Department of Electrical and Computer
Engineering
University of Thessaly, Volos, Greece

Μελέτη επίδρασης προσεγγιστικών υπολογισμών στην ποιότητα αποτελεσμάτων, την επίδοση και το ενεργειακό αποτύπωμα επιστημονικών εφαρμογών / Studying the effects of approximate computing on the quality of results, the performance and the energy footprint of scientific applications

Panos S. Koutsovasilis

March 23, 2016

ACKNOWLEDGMENTS

I would like to thank my supervisors Christos D. Antonopoulos, Nikolaos Bellas and Spyros Lalis for their help and guidance throughout this process, their ideas and feedback have been absolutely invaluable.

This work has been supported by (a) The European Commissions 7th Framework Programme (FP7/2007- 2013) under grant agreement FP7-323872 (Project SCoRPiO) and (b) The Aristeia II action (grant agreement 5211, project Centaurus of the operational program Education and Life-long Learning which is co-funded by the European Social Fund and Greek national resources).

"We cannot solve our problems with the same thinking we used when we created them"

— *Albert Einsten*

ΠΕΡΙΛΗΨΗ

Μία από τις μεγαλύτερες προκλήσεις που αντιμετωπίζουν οι μηχανικοί υπολογιστών, είναι η δημιουργία υπολογιστικών συστημάτων με ιδιαίτερα μειωμένη κατανάλωση ενέργειας, όσο αναφορά τόσο το υλικό όσο και το λειτουργικό κομμάτι του συστήματος. Μία από τις πιο δημοφιλής μεθόδους για την επίτευξη ενεργειακής αυτονομίας για το υλικό κομμάτι ενός υπολογιστικού συστήματος, είναι η ενσωμάτωση υπολογιστικών μονάδων που επιδεικνύουν διαφορετικές υπολογιστικές ικανότητες με χαμηλότερα χαρακτηριστικά κατανάλωσης ενέργειας σε σχέση με τις συμβατικές αρχιτεκτονικές (Ετερογένεια - Heterogeneity). Μια άλλη προσέγγιση είναι η παραλλαγή των υπολογιστικών πράξεων μιας εφαρμογής, με κατάλληλες που θα έχουν σαν αποτέλεσμα την μείωση της ενεργειακής κατανάλωσης αλλά και την αλλοίωση των αποτελεσμάτων της εφαρμογής (Προσεγγιστικός υπολογισμός - Approximate Computing). Πιθανές παραλλαγές είναι η απελευθέρωση των σημείων συγχρονισμού που απαιτούνται από την εφαρμογή, η παράλειψη υπολογισμού διάφορων μεταβλητών, κτλ.

Σε αυτή την Μεταπτυχιακή εργασία συνδυάζουμε Heterogeneity και Approximate Computing και αξιολογούμε την επίδραση τους στην ενεργειακή κατανάλωση και την ποιότητα αποτελεσμάτων δύο εφαρμογών. Αφού μεταφέραμε τα υπολογιστικά κομμάτια κάθε εφαρμογής σε ισοδύναμα OpenCL tasks επιλέξαμε το Centaurus API Framework ως τον διαχειριστή της εκτέλεσης για κάθε εφαρμογή. Το Centaurus API Framework προσφέρει ιδιαίτερη ευκολία στον προγραμματιστή, μιας και δεσμεύει/μεταφέρει/απελευθερώνει αυτόματα οποιοδήποτε αντικείμενο μνήμης χρειάζεται κατά την εκτέλεση των αντίστοιχων OpenCL tasks. Επίσης επιτρέπει την εύκολη μετάβαση από ακριβής σε προσεγγιστική, και αντίστροφα, εκτέλεση των διαφόρων OpenCL tasks.

Η πρώτη εφαρμογή είναι η SPSStereo (SPS) η οποία είναι μια εφαρμογή υπολογιστής οράσεως. Η SPS υπολογίζει μια εκτίμηση της πυκνότητας του βάθους, καθώς και την εξαγωγή ορίων (π.χ. ορίων έμφραξης αντικειμένων), έχοντας σαν δεδομένα ένα ζευγάρι εικόνων καθώς αυτό προκύπτει από την φωτογράφιση μιας στατικής σκηνής από μία στερεοσκοπική κάμερα. Η SPS αποτελείται από δύο μέρη, ένα μέρος υπεύθυνο για την εκτενής αναγνώριση ομοιών περιοχών ανάμεσα στις δύο εικόνες (SGM), και ένα δεύτερο για την ταξινόμηση και την εξαγωγή των ορίων. Το πιο χρονοβόρο μέρος και κατ΄ επέκταση πιο ενεργειοβόρο είναι το SGM , για το οποίο και υλοποιήσαμε μια ισοδύναμη OpenCL έκδοση, και ως προσεγγιστική εκτέλεση αφαιρέσαμε την υπολογιστικά ακριβή ανάγκη των επιμέρους υπολογιστικών διαδικασιών για συγχρονισμό.

Η δεύτερη εφαρμογή είναι η Molecular Dynamics (MD) , η οποία προσομοιώνει τις κινηματικές ιδιότητες, όπως θέση, ταχύτητα κτλ. ατόμων

υγρού Άργον τα οποία βρίσκονται υπό την επίδραση ενός δυναμικού Lennard-Jones και περιορισμένα σε ένα νοητό κουτί. Η MD αλγοριθμικά χωρίζεται σε τρία μέρη, Αρχικοποίηση, Εξισορρόπηση και Προσομοίωση. Σαν προσεγγιστικές εκτελέσεις αλλάξαμε το στάδιο της προσομοίωσης, με υπολογισμούς που εξαιρούν κάποια άτομα, και μετατοπίζουν τα άτομα στο πεδίο του χρόνου περισσότερο από αυτό που έχει οριστεί κατά το στάδιο της Αρχικοποίησης.

Στο στάδιο της αξιολόγησης των πειραματικών αποτελεσμάτων, κάθε επιμέρους προσεγγιστική εκτέλεση αξιολογείται για κάθε εφαρμογή, συγκρίνοντας την ποιότητα της εξόδου με την αντίστοιχη της ακριβής εκτέλεσης. Ως μονάδα μέτρησης ποιότητας για την SPS χρησιμοποιήσαμε την μετρική Peak Signal to Noise Ratio (PSNR) . Επιπρόσθετα θεωρούμε πως το μέγιστο όριο οπού και φέρει η ακριβής εκτέλεση είναι τα 44 db και όχι το άπειρο που είναι στην πραγματικότητα για λόγους αναπαράστασης και σύγκρισης των πειραματικών αποτελεσμάτων μεταξύ των διαφορετικών εκτελέσεων. Για την MD χρησιμοποιήσαμε την μετρική της σχετική απόκλισης για τη μέση ενέργεια του συστήματος και την μέση πίεση ανά άτομο. Βάσει των πειραματικών αποτελεσμάτων που προκύπτουν είναι δίκαιο να θεωρήσουμε πως ο συνδυασμός της Heterogeneity και Approximate Computing αποτελεί έναν αποδεκτό συμβιβασμό μεταξύ ενεργειακής κατανάλωσης και απώλειας ποιότητας αποτελεσμάτων μιας εφαρμογής.

ABSTRACT

One major concern of computer engineers is to create more energy efficient computer systems, both in terms of software and hardware. A popular way of gaining energy efficiency is to exploit heterogeneity and accelerator-based systems which combine different architectures with different computational characteristics and provide better performance in terms of energy efficiency in certain computational tasks. Another bold, yet viable alternative; is approximate computing. It tries to minimize the energy footprint of applications at the expense of output quality be either dropping some calculations, relaxing synchronization barriers etc.

