 local minima, with Cartesian moves of the backbone atoms set to a maximum value of 0.2 Å. However, due to the computational expense of the DSYEV calculations for the full monomer, no basin-hopping steps were performed for these starting structures and only 100 minimised structures were obtained. To reduce the overall time taken for the tests, the minimisations were carried out using the GPU-accelerated implementation[168] of the limited-memory BFGS (L-BFGS) algorithm.[90–94] GeForce GTX TITAN Black GPUs were used, with NVIDIA Linux driver version 375.26 and NVIDIA CUDA Toolkit 8.0.

Due to the lack of GPU-accelerated, analytical second derivatives for the AMBER potential, numerical second derivatives with calls to the GPU-accelerated AMBER energy and gradient subroutine were used in calculating the Hessian matrix. A central difference approximation of the form

$$\frac{\partial^2 V}{\partial x_i x_j} \approx \frac{\nabla_i V(\mathbf{X}+\zeta\mathbf{x}) - \nabla_i V(\mathbf{X}-\zeta\mathbf{x})}{2\zeta x_j} \tag{4.2}$$

was employed in calculating these derivatives, where $V(\mathbf{X})$ is the energy at configuration $\mathbf{X}$ in nuclear configuration space and $\zeta \ll 1$. We performed some preliminary tests to determine the optimal value of ζ for the final benchmarks. We minimised the 100 full monomeric structures for a range of values of ζ and recorded the percentage of factorisation failures resulting from a non-positive definite Hessian matrix. Based on the results in Table 4.4, we chose a value of 10^{-4} for ζ.

Table 4.4 Percentage of factorisation failures for the HA full monomer as a function of the value of ζ used in calculating the numerical second derivatives[l]

ζ	**% of factorisation failures**
10^{-2}	5
10^{-3}	3
10^{-4}	0
10^{-5}	11
10^{-6}	25
10^{-7}	9

[l]Factorisation failures result from a non-positive definite Hessian being used for the sparse Cholesky factorisation. Percentages are calculated from a total of 100 minimised structures, starting from structures taken from high temperature MD trajectories. A value of 10^{-6} kcal mol^{-1} Å^{-1} was used for the RMS force convergence threshold and the cutoff for neglect of Hessian matrix elements was set to 10^{-6} kcal mol^{-1} Å^{-2}.

We also tested the SuiteSparse GPU implementation[169] to accelerate our calculations. However, we opted not to use this approach in our benchmarks as the speedup only amounted to about 5 to 10 % for the examples considered here. This result is likely due to the relatively small size of our matrices compared to those benchmarked by the developers, for both the clusters and proteins considered here.

Tables 4.5a and 4.5b show the percentage of factorisation failures for the truncated monomer and the full monomer with various values of the RMS force convergence criterion and the Hessian cutoff. As for the atomic clusters considered in Section 4.3.1, fewer factorisation failures are observed for smaller values of the RMS force and Hessian cutoff.

Table 4.5 Percentage of factorisation failures for two protein examples as a function of RMS force convergence threshold and the cutoff for neglect of Hessian matrix elements[m]

(a) Truncated HA monomer

RMS force / kcal mol^{-1} Å^{-1}	**Hessian cutoff / kcal mol^{-1} Å^{-2}**		
	10^{-8}	10^{-5}	3.5×10^{-5}
10^{-4}	1.8	1.8	6.3
10^{-5}	0.8	0.8	5.2
10^{-6}	0.0	0.0	4.7

(b) Full HA monomer

RMS force / kcal mol^{-1} Å^{-1}	**Hessian cutoff / kcal mol^{-1} Å^{-2}**		
	10^{-9}	10^{-6}	6.0×10^{-6}
10^{-4}	3.0	3.0	24.0
10^{-5}	0.0	0.0	20.0
10^{-6}	0.0	0.0	19.0

[m]Factorisation failures result from a non-positive definite Hessian being used for the sparse Cholesky factorisation. Percentages are calculated from a total of 100 minimised structures, starting from structures taken from high temperature MD trajectories. The structures used in Table 4.5a are the same as those used in Table 4.4.

Tables 4.6a and 4.6b show the percentage errors for these calculations for the same range of values for the RMS force convergence threshold and Hessian cutoff. Again, this error is very small for all combinations of parameters.

Table 4.6 Percentage errors in the log product of positive Hessian eigenvalues for two protein examples as a function of RMS force convergence threshold and the cutoff for neglect of Hessian matrix elements[n]

(a) Truncated HA monomer

RMS force / kcal mol^{-1} Å^{-1}	**Hessian cutoff / kcal mol^{-1} Å^{-2}**		
	10^{-8}	10^{-5}	3.5×10^{-5}
10^{-4}	0.10	0.10	0.10
10^{-5}	0.10	0.10	0.10
10^{-6}	0.10	0.10	0.10

(b) Full HA monomer

RMS force / kcal mol^{-1} Å^{-1}	**Hessian cutoff / kcal mol^{-1} Å^{-2}**		
	10^{-9}	10^{-6}	6.0×10^{-6}
10^{-4}	0.06	0.06	0.06
10^{-5}	0.06	0.06	0.06
10^{-6}	0.06	0.06	0.06

[n]These results are generated from the same structures used in Table 4.5. Percentage errors are calculated for sparse Cholesky factorisation relative to a non-sparse diagonalisation procedure.

The associated speedups are shown in Tables 4.7a and 4.7b. The efficiency gains with respect to Hessian cutoff and RMS force are similar to the results we obtained for the LJ clusters.

Table 4.7 Average speedups for finding the log product of positive eigenvalues for two protein examples as a function of RMS force convergence threshold and the cutoff for neglect of Hessian matrix elements[o]

(a) Truncated HA monomer

RMS force / kcal mol^{-1} Å^{-1}	**Hessian cutoff / kcal mol^{-1} Å^{-2}**		
	10^{-8}	10^{-5}	3.5×10^{-5}
10^{-4}	7.3	7.7	8.4
10^{-5}	7.1	7.8	8.6
10^{-6}	7.3	7.7	8.5

(b) Full HA monomer

RMS force / kcal mol^{-1} Å^{-1}	**Hessian cutoff / kcal mol^{-1} Å^{-2}**		
	10^{-9}	10^{-6}	6.0×10^{6}
10^{-4}	14.6	16.9	17.3
10^{-5}	14.6	16.8	16.6
10^{-6}	14.8	16.7	18.0

[o]These results are generated from the same structures used in Table 4.5. Speedups are calculated for sparse Cholesky factorisation relative to a non-sparse diagonalisation procedure.

