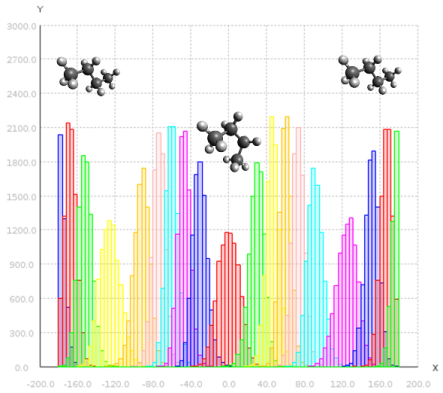
A Parallel Multidimensional Weighted Histogram Analysis Method

Andrew Potgieter

Minor Dissertation presented in partial fulfilment
of the requirements for the degree of
Masters of Information Technology

Supervisor: Michelle Kuttel
Department of Computer Science
University of Cape Town



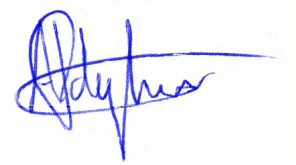
October 11, 2014

Abstract

The Weighted Histogram Analysis Method (WHAM) is a technique used to calculate free energy from molecular simulation data. WHAM recombines biased distributions of samples from multiple Umbrella Sampling simulations to yield an estimate of the global unbiased distribution. The WHAM algorithm iterates two coupled, non-linear, equations, until convergence at an acceptable level of accuracy. The equations have quadratic time complexity for a single reaction coordinate. However, this increases exponentially with the number of reaction coordinates under investigation, which makes multidimensional WHAM a computationally expensive procedure. There is potential to use general purpose graphics processing units (GPGPU) to accelerate the execution of the algorithm. Here we develop and evaluate a multidimensional GPGPU WHAM implementation to investigate the potential speed-up attained over its CPU counterpart. In addition, to avoid the cost of multiple Molecular Dynamics simulations and for validation of the implementations we develop a test system to generate samples analogous to Umbrella Sampling simulations. We observe a maximum problem size dependent speed-up of approximately 19× for the GPGPU optimized WHAM implementation over our single threaded CPU optimized version. We find that the WHAM algorithm is amenable to GPU acceleration, which provides the means to study ever more complex molecular systems in reduced time periods.

Plagarism Declaration

'I know the meaning of plagiarism and declare
that all of the work in the dissertation, save for
that which is properly acknowledged, is my own'.



Acknowledgements

I would like to thank two friends, Andrew and Theresa, who sacrificed their weekend for last minute proofreading. I would also like to express my deepest gratitude to my supervisor Michelle Kuttel for her guidance, patience and extensive support throughout this entire process. Her commentary and ideas helped improve my work far beyond what I could ever have hoped to achieve alone.

Contents

Chapter 1

Introduction

The Weighted Histogram Analysis Method (WHAM)[1] is an algorithmic technique used to solve statistical problems in computational chemistry and biochemistry. In conjunction with Molecular Dynamics, it has applications in computer-aided drug discovery[2] and is also widely used in studies of protein folding[3]. Molecular Dynamics simulates the microscopic movement, position and structure of molecules to provide insights into molecular behaviour and interactions. Simulations generate data on the molecular scale that experiments often cannot provide, such as molecular free energy. Free energy is of primary interest in molecular research since it contributes to the understanding of phenomena such as the detailed mechanism of the binding of a drug into a receptor molecule, or the preferred conformations of complex molecules such as proteins. These phenomena can be explained by analysis of the Potential of Mean Force (PMF)[4], which describes the free energy of a molecule with respect to either an external parameter or an internal reaction coordinate. However, basic Molecular Dynamics simulations are often unable to sample the full reaction coordinate space in a feasible amount of time, due to high energy barriers between molecular conformations. The quantity of samples taken at these energy barriers is often insufficient to obtain acceptable sample distributions. Efficient calculation of free energy requires special techniques to enhance sampling rates across these energy barriers. One of the most commonly employed techniques is Umbrella Sampling[5] simulations, which bias the simulations to restrict samples to a narrow window or umbrella region of the PMF. Multiple sample

distributions, which span the entire PMF, can then be obtained from a number of Umbrella Sampling simulations. WHAM combines the sample distributions and removes the bias to yield a distribution estimate from which the PMF can be computed across the entire coordinate space. It is a simple and elegant solution in which all simulations contribute.

Implementations of WHAM currently exist as part of the CHARMM[6] and GROMACS[7] Molecular Dynamics packages, and as the popular standalone Grossfield package[8] which supports one- and two-dimensional conformational space. Multiple reaction coordinates are supported by additional WHAM scripts packaged with the SMOG server[9], a web-based simulation tool, however this is not optimized for high performance. Improved performance can be achieved with the Grossfield package since recent enhancements of the two-dimensional script have dramatically reduced the run-time. A high performance one-dimensional WHAM implementation, OPT-MHM[10], for Hamiltonian Replica Exchange, uses the OpenMP API specification for parallel execution. However, there are few objective validations or evaluations of the current WHAM implementations since the majority have not prioritised optimization. Currently, a need exists to explore higher dimensional reaction coordinate spaces as this permits investigation of the conformational dynamics of larger and more complex molecules. A prerequisite for these investigations is a high performance, multidimensional implementation of WHAM, and at present no implementations satisfy these requirements.

1.1 Motivation

There is potential to exploit parallelism to improve on the performance of current WHAM implementations. At the heart of WHAM are two equations which are repeatedly evaluated until convergence. Each evaluation of the WHAM equations has quadratic time complexity, which makes it computationally expensive. Unfortunately, quadratic time limits both the size and complexity of problems that can be attempted, despite the high speeds of modern Central Processing Units (CPUs). The quadratic time limitation presents opportunity to investigate the feasibility of a parallel implementation of WHAM and specifically implementing WHAM on graphics processing units (GPUs).

Over the last decade, performance improvements of single-core CPUs have substantially reduced due to physical limitations such as power consumption and heat dissipation. Processor manufacturers turned towards the production of multi-core CPUs which initiated a shift towards multi-core and parallel application development. Even before this shift, the Molecular Dynamics community had been at the forefront of parallel application development due to the magnitude of Molecular Dynamics simulations (such as the recent 64 million atom simulation of the HIV-1 capsid[11]) which often demand the use of clusters and supercomputers.

As the computational requirements of scientific communities grew, experimentation began on the use of graphics languages to implement heavily computational algorithms on GPUs. GPUs, designed to take on the high computational demands of graphics processing, have since started to provide Application Programming Interfaces (APIs) for general purpose computing on GPUs (GPGPU). These APIs extend GPUs to allow the execution of non-graphical calculations in a highly parallel hardware environment. The rapid maturation of GPGPU and the relentless demand for computational power has fuelled significant advancements in research and development of GPU-accelerated Molecular Dynamics[12, 13]. Most of the development focus is directed towards Molecular Dynamics programs since they form the primary application domain. Until now there has been little attention devoted to parallel and GPGPU implementations of tools such as WHAM, but this requirement will arise simultaneously with the advancement of the simulation programs. GPGPU substantially increases the speed of computation for certain classes of problem[14], and WHAM is one of these problems - it is compute intensive, data parallel and there is no branching or recursion.

Development of a multidimensional GPGPU implementation of WHAM requires a benchmark implementation to assess the performance gains. Since the current implementations are not optimized for performance, an optimized sequential version will need to be developed to provide realistic benchmark times. Test data, comprising samples from Umbrella Sampling simulations, run across the entire range of reaction coordinate space, are required to validate the sequential and parallel implementations. Typically, the test data required can be generated with standard Molecular Dynamics packages, but this is impractical for two reasons: primarily because validation requires a known reference result to validate against,

but also because large multidimensional test cases would require an infeasible amount of time to generate. GPGPU algorithm speeds vary with problem size so satisfactory comparison of run-times requires a varied number of large test cases, therefore an alternative to Molecular Dynamics is required to efficiently generate data for the test cases.

1.2 Aims and Objectives

The objective is to develop a fast and correct version of WHAM on GPGPU, with a notable speed-up over the optimized CPU version. The GPGPU WHAM implementation must support multidimensional data generated from sampling of any number of reaction coordinates and must produce verifiable and accurate estimates of the Potential of Mean Force (PMF).

1.3 Approach

Development of a test environment was the initial objective. To efficiently validate the algorithm, when a WHAM implementation is executed, it must yield a known reproducible result. Evaluation is impractically time consuming with long running Umbrella Sampling simulations and even then validation is challenging since the PMF is unknown. Hence, a test harness, a system to generate test data from a known analytic reference PMF, was developed to allow for rapid and reproducible testing and validation of WHAM implementations. To validate WHAM on both small, one-dimensional data sets and large, multidimensional data, the number of dimensions and size of the data is configurable on the test harness which then generates data efficiently. A complimentary system was also developed to validate the PMF estimate generated by the implementation by measuring the output deviation from the known analytic reference PMF.

For performance comparison, two WHAM implementations were developed: a sequential, single threaded CPU optimized version, to generate benchmark times for the run-time evaluation, and the parallel GPGPU optimized version. Both versions were executed on, and validated against, single and multidimensional data.

The evaluation covered multiple test cases which ranged in size from moderate to the largest possible on the GPU. All test cases were executed on both CPU and GPGPU implementations, each of which logged run-times once the algorithm concluded. The run-times were used to calculate the speed of each implementation and hence the speed-up for each test case. The deviation of each test case's PMF estimates compared to the known PMF ensured the successful execution of all test cases.

1.3.1 Test Harness

WHAM enables us to compute an unknown PMF from samples generated by Umbrella Sampling, but from a test perspective it is possible to mimic the reverse of this process. Given a known, analytically defined PMF function, samples can be efficiently generated using a Monte Carlo approach. These samples can then be used for validation, since they originate from a known PMF. The test harness was initially developed to generate one-dimensional data but then later modified to produce data from a PMF function with any number of dimensions. The dimensions are defined by the number of parameters of the PMF function. The validation system (also multidimensional) was developed to measure the deviation of the WHAM estimated PMF from the known reference PMF function. Once both modules (data generation and validation) of the test harness were developed, the test harness itself was validated. This was performed by execution of the Grossfield implementation[8] (it is assumed this functions correctly) with a test harness generated data set, followed by visual validation of the Grossfield estimated PMF.

1.3.2 Implementations

A phased approach was taken, where a CPU version was designed, developed and tested. Common elements, such as the creation of distributions, were extracted and caching methods were developed. This was followed by optimization of the CPU version. Next the GPGPU version was developed. Initially, the entire algorithm executed on the GPU but this was only possible with small data sets. The execution of larger data sets was made possible by a hybrid implementation where a portion of the processing, such as the convergence check, was performed

on the CPU, and the majority on the GPU. Once the GPGPU design was fixed, optimization was performed in small iterations. If a change increased performance it was retained, otherwise the change was rolled back and another optimization attempted.

1.3.3 Evaluation

The performance gains achieved by porting applications from CPU to GPU architectures are typically measured in terms of speed-up - the ratio of the CPU run-time to the GPGPU run-time for a comparable problem size. To evaluate the run-time reduction, both implementations were executed on twenty two-dimensional test cases, of varying problem size, generated with the test harness which allowed us to assess the performance with regards to problem size. Each implementation logs its run-time upon completion, which is used for evaluation by comparison of the speed (time / number of elements) and speed-up. Additionally, the test harness validated all test case runs and measured the deviation of the computed PMF against the analytic PMF to record the variation of accuracy against problem size.

1.4 Contribution

The primary contribution of this work is a high performance parallel version of WHAM which can support large datasets and high dimensionality. This allows greater and more complex molecular exploration and specifically an increase in the number of reaction coordinates that can be investigated.

The second contribution is the multidimensional test harness, which is responsible for initial generation of samples from an analytically defined PMF function and the validation of WHAM PMF estimates. This can be used not only to validate our WHAM implementations, but also other WHAM implementations and other free energy measurement techniques.

1.5 Thesis Overview

Chapters 2 through 4 provide the background necessary to understand WHAM as it is applied in this evaluation. Chapter 2 contains a brief introduction to free energy and Molecular Dynamics with emphasis on the problem of Boltzmann sampling and the solution of Umbrella Sampling. This is followed by a description of the WHAM equations and algorithm (Chapter 3) and concluded with a brief overview of the CUDA architectural concepts relevant to the WHAM GPGPU implementation (Chapter 4).

Chapter 5 outlines the methods developed to validate and evaluate the CPU and GPGPU WHAM implementations. An overview and details of the test harness and the WHAM modules are described first. These sections are followed by a detailed description of the GPGPU design and implementation with a discussion of the optimization techniques employed. This section concludes with a discussion on the test cases developed to evaluate the performance of the CPU and GPGPU WHAM implementations.

Chapter 6 reports on the validation of the test harness and the WHAM implementations and presents the results of the evaluation of the WHAM implementations with size dependent speed-ups attained on the test cases. Finally, Chapter 7 contains conclusions and suggests future work.

Chapter 2

Free Energy Measurements with Molecular Dynamics

The driving force behind chemical reactions can be understood in terms of a thermodynamic quantity known as free energy. Free energy represents the energy available to produce changes in chemical substances. It explains why, in some circumstances, chemical reactions release energy in the form of heat, and in other cases, reactions occur when provided with energy. It also explains why some compounds react with each other the moment they come in contact and why some compounds require a boost of activation energy before the reaction begins. When these chemical reactions occur, it is the difference in free energy between the reactants and products which determines if energy is released or absorbed. This suggests that knowledge of free energies of chemical substances can assist in the prediction of chemical reactions - whether energy is released or absorbed, and if so, how much and at what rate.

2.1 Illustrative Example: Butane

A simple example, used throughout this document, of a method to measure free energy is the application of Molecular Dynamics to study the conformations of butane (Fig. 2.1), specifically the energy changes as a function of rotation about the central bond ($C2 - C3$). Butane has three favourable conformations which min-

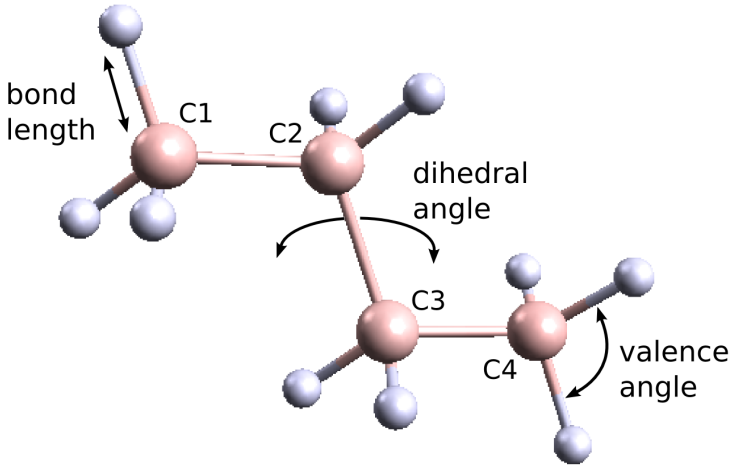


Figure 2.1: The molecular geometry of butane ($CH_3CH_2CH_2CH_3$). The three reaction coordinates: the bond length, the valence angle and the dihedral angle are illustrated with the "ball and spring" model. An example of the bond length is the distance between the larger C1 carbon atom and a smaller hydrogen atom. A valence angle is shown between two hydrogen atoms bonded to the C4 carbon. Finally a dihedral angle is shown as the angle of rotation, or twist, of the bond between the C2 and C3 carbon atoms.

imize steric clashes between atoms, and thus represent local free energy minima. These conformations are defined by the molecular geometry. Molecular geometry, the arrangement of atoms in a molecule, constitutes the three-dimensional spatial positions of the atoms. This can be specified in terms of both the angles and lengths of the bonds between atoms in a molecule. These values, collectively known as reaction coordinates, form the coordinate system which describe molecular geometry and comprise bond lengths, valence angles and dihedral angles. Fig. 2.1 illustrates the reaction coordinates of the butane molecule. Butane has four carbon atoms (C_1, C_2, C_3, C_4) surrounded by hydrogen atoms. The bond length is the distance between the nuclei of two adjoining atoms, the example indicated is the length of the C_1 carbon atom to one of its bonded hydrogen atoms. The valence angle is the angle formed between two bonds connecting three atoms, illustrated by the angle between the bonds joining carbon atom C_4 and two of its bonded hydrogen atoms. The dihedral angle is the angle of rotation, or twist, around the middle of three bonds connecting four atoms. This is indicated in the centre of the example

by the $C_2 - C_3$ dihedral angle, which is one of the principle reaction coordinates of butane. It can also be conceptualised as the angle between two planes. Planes can be defined by three points in three-dimensional space, so the $C_2 - C_3$ dihedral angle is the angle between the plane formed by atoms $C_1 - C_2 - C_3$ and the plane formed by atoms $C_2 - C_3 - C_4$. These planes intersect along the line formed by the $C_2 - C_3$ bond.

To understand the relation between free energy and molecular geometry it is helpful to conceptualize the atoms and bonds of a molecule as a system made up of spheres loosely connected by springs in constant motion. The spheres repel each other but bonded spheres are kept linked at an equilibrium bond length by the springs. In a three sphere chain, the two spheres at either end repel each other and in the absence of any other forces would form a valence angle between the two bonds of 180°. However, all spheres exert forces on all other spheres, so any other spheres connected to the system would exert forces that decrease the valence angle to below 180°. The same complex set of forces defines the equilibrium values of dihedral angles which result when the forces acting on the particles are balanced. This system of bond angles and lengths is not rigid, but constantly oscillates around the equilibrium geometry since particles continuously move and collide with each other. Another molecule, for example, could collide with a hydrogen atom and reduce the valence angle between a neighbouring hydrogen atom, thus transferring kinetic energy into free energy. At equilibrium, free energy is at its lowest, and increases as the bond angles and lengths move away from equilibrium. Equilibrium geometry represents a local energy minimum, but this is not necessarily the global energy minimum, since there can be a number of equilibrium geometries where all the forces are balanced. Equilibrium geometries at local energy minima are know as conformers, of which there can be more than one for a particular molecule. Butane has three local energy minima conformers which result from the periodic rotation around the $C_2 - C_3$ bond. These are shown in Fig 2.2a: two gauche($\pm 60°$) and an anti($\pm 180°$ - the same point in a periodic rotation) conformers. If conceptualised in this simplistic "ball and spring" manner, the energy stored in the repulsive forces of the spheres and the elastic forces of the springs is analogous to free energy.