In this Msc. thesis we exploit both heterogeneity and approximate computing on two different applications. For each application we implemented an OpenCL version of its computational tasks and we employed Centaurus API Framework as the orchestrator of application execution. Centaurus API Framework is able to invoke OpenCL kernels with minimum effort from the programmer, as it automatically allocates/-transfers any memory object needed. Also it can automatically choose between an accurate or an approximate version of computational task according to the specified energy budget.

The first application is SPSStereo (SPS) which is a computer vision application. It performs a dense depth estimation and boundary labels extraction (such as occlusion boundaries), having as input the left and the right portion of a static scene captured with a stereo camera pair. SPS derives in two parts, one for calculating an extended semi global block matching (SGM) and a plane slanted algorithm for inferring the segmentation and boundary labels extraction. As SGM is the hotspot of SPS, we implemented an OpenCL version and as approximations we relaxed synchronization barriers and drop calculations of SGM inner computational tasks.

The second application is Molecular Dynamics (MD) which simulates kinematic properties such as position, velocity etc of liquid Argon atoms when they act in a kind of force produced by a Lennard-Jones pair potential in a bounded box. MD derives in three parts, *Initialization*, *Equilibration*, *Simulation* in which we implemented the corresponding OpenCL versions and as for approximations we drop calculations in *Simulation*.

In the experimental evaluation we evaluate each exploited approximations in the aforementioned applications by comparing against the output produced by a fully accurate execution of application. For SPSStereo we use Peak Signal to Noise Ratio (PSNR) and we pick that fully accurate execution output is of 44 dB instead of actual infin-

ity dB in fairness to graphical representation and comparison of quality between executions with approximate tasks. For MD we use Relative error for average total system energy, and pressure per atom. According to our evaluation results, combination of heterogeneity and approximate computing is a viable trade-off between energy consumption and quality of output.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

Energy efficiency is critical in systems which are limited in a certain energy budget for consumption. This category of energy-constrained environments contains a variety of systems such as, wearable devices which due to their mobile nature have long up-time expectations, but also large scale HPC systems and data centers, where a significant percentage of the total cost of operation is due to power costs, as well as the cost of building, maintaining and operating the necessary cooling infrastructure. This Msc. thesis field of interest is the impact of exploiting heterogeneity and approximate computing, on an application energy footprint.

Current hardware architecture constraints such as the fact that energy consumption cant be regulated at core-level accordingly to the demands of its computational task, leads to unnecessary lower idle times of computational cores packed in the same chip and results to higher power rate. Thus, without exploiting specialized hardware power rate is not prone to reduction. As a result with current technologies the only reasonable effort to be done is try reducing the total time of application execution and by extension reducing application energy footprint.

A typical solution to achieve optimal performance per watt is heterogeneous computing, mainly expressed as the exploitation of GPUs. GPUs are programmable accelerators, efficiently targeting specific computational patterns. In contrast to conventional CPU-like architectures, GPUs offer massive, partially asynchronous parallel execution through many computational cores. Although their power footprint is slightly higher than that of a typical CPU, they are superior in terms of performance per Watt. At the expense of a few extra Watts, applications that exhibit sufficient parallelism can utilize GPUs to reduce both their execution time and energy footprint.

Another bold approach for substantially boosting energy efficiency is approximate computing. Previous work on approximate computing [1] shows that application energy footprint can substantially reduce by relaxing the need for fully precise or completely deterministic operations. Also the intrinsic resilience in several application domains, such as computer vision and computational physics, to inexactness in their computations, results to a viable trade-off between energy efficiency and acceptable results in aspect of quality. Due to direct correlation of approximation and the quality of results, approximations need

to specifically target particular computational steps which exhibit the aforementioned characteristics and usually require a domain expert to implement them, otherwise results could be unacceptable in terms of quality.

In this Msc. thesis context, we will combine both Heterogeneity and Approximate Computing, implementing Centaurus API framework. Centaurus Framework is able to invoke OpenCL kernels with minimum effort from the programmer. Also it automatically allocates/transfers any memory object needed by OpenCL kernel and can automatically select according to its energy budget if it will invoke an accurate or an approximate version of computational task. In addition all the aforementioned exploiting and the Centaurus API implementation is done on large-scale real world applications which are SPStereo and Molecular Dynamics.

SPStereo (SPS) [2] performs a dense depth estimation and boundary labels extraction (such as occlusion boundaries), having as input the left and the right portion of a static scene captured with a stereo camera pair. The application proposes a new slanted plane algorithm which in contrast to date plane proposals eliminates the time-consuming parts and delivers an algorithm which can easily find use in robotics applications such as autonomous driving. Moreover the aforementioned algorithm consists of two parts, an extended semi global block matching (SGM) part which computes an initial reference disparity semi-depth map. This semi-depth map then is used as input to a plane slanted algorithm for inferring the segmentation and boundary labels extraction.

Molecular Dynamics (MD) context is in general a computer simulation of atoms or molecules which derives from N-Body simulation. In this particular application we simulate the behaviour of liquid Argon molecules restricted in a bounded box. Specifically we simulate kinematic properties such as position, velocity etc of liquid Argon atoms when they act in a kind of force produced by a Lennard-Jones pair potential [3]. MD simulations find appliance in many science domains such as theoretical physics, biochemistry and biophysics. For example they are used to examine atomic-level effects of dynamics, that cannot be observed with naked eye or any other macroscopic technique, such as ion-subplantation.

The rest of the thesis is structured as follows. Section 2 expatiates all the related work in heterogeneous and approximate computing. Section 3 discuss detailed Heterogeneous architectures and frameworks implemented in this Msc. thesis. Section 4 presents a theoretical analysis and suitable approximations exploited of each application. Section 5 presents the experimental evaluation of both applications. Section 6 concludes the thesis and presents directions for future work.

2

BACKGROUND

This chapter introduces the basic aspects of both CPU and GPU hardware architectures and how they are coupled in a heterogeneous computer system. Finally we expose the programming frameworks used by this Msc. thesis such as OpenCL and Centaurus API.

2.1 PARALLEL PROCESSING ARCHITECTURES

As the need for more performance and general programmability from both consumer and scientific community has grown, computer industry had to evolve both in software and hardware technologies. Single core processing failed to meet the expectations for many reasons, such as hardware designs had become rather complex in order to achieve higher frequencies, fact that leads to higher yield of manufacturing process. More importantly, power dissipation proved to be proportional to clock frequency [4] , imposing a natural limit on clock rates. Although several techniques [5] emerged in the previous 15 years which were able to sustain a boost of clock speed by a factor of 4000, the ability of manufacturers to dissipate heat has reached a physical limit. Leakage power dissipation gets worse as transistor gates get smaller because gate dielectric thicknesses must proportionately decrease. As a result, a breakthrough increase in clock speed without expensive cooling infrastructures is not possible.

As viable solution, parallel processing has been introduced by the industry in order to add more parallel resources while maintaining manageable power characteristics [6]. Parallel processing is not something new, it has a long history back of the mid 1960's. The reason which made them seriously unpopular those past days, was the difficulty of programming them, but also the continuous evolution of single core processors in terms of performance. Another reason was that, parallelism as a concept wasn't rather popular, resulting many algorithms lack a parallel version. This suggests that parallel processing in past days, demanded employance of both scientists and engineers who understood the application domain and had the resources and skills to program them. The main reason for which parallel processing resurfaced is that raw performance increase can come from increasing the number of cores rather than frequency, which translates into a slower

growth in power consumption. However, this approach represents a significant gamble for the following reasons:

- Parallel programming science has not advanced nearly as fast as our ability to build parallel hardware.
- As the size of transistors will be measured in just a few tens of atoms at most, quantum effects[7] will have to be taken into account and consequently we will see in parallel processing architectures an increased unreliability of the hardware, with components failing more often and—more importantly—intermittently.
- There will be so many transistors on the chip that it will be, power wise, impossible to switch on all of these at the same time. This phenomenon is known as the dark silicon problem [8].