4.4 Conclusions

We have described the refinement and testing of an interface between our FEBH global optimisation routines and the SuiteSparse package, which allows us to perform sparse Cholesky factorisation to obtain vibrational frequencies within the FEBH framework. Tests were performed for both LJ clusters and selected biomolecules represented using the AMBER potential. A significant speedup of around 10 to 30 times was achieved for the calculation of the vibrational density of states from the product of nonzero Hessian eigenvalues, with negligible loss of accuracy, with respect to the fastest matrix diagonalisation routine that does not exploit sparsity. Exploiting sparsity will therefore accelerate structure prediction and provide access to larger and more complex systems. This capability may be important for a wide range of molecular and condensed matter systems where finite temperature effects are significant. It enables us to estimate local free energies, along with other potentially useful thermodynamic properties, such as the heat capacity.

Chapter 5

Conclusions and future work

Basin-hopping global optimisation,[8,9] a local rigid body framework,[10,11] hybrid eigenvector-following[12] (EF) and the doubly-nudged[13] elastic band[14,15] (DNEB) method have been accelerated on graphics processing units (GPUs) for systems represented by the AMBER and Lennard-Jones (LJ) potentials. Tests on biomolecular systems demonstrated an increase in performance of up to two orders of magnitude. This increase greatly extends the scope of future work on biomolecular energy landscapes and the range of computationally accessible, biologically relevant systems. Applications were presented for a coiled-coil peptide[129,130] and aggregates of addressable LJ clusters in Chapter 3.[131]

These GPU-accelerated methods have also been successfully applied to the investigation of virus capsid self-assembly through the reoptimisation of dimer configurations predicted using docking,[170,171] the study of the multifunnel energy landscapes underlying fibril elongation for the amyloid-β42 protein,[172–174] characterisation of large-scale structural transitions of the C-terminal domain of the bacterial transcription factor RfaH,[175–177] and benchmarks for a modified quasi-continuous interpolation (QCI) procedure through location of low energy transition states for epidermal growth factor receptor kinase activation.[173,178,179] Ongoing applications include the location of pathways for formation of cylindrin (an amyloid oligomer),[180] investigation into the effects of mutations on kinase activation,[181–184] the analysis of the effects of mutational changes on the folding and binding behaviour of 7SK RNA,[185] and studies of antibody folding and sequence optimisation to improve antigen binding affinities.

The most obvious avenue for further work is the implementation of other potentials on GPU. Preliminary tests for a small system represented by the LJ potential showed an order of magnitude speedup for the GPU implementation of L-BFGS relative to having the L-BFGS update operations on CPU. This result indicates that the GPU implementations of the energy landscape framework will be highly beneficial for systems where the cost of evaluating the potential is comparable to that of copying the coordinates and gradient between host and device. The tests also imply that the version of L-BFGS used in DNEB might benefit from a full implementation on GPU. It would be interesting to perform further history size analysis and comparisons to having the L-BFGS algorithm on CPU for other potentials, as the trends observed are likely to be potential dependent. The addition of new potentials to the code has been made relatively simple through the use of a C++ class for the potential, from which new potentials can be derived using inheritance. Due to the vast array of possibilities for optimisation using CUDA, and the increased time required to implement and test code for the GPU relative to code for the CPU, the sensible way to approach development is through an iterative process of profiling followed by necessary optimisations to address the current performance bottlenecks. It is therefore possible that further optimisations may need to be applied to the current CUDA code when it is used with a new potential.

Although an effort was made to include the functionality of most of the frequently used input options from GMIN and OPTIM in the CUDA code, a few options remain unimplemented and may need to be added in future. The most important development is likely to be implementation of hybrid EF when an analytical Hessian is available. This scheme is quite different from the algorithm involving Rayleigh-Ritz minimisation and involves the use of Lagrange multipliers to follow the analytically determined eigenvector uphill.[1]

Given the poor parallelisability of the L-BFGS algorithm, it might also be worth exploring alternative local minimisation algorithms more suited to implementation on GPU. One possibility to explore might be the FIRE minimiser (named for 'fast inertial relaxation engine'),[186] which is based on principles from molecular dynamics (MD) simulations, with an extra velocity term and an adaptive time step to allow acceleration in steeper directions. It is more amenable to parallelisation as it does not use a sequential history of the steps taken as L-BFGS does. However, FIRE has been shown to be three to four times slower than L-BFGS in tests of local minimisation similar to our applications on CPU,[97] which might mitigate any increase in speed due to increased parallelisability.

One effect observed from the analysis of the timings for the different history sizes and implementations with the AMBER potential was that the version of L-BFGS entirely on GPU was faster than just the potential on GPU only for sufficiently large and complex systems that required a large history size. It has been informally observed that L-BFGS minimisation often requires a larger history size when closer to a minimum, presumably because the reduced step size makes accurate estimation of the step direction more difficult. The fastest overall minimisation scheme for AMBER systems of moderate size might be an initial minimisation using a small history size, followed by another minimisation using a large history size when closer to convergence. This would be a worthwhile hypothesis to test in future work.

It is also hoped that two interesting projects initiated in association with the development of the GPU code may be completed in future work. The first of these is the application of GPU-accelerated basin-hopping global optimisation in conjunction with local rigidification to the factorised superposition approach (FSA).[140] This is an energy landscape approach to calculation of relative binding free energies of macromolecules, which has the potential for useful applications in the field of computational drug design. The initial stage of the calculation involves sampling the energy landscape to collect minima, for which basin-hopping global optimisation can be used. The fundamental idea underlying the method is that the number of minima on the surface is greatly reduced through the extensive rigidification of regions distant from the binding site in a self-consistent manner. Initial tests on GPU for the aldose reductase system have not yet managed to reproduce the convergence of the free energy demonstrated in the original paper. However, it is hoped that the efficacy of basin-hopping global optimisation in this context can be proved for a smaller system initially. Once any problems have been resolved, sampling for larger systems may be resumed.

Tests are also currently underway for the DNEB and hybrid EF approach to finding pathways between minima, using a fragment of the influenza A(H1N1) haemagglutinin glycoprotein. The process under investigation is the structural transformation of this peptide that facilitates fusion with the host cell membrane to allow viral entry, and the initial aim is to find a connection between a pre-fusion structure and a post-fusion structure. Details of the folding mechanism could then be elucidated and the effects of mutations (known to increase transmissibility or pathogenicity of the virus) on this process could be probed. If these tests are eventually successful, further runs could be performed on the whole trimeric structure.