However, energies of molecular systems are more complex than the "ball and spring" model and relate to a number of forces. Not only do the bonds contribute

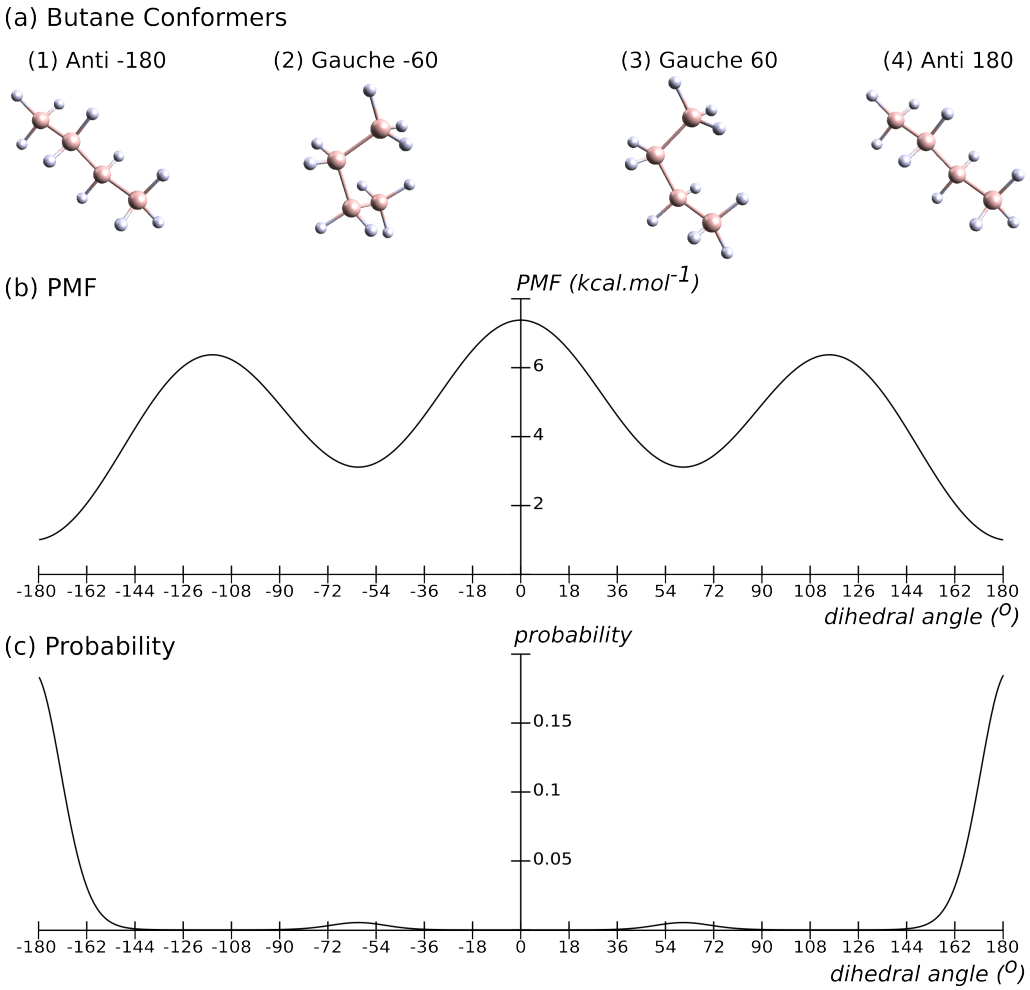


Figure 2.2: Butane: conformers, PMF and probability. Butane has three conformers (a) which result from the rotation around the $C_2 - C_3$ bond: two gauche($\pm 60°$) and one anti($\pm 180°$). The related PMF (b), the variation in the energy landscape between these three conformers, is shown below on the dihedral angle axis. This illustrates the local energy minima of the conformers. Below this, the probability distribution (c) of the states of butane - this describes the chances of finding butane at one of these local energy minima, the anti conformer being the most likely.

energies, but forces arise from other phenomenon such as van der Waals and electrostatic forces. The Potential of Mean Force (PMF) relates the energy contained in the system to the average of these forces and if plotted against a reaction coordinate demonstrates how free energy varies as a function of molecular geometry.

Fig. 2.2b is a plot of the PMF of butane against the rotation of the $C_2 - C_3$ dihedral angle from $-180°$ to $+180°$. It shows the energy minima at $\pm 180°$ and $\pm 60°$ which correspond to the anti and gauche equilibrium conformers. It also shows the energy barriers between these conformers - the peaks of energy at $\pm 120°$ and $0°$ - energy barriers which would need to be overcome to change from one equilibrium conformation to the other.

2.2 Molecular Dynamics

Over the course of the last century a number of innovative data analysis techniques have been developed to estimate free energy differences based on data from Molecular Dynamics. Molecular Dynamics simulates the structure, movement and interactions of molecules at a microscopic level based on three-dimensional mathematical models of the atoms and bonds. These models incorporate data on the position and velocity in three-dimensional space of all the atoms, and the lengths and strengths of the bonds between these atoms. In this model, classical Newtonian physical theories are employed to simulate structural dynamics by the calculation of the forces acting on the atoms and their subsequent change in velocities. The mechanics of the molecule can also be described in the context of Hamiltonian mechanics by a potential energy function. The potential function evaluates to the potential energy of the molecule as the sum of bonded (bond length, valence and dihedral angles) and non-bonded (electrostatic forces and the van der Waals forces) energy contributions.

$$E_{total} = E_{bond} + E_{valence} + E_{dihedral} + E_{electrostatic} + E_{vanderWaals} \tag{2.1}$$

The form of the mathematical expressions used for each of these potential energy terms constitutes a force-field, examples of which include CHARMM[6] and GROMACS[15]. Force-fields often model bond lengths with Hookes law, electrostatic energies with Coulombs law and van der Waals forces with Lennard-Jones potentials[16]. They differ based on the area of chemistry they are aimed at and in the degree of complexity of the functional form of each of the individual energy contributions. Force-fields form the basis of Molecular Dynamics simulations. A

simulation consists of one or more molecules with both position and momentum within the boundaries of the simulation. Initial values are assigned to the position and velocity of each atom of each molecule. Once initialised, the Molecular Dynamics simulation commences and proceeds iteratively. At each step, the forces acting on the particles are determined in the context of the force field as the negative of the gradient of the potential function (Eq. (2.1)). Newton's laws of motion (specifically the second, $F = ma$) are applied to the particles and new positions and velocities are calculated. As a simulation runs, measurements, known as samples, are taken of the reaction coordinates of interest, such as the dihedral angle.

2.3 The Boltzmann Factor

Measurement of molecular PMF, such as the PMF of butane in Fig. 2.2(b), is a common goal in computational chemistry. Structural distributions obtained from Molecular Dynamics samples provide a method to measure PMF through a physical relation known as the Boltzmann Factor. However, obtaining an acceptable number of samples across energy barriers is often also restricted by the Boltzmann Factor. Figs. 2.2(b) and (c) illustrate how the Boltzmann Factor relates PMF to the probability distribution of molecular geometric state. These distributions show that the lower the PMF is, the more likely a molecule is to be found in that geometric state. In Fig. 2.2(b) the value of the reaction coordinate defines the state the molecule is in which relates to a PMF value. However, the probability that a molecule can be in a certain state is not uniform but varies depending on the value of the reaction coordinate. This probability distribution is shown below on the same axis in Fig. 2.2(c). When the molecule is not in a state of equilibrium, it possesses a level of energy higher than one of the local minima, which means the forces acting within the molecule are unbalanced. This unbalance in forces causes the molecule to rearrange itself towards equilibrium, hence reducing its energy to the nearest local energy minimum. Since the molecules are constantly moving toward equilibrium, it suggests that lower energies have higher probabilities, the highest probabilities being at the energy minima. This is known as the Boltzmann Factor which relates the energy of a molecule (Fig. 2.2(b)) to the probability of

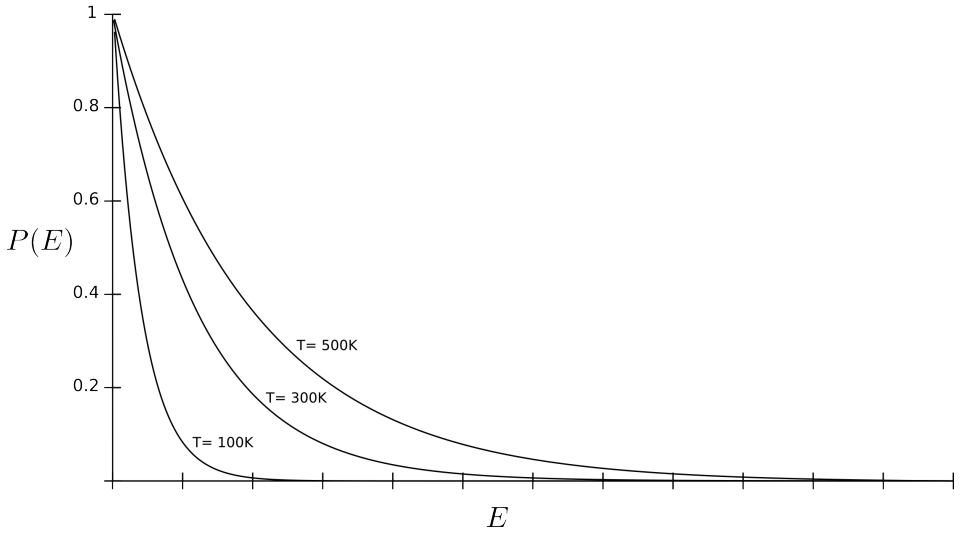


Figure 2.3: The Boltzmann Factor - the relationship between probability and energy. Eq. (2.3) is plotted against energy for three different values of T: 100K, 300K and 500K. Molecules are more likely to be found with less energy, but the higher the temperature, the greater the variance.

finding the molecule in a certain state (Fig. 2.2(c)). The Boltzmann Factor forms one of the fundamental relationships of statistical mechanics by relating probability theory to thermodynamics.

The functional form of the Boltzmann factor is required to calculate PMF. To derive this functional form we consider the additive nature of energy. Supposing that a system is in a state with energy E, we wish to find the functional form of the probability, $P(E)$, that the system has energy E. To do this consider the case of two particles with energies E_1 and E_2. Since energy is additive, this means the probability of the total energy is equal to the product of the probability of the individual energies, which gives

$$P(E_1 + E_2) = P(E_1).P(E_2). \tag{2.2}$$

This relationship defines a required property of the function $P(E)$. One such property is the exponential identity property where, using e as a base, $e^{(x+y)} = e^x e^y$. This means if $P(E)$ is an exponential function, with E as the exponent, it

satisfies Eq. (2.2). As mentioned previously, the system tends towards equilibrium - the lower the energy the higher the probability - which implies the relation is a decaying exponential, so the exponent will be negative. Another factor to take into account is the temperature. Energy is associated with temperature, in the sense that the higher the temperature the higher the energy, which implies temperature T is a denominator of the decaying exponential term. The remaining constant in this case is the Boltzmann constant k which leads to the Boltzmann Factor:

$$P(E) \propto e^{\frac{-E}{kT}}. \tag{2.3}$$

This proportional relationship is plotted in Fig 2.3 which shows probability (between zero and one) against energy E at three different temperatures ($T = 100$K, 300K and 500K). We can see that for all temperatures, as the energy increases, the probability $P(E)$ of the molecule having energy E decreases. The plots of different temperatures confirm a further intuition; that as the temperature increases it is more likely for molecules to have higher energy states.

The Boltzmann Factor provides an elegant solution to the measurement of PMF since it relates PMF to probability. Probability distributions can be determined with histograms constructed from the samples obtained from Molecular Dynamics simulations. From Eq. (2.3), these probability distributions can in turn be used to calculate the PMF with

$$W(\xi) = -kTlnP(\xi), \tag{2.4}$$

where W is the symbol for PMF. The fundamental idea here is that by taking enough samples from a Molecular Dynamics simulation, we can calculate the probability of a set of states in which the molecule moves. This probability can then be used in the estimation of the PMF of those states. Therefore Eq. (2.4) allows us to calculate the PMF from histograms of Molecular Dynamics samples given a sufficient number of samples.

Unfortunately, the Boltzmann Factor presents practical complications when attempting to measure full free energy pathways such as the PMF of butane in Fig. 2.2b. In order to obtain accurate results, a statistically significant number of measurements would need to be taken across all values of the reaction coordinate. This is trivial at reaction coordinate values around equilibrium states since the

majority of molecules are in or close to these states. The problem arises when trying to obtain enough samples at higher energy regions since there are fewer molecules with higher energy. This lack of samples leads to inaccurate results across energy barriers. Even worse, in cases where the energy barriers are narrow and steep, they may be missed entirely.

2.4 Umbrella Sampling Simulations

A number of enhanced sampling Molecular Dynamics techniques were developed to overcome the previously mentioned limitations which the Boltzmann Factor places on the complete exploration of free energy pathways. Valleau and Card[17] devised a stratification method known as multistage sampling where the total free-energy difference between two energy states is split up into a number of intermediate regions. This was shortly followed by an innovative technique from Torrie and Valleau known as Umbrella Sampling[5], which involves the introduction of an additional potential energy term to each intermediate region. This potential energy term is a biasing function (often known as the biasing, umbrella or window potential) which creates a window or umbrella around a prescribed centre point along the reaction coordinate (the umbrella centre). This window is imposed on the distribution bias of the simulation (conceptually similar to loaded dice) and confines the sampled reaction coordinates to within a small interval around the umbrella centre. A common biasing function, suggested by Roux[18], is the harmonic function

$$\frac{1}{2}K(\xi - \xi_0)^2 \tag{2.5}$$

where ξ is the variable reaction coordinate, ξ_0 is the umbrella centre point along the reaction coordinate and K is a constant know as the spring constant. A plot of this function is shown in Fig. 2.4a as $E_{bias}(\xi)$, with the umbrella centre at 60° and a spring constant K of 0.001. The width of the plot is determined by the value of the spring constant K. The function is characterized by a rapid increase in energy as ξ moves away from ξ_0 (60°), thus lowering the probability to almost zero anywhere but in the immediate area of the umbrella centre point. To apply this distribution bias, the Umbrella Sampling Molecular Dynamics simulation is

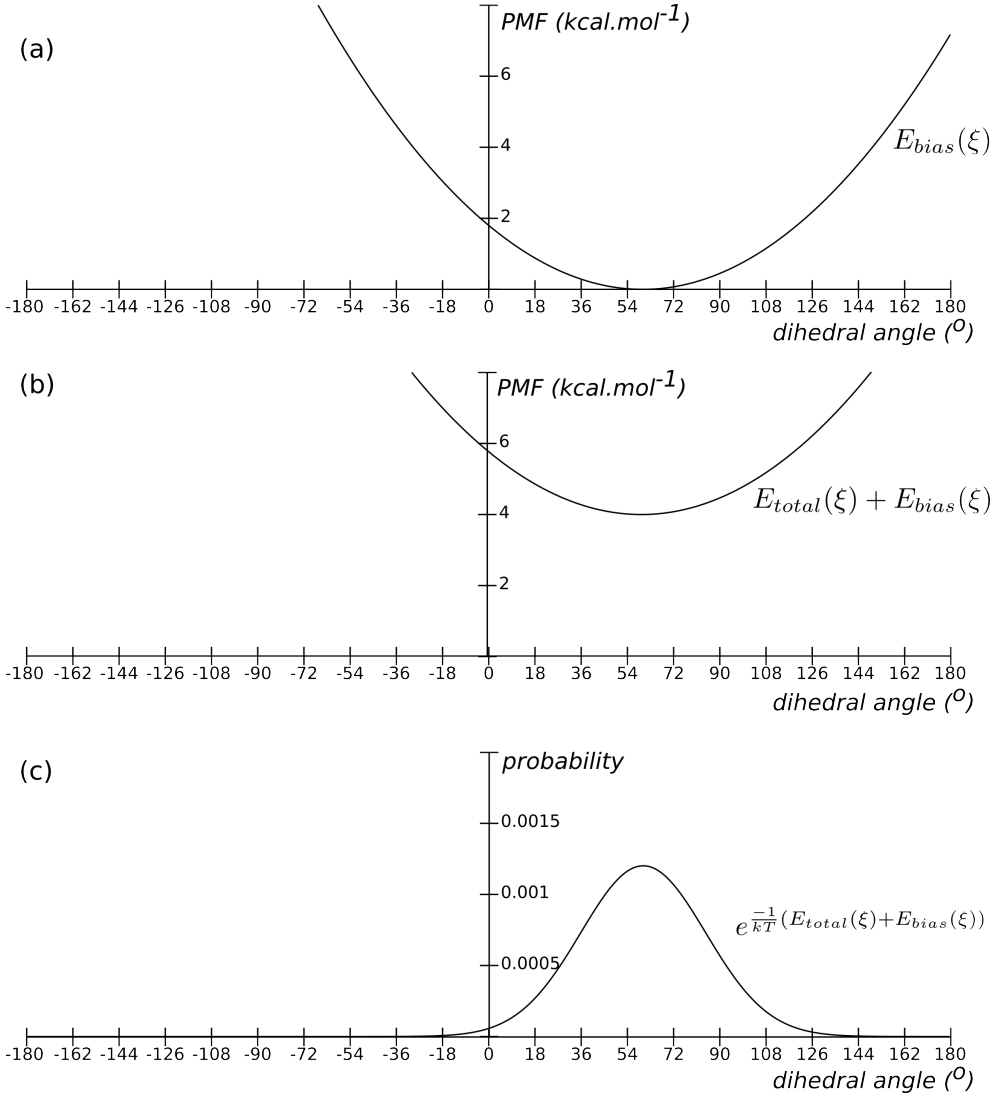


Figure 2.4: Umbrella sampling butane at 60°. Illustration of the umbrella created by applying the harmonic bias to a Molecular Dynamics simulation. (a) describes the harmonic biasing function (Eq. (2.5)), (b) the sum of the harmonic and the PMF at 60° (Eq. (2.6)) and (c) the umbrella sampling distribution created by the addition of the harmonic (Eq. (2.7)) to the Boltzmann Factor.

configured to add the harmonic function to the total energy, as in Eq. (2.6). In the example in Fig. 2.4b, the total energy along the entire reaction coordinate axis is

arbitrarily chosen to be 4 kcal/mol which shifts the entire harmonic function up.

$$E_{total}(\xi) + \frac{1}{2}K(\xi - \xi_0)^2. \tag{2.6}$$

The Umbrella Sampling probability along the reaction coordinate, shown in Fig. 2.4c, results from substituting the sum of total energy and the harmonic function (Eq. (2.6)) into the Boltzmann Factor (Eq. (2.3)), giving

$$P(E) \propto e^{\frac{-1}{kT}(E_{total}(\xi)+\frac{1}{2}K(\xi-\xi_0)^2)}, \tag{2.7}$$

or

$$P(E) \propto e^{\frac{-1}{kT}E_{total}(\xi)} \times e^{\frac{-1}{kT}(\frac{1}{2}K(\xi-\xi_0)^2)}, \tag{2.8}$$

where the second term of Eq. (2.8), $e^{\frac{-1}{kT}(\frac{1}{2}K(\xi-\xi_0)^2)}$ is known as the distribution bias throughout this thesis since it is pre-calculated. As can be seen from Fig. 2.4c, this new biased sampling probability distribution resembles an umbrella, and limits the samples taken by the Umbrella Sampling simulation to the region contained by the umbrella.