Parallel processing architectures can be classified in a number of ways. In the following sections we discuss two of the most famous kinds of multicore architectures which are trends at the time this MSc thesis is written and are often a part of heterogeneous high performance systems.

2.2 MULTICORE PROCESSORS

The first category of parallel processing is a multicore processor which typically [9] is made up of two, four, six or even eight independent processor cores in the same silicon. They are connected through an on-chip bus, which is a central intersection through which all information flow between processor cores, memory and I/O (Figure 1). Most current general purpose multicore processors are homogeneous both in instruction set architecture and performance. This means that they can execute the same binaries and that it does not really matter, from functional point of view, on which core a program runs. Also threads are executed concurrently, which is typically a boost in performance as regards intensive computational tasks and reduction in power consumption than coupling multiple single-core processors. However as more cores are added the on-chip bus creates an information traffic jam as all the data must travel through the same path, partially limiting the benefits of multiple cores.

The most common hardware with this kind of architecture is a CPU of a computer system. Specifically, Intel's base-entry chips at present, are offering four real cores and in combination with her Hyper-Threading technology, are translated up to eight logical cores. As it is shows in the Figure 1 multicore processors employ multiple levels of cache memory. Caches existed from the early days of hardware processing designing, in the form of small pools of memory that store information which is most likely to be needed by the next computation of CPU. In this way data can be accessed faster than having to be read from main memory.

Nowadays caches employ multiple levels with different access latency and size per each, in order to push further back the need of the CPU to access the main memory for data. Moreover, some levels of cache are shared between cores such as the Level 3 in Figure 1. This ability of different cores have both read and write access to a cache level demands a mechanism guaranteing cache consistency between them, such as M.E.S.I. protocol [10]. As a result, although bigger size caches is facilitating higher hit rates of data in cache and better performance, it may also lead to reduced performance due to cache misses. So another difficulty of parallel programming is the exposure of programmer to dangers like cache misses which demand extra programming effort for their minimization.

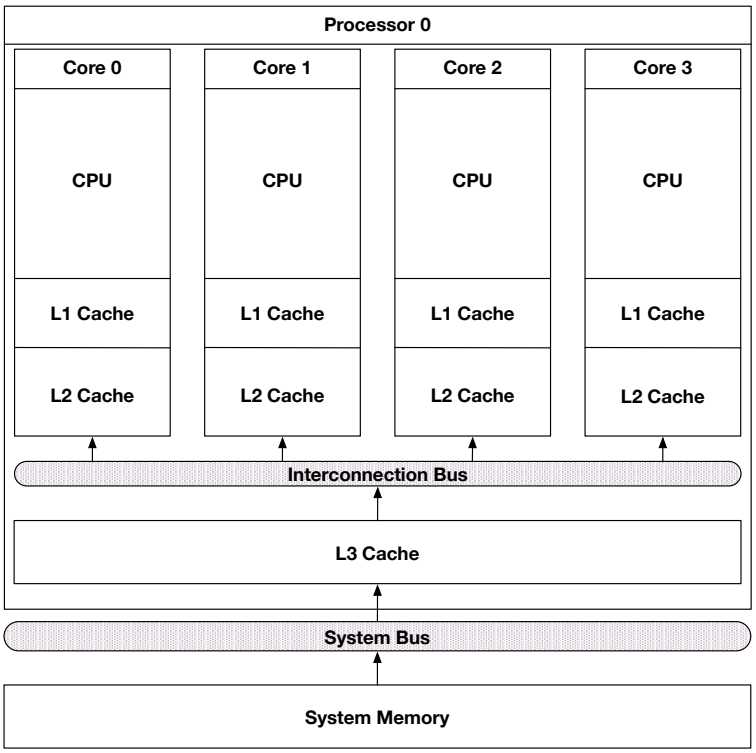


Figure 1: Four cores multicore processor level diagram.

The main characteristic of a multicore processor are:

- Limited number of real cores.
- All cores are general purpose with the same performance/power characteristics.
- Fast caches with multiple levels and different sizes.
- Cache consistency protocols.

2.3 MANYCORE PROCESSORS

A manycore processor is fair to say that in general is a multicore processor with higher number of cores up to hundreds or even thousands. In addition its cores are simpler in aspect of designing but also slower than the ones in a multicore processor. The most common manycore architecture hardware is a GPU. Nowadays GPUs employ thousands of cores, fact that makes them considered as accelerators. Due to the tremendous parallelism inherent in graphics, GPUs have long been massively parallel machines and as result parallel algorithms can really see performance gain in comparison with running on a CPU [11]. At the early days of GPUs, they were special-purpose hardwired application accelerators, suitable only for conventional graphics applications but modern GPUs are fully programmable, massively parallel, single and double point precision processing capable.

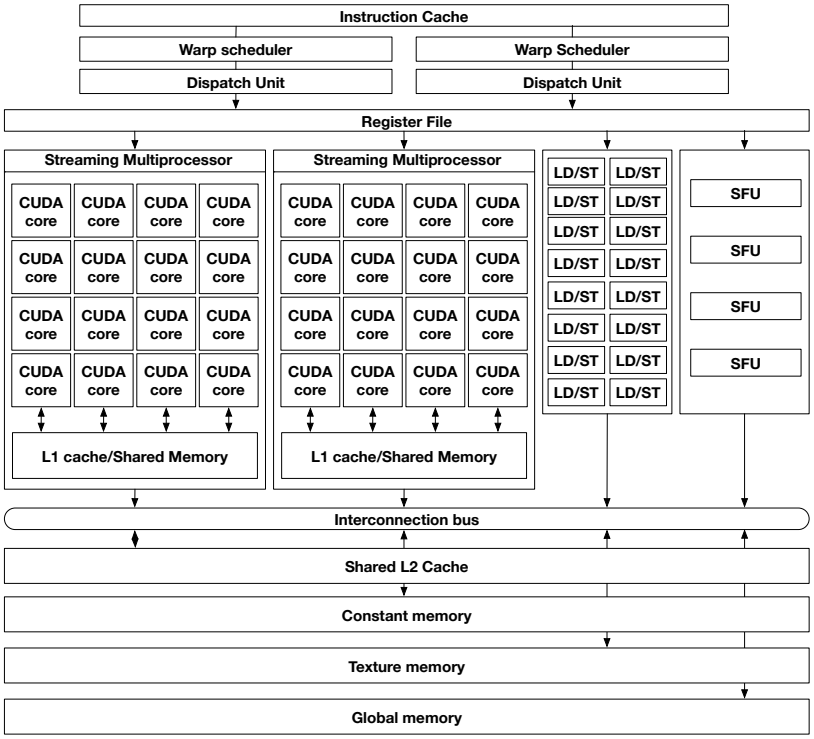


Figure 2: Nvidia Kepler architecture GPU level diagram.

As it is shown in the above Figure, a GPU also employs other units such as Load/Store units which are responsible for fetching/saving data from/to global memory of GPU. Also sports Special Function Units, which are responsible for fast approximate transcendental operations on single point precision processing. As regards cache, L1 cache is distributed per Streaming Multiprocessor (SMP) along with Shared Memory which is a special level of cache implemented in GPUs. In order to hide the latency of loading data from global memory for all these cores

and maximize the memory bandwidth, as much as possible, GPUs expose another low-access-latency Level of cache, called Shared Memory, in which the programmer is able to save data with high reusability in future computation, and as a result increase data locality. Moreover GPUs offer other kinds of memory to ease the programmer and exploit low latency data loading, such as constant memory, which as the name implies is a read only memory to GPU kernels and can only be initialized by the host. Another kind of read only memory, is texture memory which is designed mostly for graphics applications where memory access patterns exhibit a great deal of spatial locality. In a computing application, this roughly implies that a thread is likely to read from an address "near" the address that nearby threads read.