As a result of the GPU-acceleration, fresh challenges have arisen in applying the current methodology to very large systems. For example, the large number of minima and transition states that end up in the database can quickly become unwieldy and the calculation of the list of distances between all pairs of minima becomes expensive. One way of preventing the database from growing too quickly is to only add minima and transition states to the database if they are on the best path (determined using Dijkstra's algorithm[5,99]) for that connection attempt and then discard all other stationary points. Additionally, we find it is more efficient to concentrate the transition state searches on larger gaps before smaller gaps are considered when trying to find an initial connection for a very large distance.[129] One other method that has also recently been implemented aims to reduce the cost of calculating the distances between pairs of minima by initially assigning fictitious distances between pairs and only calculating the true distance when the connection appears on the best path.[187,188] Efforts are also being made to increase the efficiency of the DNEB algorithm for large systems. The current formulation uses a Cartesian interpolation to generate initial images for the band, but over large distances this approach can generate poor estimates, many of which cannot be further refined due to overflow of the forces. A refined QCI scheme is currently being modified for use with AMBER and will hopefully resolve this problem.[173,189]

The sparse Cholesky factorisation applied in the free energy basin-hopping (FEBH) procedure also represents an important step forward. Previously the most time-consuming component, the calculation of the log product of eigenvalues of the Hessian is now approximately 10 to 30 times faster. The new bottleneck in these calculations is now the calculation of the Hessian matrix of second derivatives. For potentials with analytical second derivatives, GPU-acceleration may be the best way to speed these calculations up. The numerical second derivative calculations for the AMBER potential are particularly slow due to the large number of function calls required. Analytical second derivatives would be much faster and are available on CPU within the Nucleic Acid Builder (NAB) package.[190] A large acceleration could be achieved by implementing these derivatives on GPU, although this task would be formidable, due to the complexity of the expressions involved.

In conclusion, significant acceleration has been obtained for several methods used for sampling energy landscapes. Larger system sizes are now accessible for study, opening up a wealth of new applications. Recent and ongoing applications that exploit the new CUDA interfaces have been run for a variety of systems. The corresponding results would have accounted for thousands of years of CPU time, dramatically demonstrating the new frontiers that the present work has opened up.

Bibliography

[1] D. J. Wales, *Energy Landscapes*, Cambridge University Press, Cambridge, 2003.

[2] M. Born and R. Oppenheimer, *Ann. Phys.*, 1927, **389**, 457–484.

[3] A. R. Leach, *Molecular Modelling: Principles and Applications*, Pearson Education, Harlow, 2nd edn., 2001.

[4] J. N. Murrell and K. J. Laidler, *Trans. Faraday Soc.*, 1968, **64**, 371–377.

[5] J. M. Carr, S. A. Trygubenko and D. J. Wales, *J. Chem. Phys.*, 2005, **122**, 234903.

[6] D. J. Wales and T. V. Bogdan, *J. Phys. Chem. B*, 2006, **110**, 20765–20776.

[7] F. H. Stillinger, *Phys. Rev. E*, 1999, **59**, 48–51.

[8] Z. Li and H. A. Scheraga, *Proc. Natl. Acad. Sci. U. S. A.*, 1987, **84**, 6611–6615.

[9] D. J. Wales and J. P. K. Doye, *J. Phys. Chem. A*, 1997, **101**, 5111–5116.

[10] H. Kusumaatmaja, C. S. Whittleston and D. J. Wales, *J. Chem. Theory Comput.*, 2012, **8**, 5159–5165.

[11] V. Rühle, H. Kusumaatmaja, D. Chakrabarti and D. J. Wales, *J. Chem. Theory Comput.*, 2013, **9**, 4026–4034.

[12] L. J. Munro and D. J. Wales, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1999, **59**, 3969–3980.

[13] S. A. Trygubenko and D. J. Wales, *J. Chem. Phys.*, 2004, **120**, 2082–2094.

[14] G. Henkelman, B. P. Uberuaga and H. Jónsson, *J. Chem. Phys.*, 2000, **113**, 9901–9904.

[15] G. Henkelman and H. Jónsson, *J. Chem. Phys.*, 2000, **113**, 9978–9985.

[16] K. H. Sutherland-Cash, D. J. Wales and D. Chakrabarti, *Chem. Phys. Lett.*, 2015, **625**, 1–4.

[17] General-Purpose Computation on Graphics Hardware, http://gpgpu.org/, (accessed August 2017).

[18] O. Maitre, in *Massively Parallel Evolutionary Computation on GPGPUs*, ed. S. Tsutsui and P. Collet, Springer-Verlag, Berlin, 2013, pp. 15–34.

[19] M. M. Waldrop, *Nature*, 2016, **530**, 144–147.

[20] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, Morgan Kaufmann, Waltham, 2012.

[21] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley, Boston, 2010.

[22] CUDA C Programming Guide, http://docs.nvidia.com/cuda/cuda-c-programming-guide/, (accessed August 2017).

[23] WHAT IS GPU-ACCELERATED COMPUTING?, http://www.nvidia.com/object/what-is-gpu-computing.html, (accessed August 2017).

[24] M. D. Hill and M. R. Marty, *Computer*, 2008, **41**, 33–38.

[25] E. Lindholm, J. Nickolls, S. Oberman and J. Montrym, *IEEE Micro*, 2008, **28**, 39–55.

[26] S. W. Williams and D. H. Bailey, in *Performance Tuning of Scientific Applications*, ed. D. H. Bailey, R. F. Lucas and S. Williams, CRC Press, Boca Raton, 2010, ch. 2, pp. 11–32.

[27] AMD vs. Nvidia: Who Dominates GPUs?, http://www.investopedia.com/news/amd-versus-nvdia-amd-nvda/, (accessed August 2017).

[28] Language Solutions, https://developer.nvidia.com/language-solutions, (accessed August 2017).

[29] High-Performance Computing, http://www.amd.com/en-us/products/graphics/workstation/firepro-remote-graphics/gpu-compute, (accessed August 2017).

[30] CUDA GPUs, https://developer.nvidia.com/cuda-gpus, (accessed August 2017).

[31] OpenCL$^{\text{TM}}$ Zone - Accelerate Your Applications, http://developer.amd.com/tools-and-sdks/opencl-zone/, (accessed August 2017).

[32] Whitepaper: NVIDIA's Next Generation CUDA$^{\text{TM}}$ Compute Architecture: Fermi$^{\text{TM}}$, http://www.nvidia.co.uk/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, (accessed August 2017).

[33] Whitepaper: NVIDIA's Next Generation CUDA$^{\text{TM}}$ Compute Architecture: Kepler$^{\text{TM}}$ GK110/210, http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf, (accessed August 2017).

[34] Whitepaper: NVIDIA GeForce GTX 980, https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF, (accessed August 2017).

[35] Whitepaper: NVIDIA Tesla P100, http://www.nvidia.com/object/pascal-architecture-whitepaper.html, (accessed August 2017).

[36] Pascal Compatibility Guide for CUDA Applications, http://docs.nvidia.com/cuda/pascal-compatibility-guide/index.html, (accessed August 2017).

[37] J.-H. Huang, presented in part at the NVIDIA GPU Technology Conference, San Jose, May, 2017.

[38] Whitepaper: NVIDIA TESLA V100 GPU ARCHITECTURE, http://www.nvidia.com/object/volta-architecture-whitepaper.html, (accessed August 2017).