The goal of Umbrella Sampling is to approximate a uniform sampling distribution across the reaction coordinate, so a single simulation is insufficient. In order to obtain a complete PMF estimate, samples must be taken which span the entire range of the reaction coordinate. To achieve this, multiple Umbrella Sampling simulations are required. Each simulation is configured to run with a different umbrella centre so as to evenly space the umbrellas to cover the entire reaction coordinate, as shown in Fig.2.5. Each umbrella in Fig.2.5 is the distribution bias of a single Umbrella Sampling simulation. There are seven simulations with umbrellas centred at 60° intervals from $-180°$ to 180°. In this example the spring constant K is set to 0.001 which produces umbrellas wide enough for acceptable overlap. Bereau and Swendsen[10] suggest the optimal separation between umbrellas as 2.45σ where σ is the width of the umbrellas. The effect of this overlap can be seen in the wavy line above the umbrellas in Fig.2.5, which shows the sum of all seven distribution biases. This sum of distributions approximates a uniform sampling distribution from $-180°$ to 180°, which means that samples can be taken from the entire range of the reaction coordinate, thus overcoming the limitation

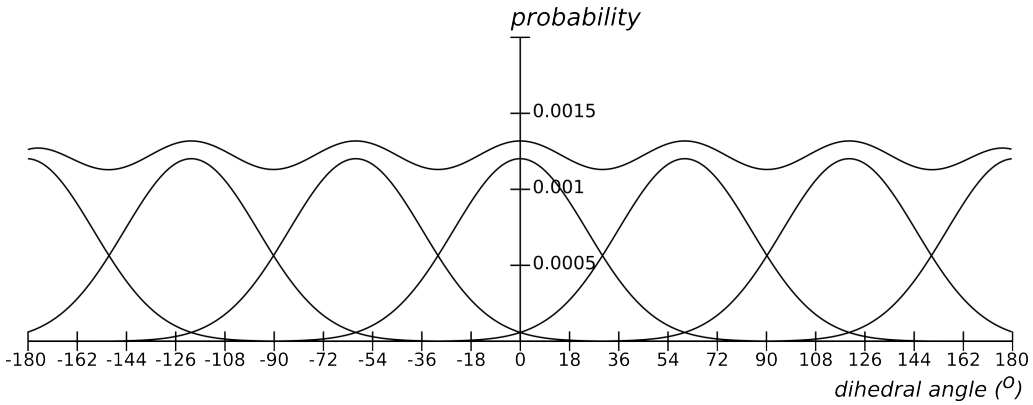


Figure 2.5: Multiple Umbrella Sampling simulations and the effective probability distribution. Seven Umbrella Sampling simulations are configured at intervals of 60°, each with a distribution bias shown as an umbrella. The sum of all the distributions results in the wavy line above the umbrellas, which forms a reasonable approximation to a uniform sampling distribution. This uniform distribution allows sampling across the entire reaction coordinate.

imposed by the Boltzmann Factor.

Chapter 3

WHAM

The previous chapter demonstrates how the problematic Boltzmann Factor can be overcome with the use of the Umbrella Sampling technique. Umbrella Sampling allows us to take samples from the entire reaction coordinate space and thus obtain enough samples across this space, but it introduces a new problem: the now biased uniform probability distribution invalidates any meaningful calculation of PMF with Eq. (2.4). What is required is a technique which removes the bias introduced by Umbrella Sampling after the samples are acquired. This is precisely what WHAM does. WHAM combines distributions from all the Umbrella Sampling simulations and removes the distribution bias originally introduced, to yield an estimate of the global unbiased distribution.

WHAM originates from Umbrella Sampling and histogram techniques which began with the development of histogram re-weighting. Histogram re-weighting allows the results of a single simulation to extend to estimations of nearby properties of the system at conditions other than that of the original simulation, i.e. a simulation at one temperature allows the estimation of the system at a nearby temperature. Based on earlier work by Salsburg et al.[19], this technique was successfully demonstrated by Ferrenberg and Swendsen[20, 21] in finite size scaling investigations of phase transitions by the comparison of histogram re-weighting results to experimental results. Unfortunately, histogram re-weighting is only suitable for small variations in conditions. This limitation was soon overcome in Ferrenberg and Swendsen's follow up paper[22] which introduced the multiple

histogram method, and with it the first versions of the WHAM equations. The multiple histogram method relies on histograms obtained from a number of simulations. Each of the histograms is appropriately weighted according to its statistical contribution to the target point. So the closer the umbrella to a point along a reaction coordinate the more the histogram contributes to the PMF at that point. This allows all histograms to contribute to the result thus maximising the amount of information which can be obtained from the simulations. Kumar et al.[1] then extended the multiple histogram method by combining it with Umbrella Sampling in a study of the PMF of the sugar ring in deoxyadenosine. This publication also introduced the name "Weighted Histogram Analysis Method" and the acronym WHAM.

3.1 Current Implementations

Current WHAM implementations exist either as standalone applications or integrated into Molecular Dynamics packages. The Molecular Dynamics package CHARMM includes WHAM functionality as an option in the free energy perturbation module[6], and the GROMACS distribution ships with the feature rich g_wham[7] tool. The interoperability of the simulation and analysis tools in these packages make them convenient when sampling is performed with the same package. However, standalone applications tend to be more versatile. The well-documented and concise Grossfield package[8] is one of the most popular standalone implementations and applies to both one- and two-dimensional WHAM. In addition, Sindhikara's Modular Reweighting [23] standalone package supports a variety of equilibrium biasing techniques (examples of extensions for Umbrella Sampling and Replica Exchange are provided). Parallel tempering simulation analysis can be performed with Chodera's PTWHAM[24]. It is however, deprecated in favour of the multistate Bennett acceptance ratio method (MBAR) [25] which is a similar method to WHAM with zero bin widths[26]. Hamiltonian Replica Exchange analysis can be performed with Bereau's OPT-MHM implementation of WHAM[10], which employs the OpenMP pragmas for high performance. The SMOG server[9], has an additional WHAM script which supports the measurement of any number of reaction coordinates but is not optimized for high performance.

Although there are numerous implementations of WHAM currently available, there is a lack of high performance implementations which support multi-dimensional reaction coordinate space, the closest being the two-dimensional Grossfield implementation.

3.2 WHAM Equations and Algorithm

The WHAM algorithm requires a histogram of each Umbrella Sampling simulation before the equations can be executed. The reaction coordinate is divided into a number of bin intervals and the samples from each simulation are used to create histograms defined by these bin intervals. These histograms together with the values of the distribution bias that fall within the bin intervals, form the input to the WHAM algorithm. The WHAM algorithm runs as a series of iterations of two equations that refine the results over each iteration until an acceptable level of accuracy is achieved. The body of an iteration comprises two, coupled, non-linear equations. The first, Eq. (3.1), computes new unbiased probability estimates, $p_{\bar{b}}$, from a set of weighting factors, f_h. The unbiased probability estimates are WHAM's estimates of the sampling probability over the entire reaction coordinate space if the bias were removed. The weighting factors represent the relative contribution of each histogram, and are all set to 1 to start with, but new weighting factors are used in each iteration. The second equation, Eq. (3.2), computes these new weighting factors, f_h, given the unbiased probability estimates, $p_{\bar{b}}$, computed previously. With the simulation histograms indexed by h and bin intervals indexed by $\bar{b}$ (the bar indicates potential n-dimensionality [1]) the WHAM equations are:

$$p_{\bar{b}} = \frac{B_{\bar{b}}}{\sum_h H_h f_h c_{h,\bar{b}}} \tag{3.1}$$

$$f_h = \frac{1}{\sum_{\bar{b}} c_{h,\bar{b}} p_{\bar{b}}}. \tag{3.2}$$

[1] n-dimensionality implies an index with any number of dimensions. This occurs when more than one reaction coordinate is investigated. For example, in other compounds, where the conformers primarily depend on two reaction coordinates, simulations would be configured to sample both coordinates. This would lead to a two dimensional histogram, where bins would be indexed by a pair of integers.

Given $n_{h,\bar{b}}$ as the number of samples in histogram h which fall into bin interval $\bar{b}$, $B_{\bar{b}}$ is the sum of samples of bin interval $\bar{b}$ across all histograms, or

$$B_{\bar{b}} = \sum_{h} n_{h,\bar{b}}, \tag{3.3}$$

and H_h is the sum of all samples in simulation h,

$$H_h = \sum_{b} n_{h,\bar{b}}. \tag{3.4}$$

H_h and $B_{\bar{b}}$ are referred to as the histogram aggregates in this thesis since they are also pre-calculated. $c_{h,\bar{b}}$ (the second term of Eq. (2.8)) is the value of the distribution bias[2] applied to simulation h (with umbrella centred at ξ_{h0}) at a point along the reaction coordinate, $\xi_{\bar{b}}$, which falls within bin interval $\bar{b}$ (usually the center), given by

$$c_{h,\bar{b}} = e^{\frac{-1}{kT}(\frac{1}{2}K(\xi_{\bar{b}}-\xi_{h0})^2)}. \tag{3.5}$$

The values of the histogram aggregates H_h and $B_{\bar{b}}$ and the distribution bias $c_{h,\bar{b}}$ are not modified by the WHAM equations so can be calculated before the WHAM algorithm proceeds.

As mentioned previously, the initial weighting factors, f_h, are all set to 1, which along with values of the histogram aggregates and distribution bias complete the data requirements to run the WHAM algorithm. An iteration of the WHAM algorithm requires evaluation of Eq. (3.1) for each bin interval $\bar{b}$, to yield an array of unbiased probability estimates, $p_{\bar{b}}$. These estimates are used as input to Eq. (3.2), which yields a new weighting factor f_h for each histogram h. At this point convergence is determined. The difference between each initial weighting factor, used in Eq.(3.1), and its corresponding new weighting factor, calculated with Eq.(3.2), determines convergence. When all differences are less than a prescribed tolerance level (the default value for the Grossfield WHAM package[8] is 0.001), the algorithm has converged. This indicates that the variation of weighting factors between iterations has dropped to a tolerable level. If the difference between any initial

[2]The n-dimensional expansion of Eq. (2.8) is given by $c_{h,\bar{b}} = e^{\frac{-1}{kT}\sum_{d=1}^{n}(\frac{1}{2}K_d(\xi_{\bar{b}_d}-\xi_{h0_d})^2)}$, where n is the number of dimensions.

and new weighting factor is greater than the tolerance level then the algorithm has not converged. In this case the new weighting factors are substituted into the initial weighting factors as input to Eq.(3.1), and another iteration begins. Upon convergence the most recent values of $p_{\bar{b}}$ are taken as the final global unbiased distribution estimate. This final distribution can then be used to calculate the estimate of the PMF at each bin interval $\bar{b}$ with Eq. (2.4).

The algorithm is detailed as pseudo code in Alg. 1. The top level loop forms the convergence check where at least one iteration must complete (repeat...until loop). For this reason the check is at the end of the loop. Eq.(3.1) is expressed as a nested loop pair. The outer loop is necessary to calculate a value for $p_{\bar{b}}$ for each bin interval $\bar{b}$ and the inner loop computes the summation in the denominator. The same pattern applies to Eq.(3.2), where the outer loop calculates a weighting factor for each histogram and the inner loop computes the summation. These loops determine the overall WHAM time complexity. The number of iterations executed by the top level convergence loop has no dependency on the number of histograms or bins which means it has constant time complexity. Nested loop pairs have quadratic time complexity or $O(n^2)$, which means the overall time complexity of WHAM is quadratic. Quadratic complexity means execution time is proportional to the square of the input size, so if the problem size doubles, execution takes four times longer.

Problem size is specified by the product of the number of histograms and the number of bins. These are typically prescribed according to the requirements of the experiment. For example, the choice of seven simulations for the butane example (Fig. 2.5) produces acceptable results to illustrate the concepts in this thesis, yet only produces PMF estimates every 60°. However, the most significant factor influencing problem size is not the number of simulations or bins, but the number of reaction coordinates under investigation. Butane is a simple example, measuring only one dihedral angle, but in more complex scenarios, multiple reaction coordinates may be explored. The number of simulations required and so the number of histograms processed by WHAM both increase exponentially with the number of reaction coordinates measured. For example, if the experiment requires PMF estimates around a dihedral angle every 10°, this necessitates 36 umbrella sampling simulations. To measure two such reaction coordinates at that

```
// Begin iteration loop
repeat
    // Equation (3.1)
    foreach b̄ do
        denom ← 0;
        foreach h do
            denom ← denom + H_h * c_{h,b̄} * f_{h_old};
        end
        p_b̄ ← B_b̄ / denom;
    end
    // Equation (3.2)
    foreach h do
        finv ← 0;
        foreach b̄ do
            finv ← finv + c_{h,b̄} * p_b̄ ;
        end
        f_{h_new} ← 1 / finv;
    end
    // Check for convergence
until converged(f_{h_old}, f_{h_new});
```

Algorithm 1: The WHAM algorithm. Computation of each equation requires a loop through both the histograms and bins to calculate the summation factor in each equation. The 'converged' function, at the end of the outer iteration loop, compares $f_{h_{new}}$ and $f_{h_{old}}$ and returns *true* if all differences are below the tolerance level.

interval requires 1296 (36^2) simulations. Furthermore, two reaction coordinates would lead to two-dimensional histograms, meaning the number of bins also increases exponentially. Since all simulations are processed by WHAM, which has a quadratic time complexity, this rapidly leads to unreasonable run-time durations for dimensions greater than two. The time complexity of the WHAM algorithm and requirements to explore multiple reaction coordinates, are strong reasons in favour of the argument to develop a parallel implementation of WHAM.

Chapter 4

GPGPU and CUDA

Graphics Processing Units (GPUs) have recently introduced a parallel programming paradigm aimed at high performance computing. The GPU architecture was originally designed for graphics acceleration, but the interface has since been extended to support general purpose computing (GPGPU), which facilitates the development of scientific applications, such as Molecular Dynamics, on GPUs[12, 27]. Recent enhancements of the Grossfield package have dramatically reduced the run time, but as of yet there are no GPGPU implementations of WHAM. Therefore, the primary contribution of this study is a high performance version of WHAM which can support multiple dimensions and large datasets by taking advantage of the parallelism offered by GPGPU architecture.