The main characteristic of a manycore processor are:

- Hundreds or thousands cores.
- Simple and low frequency cores.
- Fast caches with multiple levels and different sizes.
- Extra level of cache shared memory.

2.4 OPENCL FRAMEWORK

OpenCL[12], first released in December of 2008, is an industry standard framework for programming heterogeneous systems composed by a variety of processors such as CPUs and GPUs. OpenCL is the first of its kind and gives the ability to programmers to write a single program that can run on a wide range of systems, from conventional computers to nodes in massive supercomputers, without necessary extra tweaking. But this code portability comes at expense of programming effort. In OpenCL the programmer is exposed to low-level parametrization and he must explicitly define the platform, its context, and how computational work is scheduled onto different devices. This is the main reason OpenCL receives a lot of criticism, many programmers don't need or even want the detailed control OpenCL provides.

OpenCL defines a host scope and independent scopes for each compatible processor/device found installed in system. Programmers react with device scope by invoking OpenCL kernels (Figure 3). All the aforementioned necessary parametrization and kernel scheduling by the programmer is done in the host scope. As a result an OpenCL application consists of two distinct parts, the host program and a collection of one or more kernels. Moreover, OpenCL framework is divided into the following three components:

- **OpenCL platform API:** It is up to device vendor to define its own device corresponding OpenCL framework. As a result multiple OpenCL platforms can exist on a single heterogeneous

computer at one time. For example, the CPU vendor and the GPU vendor may define their own OpenCL frameworks. Each supplied OpenCL framework is recognized as a platform by an OpenCL program and it is a struct which refers to the installed devices and their OpenCL capabilities. This API provides the programmers a way to query the system about the available platforms and create the appropriate context for it.

- **OpenCL runtime API:** Provides the necessary functions to programmers to set up the command-queues, define memory objects and build the dynamic libraries from which kernels are defined. Programmers can attach either one command-queue to a single device or multiple within a single context. Besides memory object definition it also exposes functions for managing memory objects, such as transferring from host to device and vice versa. It also keeps track of how many instances of kernels use these objects (e.g retain a memory object) and when kernels are finished with a memory object (e.g. release a memory object). Another feature is synchronization points for managing data sharing and enforce of constraints on the execution of kernels.

- **OpenCL programming language:** The kernel programming language in OpenCL is called the OpenCL C programming language because which is derived from the ISO C99 language.

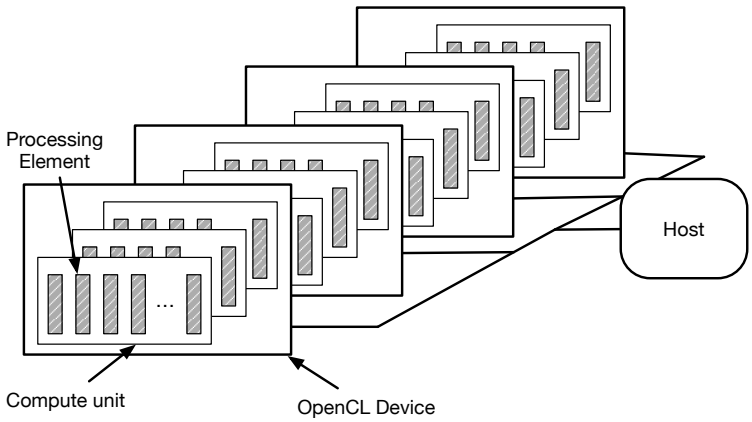


Figure 3: OpenCL platform model with one host and multiple devices.

2.5 CENTAURUS FRAMEWORK

Centaurus Framework is an energy aware framework which is able to choose between approximate and accurate execution in order to meet either a programmer defined energy frame or behave accordinly to an internal policy for energy consumption minimization. For task creation, Centaurus Framework offers its own programming model which is built

on top of LLVM[13] compiler, and it adopts a task-based paradigm, using *#pragma* directives. Tasks are implemented as OpenCL kernels, facilitating execution on heterogeneous systems.

Centaurus programming model offers flexible execution on heterogeneous systems, without the exposure of the programmer to many low-level concepts, such as inter-task synchronization, scheduling and data manipulation. This combination of heterogeneous tasks and energy-wise execution of tasks (accurate/approximate) is a feature that is introduced for the first time in a framework.

Moreover, Centaurus Framework is divided into the following components:

- **Centaurus platform API:** The Centaurus platform inherits the typical OpenCL platform model, depicted in Figure 3, providing an abstract execution model over all available devices. Due to the fact that Centaurus framework hides from programmer all the tedious parametrization needed in OpenCL, programmer cant specify exact task per device bindings but he can specify on which type of device the task must run (e.g. CPU, GPU etc.).

- **Centaurus runtime API:** Centaurus runtime system is responsible for orchestrating the execution and the selection of the accurate or approximate implementation for each task, based on the active policy among a number of scheduling policies and the additional programmer's/user's information through directive clauses. Tasks are implemented as OpenCL kernels. The tasks execute on one or more devices and host program manages their creation and synchronization. Also in host scope, the programmer defines and spawns tasks annotated with information about significance and data dependencies. Significance can also be defined within device code in the form of subtasks. The compiler generates different code versions for each case, accurate and approximate. Figure 4 depicts the life cycle of a task, from creation to completion.

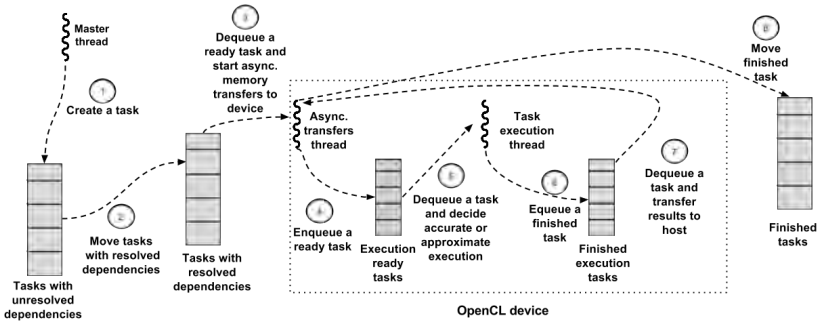


Figure 4: Task life in the Centaurus framework.

- **Centaurus memory model:** As in OpenCL, independent scopes are defined for host and all the available devices in system. By us-

ing directives the programmer from host scope can transfer host-to-device, device-to-host or device-to-device data. As for memory objects release, Centaurus framework keeps track of task executions and when a memory object is not needed by a future task is deleted in any device scope which is allocated.

- **Centaurus programming model:** The task programming language is OpenCL C language and the host counterpart language is ISO C99 language. It also exposes *#pragma* directives for necessary task and memory management between host and devices scopes.

3

SPSTEREO DISPARITY IMAGE

SPStereo(SPS) is a stereo vision application producing a dense disparity map of a static scene captured with a stereo pair camera. It combines a Stereo Global Matching (SGM) algorithm [14] and a slanted plane algorithm which assumes that the 3D scene is piece-wise planar and the motion is rigid or piece-wise rigid [15, 16]. A robust and dense computation depth information is necessary in many (automated) machine vision applications such as driver assistance systems, robotics and is a cheap effective alternative to RGB-D cameras. In the sections to follow we analyse the original algorithm used and its hotspots. Next we will discuss the transition to a more GPU oriented version exploiting Centaurus Framework and the implementation of viable approximations.