[39] Inside Volta: The World's Most Advanced Data Center GPU, https://devblogs.nvidia.com/parallelforall/inside-volta/, (accessed August 2017).

[40] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*, Addison-Wesley, Upper Saddle River, 2013.

[41] Tesla, http://www.nvidia.co.uk/object/tesla-high-performance-computing-uk.html, (accessed August 2017).

[42] L. Polok and P. Smrz, in *Proceedings of the 24th High Performance Computing Symposium*, Society for Computer Simulation International, San Diego, 2016, pp. 1–8.

[43] NVIDIA® TESLA® GPU ACCELERATORS, http://www.nvidia.co.uk/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf, (accessed August 2017).

[44] P. Zhang and Y. Gao, in *High Performance Computing*, ed. J. M. Kunkel and T. Ludwig, Springer International Publishing, Cham, 2015, vol. 9137, pp. 17–30.

[45] GeForce GTX TITAN Black, http://www.nvidia.co.uk/gtx-700-graphics-cards/gtx-titan-black/, (accessed August 2017).

[46] J. Kussmann and C. Ochsenfeld, *J. Chem. Theory Comput.*, 2017, **13**, 2712–2716.

[47] A. Charara, H. Ltaief and D. Keyes, in *Euro-Par 2016: Parallel Processing*, ed. P.-F. Dutot and D. Trystram, Springer International Publishing, Cham, 2016, vol. 9833, pp. 477–489.

[48] D. D. Donno, A. Esposito, L. Tarricone and L. Catarinucci, *IEEE Trans. Antennas Propag.*, 2010, **52**, 116–122.

[49] Board Specification: NVIDIA Tesla C870 GPU Computing Processor Board, http://www.nvidia.co.uk/docs/IO/43395/C870-BoardSpec_BD-03399-001_v04.pdf, (accessed August 2017).

[50] J. L. Hennessy and D. A. Patterson, *Computer Organization and Design: The Hardware / Software Interface*, Morgan Kaufmann, Amsterdam, 5th edn., 2013.

[51] R. Farber, *CUDA Application Design and Development*, Morgan Kaufmann, Amsterdam, 2011.

[52] I. S. Haque and V. S. Pande, in *GPU Computing Gems Emerald Edition*, ed. W.-m. W. Hwu, Morgan Kaufmann, Boston, 2011, ch. 2, pp. 19–34.

[53] O. Villa, M. Fatica, N. Gawande and A. Tumeo, in *Euro-Par 2013 Parallel Processing*, ed. F. Wolf, B. Mohr and D. an Mey, Springer International Publishing, Cham, 2013, vol. 8097, pp. 813–825.

[54] Faster Parallel Reductions on Kepler, https://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/, (accessed August 2017).

[55] V. Volkov and J. W. Demmel, *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE, 2008, pp. 1–11.

[56] D. Storti and M. Yurtoglu, *CUDA for Engineers: An Introduction to High-Performance Parallel Computing*, Addison-Wesley, New York, 2015.

[57] J. E. Jones, *Proc. R. Soc. A*, 1924, **106**, 463–477.

[58] F. M. Mourits and F. H. A. Rummens, *Can. J. Chem.*, 1977, **55**, 3007–3020.

[59] J. D. Honeycutt and H. C. Andersen, *J. Phys. Chem.*, 1987, **91**, 4950–4963.

[60] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang and R. J. Woods, *J. Comput. Chem.*, 2005, **26**, 1668–1688.

[61] Amber Home Page, http://ambermd.org, (accessed August 2017).

[62] P. K. Weiner and P. A. Kollman, *J. Comput. Chem.*, 1981, **2**, 287–303.

[63] A. W. Götz, M. J. Williamson, D. Xu, D. Poole, S. Le Grand and R. C. Walker, *J. Chem. Theory Comput.*, 2012, **8**, 1542–1555.

[64] J. W. Ponder and D. A. Case, in *Advances in Protein Chemistry*, ed. V. Daggett, Academic Press, San Diego, 2003, vol. 66, pp. 27–85.

[65] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel and P. Kollman, *Comput. Phys. Commun.*, 1995, **91**, 1–41.

[66] Y.-P. Pang, *Biochem. Biophys. Res. Commun.*, 2015, **457**, 183–186.

[67] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell and P. A. Kollman, *J. Am. Chem. Soc.*, 1996, **118**, 2309–2309.

[68] S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta and P. Weiner, *J. Am. Chem. Soc.*, 1984, **106**, 765–784.

[69] W. D. Cornell, P. Cieplak, C. I. Bayly and P. A. Kollmann, *J. Am. Chem. Soc.*, 1993, **115**, 9620–9631.

[70] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg and C. Simmerling, *Proteins: Struct., Funct., Bioinf.*, 2006, **65**, 712–725.

[71] J. A. Maier, C. Martinez, K. Kasavajhala, L. Wickstrom, K. E. Hauser and C. Simmerling, *J. Chem. Theory Comput.*, 2015, **11**, 3696–3713.

[72] E. Małolepsza, B. Strodel, M. Khalili, S. Trygubenko, S. N. Fejer and D. J. Wales, *J. Comput. Chem.*, 2010, **31**, 1402–1409.

[73] E. Małolepsza, B. Strodel, M. Khalili, S. Trygubenko, S. Fejer, J. M. Carr and D. J. Wales, *J. Comput. Chem.*, 2012, **33**, 2209–2209.

[74] A. Dejoux, P. Cieplak, N. Hannick, G. Moyna and F.-Y. Dupradeau, *J. Mol. Model.*, 2001, **7**, 422–432.

[75] Amber 12 Reference Manual, http://ambermd.org/doc12/Amber12.pdf, (accessed August 2017).

[76] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand and R. C. Walker, *J. Chem. Theory Comput.*, 2013, **9**, 3878–3888.

[77] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, Oxford, 1991.

[78] T. Darden, D. York and L. Pedersen, *J. Chem. Phys.*, 1993, **98**, 10089–10092.

[79] A. Onufriev, D. Bashford and D. A. Case, *Proteins: Struct., Funct., Bioinf.*, 2004, **55**, 383–394.

[80] A. Onufriev, D. Bashford and D. A. Case, *J. Phys. Chem. B*, 2000, **104**, 3712–3720.

[81] J. Srinivasan, M. W. Trevathan, P. Beroza and D. A. Case, *Theor. Chem. Acc.*, 1999, **101**, 426–434.

[82] W. C. Still, A. Tempczyk, R. C. Hawley and T. Hendrickson, *J. Am. Chem. Soc.*, 1990, **112**, 6127–6129.

[83] A. Onufriev, D. A. Case and D. Bashford, *J. Comput. Chem.*, 2002, **23**, 1297–1304.