The basis of parallel programming is the division of loop iterations between multiple processors. This makes nested loop pairs, such as those in WHAM, well suited to parallel development. The traditional approach is to divide the problem by the number of available processors and let each processor loop through a section of the workload. The NVIDIA® GPGPU platform, the Compute Unified Device Architecture (CUDA) model, allows the program to declare the number of processors, or threads, each of which executes a single iteration of the loop. Rather than a loop counter, each thread has an index which is typically used to identify the location of data elements utilized in thread computation. These indices can be declared as one-, two- or three-dimensional, depending on the dimensionality of the problem domain. A nested loop pair, for example, maps to a two-dimensional

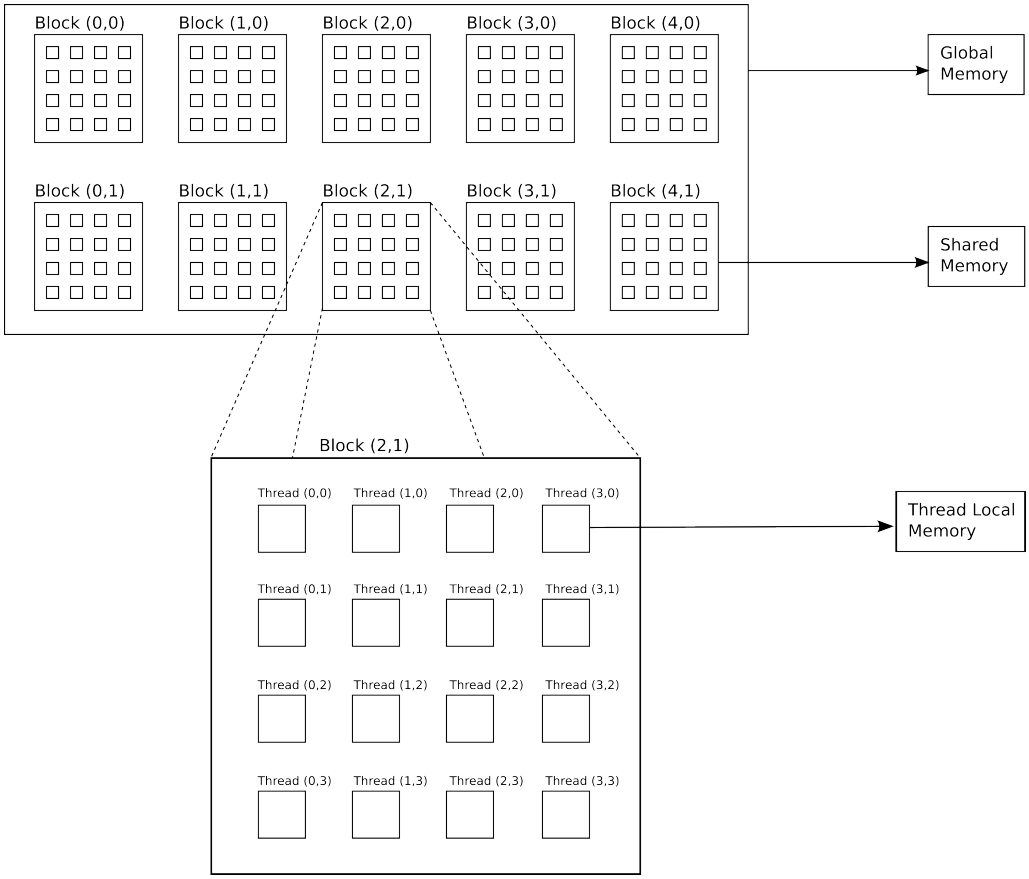


Figure 4.1: CUDA programming model and memory hierarchy. A grid is divided into blocks which are further divided into threads. This allocation of threads illustrates the division of a two-dimensional problem domain, such as a nested loop pair, into the CUDA programming model. All threads in the entire grid can access a single global memory, all threads in a block can access a shared memory space exclusive to each block, and each thread has its own private local memory.

CUDA domain as illustrated in Fig. 4.1. This domain comprises a grid divided into two dimensions of blocks, which in turn are divided into two dimensions of threads. Each thread performs execution of C or C++ code fragments known as kernels. Kernels have access to multiple memory spaces during their execution: local memory, shared memory and global memory, each of which are accessible as indicated in the figure. Registers provide private local memory exclusive to a thread. Within a block all threads have access to a common shared memory space, of which there is one shared memory space for each block. Shared memory

and thread barrier synchronization within the block allow threads to communicate and share data. A single global memory space, available to all threads in the grid is unsuitable for communication since synchronization is unavailable on the grid level. Only upon completion of kernel execution can blocks be considered synchronized. This limitation is imposed by the architecture in scenarios where devices require all threads in a block to run to completion before another block commences execution. Of the three memory types, local and shared memory have the smallest capacity, but fastest access times. Global memory is large and slow, yet it facilitates data transfer between CPU and GPU and persists on the GPU between kernel invocations. Chapter 5 describes in detail the design of the GPGPU implementation of WHAM: how the data is divided into blocks and threads and how synchronisation is achieved at the grid level by performing portions of the computation on the CPU.

Chapter 5

Design and Implementation

To consider development of a parallel algorithm successful, the parallel algorithm must execute faster than its sequential counterpart and its outcome must be comparable to that of the sequential algorithm. WHAM is a challenging algorithm to both evaluate and validate. Evaluation is laborious since WHAM follows on from long running Molecular Dynamic simulations, and even then validation is not possible since simulations provide no certainty that the WHAM outcome is correct. Molecular Dynamics cannot adequately support evaluation nor validation of WHAM, so for this reason a system was developed to replace the Molecular Dynamic simulations. This system, the test harness, assists in evaluation by generating samples fast and efficiently from a known analytic reference PMF. Additionally, the analytic PMF enables validation by comparison of the WHAM estimated PMF to a known reference PMF. Development of this system led to the logical separation of two modules: the test harness and the WHAM module, illustrated in Fig.5.1. The test harness forms a platform on which WHAM can be comprehensively evaluated.

The functionality of each module is decomposed into components each with a defined responsibility. The test harness has a dual purpose: sample generation and WHAM validation. These tasks occur before and after WHAM and both make use of the analytic reference PMF. The WHAM module is designed to function in a standalone manner. If provided with samples generated by either the test harness or Umbrella Sampling simulations then the WHAM module will generate PMF

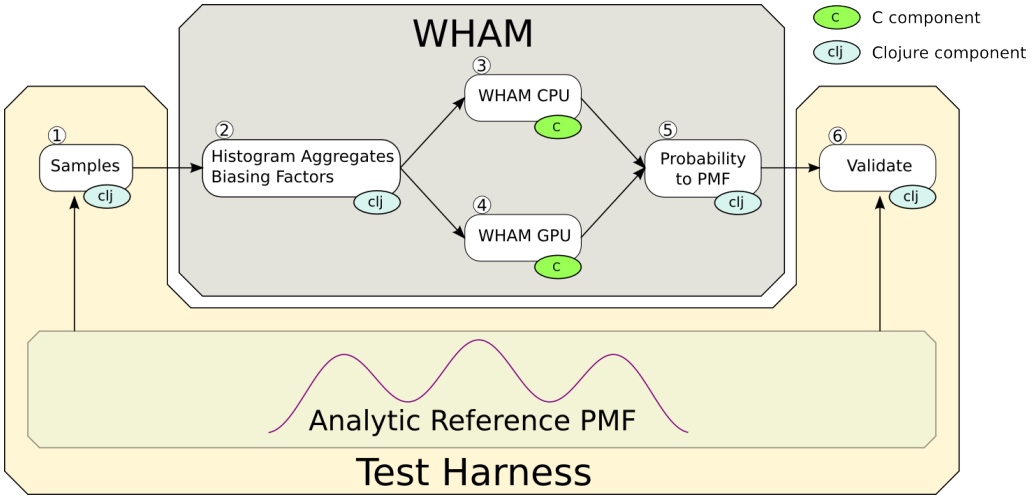


Figure 5.1: Components of WHAM and the test harness. The six components, developed to evaluate WHAM implementations, belong to WHAM or the test harness. The components are written in C or Clojure, tagged with C or clj respectively.

estimates. It allows selective execution of a WHAM implementation and performs common functionality: it creates input data from the samples and processes the WHAM implementation output data. In order to compare the speed of CPU and GPGPU WHAM implementations each is built as a separate component.

Fig.5.1 illustrates the constituent components (numbered ① - ⑥) of both modules and the data flow between them. Each component is written in either C or Clojure[28], and is tagged with either C (C) or clj (Clojure) to identify the programming language used. The WHAM implementations (③ and ④) are written in C since this is the prescribed CUDA kernel language and it offers the best performance. The test harness (① and ⑥) and the components which create input data from the samples(②) and process WHAM output data(⑤) were written in Clojure because it is a functional language and the processing performed by these components is functional. To facilitate the interaction between programming language environments, each component is run as a separate operating system process which reads its input from files and writes its output back to files.

The component's responsibilities are summarized below and explained further in the relevant section.

① Samples, analogous to Umbrella Sampling simulations, are generated by the test harness. These are equivalent to samples which would be generated by Umbrella Sampling simulations of a molecule whose PMF was that of the analytic reference PMF.

② Input data, suitable for the WHAM iterations, is generated. Histograms are created from the samples which are in turn used to create the histogram aggregates $B_{\bar{b}}$ (Eq. (3.3)) and H_h (Eq. (3.4)), and the distribution bias values $c_{h,\bar{b}}$ (Eq. (3.5)) are computed.

③ and ④ The CPU and GPGPU implementations are executed separately with the data created by component ②. As the algorithm completes, the probability estimates, $p_{\bar{b}}$, are computed and written to disk.

⑤ The PMF is computed with Eq. (2.4), from the probability estimates, $p_{\bar{b}}$, generated by each implementation.

⑥ The second component of the test harness. The PMF output of the WHAM module is validated by comparison to the analytic reference PMF.

This chapter expands on each component and describes the end-to-end process of WHAM evaluation. This spans the creation of umbrella samples to validation and performance assessment of the WHAM implementations. First, an illustrative example using the simple butane molecule is used to explain the test harness. Details are provided of sample generation and validation based on the analytic reference PMF. Next the WHAM module is described, again with the butane example, starting with the steps required to create WHAM input data from the samples. The WHAM implementations are then discussed and additional details are provided for the GPGPU algorithm to demonstrate the modifications made to convert the algorithm from a sequential to parallel paradigm. The WHAM module description ends with the conversion of the probability estimates to PMF. To conclude the chapter, we describe the more complex analytic PMF function used to generate samples, the complete set of test cases created and the metrics required to evaluate performance enhancements gained from porting WHAM to GPGPU.

5.1 Test Harness

5.1.1 Samples

Generation of samples analogous to those taken during Umbrella Sampling, is the first responsibility of the test harness (Fig.5.1 component ①). These samples are required to be similar to those generated by Umbrella Sampling simulations of a molecule whose PMF is equivalent to the analytic reference PMF. Given a set of Umbrella Sampling configurations (the umbrella centre and the value of the spring constant) and an analytic PMF (both defined as code), the test harness derives a probability distribution function for each configuration. A prescribed number of random samples are drawn from this probability distribution using a Monte Carlo method, and the values of the reaction coordinates of each sample is recorded.

Fig. 5.2(a) and (b) illustrate sample generation with the butane example. Initially the analytic PMF function $pmf(\xi)$ is composed. In the example (Fig. 5.2(a)) this is a simple function of a single reaction coordinate ξ, which resembles the PMF of butane. The reaction coordinate ξ is the $C_2 - C_3$ dihedral angle between $-180°$ and $180°$. In the example, seven Umbrella Sampling simulations are performed, however the effect of only a single Umbrella Sampling simulation, centred at $60°$, is illustrated in Fig. 5.2(a). Umbrella Sampling entails the addition of a restraint to the simulation, which biases the energy of the molecular system to within a certain region of the reaction coordinate. The energy bias is the harmonic function $\frac{1}{2}k(\xi - \xi_0)^2$, where k, the spring constant, affects the gradient, and ξ_0 determines the center of the harmonic. The result of the energy bias is to effectively add the harmonic function to the PMF, which yields an analytical term for the biased energy curve. Fig. 5.2(a) shows the sum of the energy bias and the PMF function centred at point $\xi_0 = 60°$ with spring constant 0.001. The resulting biased energy curve, $pmf(\xi) + \frac{1}{2}k(\xi - \xi_0)^2$, falls above the PMF function, and touches the PMF at $\xi_0 = 60°$, where the bias is equal to zero. The relationship between PMF and probability (Eq. (2.4)) associates the biased energy curve to a probability distribution function which resembles a bell curve, (Fig. 5.2(b)), given by

$$P(\xi) = e^{\frac{-1}{kT}(pmf(\xi) + \frac{1}{2}k(\xi - \xi_0)^2)} \tag{5.1}$$

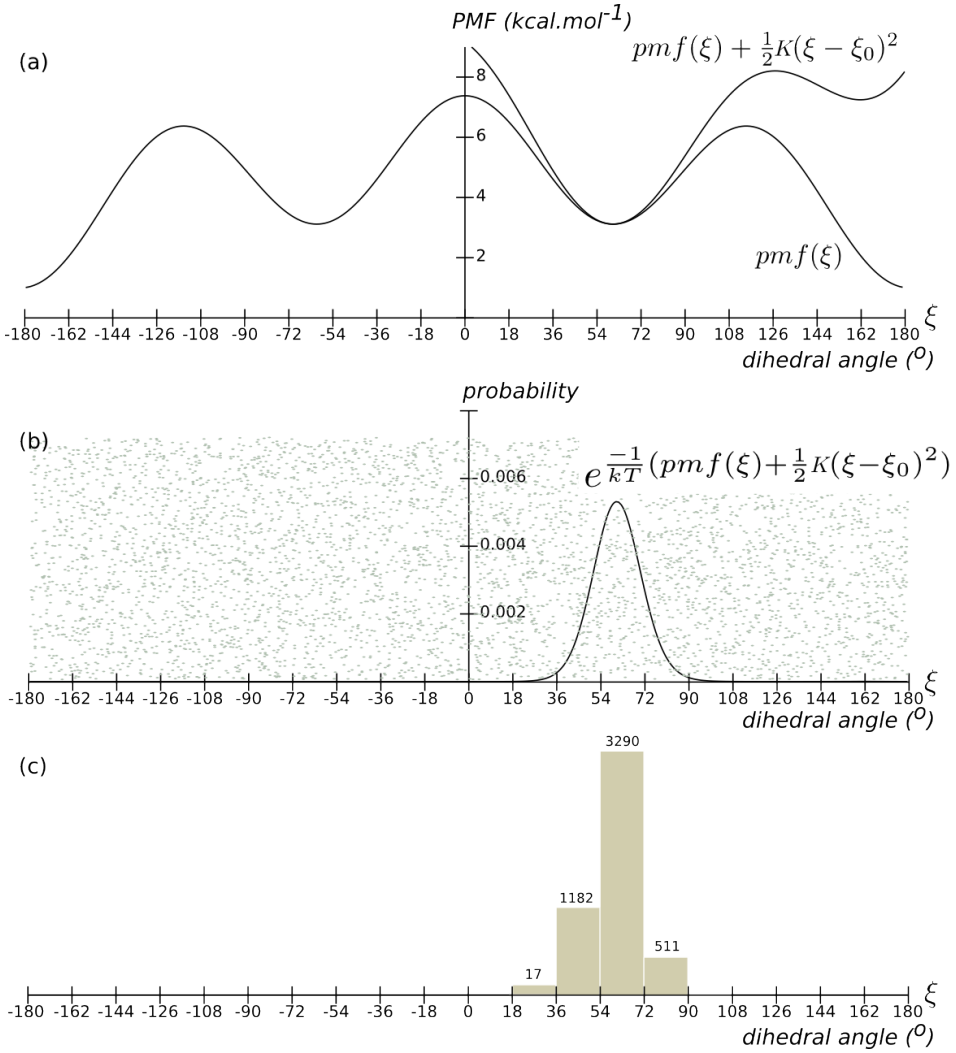


Figure 5.2: Test harness sample generation. Illustration of the artificial generation of Umbrella Sampling samples from an analytic PMF and an umbrella bias. (a) The bias, centred at 60° along the reaction coordinate ξ is added to the PMF function, $pmf(\xi)$, to yield the biased energy curve, $pmf(\xi) + \frac{1}{2}K(\xi - \xi_0)^2$. (b) This curve is then evaluated as a probability distribution function with Eq. (5.1). Reaction coordinate samples, analogous to samples taken during Umbrella Sampling, are produced by generating random points in the probability space. (c) The value of ξ for each point that falls within the distribution is binned at 18° intervals to create a histogram of umbrella samples.

This probability distribution is analytic, thus suitable for generating samples with a Monte Carlo simulation. Two random numbers are generated to produce each

sample. The first, ξ_{rand}, is a random number within the bounds of the reaction coordinate (between $-180°$ and $180°$). The value of the distribution at ξ_{rand}, $P(\xi_{rand})$, is evaluated with Eq. (5.1). The second is a random number between 0 and the maximum of the distribution (approx 0.006) which is more efficient than a random number between 0 and 1. ξ_{rand} is recorded as a sample if the second number is less than $P(\xi_{rand})$, i.e. if the point $(\xi_{rand}, P(\xi_{rand}))$ falls within the distribution. Random points are continuously generated and all those which fall within the distribution are included as samples until the required quantity have accumulated. In Fig. 5.2(b), the points under the bell curve are samples, whose values of ξ correspond to measurements of the reaction coordinate ξ taken during Umbrella Sampling. These samples are later grouped into histograms by the WHAM module.

5.1.2 Validation

Validation of the PMF estimates generated by the WHAM module is the second responsibility of the test harness (component ⑥ of Fig. 5.1). The analytically defined PMF function initially used to generate samples provides a reference against which the calculated PMF can be compared. Since these results are estimates and dependent on factors such as the number of samples and values of the harmonic spring constants, a simple comparison within some epsilon value is insufficient for validation. Instead, an initial visual confirmation of the correctness of the calculated PMF is made by comparison of plots of the calculated PMF against the analytic PMF function. Next, a more exact measure of accuracy is computed with the Root-Mean-Square Deviation method (RMSD). This method gives an indication of the overall difference in PMF predicted by the analytical function and the WHAM calculated values in which a value of zero indicates a perfect match and the greater the RMSD value is, the less accurate is the result.

5.2 WHAM

The WHAM module constitutes several processes depicted as components ② to ⑤ in Fig.5.1. Component ② is the initial pre-processing stage which creates input data for the WHAM implementations. Next, the algorithm is executed with the input data by one of the implementations, either CPU or GPGPU (components ③ and ④, respectively). Once complete, the implementation writes the unbiased probability estimates to disk from which the PMF is calculated by component ⑤.

Once all samples are generated by the test harness, component ② is responsible for the processes required to create the input data - structured data appropriate for the WHAM equations. The input for Eq. (3.1) comprises initial weighting factors f_h, histogram aggregates H_h and $B_{\bar{b}}$ and the biasing factors $c_{h,\bar{b}}$. In these terms, h and $\bar{b}$ are both array indexes, where h is the histogram number (corresponding to the simulation number), and $\bar{b}$ the bin number. The bar above $\bar{b}$ indicates potential multi-dimensionality, for example, the measurement of two reaction coordinates elicits two dimensional histograms whose bins are indexed as ordered pairs. The initial values of the weighting factors f_h are all simply set to a single value (usually 1) and component ② computes the values for the histogram aggregates, H_h, $B_{\bar{b}}$, and the distribution biases, $c_{h,\bar{b}}$.