3.1 SEMI-GLOBAL MATCHING

After profiling the original implementation with Intel Vtune Performance Analyzer we discovered that the most time consuming task is the first section of application, the Semi Global Matching (SGM). SGM considers pairs of images with known intrinsic and extrinsic orientation (stereo-camera characteristics) and is responsible for an initial disparity image estimation. Moreover, disparity estimation is the task of identifying the projection point of the same 3D real-world point in two or more images taken from distinct viewpoints. The most challenging task of a disparity estimation algorithm is to identify correctly a point and its different references from different viewpoints in the same 3D world despite the high level of ambiguity.

As Figure 5 depicts, the stereo image pair is facilitating an epipolar geometry and as a result the further analysis of similarity between pixels from left and right image (base and match image, respectively) is done in 1D epipolar lines (scanlines) level. In order to deal with non-unique or wrong correspondences due to low texture and ambiguity, the semi-global matching facilitates other computer vision techniques to check pixel consistency and produce proper matching costs between base and match image. The detailed steps of SGM algorithm are the following:

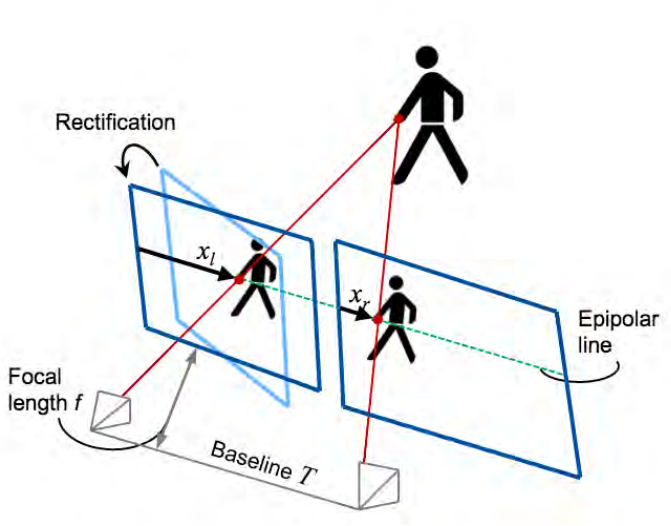


Figure 5: Stereo camera geometry showing epipolar line analysis (scan-lines)

3.1.1 *Capped Sobel Filter*

Is a high pass filter and especially in computer vision and it highlights edges and transitions of pixels intensity by applying the following operator per image pixel.

$$P = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{1}$$

Moreover the final pixel in the resulting sobel image is produced through the following transformation

$$R_{sobel}(P) = \begin{cases} 2 \cdot cap & P > cap \\ P + cap & -cap < P < cap \\ 0 & P < -cap \end{cases} \tag{2}$$

where *cap* is an algorithm defined constant which further highlights edges.

3.1.2 *Census Transform*

It defines a central pixel which is surrounded by a virtual square window D of size $M \times M$. The intensity of the corresponding pixels in the window is mapped to a bit string and then the central window pixel

value is replaced by this string accordingly the movement of the window (e.g. left, right etc).

$$R_{census}(P) = \underset{[i,j]\epsilon D}{\otimes} \xi(P, P + [i,j]) \tag{3}$$

where $\otimes$ is the contetation of corresponding bits and function ξ is defined as

$$\xi(P, P') = \begin{cases} 1 & P \geq P' \\ 0 & P < P' \end{cases} \tag{4}$$

3.1.3 *Sum of Absolute Difference (SAD)*

Is used to indicate the similarity between image blocks and in this application context, is measured by taking the absolute difference between each pixel in the left image block D in accordance with the corresponding right image block, resulting as a cost factor corresponding to the similarity of pixel regions.

$$C_{sad}(x,y) = \sum_{[i,j]\epsilon D} \Big| I_l[x,y] - I_r[x+i, y+i] \Big| \tag{5}$$

where I is the corresponding left and right image respectively.

3.1.4 *Hamming Distance*

Although Hamming Distance is mainly used in information theory, in this context is implemented as an extra indication of block similarities calculating the minimum number of substitutions required to change one pixel of left image block D into the corresponding right image block pixel.

$$C_{hamming}(x,y) = \sum_{[i,j]\epsilon D} \Big| I_l[x,y] \oplus I_r[x+i, y+j] \Big| \tag{6}$$

where $\oplus$ bitwise XOR operation between two pixels.

3.1.5 *Image Row Costs Calculation*

Combines the result of previous SAD and Hamming image block analysis and calculates final pixelwise costs for each image. Also in order to produce more accurate results cost pixels from different image rows are being aggregated and the final cost is defined as

$$C_r(p,d) = C_{final}(p,d) + C_r(p, d-1) + \underset{i}{C_r}(p,d) - \underset{j}{C_r}(p,d) \tag{7}$$

where $\underset{i}{C_r}$ and $\underset{j}{C_r}$ are the aggregated costs from different rows and $C_r(p, d-1)$ is the exactly previous calculated cost. Also C_{final} is defined as

$$C_r(p,d) = C_{hamming}(p,d) + C_{sad}(p,d) \tag{8}$$

3.1.6 *Semi-Global Matching*

Every image row is considered a scanline which consists of costs paths. The path costs $L_r(P,d)$ are aggregated along a path r according to

$$L_r(p,d) = C_r(p,d) + min \begin{cases} L_r(p-r,d) \\ L_r(p-r,d-1) + P1 \\ L_r(p-r,d+1) + P1 \\ \underset{i}{min} L_r(p-r,i) + P2 \end{cases} - \underset{l}{min} L_r(p-r,l) \tag{9}$$

Where d is the disparity and i, l are indexes of different paths r. The first term is the calculated costs for the pixel from previous analysis. The second term adds the minimal path costs of the previous path r including a penalty $P1$ for disparity changes and P2 for disparity discontinuities, respectively. The third term in order to fix any wrong correspondences due to low texture and ambiguity removes the minimal path costs of previous paths. Moreover SGM is applied forward and backwards on both images and the resulting disparity image pixel is defined as

$$d_f(p,r) = Factor_d \left(d_{best}(p,r) + \begin{cases} \frac{d(p+1,r) - d(p-1,r)}{4 \cdot (d(p,r) - d(p-1,r))} & , \frac{d(p-1,r)}{d(p+1,r)} < 1 \\ \frac{d(p+1,r) - d(p-1,r)}{4 \cdot (d(p,r) - d(p+1,r))} & , \frac{d(p-1,r)}{d(p+1,r)} \geq 1 \end{cases} \right) \tag{10}$$

Let d_{best} be the disparity with the lower cost associated in this path p of row r.

3.1.7 *Speckle Filter*

Searches in image for tearing regions which is shown as noise and enhances it in order to eliminate these regions.

3.1.8 *Enforce Image Consistency*

Enhances both left and right produced images, so they have matching disparity intensity and lighting conditions.

In Figure 6 is shown the flowchart of the original SGM sequential implementation.