[84] G. D. Hawkins, C. J. Cramer and D. G. Truhlar, *J. Phys. Chem.*, 1996, **100**, 19824–19839.

[85] M. Schaefer and C. Froemmel, *J. Mol. Biol.*, 1990, **216**, 1045–1066.

[86] V. Tsui and D. A. Case, *Biopolymers*, 2000, **56**, 275–291.

[87] S. Le Grand, A. W. Götz and R. C. Walker, *Comput. Phys. Commun.*, 2013, **184**, 374–380.

[88] D. J. Wales, *Phys. Biol.*, 2005, **2**, S86–S93.

[89] GMIN: A program for finding global minima and calculating thermodynamic properties from basin-sampling., http://www-wales.ch.cam.ac.uk/GMIN/, (accessed August 2017).

[90] J. Nocedal, *Math. Comput.*, 1980, **35**, 773–782.

[91] C. G. Broyden, *IMA J. Appl. Math.*, 1970, **6**, 222–231.

[92] R. Fletcher, *Comput. J.*, 1970, **13**, 317–322.

[93] D. Goldfarb, *Math. Comput.*, 1970, **24**, 23–26.

[94] D. F. Shanno, *Math. Comput.*, 1970, **24**, 647–656.

[95] J. E. Dennis, Jr. and J. J. Moré, *SIAM Rev.*, 1977, **19**, 46–89.

[96] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, 2nd edn., 2006.

[97] D. Asenjo, J. D. Stevenson, D. J. Wales and D. Frenkel, *J. Phys. Chem. B*, 2013, **117**, 12717–12723.

[98] OPTIM: A program for optimizing geometries and calculating reaction pathways, http://www-wales.ch.cam.ac.uk/OPTIM/, (accessed August 2017).

[99] E. W. Dijkstra, *Numer. Math.*, 1959, **1**, 269–271.

[100] D. J. Wales, J. M. Carr, M. Khalili, V. K. de Souza, B. Strodel and C. S. Whittleston, in *Proteins: Energy, Heat and Signal Flow*, ed. D. M. Leitner and J. E. Straub, CRC Press, Boca Raton, 2010, vol. 1, ch. 14, pp. 318–319.

[101] J. A. Joseph, C. S. Whittleston and D. J. Wales, *J. Chem. Theory Comput.*, 2016, **12**, 6109–6117.

[102] A. Kitao and N. Go, *Curr. Opin. Struct. Biol.*, 1999, **9**, 164–169.

[103] O. F. Lange and H. Grubmüller, *J. Phys. Chem. B*, 2006, **110**, 22842–22852.

[104] D. J. Jacobs, A. J. Rader, L. A. Kuhn and M. F. Thorpe, *Proteins: Struct., Funct., Bioinf.*, 2001, **44**, 150–165.

[105] M. F. Thorpe, M. Lei, A. J. Rader, D. J. Jacobs and L. A. Kuhn, *J. Mol. Graphics Modell.*, 2001, **19**, 60–69.

[106] D. J. Wales, *Phil. Trans. R. Soc. A*, 2005, **363**, 357–377.

[107] ROTATION, http://www.mech.utah.edu/~brannon/public/rotation.pdf, (accessed August 2017).

[108] D. Chakrabarti and D. J. Wales, *Phys. Chem. Chem. Phys.*, 2009, **11**, 1970–1976.

[109] B. Strodel and D. J. Wales, *Chem. Phys. Lett.*, 2008, **466**, 105–115.

[110] D. J. Wales, *Mol. Phys.*, 1993, **78**, 151–171.

[111] F. Calvo, D. Schebarchov and D. J. Wales, *J. Chem. Theory Comput.*, 2016, **12**, 902–909.

[112] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 2nd edn., 1992.

[113] K. Singh, *Linear Algebra: Step by Step*, Oxford University Press, Oxford, 2014.

[114] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The John Hopkins University Press, Baltimore, 3rd edn., 1996.

[115] N. J. Higham, *Wiley Interdiscip. Rev. Comput. Stat.*, 2009, **1**, 251–254.

[116] A. Krishnamoorthy and D. Menon, in *Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, IEEE, 2013, pp. 70–72.

[117] SuiteSparse, http://faculty.cse.tamu.edu/davis/suitesparse.html, (accessed August 2017).

[118] GPU Applications, http://www.nvidia.com/object/gpu-applications.html, (accessed August 2017).

[119] L. D'Amore, G. Laccetti, D. Romano, G. Scotti and A. Murli, *Int. J. Comput. Math.*, 2015, **92**, 59–76.

[120] Y. Fei, G. Rong, B. Wang and W. Wang, *Comput. Graph.*, 2014, **40**, 1–9.

[121] Q. Zhang, H. Bao, C. Rao and Z. Peng, *Opt. Rev.*, 2015, **22**, 741–752.

[122] J. Wetzl, O. Taubmann, S. Haase, T. Köhler, M. Kraus and J. Hornegger, in *Bildverarbeitung für die Medizin 2013*, ed. H.-P. Meinzer, M. T. Deserno, H. Handels and T. Tolxdorff, Springer, Berlin, 2013, pp. 21–26.

[123] S. Yatawatta, S. Kazemi and S. Zaroubi, in *2012 Innovative Parallel Computing (InPar)*, IEEE, 2012, pp. 1–6.

[124] B. Sukhwani and M. C. Herbordt, in *Numerical Computations with GPUs*, ed. V. Kindratenko, Springer International Publishing, Cham, 2014, ch. 18, pp. 379–405.

[125] M. Gates, M. T. Heath and J. Lambros, *Int. J. High Perform. Comput. Appl.*, 2015, **29**, 92–106.

[126] J. Gu, M. Zhu, Z. Zhou, F. Zhang, Z. Lin, Q. Zhang and M. Breternitz, in *APSys '14 Proceedings of 5th Asia-Pacific Workshop on Systems*, ACM, New York, 2014, pp. 1–7.

[127] G. Rong, Y. Liu, W. Wang, X. Yin, D. Gu and X. Guo, *IEEE Trans. Vis. Comput. Graphics*, 2011, **17**, 345–356.

[128] J. Martinez, F. Claux and S. Lefebvre, *Raster2Mesh: Rasterization based CVT meshing*, [Research Report] RR-8684, INRIA, Nancy, 2015.

[129] K. Röder and D. J. Wales, *J. Chem. Theory Comput.*, 2017, **13**, 1468–1477.

[130] J. A. Joseph, K. Röder, D. Chakraborty, R. G. Mantell and D. J. Wales, *Chem. Commun. (Cambridge, U. K.)*, 2017, **53**, 6974–6988.

[131] S. N. Fejer, R. G. Mantell and D. J. Wales, *Phys. Chem. Chem. Phys.*, submitted.