Arrays H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ constitute the bulk of the input data, and conveniently, their calculation is only required once. Inspection of the WHAM Eqs. (3.1) and (3.2) shows that only the weighting factors f_h, and the unbiased probability estimates $p_{\bar{b}}$, change during an iteration of the algorithm. This means all calculations of H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ can be performed prior to the WHAM iterations.

5.2.1 Histogram Aggregates

Two phases are required to create the histogram aggregates H_h and $B_{\bar{b}}$. Firstly, the samples are grouped by reaction coordinate intervals, or bins, to create histograms. Fig. 5.2(b) and (c) illustrate how the random points generated by the test harness form the histogram for the simulation centred at 60°. For the butane example, 5000 samples are generated at each umbrella centre and binned into 18° intervals. Next, the histogram aggregates H_h and $B_{\bar{b}}$ are computed from these

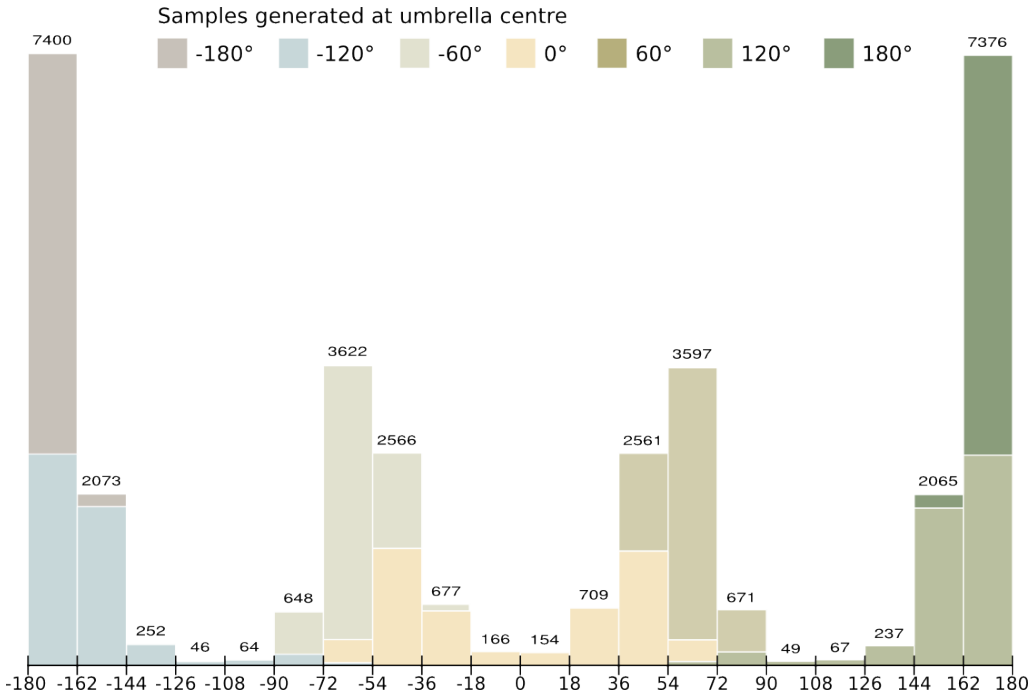


Figure 5.3: Global histogram, $B_{\bar{b}}$, of the butane example. The histogram aggregate $B_{\bar{b}}$ is calculated with Eq. (3.3) and depicted as a histogram. This is the histogram of all samples from all test harness simulations, layered to show the individual contribution of each simulation.

histograms with Eqs. (3.3) and (3.3). Each element of array H_h is the total number of samples in histogram h. For the butane example, H_h is an array of size 7 (from 7 simulations) where each element has a value of 5000, since this is the number of samples configured in the test harness. Array $B_{\bar{b}}$ amounts to a global histogram, where the value of the element at bin $\bar{b}$ is the sum of the bin count $\bar{b}$ of the individual histograms. Fig. 5.3 shows the butane example's global histogram $B_{\bar{b}}$ where there are 20 bins with the total bin count indicated above. The bins are layered so as to demonstrate the individual contributions from the test harness sample generation at each umbrella centre.

5.2.2 Distribution Biases

The distribution bias indicates the extent to which the sampling probability is modified along the reaction coordinate during an Umbrella Sampling simulation.

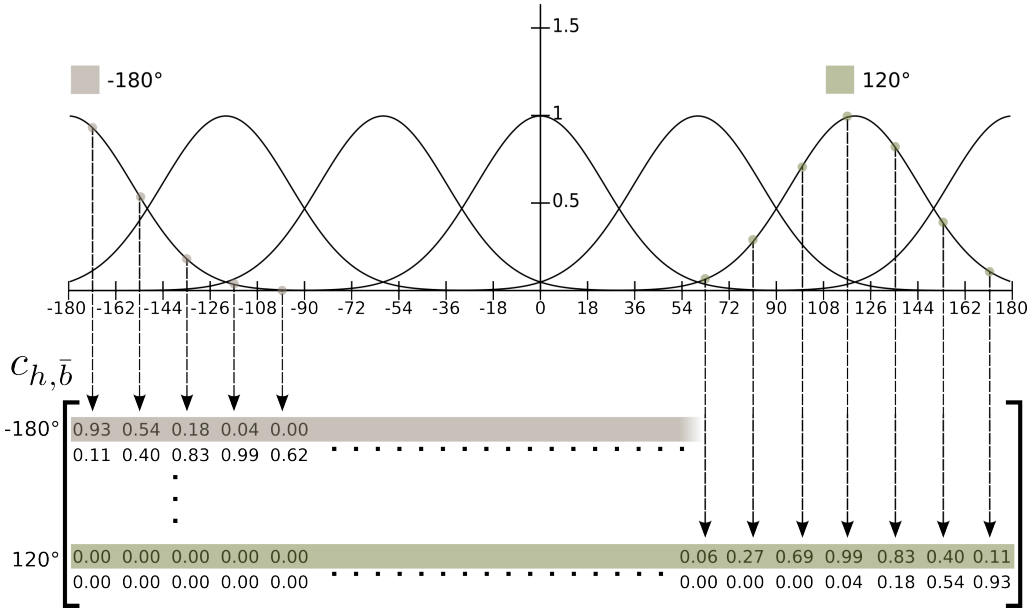


Figure 5.4: Computation of the distribution biases $c_{h,\bar{b}}$, of the butane example. The 7 simulations produce an array with 7 rows, where each element is computed with Eq. (3.5) at the centre of the corresponding bin.

It is the distribution introduced by the biasing potential Eq. (2.5), and hence it is computed from the Umbrella Sampling configuration, rather than the samples. Each element of the distribution bias array, $c_{h,\bar{b}}$, is the value of the bias at bin $\bar{b}$ for simulation h. This is calculated with Eq. (3.5) where $\xi_{\bar{b}}$ is the value of the reaction coordinate at the centre of bin $\bar{b}$. Consequently, the $c_{h,\bar{b}}$ array is two dimensional, where each row corresponds to a simulation and each column to a bin.

Fig. 5.4 illustrates how the $c_{h,\bar{b}}$ array of the butane example is computed for the simulations centred at $-180°$ and $120°$. The top row of $c_{h,\bar{b}}$ relates to the $-180°$ simulation where the first 5 values, each from the centre of the bin, are indicated on the graph. The remainder of the row is approximately zero. The second last row is computed from the second last simulation centred at $120°$. The same pattern applies to all simulations to generate an array of 7 rows and 20 columns from the 7 simulations with 20 bins each.

5.2.3 Extension to n Dimensions

An additional benefit of pre-calculating the H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ arrays is the convenient extension to multiple dimensions. The butane example is based on the one dimensional case, where only one reaction coordinate, the dihedral angle, is measured as the simulations proceed. In cases where numerous reaction coordinates are measured, the histograms acquire the same number of dimensions as the number of reaction coordinates, so the bin indices $\bar{b}$ become n-dimensional (n being the number of reaction coordinates). All data structures utilised by the algorithm can apply to n-dimensional scenarios by simply flattening the bin index $\bar{b}$ in a row major order. In the two-dimensional histogram case, rows of bins are laid out contiguously to form a linear structure, thus flattening $\bar{b}$ to a single dimension. This is possible since there are no dependencies in the WHAM equations between data in different bins and therefore applies to arrays $B_{\bar{b}}$, $c_{h,\bar{b}}$ and $p_{\bar{b}}$. No modifications are necessary for H_h or f_h. The n-dimensional values of array $c_{h,\bar{b}}$ are computed before flattening where the exponent of Eq. (3.5) is modified to include the sum of n harmonics, given by

$$c_{h,\bar{b}} = e^{\frac{-1}{kT}\sum_{d=1}^{n}\frac{1}{2}K_d(\xi_{\bar{b}_d}-\xi_{h0_d})^2} \tag{5.2}$$

where d is the dimension number inclusive of the upper bound n.

Upon completion of the algorithm, the final unbiased probability estimates $p_{\bar{b}}$ are reconstructed into a structure with the original dimensions. If the arrays are flattened during construction, H_h and $B_{\bar{b}}$ remain one dimensional and $c_{h,\bar{b}}$ remains two dimensional therefore any number of reaction coordinates can be measured without the need to modify the algorithm.

5.2.4 WHAM Implementations

Once arrays H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ are computed by component ②, iterative execution of the WHAM equations can begin, a task performed by the CPU and GPGPU implementations (components ③ and ④). The first iteration of the butane example (Fig. 5.5) illustrates the input and output arrays of a WHAM implementation which leads to the calculation of new weighting factors f_h.

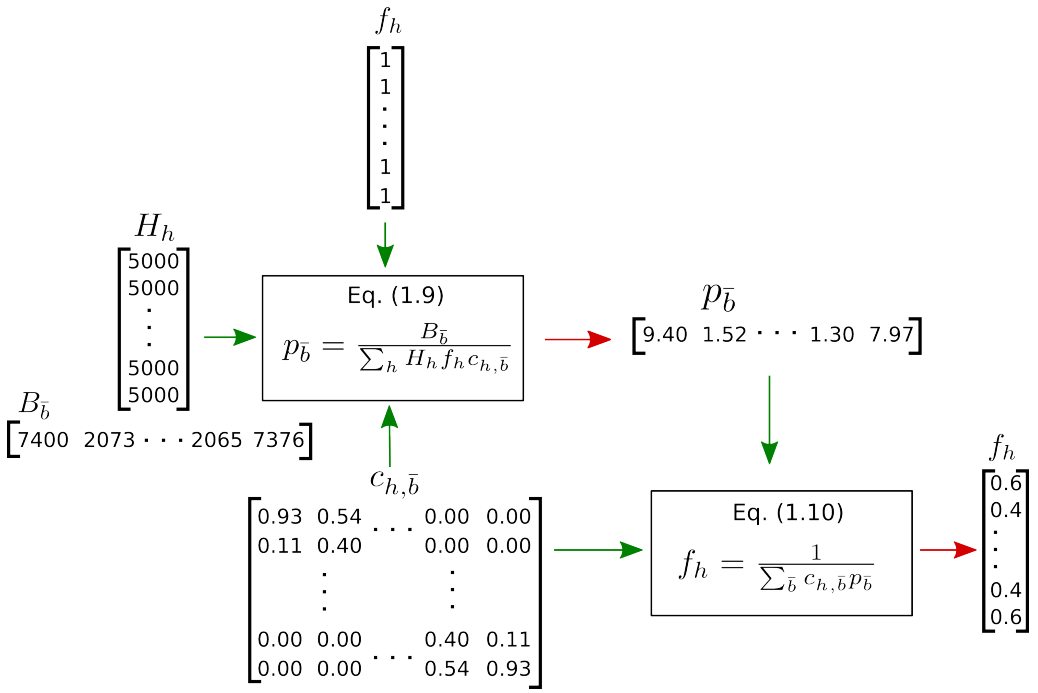


Figure 5.5: Schematic illustrating the first WHAM iteration of the butane example. Eqs. (3.1) and (3.2) show the two phases of the algorithm, described by Alg. 1 indicating their inputs (green) and outputs (red). The arrays H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$, computed earlier, comprise input to Equation (3.1) along with the initial weighting factor values, f_h (all set to 1 to start with). The distribution bias array, $c_{h,\bar{b}}$, is used again in Equation (3.2) with the unbiased probability estimates $p_{\bar{b}}$ (the output of Equation (3.1)) to compute new values for f_h.

Convergence is checked at the end of an iteration where a comparison is made between the old values of f_h and the new values of f_h. If the absolute difference between every old value and its corresponding new value of f_h is less than a certain tolerance value, the algorithm terminates and the latest values of $p_{\bar{b}}$ provide the final probability estimates. These values are written to disk to be picked up by component ⑤ which converts the probability to PMF.

Since the two implementations were developed to compare performance, they are written to be as similar as possible. Both implementations read H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ from the file system and store the data in low level single precision floating point arrays. Both follow Alg. 1: the CPU version loops through the arrays whereas the GPGPU version employs threads to perform the operations on each array element.

Both implementations are compiled with the NVidia CUDA nvcc complier V5.5.0 set to the highest level of optimization to ensure the GPU version is not faster due to compiler optimizations.

5.2.5 Probability Estimates to PMF

After convergence the final task of the WHAM module, the calculation of PMF from probability, is executed by component ⑤. When the WHAM implementation terminates the most recent values of the unbiased probability estimates $p_{\bar{b}}$ are written to disk by the implementations (components ③ and ④). The probability estimates are read by component ⑤ which are normalized and then PMF is calculated with Eq. (2.4) to yield one PMF estimate per bin. These estimates are then written to disk for validation and to create plots. To facilitate the extension to n-dimensions, component ⑤ is written in Clojure to take advantage of its n-dimensional array library (NDArray).

5.3 GPU Design

CUDA's single-instruction, multiple-thread[29] (SIMT) architecture extends the single-instruction multiple-data (SIMD) paradigm to allow programming at the thread execution level. For example, to decompose a sequential loop through a 100 element array onto a 4 processor multi-core CPU requires splitting the array into 4 sub-arrays of 25 elements. Alternatively, in the SIMT paradigm, 100 threads are assigned, each to a single element.

Data decomposition is typically the initial step[30] in the procedure to port sequential algorithms to the SIMT paradigm. The subsequent steps are: algorithm selection, implementation and optimization. Algorithm selection entails extraction of computations or instructions which are amenable to parallelism. For WHAM, these are the summations of the products in the denominators of Equation (3.1) and Equation (3.2). The GPGPU implementation of WHAM was approached in two stages. Initially a prototype implementation was written and validated with a minimal data set on a single block. This was followed by extension of the prototype across multiple blocks to allow for larger problem sizes. This extension

necessitated a shift to a hybrid approach where sections of the computation are performed on CPU and others on GPU. The final stage of the development procedure, optimization, was performed incrementally by applying an optimization and taking a run-time profile of the algorithm executed on a large data set. If run-time was reduced another optimization iteration was repeated on the improved code base.

5.3.1 Data Decomposition

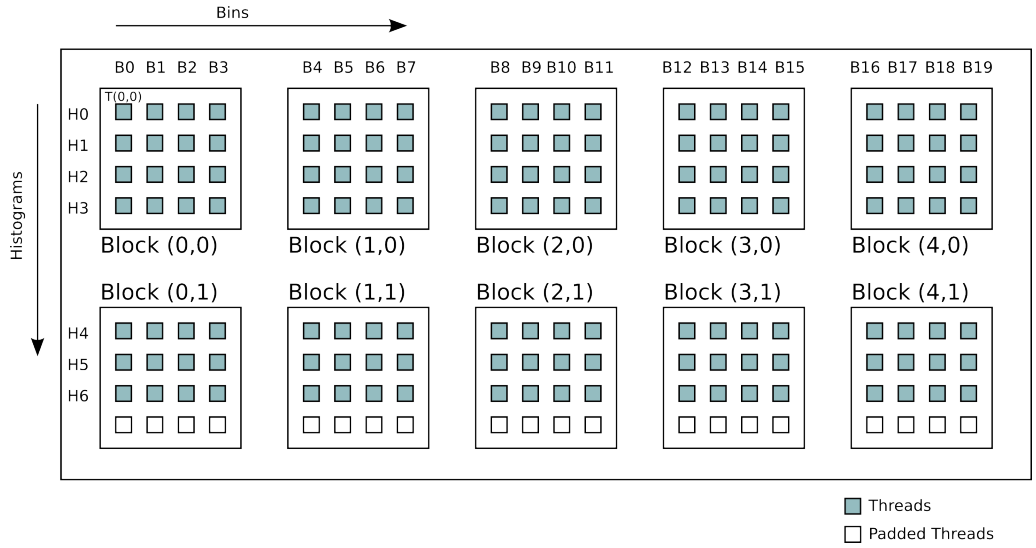


Figure 5.6: CUDA domain decomposition for the butane example. The seven histograms, indexed H0 to H6 are decomposed into rows, each with 20 bin intervals B0 to B19. The block size is 4×4 which breaks the problem into 2×5 blocks, where the last row is padded with zeros.

Identification of computationally expensive loops is a fundamental parallel design technique to reveal efficient GPGPU data decompositions. Alg. 1 contains two expensive nested loop pairs, one for each of Eqs. (3.1) and (3.2), which both cycle through the histograms and bins. Nested loop pairs suggest a two-dimensional parallel data decomposition. For WHAM this decomposition is employed on the grid and block level as illustrated by the decomposition of the butane example in Fig. 5.6. The butane example comprises 7 histograms, each with 20 bins spaced at 18° intervals. Each histogram is assigned a row in the two-dimensional data struc-

ture, where the histogram index h points to the y-axis and the flattened bin index $\bar{b}$ points to the x-axis (similar to the distribution bias array $c_{h,\bar{b}}$). This assigns one thread to each bin of each histogram, depicted as the smallest squares (thread (0,0) is labelled) in the diagram. The block size must be a power of two, so in this example it is chosen as 4×4, but this can vary and is limited by the maximum number of threads allowed on a block. Twenty bins on the x-axis fits exactly into 5 blocks, but seven histograms is one short of two blocks on the y-axis. A common strategy is to build a range check into the kernels, but our implementation pads the last row with zeros instead (indicated as padded threads), since this does not affect the outcome and simplifies the code by removing branch divergence.