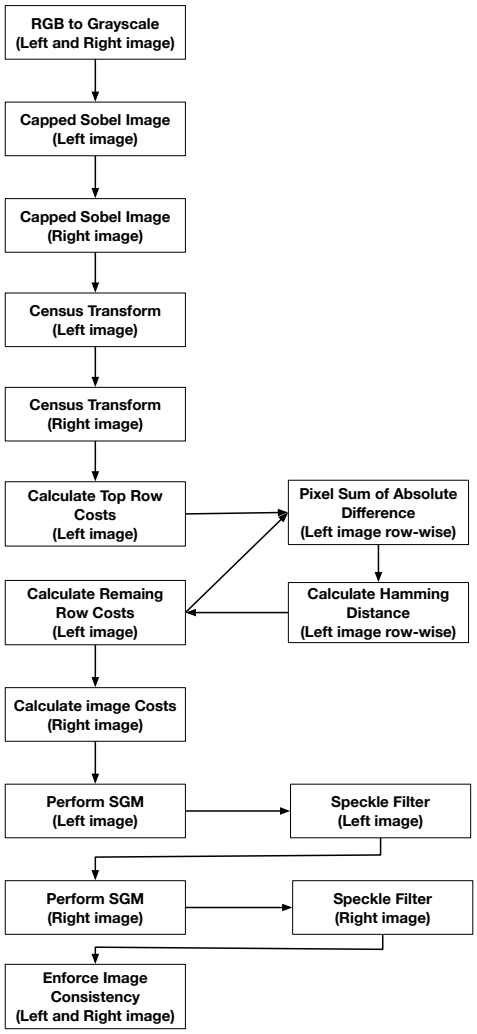


Figure 6: Flowchart of original SGM implementation

3.2 IMPLEMENTATION

SPS original implementation was written in C++, a programming language which is not supported by Centaurus Framework. Also it was exploiting Intel SIMD extensions in order to achieve fast and robust results, and shrink as much as possible any time-consuming task. In order to begin with the migration to a GPU version and Centaurus Framework implementation, the application was completely rewritten in C language along with the corresponding originally utilized classes (e.g. vector, stack etc). After affirming the exactness of results between C and C++ versions we translated all previous mentioned functionalities to corresponding OpenCL kernels.

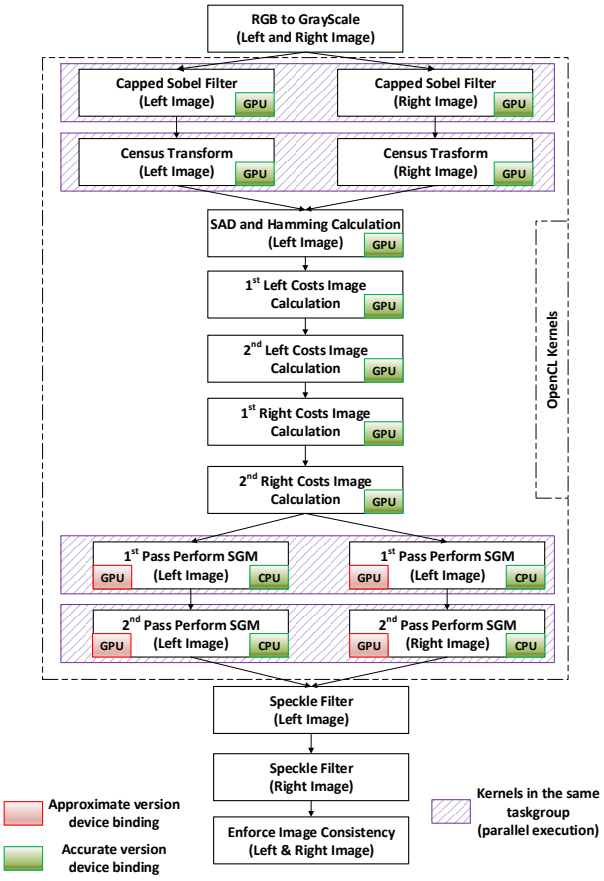


Figure 7: Centaurus implementation flowchart of SGM section

Firstly the fact that original version was exploiting SIMD extensions made the code hard to understand and trace data dependencies. Also despite that SIMD by definition is indicating some level of data parallelism, in most cases the level of parallelism is not sufficient of fully utilizing a GPU, so geometry of each functionality had to be revisited. Pretty much most of the corresponding OpenCL kernels are implementing the same algorithmic steps, but there are also functionalities which are implemented with step-reduction kernels due to prohibiting data dependencies.

Specifically, the function of calculating image row costs as is shown in equation 7 exhibits both vertical and horizontal data dependencies. Horizontal dependencies are due to the second term which is the exact previous calculated cost path in the same row, and vertical because the last two terms are adding and subtracting respectively, the corresponding path costs of previous calculated rows. Thus this function is broken up to two OpenCL kernels, each one responsible for satisfying one of data dependencies. At the end both their results are summed resulting the same function.

In addition, there are functionalities that have sequential dependencies across consecutive image rows that make it impossible to exploit any level of data parallelism higher than the already implemented. This kind of function is SGM and as shown in equation 9 the second term requires the local minimum between different path costs of rows in the image, fact that establishes an uncertainty of data access pattern and leads to limited level of parallelism. Thus due to insufficient parallelism of utilizing a GPU the corresponding function is translated to a CPU oriented OpenCL kernel in which original SIMD is replaced with OpenCL vectorization extensions, in order to maintain the originally implemented level of data parallelism.

Concluding, the resulting flowchart of our SGM implementation in Centaurus Framework is shown in Figure 7. As it depicts some of functions run concurrently and there are appropriate versions of tasks for both accurate and approximate executions which will be detailed in the following section.

3.3 APPROXIMATIONS

After successfully implementing the accurate version in Centaurus Framework we tried to exploit different approximations in order to produce acceptable output in aspect of quality (PSNR) and reduce total execution time. As a baseline approximation we exploited the relaxation in dependencies of calculated paths between previous scanlines (SGMDR). Furthermore each path cost is considered independent from previous calculated path cost in the same row, and for every path cost emerges one local minimum. As a result, horizontal dependencies from equation 9 are eliminated and the equation of SGM function becomes

$$L_r(p,d) = C_r(p,d) + min \begin{cases} L_r(p-r,d) \\ \min\limits_i L_r(p-r,i) + P2 \end{cases} - \min\limits_l L_r(p-r,l) \tag{11}$$

This approximated version of SGM function exhibits high level of parallelism, thus the corresponding kernel is GPU oriented. Another approximation injected in combination with SGMDR is the reduction of necessary forward and backwards analysis of SGM function to just backwards (SGMRS). Our last approximation is the elimation of computation intensiveness in SGM pixel region similarity functions by excluding completely Census function execution (SGMEPS). Each approximation evaluation is presented in the following section.

3.4 EXPERIMENTAL EVALUTION

In this section all aforementioned approximations will be evaluated in aspect of execution time, quality and energy consumption. Firstly Figure 8 shows the input images on which our approximations will be evaluated.

(a)

(b)

Figure 8: (a) Left (Base) image (b) Right (Match) Image

Each approximation output quality is evaluated in metric of PSNR (dB). We pick that fully accurate execution output is of 44 dB instead of actual infinity dB in fairness to graphical representation and comparison of quality between executions with approximate tasks. This is valid because in image benchmarking, images with 44 dB are considered to have high fidelity. Figure 9 depicts the output of an accurate execution for image size 1241×376 which is considered the gold standard in the following approximations evaluation.

Figure 9: Output of fully accurate execution for image size 1241×376

3.4.1 *SGMDR approximation*

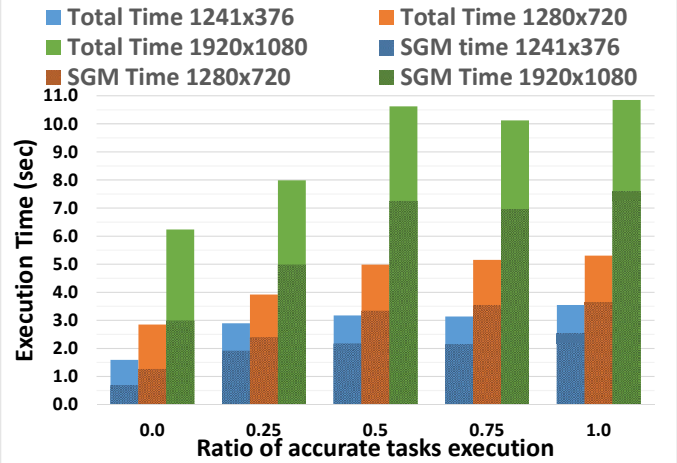


Figure 10: SGMDR time execution vs ratio of accurate tasks

In Figure 10 the total time of SPS execution is shown for each ratio configuration. In ratio 1.0 configuration it is clearly that SGM section of SPS takes almost the one fourth of total time execution. As more approximate task executions are injected and for configurations of ratio 0.75 and 0.5 we notice a small reduction of SGM section time that doesn't affect total time execution. This is due to the fact that approximate version of tasks is GPU execution binded in contrast with accurate version which is CPU binded. This mixture of device binded executions causes extra memory transfers between GPU and CPU which prohibit the further reduction of total execution time. In ratio 0.25 we see that time execution gets reduced and even more in fully approximate execution of ratio 0.0. Another noticeable effect is that the second section of SPS application is getting slower as more tasks are being approximately executed and this is due to the fact that more erroneous regions of initial disparity image are surfaced.