[132] J. Weidendorfer, M. Kowarschik and C. Trinitis, in *Computational Science - ICCS 2004*, ed. M. Bubak, G. D. van Albada, P. M. A. Sloot and J. Dongarra, Springer, Berlin, 2004, vol. 3038, pp. 440–447.

[133] D. De Sancho and R. B. Best, *J. Am. Chem. Soc.*, 2011, **133**, 6809–6816.

[134] cudaLBFGS, https://github.com/jwetzl/CudaLBFGS, (accessed October 2013).

[135] cuBLAS, https://developer.nvidia.com/cublas, (accessed August 2017).

[136] Y. Kumeda, D. J. Wales and L. J. Munro, *Chem. Phys. Lett.*, 2001, **341**, 185–194.

[137] J. Yu, L. Zhang, P. M. Hwang, K. W. Kinzler and B. Vogelstein, *Mol. Cell*, 2001, **7**, 673–682.

[138] C. L. Day, C. Smits, F. C. Fan, E. F. Lee, W. D. Fairlie and M. G. Hinds, *J. Mol. Biol.*, 2008, **380**, 958–971.

[139] AMBER 12 NVIDIA GPU ACCELERATION SUPPORT: Benchmarks, http://ambermd.org/gpus12/benchmarks.htm, (accessed August 2017).

[140] K. Mochizuki, C. S. Whittleston, S. Somani, H. Kusumaatmaja and D. J. Wales, *Phys. Chem. Chem. Phys.*, 2014, **16**, 2842–2853.

[141] C. D. Bahl, K. L. Hvorecny, A. A. Bridges, A. E. Ballok, J. M. Bomberger, K. C. Cady, G. A. O'Toole and D. R. Madden, *J. Biol. Chem.*, 2014, **289**, 7460–7469.

[142] S. Chutinimitkul, S. Herfst, J. Steel, A. C. Lowen, J. Ye, D. v. Riel, E. J. A. Schrauwen, T. M. Bestebroer, B. Koel, D. F. Burke, K. H. Sutherland-Cash, C. S. Whittleston, C. A. Russell, D. J. Wales, D. J. Smith, M. Jonges, A. Meijer, M. Koopmans, G. F. Rimmelzwaan, T. Kuiken, A. D. M. E. Osterhaus, A. García-Sastre, D. R. Perez and R. A. M. Fouchier, *J. Virol.*, 2010, **84**, 11802–11813.

[143] C. Shang, C. S. Whittleston, K. H. Sutherland-Cash and D. J. Wales, *J. Chem. Theory Comput.*, 2015, **11**, 2307–2314.

[144] C. S. Whittleston, unpublished work.

[145] CUDA Toolkit 6.5, https://developer.nvidia.com/cuda-toolkit-65, (accessed August 2017).

[146] D. J. Wales and T. R. Walsh, *J. Chem. Phys.*, 1996, **105**, 6957–6971.

[147] P. B. Harbury, T. Zhang, P. S. Kim and T. Alber, *Science*, 1993, **262**, 1401–1407.

[148] M. K. Yadav, L. J. Leman, D. J. Price, C. L. Brooks III, C. D. Stout and M. R. Ghadiri, *Biochemistry*, 2006, **45**, 4463–4473.

[149] J. M. Carr and D. J. Wales, *J. Chem. Phys.*, 2005, **123**, 234901.

[150] B. Strodel, C. S. Whittleston and D. J. Wales, *J. Am. Chem. Soc.*, 2007, **129**, 16005–16014.

[151] O. M. Becker and M. Karplus, *J. Chem. Phys.*, 1997, **106**, 1495–1517.

[152] D. J. Wales, M. A. Miller and T. R. Walsh, *Nature*, 1998, **394**, 758–760.

[153] D. J. Wales, *J. Chem. Phys.*, 2017, **146**, 054306.

[154] A. L. Mackay, *Acta Crystallogr.*, 1962, **15**, 916–918.

[155] G. M. Torrie and J. P. Valleau, *J. Comp. Phys.*, 1977, **23**, 187–199.

[156] A. P. Lyubartsev, A. A. Martsinovski, S. V. Shevkunov and P. N. Vorontsov-Velyaminov, *J. Chem. Phys.*, 1992, **96**, 1776–1783.

[157] B. A. Berg and T. Neuhaus, *Phys. Rev. Lett.*, 1992, **68**, 9–12.

[158] F. Wang and D. P. Landau, *Phys. Rev. Lett.*, 2001, **86**, 2050–2053.

[159] A. Laio and M. Parrinello, *PNAS*, 2002, **99**, 12562–12566.

[160] S. V. Krivov and M. Karplus, *J. Chem. Phys.*, 2002, **117**, 10894–10903.

[161] D. Gfeller, P. De Los Rios, A. Caflisch and F. Rao, *Proc. Natl. Acad. Sci. U. S. A.*, 2007, **104**, 1817–1822.

[162] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 3rd edn., 1999.

[163] I. Georgescu and V. A. Mandelshtam, *J. Chem. Phys.*, 2012, **137**, 144106.

[164] D. A. Case, T. A. Darden, T. E. Cheatham III, C. L. Simmerling, J. Wang, R. E. Duke, R. Luo, R. C. Walker, W. Zhang, K. M. Merz, B. Roberts, S. Hayik, A. Roitberg, G. Seabra, J. Swails, A. W. Goetz, I. Kolossvai, K. F. Wong, F. Paesani, J. Vanicek, R. M. Wolf, J. Liu, X. Wu, S. R. Brozell, T. Steinbrecher, H. Gohlke, Q. Cai, X. Ye, J. Wang, M. J. Hsieh, G. Cui, D. R. Roe, D. H. Mathews, M. G. Seetin, R. Salomon-Ferrer, C. Sagui, V. Babin, T. Luchko, S. Gusarov, A. Kovalenko and P. A. Kollman, AMBER 12, University of California, San Francisco, 2012.

[165] M. Page and J. W. McIver, *J. Chem. Phys.*, 1988, **88**, 922–935.

[166] Y. Chen, T. A. Davis, W. W. Hager and S. Rajamanickam, *ACM Trans. Math. Softw.*, 2008, **35**, 22.

[167] suitesparse-metis-for-windows, https://github.com/jlblancoc/suitesparse-metis-for-windows, (accessed August 2017).

[168] R. G. Mantell, C. E. Pitt and D. J. Wales, *J. Chem. Theory Comput.*, 2016, **12**, 6182–6191.

[169] S. C. Rennich, D. Stosic and T. A. Davis, *Parallel Comput.*, 2016, **59**, 140–150.

[170] Z. Antal, J. Szoverfi and S. N. Fejer, *J. Chem. Inf. Model.*, 2017, **57**, 910–917.

[171] J. A. Speir, B. Bothner, C. Qu, D. A. Willits, M. J. Young and J. E. Johnson, *J. Virol.*, 2006, **80**, 3582–3591.