5.3.2 SIMT algorithm

`// Begin iteration loop`
repeat
  `// Equation` (3.1)
  $HCF_{h,\bar{b}} \leftarrow H_h * c_{h,\bar{b}} * f_{h_{old}}$;
  $colSum_{\bar{b}} \leftarrow sumColumns(HCF_{h,\bar{b}})$;
  $p_{\bar{b}} \leftarrow \frac{B_{\bar{b}}}{colSum_{\bar{b}}}$;
  `// Equation` (3.2)
  $CP_{h,\bar{b}} \leftarrow c_{h,\bar{b}} * p_{\bar{b}}$;
  $rowSum_h = sumRows(CP_{h,\bar{b}})$;
  $f_{h_{new}} \leftarrow \frac{1}{rowSum_h}$;
until *converged*$(f_{h_{old}}, f_{h_{new}})$;

Algorithm 2: The WHAM algorithm in the SIMT paradigm. Initial prototype of the WHAM algorithm where the entire algorithm is executed to completion on a single block.

The SIMT paradigm exposes a view of the WHAM algorithm which resembles the equations more closely than the sequential algorithm. Each thread, indexed by $(h, \bar{b})$, handles execution of the GPGPU kernel described by Alg. 2. The first step is to populate an intermediate, two-dimensional, shared memory array, HCF,

with the product $H_h * c_{h,\bar{b}} * f_{h_{old}}$. The summation in the denominator of Eq. (3.1) is across histograms so each column of HCF is summed and the result applied to the remainder of the equation to yield the unbiased probability estimates $p_{\bar{b}}$. Next, for Eq. (3.2), another intermediate array, CP, is computed from the product $c_{h,\bar{b}} * p_{\bar{b}}$. Since summation is across bins, the sum of the rows of CP are used as the denominator to calculate the new weighting factors, $f_{h_{new}}$. Finally $f_{h_{new}}$ is compared with $f_{h_{old}}$ and the iteration restarted if convergence is not reached.

Written as a proof of concept, the single block implementation (where thread communication is synchronised) maintains numerous advantages over the multiple block implementation: a single kernel invocation iterates until convergence, there is minimal data transfer between CPU and GPU, the convergence loop executes inside the kernel and all data structure indices are simple. However, a single block limits the problem size to the maximum number of threads which can run on a block. For current high end NVidia GPUs with compute capability 3.5 this limit is 1024. This is sufficient for only the simplest one-dimensional scenarios and does not justify parallelism from the outset. Therefore, multiple blocks are mandatory to address larger, more complex problems.

5.3.3 Extension to multiple blocks

Parallel summation, such as the column and row summations ($sumColumns()$ and $sumRows()$) of Alg. 2, employs reduction which relies on thread synchronization. However, accomplishing thread synchronization across multiple blocks requires that kernel invocations return prior to any subsequent dependent calculations. This limitation exists since block synchronisation, and hence global thread synchronisation is guaranteed only upon kernel returns. Consequently, a multiple block WHAM implementation calls for two kernels, described by Alg. 3, one of which terminates after $sumColumnsGPU()$ and the other after $sumRowsGPU()$. Each kernel performs shared memory reduction contained within a block and writes the intermediary block sums to global memory. To complete the summations the intermediary sums are transferred back to the CPU, which performs the final summation. This interleaves the kernel and equation boundaries as can be seen in Alg. 3 where kernels 1 and 2 are shaded, the CPU code is unshaded, and kernel

// Begin iteration loop
repeat
  // Kernel 1
  // Equation (3.1)
  $HCF_{h,\bar{b}} \leftarrow H_h * c_{h,\bar{b}} * f_{h_{old}}$;
  $interColSum_{\bar{b}} \leftarrow sumColumnsGPU(HCF_{h,\bar{b}})$;
  $colSum_{\bar{b}} \leftarrow sumColumnsCPU(interColSum_{\bar{b}})$;
  // Kernel 2
  $p_{\bar{b}} \leftarrow \frac{B_{\bar{b}}}{colSum_{\bar{b}}}$;
  // Equation (3.2)
  $CP_{h,\bar{b}} \leftarrow c_{h,\bar{b}} * p_{\bar{b}}$;
  $interRowSum_h = sumRowsGPU(CP_{h,\bar{b}})$;
  $rowSum_h = sumRowsCPU(interRowSum_h)$;
  $f_{h_{new}} \leftarrow \frac{1}{rowSum_h}$;
until *converged*$(f_{h_{old}}, f_{h_{new}})$;

Algorithm 3: The WHAM algorithm - Hybrid. Two kernels which perform sections of computation are shown in the shaded regions and the unshaded lines are executed on the CPU. After completion of each kernel, intermediary summation values are transferred to the CPU which completes the summation. In the hybrid version the computation of the new weighting factors, $f_{h_{new}}$, and the convergence check loop is also executed on the CPU.

2 implements the final calculation $p_{\bar{b}}$ from Eq. (3.2). The final row summation, the computations of $f_{h_{new}}$ and the convergence check are performed on the CPU, where the new values of $f_{h_{new}}$ are transferred back to the GPU if another iteration is initiated. If not, the probability estimates $p_{\bar{b}}$ are retrieved from GPU memory if the algorithm converges.

5.3.4 Optimizations and Design Discussion

The strategy employed to gauge the effect of optimizations was to take performance profiles with the Nvidia Visual Profiler and comment out individual code sections

to measure their run-time contributions. This allows profiling at the "line of code" granularity. Times taken by the individual code fragments are shown in the comments of the kernel code listings in Appendix A.

Initially, the histogram aggregates and the distribution biases, arrays H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$, are transferred to the GPU before any kernel calls. This proves to be a considerable advantage of the WHAM algorithm compared with other iterative algorithms because these form the majority of the data transfer and are unchanged from iteration to iteration. Hence the arrays are only transferred once at start-up. It is possible to pre-compute and transfer $H_h * c_{h,\bar{b}}$ to the GPU to speed up the calculation of the HCF array ($H_h * c_{h,\bar{b}} * f_{h_{old}}$) by removing one multiplication operation. This was attempted and it was found to speed up the algorithm by 4 ms but it also drastically reduces the maximum problem size which can be attempted due to memory constraints on the GPU device. As a result the decision was made to omit this optimization.

The standard set of reduction optimizations[31] were applied to the summation sections of the kernel with the exception of loop unrolling since this had a negligible effect. However, the discrepancy between the column summation and the row summation times was of particular interest. The column summations of Kernel 1 were approximately twice as fast as that of the row summation of Kernel 2. This discrepancy is counter intuitive to the concept of memory access coalescence. The row summations access contiguous memory locations whereas the column summations access spanned memory locations, so the row summations are expected to run faster. However, the reason for this discrepancy can be explained when taking into account the number of active warps. During each reduction step the second half of the row (or column) is added to the first half. In the case of the row summation all warps will remain active at each reduction step since the entire warp must remain active if there is one active thread. At each column reduction step however, the number of active warps is halved since there are no active threads in the bottom half of the rows and these inactive threads generate no read cycles. To achieve further run-time reduction based on this observation, the row summation of Kernel 2 was replaced with a column summation of transposed values.

Another optimization made was to delay the writing of the unbiased probability estimates $p_{\bar{b}}$ to global memory. The estimates are only required at algorithm

termination and since global memory writes are slow, Kernel 2 does not write $p_{\bar{b}}$ to global memory. Instead, upon convergence, a separate finalize kernel runs the Kernel 2 algorithm up to the point of computing $p_{\bar{b}}$ and then writes these values to global memory, which is then transferred to the CPU.

Further small scale optimizations were considered such as employing constant memory to broadcast the histogram aggregates, but the contributions of these optimizations were deemed too insignificant to justify the time spent on development.

5.4 Measurements and Test Cases

Speed-up is defined as the ratio of the time taken on the GPU to the time taken on the CPU. This ratio is obtained from time measurements taken by the CPU and GPGPU implementations as part of their run-time process. Both versions take measurements with the same timing API, at the same points in the algorithm, and record the average of ten runs to account for any background process interference. The total time and number of iterations are recorded which yields the average time per iteration. This metric is comparable for iterative algorithms such as WHAM since the number of iterations may vary.

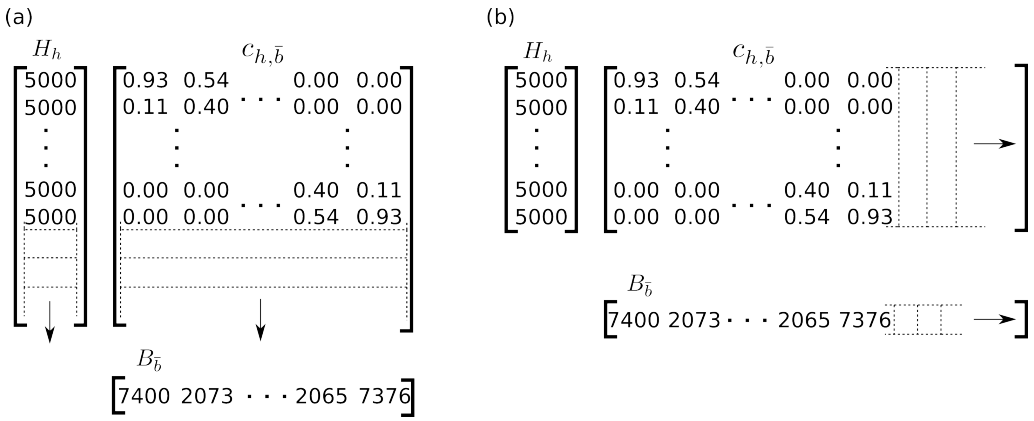


Figure 5.7: Variation of the problem size by (a) increasing the number of histograms (size of H_h) or (b) increasing the number of bins (size of $B_{\bar{b}}$)

In a sequential, single threaded environment, computational operations typically execute at a roughly consistent speed which depends on the speed of the CPU. This is not the case on the GPU architecture. As problem size increases on the

GPU the occupancy increases, which effectively increases the speed. This increase tends to converge to a maximum as the problem size approaches the maximum occupancy of the device. For this reason, to gain a thorough understanding of the potential speed-up, a number of test cases are required with increasing problem size. For WHAM, the problem size increases as the input arrays H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ increase in size, which can happen in two ways: either an increase in the number of histograms, which increases the size of H_h and $c_{h,\bar{b}}$, or an increase in the number of bins, which increases $B_{\bar{b}}$ and $c_{h,\bar{b}}$. Fig. 5.7(a) and (b) depict increases in the number of histograms and bins respectively. The test cases were designed in such a way that the size of H_h and the size of $B_{\bar{b}}$ varied independently, thus allowing us to isolate the effect of the variation on the speed-up.

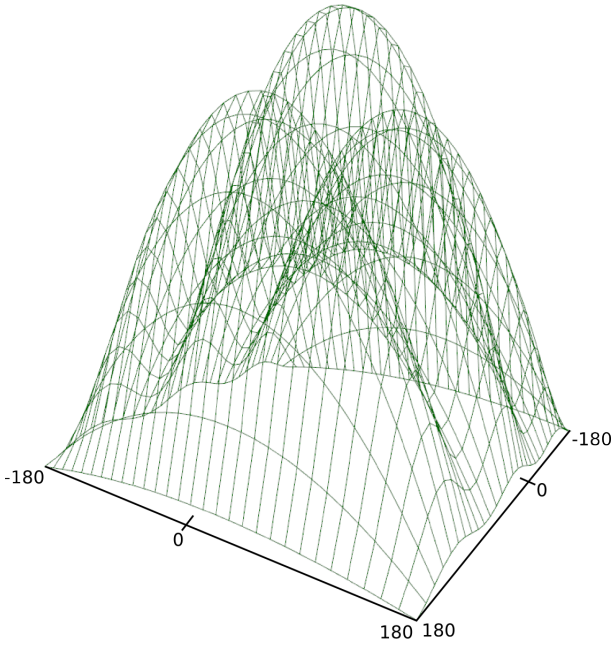


Figure 5.8: Plot of the analytical PMF function used to generate samples for the test cases and for validation. A two-dimensional extension of the butane example was created, which ranges from $-180°$ to $180°$ in the x and y dimensions.

Test cases were designed to evaluate the speed-up variation from small to large problem sizes and to validate the correctness of the implementations with multi-dimensional problems. For these reasons, and for easy visualization, a two dimensional analytic reference PMF function was created (Fig. 5.8), imitating the

measurement of dihedral angles. To mimic the measurement of two dihedral angles, a periodic function, extended from the butane example, ranging from $-180°$ to $180°$ in both the x and y dimensions was created. A total of 20 test cases were created: 4 variations in the number of histograms (19×19, 37×37, 61×61, 91×91) each with 5 variations of the number of bins (30×30, 60×60, 90×90, 120×120, 180×180). Each histogram was created from 50000 test harness generated samples. The irregular numbers of histograms is due to the fact that biases are centred at the boundaries of each reaction coordinate ($-180°$ and $180°$), as they are in the butane example. For the 19×19 case, the biases are spaced at $20°$ in each dimension, starting at $-180°$ and ending at $180°$, giving a total of 19×19 histograms. Similarly, biases spaced at $4° \times 4°$ results in 91×91 histograms. The histogram bounds are consistent across all test cases, so variation in the number of bins is inversely proportional to the bin width, hence the bin widths vary from $12° \times 12°$ (30×30 bins) to $2° \times 2°$ (180×180 bins).

Each test case was executed on both CPU and GPGPU implementations to produce corresponding performance metrics and the final probability estimates, $p_{\bar{b}}$. The test cases were run 10 times each and the average times were calculated from these 10 runs. The CPU implementation test cases were run on an Intel Core i7-3820 processor running at 3.60GHz on Ubuntu 12.10 (64 bit) with 8GB of ram. The processor has 8 cores but only a single core was active during the execution. The performance metrics include the total time taken by the algorithm and the number of iterations, which determines the average time per iteration. For the GPGPU implementation, the total time includes the time taken to allocate and initialize GPU memory, all data transfers between host and device, and the WHAM compute times. An Nvidia GTX670 with 1GB of on chip memory was used on the same machine to execute the GPGPU test cases. The size of the on chip memory enforces the maximum size of the GPGPU test cases which can be run. Data transfer time comprises: initial H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ array transfers, intermediary summation data between kernel calls, the weighting factors f_h between iterations, and finally the probability estimates $p_{\bar{b}}$ once convergence is reached. For the test cases, block dimensions were set to 32×32, which breaks each single dimensional array (H_h, $B_{\bar{b}}$, f_h, $p_{\bar{b}}$) into 32 element sections and the $c_{h,\bar{b}}$ into 32×32 elements.

Chapter 6

Results and Discussion

Our evaluation focuses on two key aspects of WHAM: validation of the PMF estimates and measurement of performance gained from porting the algorithm from CPU to GPGPU architectures. Validation is required to prove the test harness generates samples comparable to Umbrella Sampling simulations and to verify that the WHAM implementations compute the correct PMF from these samples. A quantitative measure of accuracy (RMSD) is also crucial to ensure there is minimal PMF estimate variation across all test cases, since any speed-up is inconsequential if the trade off is a significant reduction in PMF accuracy. Additionally, the RMSD indicates the expected error range of the WHAM implementations.

As part of the evaluation both speed and speed-up are analysed. A fundamental measurement of performance of a parallel algorithm is the speed-up attained over its sequential counterpart. Speed-up is the ratio of the parallel to sequential execution speed, where, for this evaluation, speed is defined as the problem size (product of number of histograms and bins) over the average time per iteration for each test case.

6.1 Validation of the Test Harness

Validation affirms that both modules developed to evaluate WHAM (Chapter 5) function as expected. Validation of the test harness and our WHAM implementations is done by comparison of the PMF produced from the statistical sam-

ples (discussed in Chapter 5) by the Grossfield reference WHAM and our GPU WHAM implementations with the original analytical PMF (Fig. 6.1). The maximum deviation of the Grossfield implementation (red line), at the 9° bin centre, is approximately 8% and that of the GPGPU implementation (green line), at 99°, approximately 11%. Taking into account the minimal number of samples, histograms and bins employed, the figure clearly shows that both implementations exhibit very similar deviations from the analytic PMF, with an RMSD of 0.43 for the Grossfield implementation and 0.46 for our implementation. The test harness

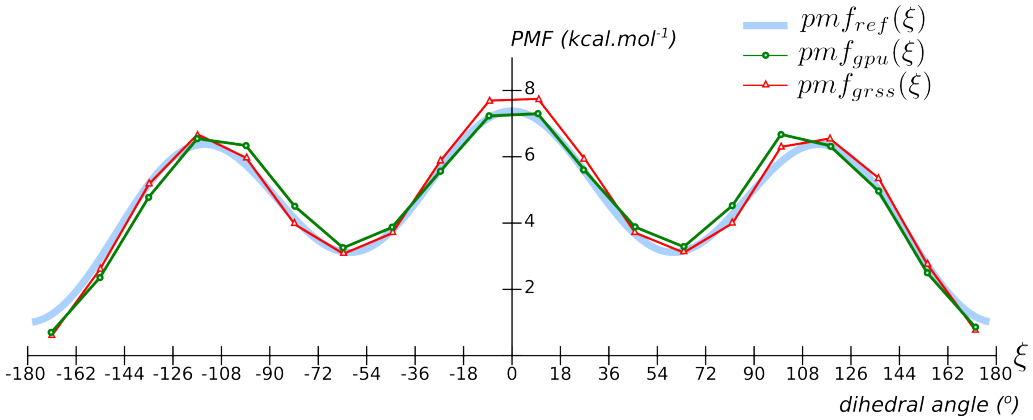


Figure 6.1: Comparison of three PMFs of the butane example: the analytic reference PMF($pmf_{ref}(\xi)$), used to generate samples, and estimates computed by the Grossfield($pmf_{grss}(\xi)$) and GPGPU($pmf_{gpu}(\xi)$) WHAM implementations. The reference PMF is rendered as a thick, smooth curve and the estimates as connected points in the center of each 18° bin interval.

is validated by the fact that the Grossfield WHAM PMF estimates align with the analytic PMF used by the test harness to generate samples. With the test harness validated, the samples were then employed to validate the CPU and GPGPU implementations. This proved a valuable tool during the development and optimization phases of both implementations, since all modifications made could be rapidly evaluated. In addition, the test harness assists in efficient selection of Umbrella Sampling parameters (umbrella spacing and spring constants) since it can rapidly expose unreliable configurations.