As Figure 11 shows, quality decreases in accordance to the number of approximate tasks execution reaching a minimum of 36.267 dB for image size 1241×376 and 27.537 dB for size 1920×1080. This difference is due to the fact that SGMDR approximation is highly correlated with image size. While the size of image gets larger, then more local minimum path costs emerges as the scalines get wider. Thus the error injection rate is bigger and the output is further differentiated from the golden.

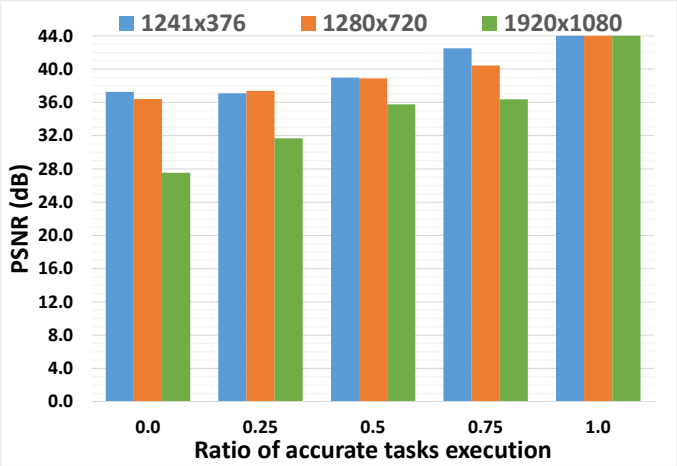
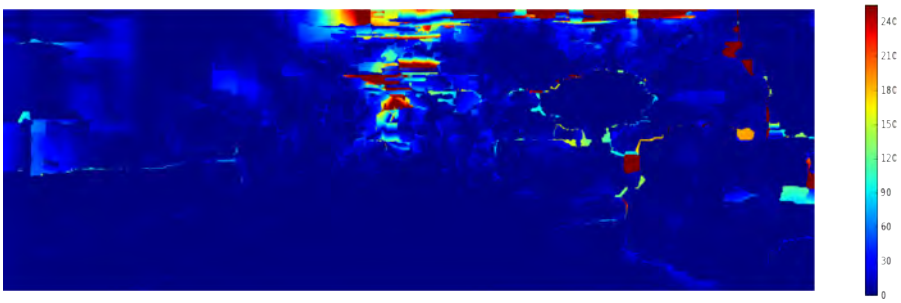


Figure 11: SGMDR quality (PSNR) vs ratio of accurate tasks

(a)



(b)

Figure 12: (a) Output of fully approximate SGMDR execution. (b) Difference image between fully approximate and fully accurate executions.

We can observe in Figure 12b that we lose shape detail of objects. In accurate version we preserve the total minimum path cost of a scanline and thus any change in disparity by the existence of object is located correctly. In the approximate version total minimum is replaced by local minimums and this likely difference in minimum value of different pathcosts is resulting to injection of non-existent artifacts and alternation of objects shape. However, object positioning and disparity in the scene are still correct.

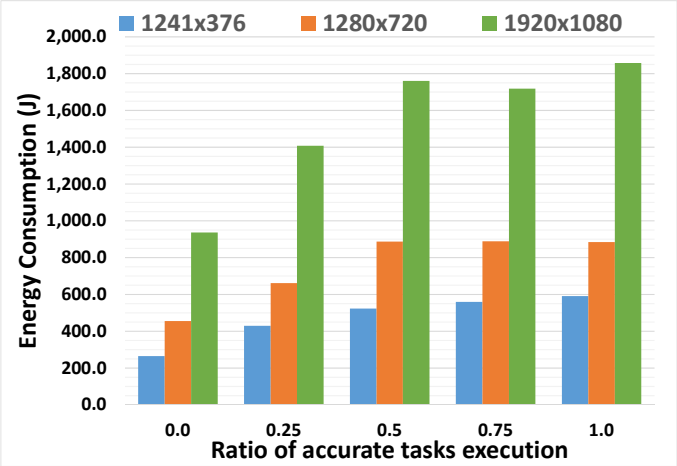


Figure 13: SGMDR energy consumption vs ratio of accurate tasks

Energy consumption as shown in Figure 13, is highly correlated with total execution time. We see a 55% reduction for 1241×376 size and 49% for both 1280×720 and 1920×1080 sizes.

3.4.2 *SGMRS approximation*

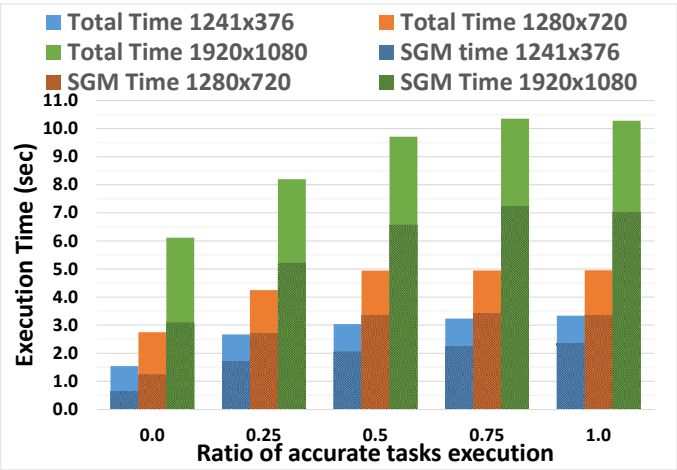


Figure 14: SGMRS time execution vs ratio of accurate tasks.

Time perfomance of SGMRS approximation is almost the same with SGMDR. The final results show the already implemented gains from SGMDR approximation and an extra 0.06 seconds reduction (ratio 0.0) due to the exclusion of backwards analysis. SGMRS has such low impact on total time because the excluded kernel are too small in duration.