[172] K. Röder and D. J. Wales, unpublished work.

[173] K. Röder and D. J. Wales, unpublished work.

[174] T. Lührs, C. Ritter, M. Adrian, D. Riek-Loher, B. Bohrmann, H. Döbeli, D. Schubert and R. Riek, *Proc. Natl. Acad. Sci. U. S. A.*, 2005, **102**, 17342–17347.

[175] J. A. Joseph and D. J. Wales, unpublished work.

[176] G. A. Belogurov, M. N. Vassylyeva, V. Svetlov, S. Klyuyev, N. V. Grishin, D. Vassylyev and I. Artsimovitch, *Molecular Cell*, 2007, **26**, 117–129.

[177] B. Burmann, S. Knauer, A. Sevostyanova, K. Schweimer, R. Mooney, R. Landick, I. Artsimovitch and P. Rösch, *Cell*, 2012, **150**, 291–303.

[178] C.-H. Yun, T. J. Boggon, Y. Li, M. S. Woo, H. Greulich, M. Meyerson and M. J. Eck, *Cancer Cell*, 2007, **11**, 217–227.

[179] Y. Kawakita, M. Seto, T. Ohashi, T. Tamura, T. Yusa, H. Miki, H. Iwata, H. Kamiguchi, T. Tanaka, S. Sogabe, Y. Ohta and T. Ishikawa, *Bioorg. Med. Chem.*, 2013, **21**, 2250–2261.

[180] A. Laganowsky, C. Liu, M. R. Sawaya, J. P. Whitelegge, J. Park, M. Zhao, A. Pensalfini, A. B. Soriaga, M. Landau, P. K. Teng, D. Cascio, C. Glabe and D. Eisenberg, *Science*, 2012, **335**, 1228–1231.

[181] S. W. Cowan-Jacob, G. Fendrich, P. W. Manley, W. Jahnke, D. Fabbro, J. Liebetanz and T. Meyer, *Structure*, 2005, **13**, 861–871.

[182] W. Xu, A. Doshi, M. Lei, M. J. Eck and S. C. Harrison, *Mol. Cell*, 1999, **3**, 629–638.

[183] C. W. Müller, G. J. Schlauderer, J. Reinstein and G. E. Schulz, *Structure*, 1996, **4**, 147–156.

[184] M. B. Berry, B. Meador, T. Bilderback, P. Liang, M. Glaser and G. N. Phillips, *Proteins: Struct., Funct., Bioinf.*, 1994, **19**, 183–198.

[185] D. Martinez-Zapien, P. Legrand, A. G. McEwen, F. Proux, T. Cragnolini, S. Pasquali and A.-C. Dock-Bregeon, *Nucleic Acids Res.*, 2017, **45**, 3568–3579.

[186] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler and P. Gumbsch, *Phys. Rev. Lett.*, 2006, **97**, 170201.

[187] F. Noé, M. Oswald, G. Reinelt, S. Fischer and J. Smith, *Multiscale Model. Simul.*, 2006, **5**, 393–419.

[188] F. Noé, D. Krachtus, J. C. Smith and S. Fischer, *J. Chem. Theory Comput.*, 2006, **2**, 840–857.

[189] D. J. Wales and J. M. Carr, *J. Chem. Theory Comput.*, 2012, **8**, 5020–5034.

[190] R. A. Brown and D. A. Case, *J. Comput. Chem.*, 2006, **27**, 1662–1675.

Appendix A

Shifting the Hessian eigenvalues

The translational and rotational eigenvalues must be shifted from zero to one at each local minimum of interest to obtain the product of positive Hessian eigenvalues. To do this, we must find normal mode vectors for translation and rotation, where the molecule is aligned so that the principal axes and the centre of mass coincide with the fixed axis system and the origin. The corresponding Hessian eigenvectors are then orthogonal, as detailed below.

Let $X_{\alpha x}$ be the x coordinate of atom α, etc. A rotation of the position vector of atom α, $\mathbf{X}_\alpha^0$, through angle θ about an axis defined by the unit vector $\hat{\mathbf{n}}$ is given by

$$\mathbf{X}_\alpha = \mathbf{X}_\alpha^0 \cos\theta + \hat{\mathbf{n}}(\hat{\mathbf{n}}\cdot\mathbf{X}_\alpha^0)(1-\cos\theta) + \mathbf{X}_\alpha^0 \wedge \hat{\mathbf{n}} \sin\theta. \tag{A.1}$$

Taking a Taylor series about $\theta = 0$, the displacement vector for this rotation is

$$\boldsymbol{\Delta}_\alpha = \mathbf{X}_\alpha - \mathbf{X}_\alpha^0 = \theta \begin{pmatrix} \hat{n}_z X_{\alpha y}^0 - \hat{n}_y X_{\alpha z}^0 \\ \hat{n}_x X_{\alpha z}^0 - \hat{n}_z X_{\alpha x}^0 \\ \hat{n}_y X_{\alpha x}^0 - \hat{n}_x X_{\alpha y}^0 \end{pmatrix} + \frac{1}{2}\theta^2 \begin{pmatrix} -X_{\alpha x}^0 + \hat{n}_x^2 X_{\alpha x}^0 + \hat{n}_x(\hat{n}_y X_{\alpha y}^0 + \hat{n}_z X_{\alpha z}^0) \\ -X_{\alpha y}^0 + \hat{n}_y^2 X_{\alpha y}^0 + \hat{n}_y(\hat{n}_x X_{\alpha x}^0 + \hat{n}_z X_{\alpha z}^0) \\ -X_{\alpha z}^0 + \hat{n}_z^2 X_{\alpha z}^0 + \hat{n}_z(\hat{n}_x X_{\alpha x}^0 + \hat{n}_y X_{\alpha y}^0) \end{pmatrix} \tag{A.2}$$

A Taylor expansion of the potential energy gives

$$V(\mathbf{X}^0) = V(\mathbf{X}^0 + \boldsymbol{\Delta}) = V(\mathbf{X}^0) + \nabla V(\mathbf{X}^0)^T \boldsymbol{\Delta} + \frac{1}{2}\boldsymbol{\Delta}^T H(\mathbf{X}^0)\boldsymbol{\Delta} + \ldots, \tag{A.3}$$

where $\boldsymbol{\Delta} = \{\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2, \ldots, \boldsymbol{\Delta}_N\}^T$, $\nabla V(\mathbf{X}^0) = \partial V(\mathbf{X}^0)/\partial X_\alpha$, and $H_{\alpha\beta}(\mathbf{X}^0) = \partial^2 V(\mathbf{X}^0)/\partial X_\alpha \partial X_\beta$. The potential energy of an isolated molecule must be invariant to rotation, so the terms in $\boldsymbol{\Delta}$ must be equal to zero. At a stationary point, $\nabla V(\mathbf{X}^0) = \mathbf{0}$ and equating terms in θ^2 to zero gives

$$H(\mathbf{X}^0)\boldsymbol{\Delta} = \begin{pmatrix} \hat{n}_z X^0_{1y} - \hat{n}_y X^0_{1z} \\ \hat{n}_x X^0_{1z} - \hat{n}_z X^0_{1x} \\ \hat{n}_y X^0_{1x} - \hat{n}_x X^0_{1y} \\ \vdots \end{pmatrix} = \mathbf{0}. \tag{A.4}$$