6.2 Validation of the Test Case PMFs

The PMF estimates generated with the test cases were validated against the two-dimensional analytic reference PMF (Sec. 5.4). This was carried out by calculation of the RMSD for each test case, to confirm consistency across all cases. Table 6.1 verifies that all test cases were equally successful, as seen by the small deviation in RMSD values. The RMSD values are presented in grid format where each row lists the number of histograms (19×19 to 91×91) and each column list the bin intervals of histograms (30×30 to 180×180). PMF estimates of the four test cases labelled (a), (b), (c), (d) are plotted in Fig. 6.2 below.

		RMSD				
		No. of Bins				
		30×30	60×60	90×90	120×120	180×180
No. of Histograms	19×19	(a) 1.76	1.76	1.75	1.73	(b) 1.72
	37×37	1.61	1.61	1.61	1.61	1.60
	61×61	1.61	1.61	1.61	1.61	1.61
	91×91	(c) 1.61	1.60	1.60	1.60	(d) 1.60

Table 6.1: Grid of RMSD measurements of all test cases against the 2-dimensional analytic PMF. Each row represents an increase in the number of histograms with the 5 variations of bin intervals. Four test case PMFs, indicated with (a), (b), (c) and (d) are illustrated below in Fig. 6.2.

The overall consistency in RMSD values across the range of test cases suggest that problem size has very little influence on accuracy of the PMF estimates. In practice, accuracy depends on more factors than the number of histograms and bins, such as the number of samples taken and the umbrella overlap. The greatest variation in the table is from the 19×19 to 37×37 histogram group. However, after this, there is very little change, which suggests that accuracy approaches its limit between these histogram groups. The increase in number of bins appears to have a minimal effect on accuracy, with the only notable change occurring in the 19×19 histogram group, which increases in accuracy as the number of bins increases. This increase in accuracy can be misleading, as will be demonstrated next by the plots in Fig. 6.2. Nonetheless, the overall consistency in accuracy, shows little correlation between problem size and accuracy which implies the problem size can

increase beyond that of the test cases. Of prime importance, however, is that the RMSD values verify that all test cases generate valid PMF estimates.

In addition, surface plots of four test case PMFs (19×19 and 37×37 histograms with 30×30 and 180×180 bins each) illustrate how the accuracy of the PMF estimates varies with problem size. Fig. 6.2 shows surface plots from four of the twenty test cases (Table 6.1 (a) - (d)), to illustrate the effect of the variation of histograms and bins. These four plots, arranged in a grid, show the combination of the minimum and maximum numbers of histograms (y-axis) and bins (x-axis).

Fig. 6.2 (a) and (c) illustrate PMF estimates generated with the minimum number of bins (30×30), which produce sparser results. These sparse estimates are adequate for the two-dimensional test PMF, since the energy barriers are far wider than the bins. However, in scenarios with steep energy barriers, a greater number of bins would be required, such as those of plots (b) and (d) which produce finer grained results allowing sharper PMF curves. The peak PMF regions of (a) and (b), the minimum number of histograms, are less smooth than the bases, which accounts for the higher RMSD values. However these RMSD values are only 2% less accurate than the substantially higher number of histograms (91×91) in (c) and (d). RMSD values give a measure of overall accuracy and do not necessarily equate to better accuracy at specific points. In the 19×19 histogram case (b), more bins may result in a less accurate estimate due to the variance of the estimates in the higher PMF regions, a consequence of the decrease in sample density per bin. As mentioned earlier, this can be misleading, since the overall accuracy is good but readings taken at points where the PMF estimates fluctuate greatly may yield unpredictable results. The choice of the number of histograms and bins is best determined on a case-by-case basis. If there is any prior indication of the PMF landscape, then the test harness can prove a useful tool to estimate the most efficient choice of Umbrella Sampling configuration.

The graphs of PMF estimates in Fig. 6.2 verify that all test cases are consistent with the analytic reference PMF. The consistency of the PMF estimates of all test cases provides assurance that the performance measurements are not biased by accuracy deviations. In addition, these graphs also verify that the test harness and WHAM implementations function correctly when extended to two-dimensional PMFs.

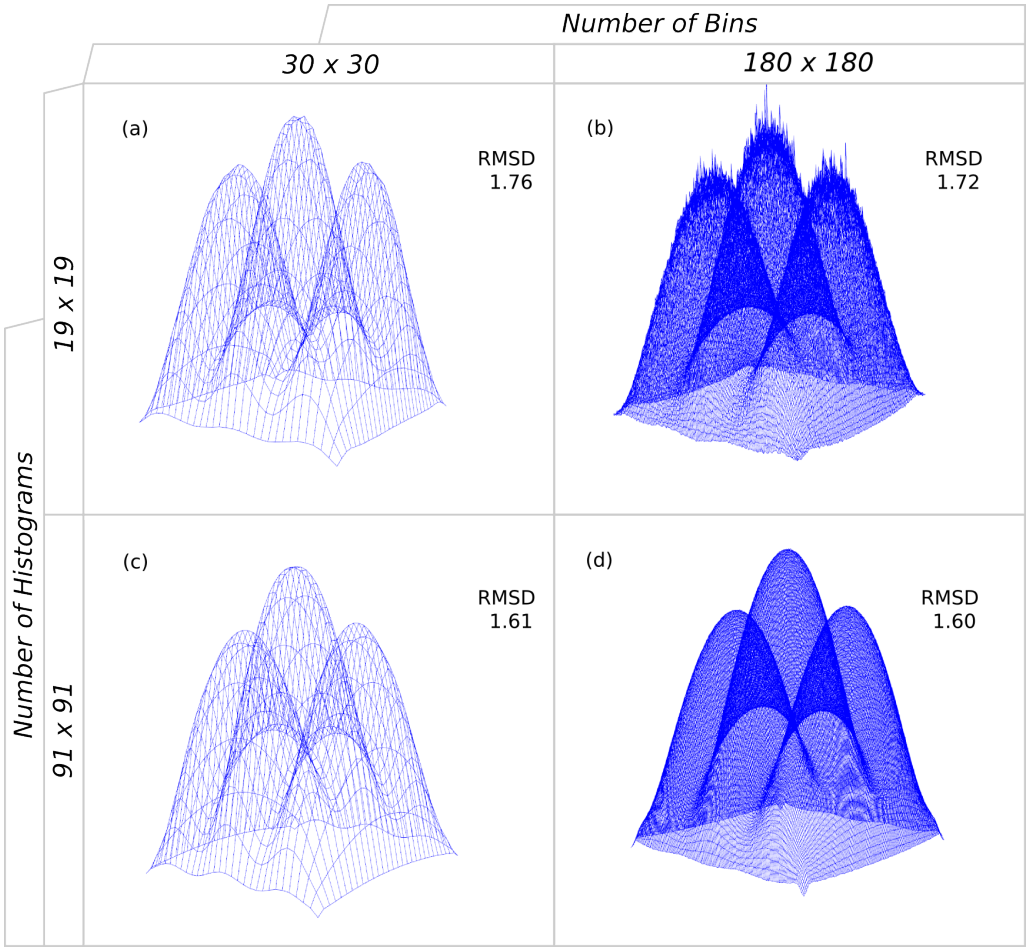


Figure 6.2: Surface plots of a 2D PMF computed by WHAM. Four test cases are shown to illustrate effects of variation of the sizes of H (y-axis) and B (x-axis). The Root Mean Square Deviation (RMSD), given alongside, represents the difference between the PMF computed by WHAM and the reference PMF. The smaller the value of the RMSD, the better the overall accuracy.

6.3 GPGPU WHAM Run-time Profile

Fig. 6.3 shows the run-time profile of 19×19 histograms and 180×180 bins test case (Table 6.1(b)), demonstrating WHAM's suitability to the GPUPGU architecture. The profile illustrates the time taken by the various stages of GPGPU program execution. Initial memory transfer (MemCpy), the data transfer of the H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ arrays from the CPU to GPU, takes a substantial block of time at start-up (the initial CPU start-up time of approximately 0.032s, which includes reading

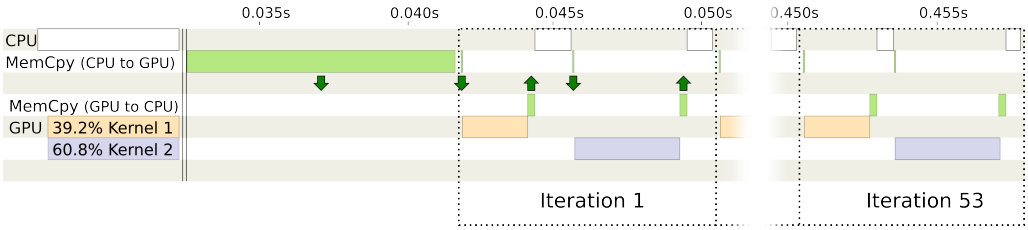


Figure 6.3: Visual profile of the run of a single test case. The x-axis shows the timeline in seconds and each row distinguishes a stage of execution: CPU execution, memory transfers and CUDA kernels. The top row identifies all CPU processing times. The row labelled MemCpy (CPU to GPU) indicates times taken to transfer memory from the CPU to GPU (downwards arrows) and vice versa for row MemCpy (GPU to CPU) (upwards arrows). The time blocks for kernel 1 and kernel 2, are indicated for each iteration (the first and last iterations are illustrated).

the array data files, is omitted from the profile for brevity). MemCpy is followed by a cycle of WHAM iterations, demarcated by dashed lines, which repeat until convergence, at 53 iterations in this case (the first and last iterations are shown). The initial data transfer is followed by the GPGPU iteration steps (Alg. 3). For the first iteration, data transfer (up and down arrows) of the weighting factors (f_h), the intermediary and final column summations ($interColSum_{\bar{b}}$ and $interColSum_{\bar{b}}$), and the intermediary row summations ($interRowSum_h$) account for approximately 7% of the iteration time. CPU processing accounts for approximately 25%, which includes the final column and row summations, the calculation of new weighting factors and the convergence check. The GPU computations take the majority share (68%) of total iteration time, spilt into Kernels 1 and 2 by the ratio indicated in the process names. For this test case, kernel 2 takes the majority of the time since test case (b) has a far greater number of bins (180×180) than histograms 19×19, making the calculation of the intermediary row summations ($interRowSum_h$) the dominant GPGPU step. This would alter for different test cases.

As is clear in Fig. 6.3, initial memory transfer takes a longer time than a single iteration. However, it is unusual that the WHAM algorithm converges in a single iteration, for example, the figure depicts 53 iterations, and the larger test cases take over 400. Typically, the initial data transfer accounts for a small portion of the total time, e.g. less than 2% for test case (b). This proves to be a

considerable advantage of the WHAM algorithm, since the bulk of the data are transferred at start-up and are unchanged from iteration to iteration. This profile clearly demonstrates that WHAM is amenable to parallelism, since the GPU computation predominates and there is minimal data transfer time per iteration. The CPU computation accounts for the next highest time component, which suggests opportunity for further heterogeneous parallelism. The CPU calculations of our implementations are all single threaded, but this can easily be extended to multiple threads on a multi-core CPU. Profiles, such as the one in Fig. 6.3, are valuable development tools since they identify bottle necks which prioritize opportunities for improvement and they verify that the WHAM algorithm is suitable for GPGPU implementation.

6.4 Implementation Performance

Performance improvement of a parallel algorithm is measured in terms of speed-up, and, in addition, GPGPU parallelism speed-up is problem size dependent, due to data transfer and occupancy. Table 6.2 lists the size dependent speed-ups of the test cases. The maximum speed-up achieved is 19.14× for the test case with a problem size of 61×61 histograms and 180×180 (shaded block). Problem size for each test case, the combination of the number of histograms and the number of bins, is shown as the first two columns of the table. The raw measurements, total execution time and the number of iterations (taken from an average of 10 runs) were recorded for the CPU and GPGPU implementations as part of their run-time process. The average time per iteration, one column for the CPU (column 5) and one for the GPU (column 7), is computed from the total execution time divided by the number of iterations. Average time per iteration is a comparable time metric since the test cases have unequal numbers of iterations (column 3).

Surprisingly, the maximum speed-up (19.14×) is not achieved with the maximum problem size. Speed-up tends to increase as the problem size increases so it would be reasonable to assume that the largest problem size exhibits the greatest speed-up. However, there is an unexpected decrease in the last histogram group (91×91). Although not obvious from the table, this is a consequence of an increase in CPU speed rather than a decrease in GPU speed and, illustrated later

			CPU		GPU		
No. of Histograms	No. of Bins	No. of Iterations[1]	Average Total Time (ms)	Average Time per Iteration (ms)	Average Total Time (ms)	Average Time per Iteration (ms)	Speed-up CPU/ GPU
19×19	30×30	52	173	3	68	1.3	2.55
	60×60	53	756	14	102	1.9	7.39
	90×90	53	1767	33	160	3.0	11.03
	120×120	53	3205	60	240	4.5	13.34
	180×180	53	7334	138	461	8.7	15.89
37×37	30×30	232	2888	12	244	1.1	11.82
	60×60	233	12599	54	784	3.4	16.08
	90×90	234	29021	124	1686	7.2	17.21
	120×120	234	52056	222	2878	12.3	18.09
	180×180	234	118822	508	6322	27.0	18.79
61×61	30×30	227	8024	35	545	2.4	14.73
	60×60	228	33574	147	1928	8.5	17.42
	90×90	228	77084	338	4246	18.6	18.15
	120×120	228	138116	606	7330	32.1	18.84
	180×180	228	315983	1,386	16511	72.4	19.14
91×91	30×30	408	25363	62	1983	4.9	12.79
	60×60	410	104298	254	7346	17.9	14.20
	90×90	411	241101	587	16394	39.9	14.71
	120×120	411	428575	1,043	28952	70.4	14.80
	180×180	411	985780	2,398	65282	158.8	15.10

[1] The Number of iterations is the same for both CPU and GPU versions.

Table 6.2: WHAM performance metrics for 20 test cases: Each block delineates the number of histograms with five variations of the number of bins. The problem size is indicated by the first two columns, the combination of the number of histograms and number of bins. The number of iterations each test case took is listed in the next column followed by the total time and average time per iteration, first for the CPU and then the GPU. The speed-up of the GPU over the CPU implementation is listed in the final column with the highest value highlighted.

in Fig. 6.4.

An interesting observation is the short total times (985780 ms and 65282 ms) of the largest problem size (91×91 histograms and 180×180 bins - roughly 268M elements). The total run-time for this test case was approximately 16 minutes on CPU and 1 minute on the GPU. Both these times are significantly shorter than the time required to run the Umbrella Sampling simulations (91×91 is 8281 simulations). Seemingly, this fact suggests that parallel implementation is not justified, however there are a number of factors to take into consideration. Firstly,

the test cases are based on a smooth analytical reference PMF with low energy barriers, resulting in few iterations required to converge. In cases where the number of iterations may reach upwards of 10000, the total CPU run-time exceeds 6 hours, while the GPU run-time is approximately 26 minutes. Another factor to consider is the impracticality of running many Umbrella Sampling simulations. In scenarios where multiple reaction coordinates are under investigation, special techniques are required to selectively sample reaction coordinate regions to reduce the number of simulations to a practical number. An informal exercise was conducted where 30 simulations were run measuring 12 reaction coordinates. The n-dimensional GPGPU WHAM implementation was then applied by padding the unsampled regions of the H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ arrays with zeros. This significantly reduced the disparity between simulation time and WHAM time, however, the problem of sparse PMF estimates this generates is currently being addressed by the author.

Fig. 6.4 shows the speeds and speed-ups for the test cases as the number of histograms and number of bins varies. This indicates the problem size where GPGPU execution becomes viable and the maximum speed-up expected beyond the tested problem sizes. Fig. 6.4(a) and (b) show the problem size dependent GPU (blue) and CPU (green) speed variation as the number of histograms increases (a) and as the number of bins increases (b). Figs. (c) and (d) show the corresponding speed-up against the same histogram and bin increases (problem size variations are described in Sec. 5.4 with Fig. 5.7).

The speed graphs (Fig. 6.4(a) and (b)) confirm that GPGPU performance varies with problem size, whereas the CPU version is approximately constant. The jump from the 19×19 to 37×37 histogram groups exhibits the largest GPU speed increase. Over the subsequent histogram groups this speed converges to a value of about 1800K elements/msec. This convergence suggests that the GPGPU implementation will not exceed a speed of 1800K elements/msec as the problem size increases beyond what is tested, which confirms the test case problem sizes are sufficient. The speed-up graphs (c) and (d) indicate approximately 19× maximum parallel speed-up, where (d) suggests that this speed-up will converge to this value.