This result implies that $\boldsymbol{\Delta}$ is an eigenvector of H with eigenvalue zero. We require eigenvectors of the mass-weighted Hessian corresponding to mass-weighted coordinates, $\mathbf{Q}_X = \mathbf{X}_\alpha \sqrt{m_\alpha}$, where m_α is the mass of atom α, and the corresponding components are

$$\begin{pmatrix} \sqrt{m_1}(\hat{n}_z X^0_{1y} - \hat{n}_y X^0_{1z}) \\ \sqrt{m_1}(\hat{n}_x X^0_{1z} - \hat{n}_z X^0_{1x}) \\ \sqrt{m_1}(\hat{n}_y X^0_{1x} - \hat{n}_x X^0_{1y}) \\ \vdots \end{pmatrix}. \tag{A.5}$$

We can prove that the rotational eigenvectors are orthogonal if we choose rotations about directions corresponding to eigenvectors of the moment of inertia tensor:

$$\tilde{I} = \sum_\alpha m_\alpha \begin{pmatrix} (X^0_{\alpha y})^2 + (X^0_{\alpha z})^2 & -X^0_{\alpha x} X^0_{\alpha y} & -X^0_{\alpha x} X^0_{\alpha z} \\ -X^0_{\alpha y} X^0_{\alpha x} & (X^0_{\alpha x})^2 + (X^0_{\alpha z})^2 & -X^0_{\alpha y} X^0_{\alpha z} \\ -X^0_{\alpha z} X^0_{\alpha x} & -X^0_{\alpha z} X^0_{\alpha y} & (X^0_{\alpha x})^2 + (X^0_{\alpha y})^2 \end{pmatrix}. \tag{A.6}$$

Using equation (A.5) with two eigenvectors of $\tilde{I}$ corresponding to $\tilde{I}\hat{\mathbf{e}}_a = \lambda_a \hat{\mathbf{e}}_a$ and $\tilde{I}\hat{\mathbf{e}}_b = \lambda_b \hat{\mathbf{e}}_b$, replacing $\hat{\mathbf{n}}$, we can construct the dot product as:

$$\begin{aligned} &\sum_\alpha m_\alpha \big[(\hat{e}_{az} X^0_{\alpha y} - \hat{e}_{ay} X^0_{\alpha z})(\hat{e}_{bz} X^0_{\alpha y} - \hat{e}_{by} X^0_{\alpha z}) + (\hat{e}_{ax} X^0_{\alpha z} - \hat{e}_{az} X^0_{\alpha x})(\hat{e}_{bx} X^0_{\alpha z} - \hat{e}_{bz} X^0_{\alpha x}) \\ &\qquad + (\hat{e}_{ay} X^0_{\alpha x} - \hat{e}_{ax} X^0_{\alpha y})(\hat{e}_{by} X^0_{\alpha x} - \hat{e}_{bx} X^0_{\alpha y}) \big] \\ &= \sum_\alpha m_\alpha \big[\hat{e}_{ax}\hat{e}_{bx} \left\{ (X^0_{\alpha z})^2 + (X^0_{\alpha y})^2 \right\} - \hat{e}_{by}\hat{e}_{ax} X^0_{\alpha x} X^0_{\alpha y} - \hat{e}_{bz}\hat{e}_{ax} X^0_{\alpha x} X^0_{\alpha z} \\ &\qquad + \hat{e}_{ay}\hat{e}_{by} \left\{ (X^0_{\alpha z})^2 + (X^0_{\alpha x})^2 \right\} - \hat{e}_{bz}\hat{e}_{ay} X^0_{\alpha y} X^0_{\alpha z} - \hat{e}_{bx}\hat{e}_{ay} X^0_{\alpha y} X^0_{\alpha x} \\ &\qquad + \hat{e}_{az}\hat{e}_{bz} \left\{ (X^0_{\alpha x})^2 + (X^0_{\alpha y})^2 \right\} - \hat{e}_{bx}\hat{e}_{az} X^0_{\alpha z} X^0_{\alpha x} - \hat{e}_{by}\hat{e}_{az} X^0_{\alpha z} X^0_{\alpha y} \big] \\ &= \hat{\mathbf{e}}_a^T \tilde{I} \hat{\mathbf{e}}_b = \hat{\mathbf{e}}_a^T \lambda_b \hat{\mathbf{e}}_b = \lambda_b \hat{\mathbf{e}}_a^T \hat{\mathbf{e}}_b = 0. \end{aligned} \tag{A.7}$$

The dot product of an infinitesimal rotation and a translation corresponding to axes defined by eigenvectors $\hat{\mathbf{e}}_a$ and $\hat{\mathbf{e}}_b$ of $\tilde{I}$ is

$$\begin{aligned}
&\sum_\alpha m_\alpha [(\hat{e}_{az}X^0_{\alpha y} - \hat{e}_{ay}X^0_{\alpha z})\hat{e}_{bx} + (\hat{e}_{ax}X^0_{\alpha z} - \hat{e}_{az}X^0_{\alpha x})\hat{e}_{by} + (\hat{e}_{ay}X^0_{\alpha x} - \hat{e}_{ax}X^0_{\alpha y})\hat{e}_{bz}] \\
=&(\hat{e}_{ay}\hat{e}_{bz} - \hat{e}_{az}\hat{e}_{by})\sum_\alpha m_\alpha X^0_{\alpha x} + (\hat{e}_{az}\hat{e}_{bx} - \hat{e}_{ax}\hat{e}_{bz})\sum_\alpha m_\alpha X^0_{\alpha y} + (\hat{e}_{ax}\hat{e}_{by} - \hat{e}_{ay}\hat{e}_{bx})\sum_\alpha m_\alpha X^0_{\alpha z} \\
=&(\hat{e}_a \wedge \hat{e}_b)\cdot \sum_\alpha m_\alpha \mathbf{X}^0 = 0, \qquad \text{(A.8)}
\end{aligned}$$

when the centre of mass is at the origin, because $\sum_\alpha m_\alpha \mathbf{X}^0 = 0$. Hence the Hessian eigenvectors corresponding to infinitesimal rotations and translations are orthogonal if the system centre of mass is at the origin and the system is oriented so that the principal axes of the inertia tensor are aligned with the global frame. This alignment enables us to shift the six corresponding eigenvalues to unity individually.