An unexpected observation is the decrease in speed-up of the 91×91 histogram test cases in graph (c). This is explained by the implementations' speeds in graph (a). This graph shows that the GPU speeds of all series increases from the 61×61

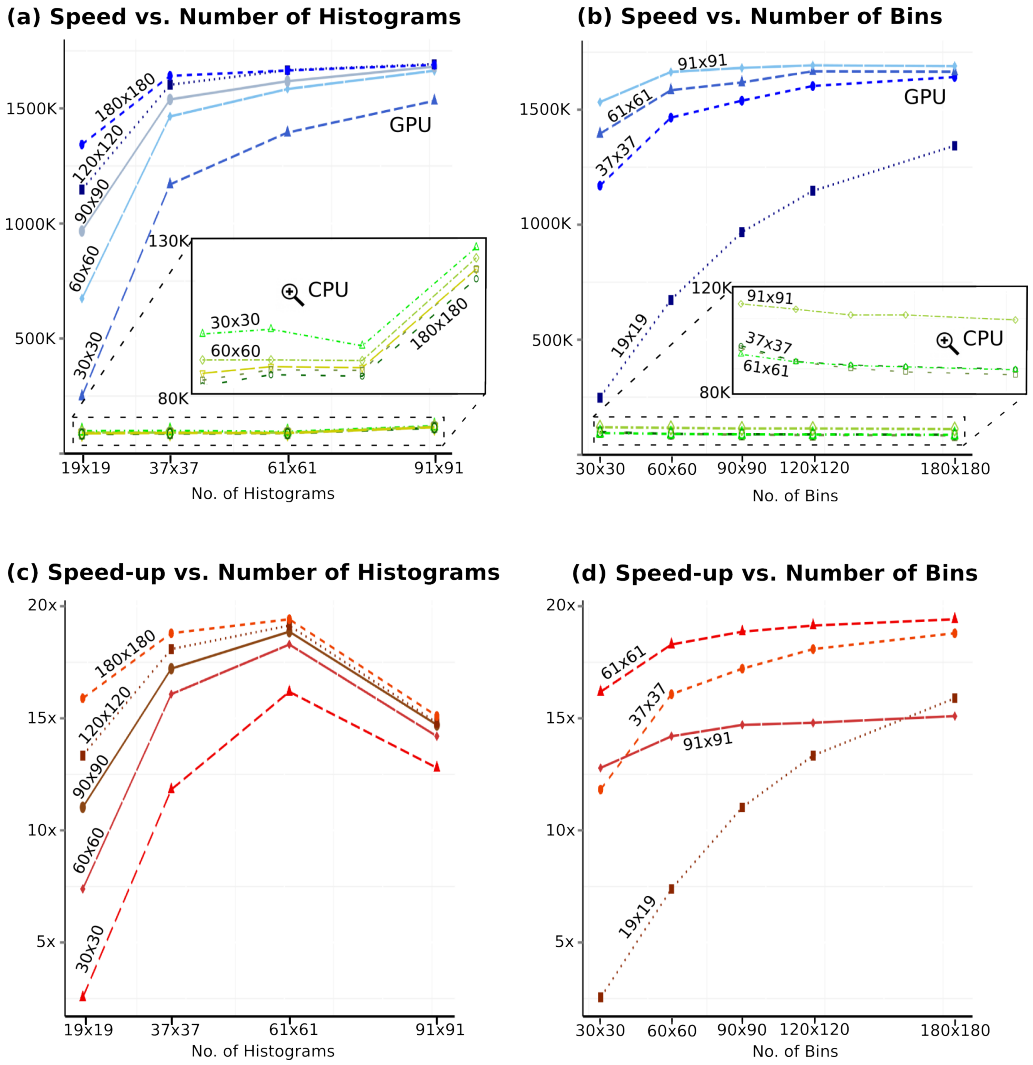


Figure 6.4: Performance of the CPU and GPU implementations versus the number of histograms and number of bins. Performance is measured by two factors: the speed of the implementations (a and b) and the speed-up (c and d). GPU and CPU speeds are shown on the same axis with the inset providing finer detail of the CPU speeds. The left hand side (a and c) illustrates the performance as the number of histograms is varied, (Fig. 5.7 (a)) where the number of bins is held fixed for each line. The right hand side shows the speed and speed-up as the number of bins is varied where each line represents a fixed number of histograms (Fig. 5.7 (b)).

to the 91×91 histogram groups, as expected. However, the inset shows that, for all bin series, the CPU speed of the 91×91 histogram group increases unexpect-

edly by a relatively constant amount (approximately 30%). The reason for this is unknown. Hence, the decrease in speed-up is a consequence of the increased CPU implementation speed, rather than a decrease in GPGPU implementation performance. The effect of the CPU speed increase is visible in the other graphs: (b) shows the jump in CPU speed in the zoom box and (d) shows the overall consistent decrease in speed-up for the 91×91 histogram group.

The results affirm the validity of the test harness and of the WHAM implementations by comparison with the reference Grossfield WHAM implementation. A performance comparison was not conducted against the Grossfield implementation since it has not been optimized for performance and our WHAM implementations do not parallelize all phases of WHAM but only the iteration of the equations. However, the maximum speed-up (19.14×) of the GPGPU implementation confirms the feasibility of a parallel implementation of WHAM, and future steps comprise parallelizing the creation of the histogram aggregates (H_h, $B_{\bar{b}}$) and distribution biases ($c_{h,\bar{b}}$).

6.5 Comparison Against Alternative Sequential Implementations

Grossfield WHAM has applications for one and two dimensional WHAM calculations. Version 2.0.8, released on October 8th 2013, contains performance enhancements for the two dimensional implementation which increases calculation speed by up to 100 times faster than the previous version. When initial investigations of the available WHAM implementations was conducted, the sequential version written for this research far outperformed the Grossfield implementation. This was expected since high performance was not the goal of the Grossfield applications. However, a measurement of the WHAM times for the 61×61 histograms and 180×180 bins data set was recently taken and it was found to be approximately 3 times faster than the sequential version used to obtain the results in this thesis. This reduces the GPU implementation speed-up to approximately 6 times faster than the fastest current sequential two dimensional WHAM implementation. Inspection of the code revealed that a portion of this performance improvement

can be attributed to similar caching mechanisms used in our algorithm to avoid repeated calculation of the distribution biases ($c_{h,\bar{b}}$).

A number of alternative sequential implementations were developed during the initial investigation stages to gain an understanding of the performance of different technology options. Three Java implementations were developed: a sequential version, a classic Java multi-threaded version and multi-core version which employed the more recent Fork/Join framework. When run on an 8 core Intel i7 processor a $4\times$ speed-up was observed by the two parallel versions compared to the sequential version. There was little observable difference between the two parallel versions since the operating system allowed for the Java threads to run on separate cores. Sequential C and parallel OpenMP versions were developed with the same algorithm as the Java versions. A similar $4\times$ speed-up was observed from the sequential to parallel versions. The half-linear speed-ups observed in both cases indicated that a modification of the computational model was required to achieve increased performance. Additionally, these observations led to the decision to focus on a sequential C implementation compared to a parallel GPGPU implementation to constrain the scope of the thesis.

Chapter 7

Conclusions

We found that the WHAM algorithm is well suited to SIMT parallelism and yields a speed-up of approximately 19 times over the CPU version, making the GPGPU implementation a feasible option. One of the key advantages of the WHAM algorithm comes from the fact that data sets need only be transferred to the GPU once, before the algorithm commences, after which point the compute time outweighs the intermediary data transfer times. This, combined with pre-computing the H_h, $B_{\bar{b}}$ and $c_{h,\bar{b}}$ arrays, allows large, n-dimensional data sets to execute successfully and efficiently. The facility to process n-dimensional simulation data provides the means to study more complex molecular systems, such as systems where the PMF is dependent on multiple reaction coordinates. However, there are still practical limits on the number of reaction coordinates measurable, since every extra reaction coordinate results in an exponential rise in problem size. This exponential increase is especially problematic when considering the number of Molecular Dynamics simulations required; too many would become impractically time consuming. In this regard, the test harness proved a valuable alternative to generating samples on Molecular Dynamics platforms and for validation of the WHAM implementations. The test harness has potential use in future development of Molecular Dynamics simulation and post simulation data analysis systems, for example, systems which allow selective Umbrella Sampling in isolated regions of conformational space. These systems are not limited to Umbrella Sampling simulations since the test harness can be easily extended to generated samples analogous to basic Molecular

Dynamics simulations or to other enhanced sampling techniques.

The speed-up that may have been obtained from paralellization of the histogram creation and array construction phases were purposefully omitted from this study. This was to restrict the focus primarily to the speed-up obtained with the WHAM equations. Future enhancements could include GPU implementation of all pre- and post-processing to provide a complete WHAM package, along with mechanisms to allow a reduced number of simulations as inputs. The utility of the test harness and the viable performance increase of the WHAM algorithm demonstrated in this study provide valuable starting points towards analysis of ever more complex molecular systems.

Appendix A

CUDA kernels

Listings of the CUDA Kernels used in the GPGPU implementation (Alg. 3). Each kernel begins with calculation of the required indices to retrieve data from global memory. Next a shared memory block is populated (s_HFC and s_CP) and sum reduced. The summations in each block are saved to global memory and transferred to the CPU to complete the summation.

Kernel 1 : Calculate $H_h * c_{h,\bar{b}} * f_{h_{old}}$ and sum across histograms.

```
static __global__ void dColSum(int* d_H, float* d_C, float↩
   * d_Fold, float* d_col_sum_inter, unsigned int H, ↩
   unsigned int B) {

        // Get indices
        unsigned int s_tx = threadIdx.x;
        unsigned int d_tx = BLOCKSIZE * blockIdx.x + ↩
           threadIdx.x;
        unsigned int d_size_x = gridDim.x * BLOCKSIZE;
        unsigned int s_ty = threadIdx.y;
        unsigned int d_ty = BLOCKSIZE * blockIdx.y + ↩
           threadIdx.y;
        unsigned int d_x_y_flat = d_ty * d_size_x + d_tx;

        // Create shared mem HFC
        __shared__ float s_HFC[BLOCKSIZE][BLOCKSIZE];
```

```
        // Calculate HFC
        // @@ 14 ms  - 10 ms with pre-cached HC
        s_HFC[s_ty][s_tx] = d_H[d_ty] * d_Fold[d_ty] * d_C↩
           [d_x_y_flat];

        __syncthreads();

        // Sum reduce each column of HFC
        // @@ 16 ms
        for (unsigned int stride = blockDim.y / 2; stride ↩
           >= 1; stride >>= 1) {
                __syncthreads();
                if (s_ty < stride) {
                        s_HFC[s_ty][s_tx] += s_HFC[s_ty + ↩
                           stride][s_tx];
                }
        }
        __syncthreads();

        // Save row 0 to global mem
        // @@ 4 ms
        if (s_ty == 0) {
                d_col_sum_inter[d_size_x * blockIdx.y + ↩
                   d_tx] = s_HFC[0][s_tx];
        }

}
```

Kernel 2 : Calculate $P_{\bar{b}}$ and sum across bins.

```
static __global__ void dRowSum(int* d_B, float* d_C, float* d_P, float* d_col_sum_inter, float* d_row_sum_inter, unsigned int H, unsigned int B) {

	// get indecies
	unsigned int s_tx = threadIdx.x;
	unsigned int d_tx = BLOCKSIZE * blockIdx.x + threadIdx.x;
	unsigned int d_size_x = gridDim.x * BLOCKSIZE;

	unsigned int s_ty = threadIdx.y;
	unsigned int d_ty = BLOCKSIZE * blockIdx.y + threadIdx.y;

	unsigned int d_x_y_flat = d_ty * d_size_x + d_tx;

	// Work out P for each bin
	float P = 0;
	float denom = d_col_sum_inter[d_tx];

	// @@ 4 ms
	if (denom != 0) {
		P = d_B[d_tx] / denom;
	}
	__syncthreads();

	// Create shared mem CP
	__shared__ float s_CP[BLOCKSIZE][BLOCKSIZE];

	// @@ 8 ms
	s_CP[s_tx][s_ty] = P * d_C[d_x_y_flat];
	__syncthreads();

	// Sum reduce each column of the CP Transposed - column summation is faster than row summation.
	// @@ 20 ms
	for (unsigned int stride = blockDim.y / 2; stride >= 1; stride >>= 1) {
		__syncthreads();
		if (s_ty < stride) {
			s_CP[s_ty][s_tx] += s_CP[s_ty + stride][s_tx];
```

```
                }
        }
        __syncthreads();

        // Save row 0 to global mem - transposed
        // @@4 ms
        if (s_tx == 0) {
                d_row_sum_inter[d_ty * gridDim.x + ↩
                   blockIdx.x] = s_CP[0][s_ty];
        }

}
```

Bibliography

[1] Shankar Kumar, John M Rosenberg, Djamal Bouzida, Robert H Swendsen, and Peter A Kollman. The Weighted Histogram Analysis Method for Free-Energy Calculations on Biomolecules. I. The Method *Journal of Computational Chemistry*, 13(8):1011–1021, October 1992.

[2] Jacob D. Durrant and J. Andrew McCammon. Molecular dynamics simulations and drug discovery. *BMC Biology*, 9(1):71, October 2011.

[3] Michael Levitt and Arieh Warshel. Computer simulation of protein folding. *Nature*, 253(5494):694–698, February 1975.

[4] John G. Kirkwood. Statistical Mechanics of Liquid Solutions. *Chemical Reviews*, 19(3):275–307, December 1936.

[5] G. M. Torrie and J. P. Valleau. Nonphysical Sampling Distributions in Monte Carlo Free-Energy Estimation: Umbrella Sampling. *Journal of Computational Physics*, 23(2):187–199, February 1977.

[6] B. R. Brooks, C. L. Brooks, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus. CHARMM: The Biomolecular Simulation Program. *Journal of Computational Chemistry*, 30(10):1545–1614,2009.

[7] Jochen S. Hub, Bert L. de Groot, and David van der Spoel. g_wham A Free Weighted Histogram Analysis Implementation Including Robust Error and Autocorrelation Estimates. *Journal of Chemical Theory and Computation.*, 6(12):3713–3720, August 2010.

[8] Grossfield, Alan, "WHAM: the weighted histogram analysis method", version 2.0.7, http://membrane.urmc.rochester.edu/content/wham

[9] Jeffrey K. Noel, Paul C. Whitford, Karissa Y. Sanbonmatsu, and José N. Onuchic. SMOG@ctbp: simplified deployment of structure-based models in GROMACS. *Nucleic Acids Research*, 38:W657–W661, July 2010.

[10] Tristan Bereau and Robert H. Swendsen. Optimized convergence for multiple histogram analysis. *Journal of Computational Physics*, 228(17):6119–6129, September 2009.

[11] Gongpu Zhao, Juan R. Perilla, Ernest L. Yufenyuy, Xin Meng, Bo Chen, Jiying Ning, Jinwoo Ahn, Angela M. Gronenborn, Klaus Schulten, Christopher Aiken and Peijun Zhang. Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics. *Nature*, 497(7451):643–646, May 2013.

[12] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227(10):5342–5359, January 2008.

[13] John E. Stone, David J. Hardy, Ivan S. Ufimtsev, and Klaus Schulten. GPU-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modelling*, 29(2):116–125, September 2010.

[14] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[15] Berk Hess, Carsten Kutzner, David van der Spoel, and Erik Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, March 2008.

[16] Terrell L. Hill. Steric Effects. I. Van Der Waals Potential Energy Curves. *The Journal of Chemical Physics*, 16(4):399–404, April 1948.

[17] J. P. Valleau and D. N. Card. Monte Carlo Estimation of the Free Energy by Multistage Sampling. *The Journal of Chemical Physics*, 57(12):5457–5462, December 1972.

[18] Benoît Roux. The calculation of the potential of mean force using computer simulations. *Computer Physics Communications*, 91(13):275–282, September 1995.

[19] Z. W. Salsburg, J. D. Jacobson, W. Fickett, and W. W. Wood. Application of the Monte Carlo Method to the Lattice-Gas Model. I. Two-Dimensional Triangular Lattice. *The Journal of Chemical Physics*, 30(1):65–72, January 1959.

[20] Alan M. Ferrenberg and Robert H. Swendsen. New Monte Carlo Technique for Studying Phase Transitions. *Physical Review Letters*, 61(23):2635–2638, December 1988.

[21] Alan M. Ferrenberg and Robert H. Swendsen. New Monte Carlo Technique for Studying Phase Transitions ERRATA. *Physical Review Letters*, 63(15):1658, October 1989.

[22] Alan M. Ferrenberg and Robert H. Swendsen. Optimized Monte Carlo Data Analysis. *Physical Review Letters*, 63(12):1195–1198, September 1989.

[23] Daniel J. Sindhikara. Modular reweighting software for statistical mechanical analysis of biased equilibrium data. *Computer Physics Communications*, 183(7):1560–1561, July 2012.

[24] John D. Chodera, William C. Swope, Jed W. Pitera, Chaok Seok, and Ken A. Dill. Use of the Weighted Histogram Analysis Method for the Analysis of Simulated and Parallel Tempering Simulations. *Journal of Chemical Theory and Computation*, 3(1):26–41, 2007.

[25] Michael R. Shirts and John D. Chodera. Statistically optimal analysis of samples from multiple equilibrium states. *The Journal of Chemical Physics*, 129(12), September 2008.

[26] Zhiqiang Tan, Emilio Gallicchio, Mauro Lapelosa, and Ronald M Levy. Theory of binless multi-state free energy estimation with applications to protein-ligand binding. *The Journal of Chemical Physics*, 136(14), April 2012.

[27] Peter Eastman, Mark S. Friedrichs, John D. Chodera, Randall J. Radmer, Christopher M. Bruns, Joy P. Ku, Kyle A. Beauchamp, Thomas J. Lane, Lee-Ping Wang, Diwakar Shukla, Tony Tye, Mike Houston, Timo Stich, Christoph Klein, Michael R. Shirts, and Vijay S. Pande. OpenMM 4: OpenMM 4: A Reusable, Extensible, Hardware Independent Library for High Performance Molecular Simulation. *Journal of Chemical Theory and Computation*, 9(1):461–469, January 2013.

[28] Rich Hickey. The Clojure programming language. In *Proceedings of the 2008 Symposium on Dynamic Languages*, July 2008.

[29] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable Parallel Programming with CUDA. *ACM Queue*, 6(2):4053, March 2008.

[30] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier, February 2010.

[31] Mark Harris. Optimizing parallel reduction in CUDA. *NVIDIA Developer Technology*, 2, September 2007.