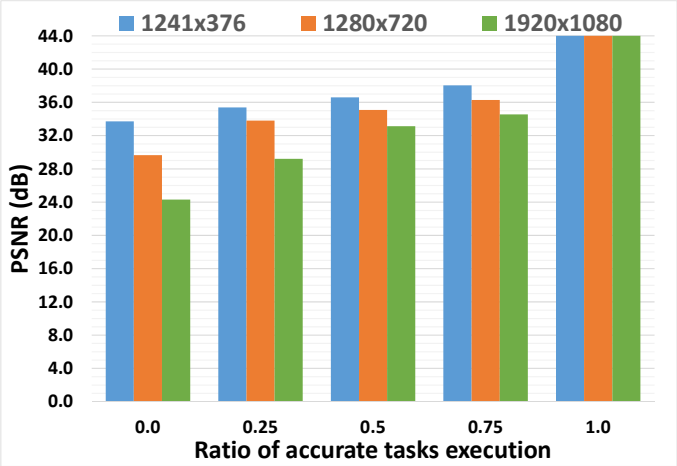
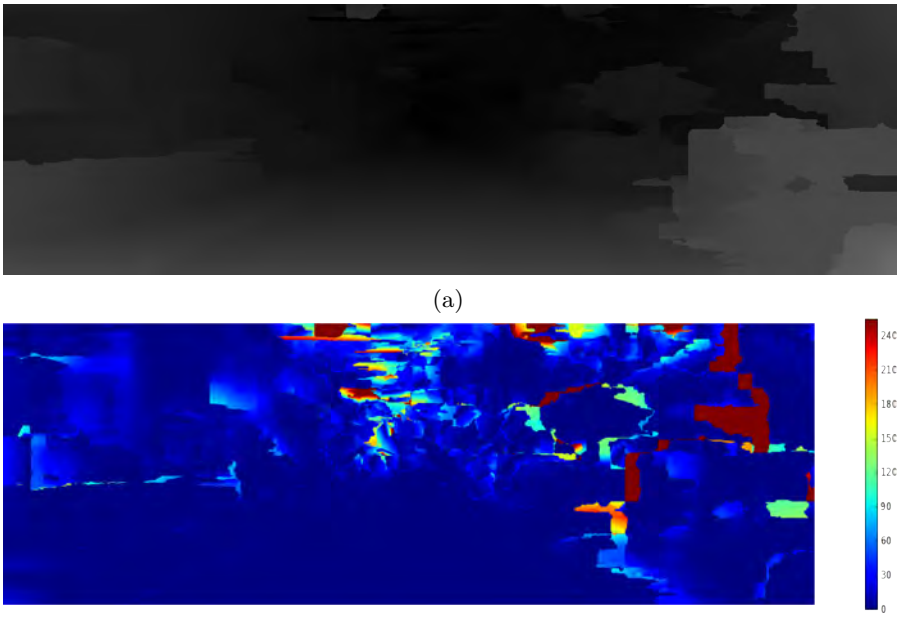


Figure 15: SGMRS quality (PSNR) vs ratio of accurate tasks.

Quality of fully approximate execution is reduced to 33.713 dB of image size 1241×376. This is due to the fact that as backward SGM analysis is excluded are also excluded the corresponding minimums and as a result we have more erroneous regions.



(a)

(b)

Figure 16: (a) Output of fully approximate SGMRS execution. (b) SGMRS difference image between fully approximate and fully accurate executions.

As regards object shape correctness we can see that in this approximation objects shape is transformed even more from SGMDR approximation. The lack of backward analysis leave false disparities which leads to further distortion of objects shape.

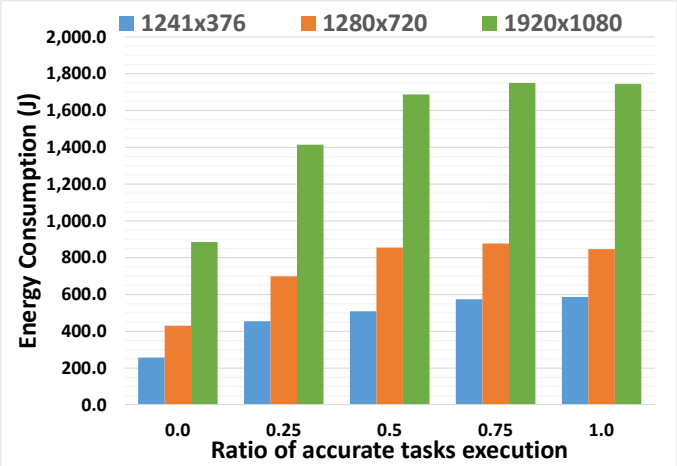


Figure 17: SGMRS energy consumption vs ratio of accurate tasks.

Energy consumption of ratio 0.0 configuration is 257.92 J and SGMRS is by 2.5% more energy efficient than SGMDR and by 57% from fully accurate execution. As for sizes 1280×720 and 1920×1080 the gain is 5.5% and 54.5% from SGMDR and fully accurate execution respectively.

3.4.3 *SGMEPS approximation*

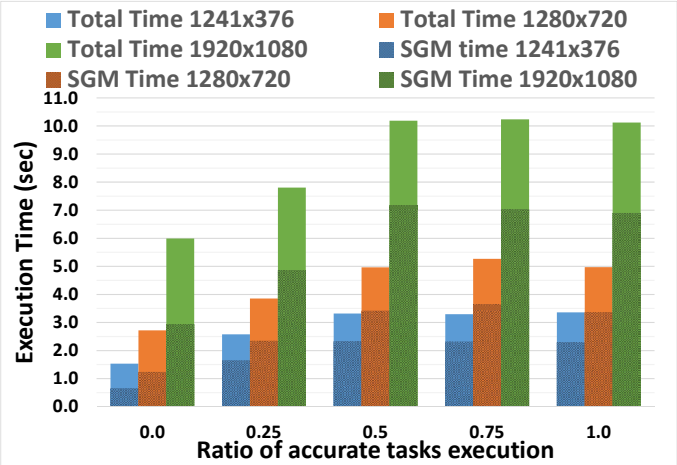


Figure 18: SGMEPS time execution vs ratio of accurate tasks.

Time performance of SGMEPS is the same with SGMRS. Although in SGMEPS, kernels responsible of Census function are reduced, their duration are in scale of 20 msec thats why their reduction doesn't affect the total execution time.

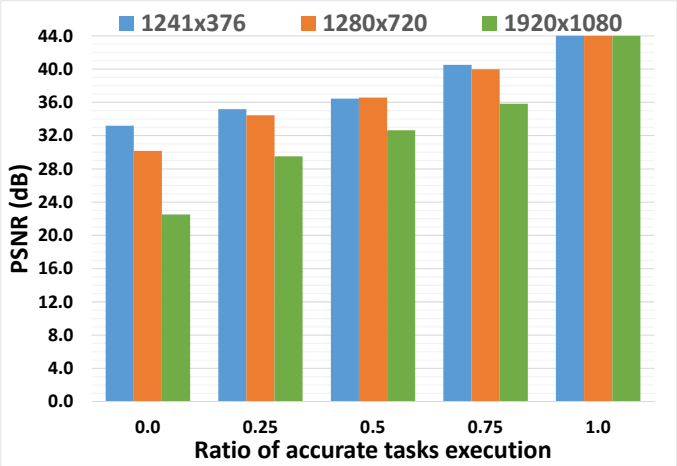
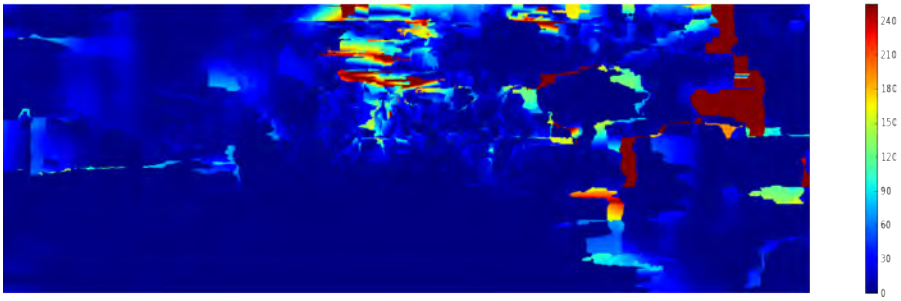


Figure 19: SGMEPS quality (PSNR) vs ratio of accurate tasks.

Quality for image size 1241×376 stays the same with SGMRS and for other two remaining sizes is slightly improved. This is due the fact that SGMEPS is reducing the functions responsible for determining similarities between pixel regions. This relaxation cannot improve the distortion of objects shape but as Figure 20b shows, can improve the calculated disparity of erroneous regions to be closer to ground truth.

(a)



(b)

Figure 20: (a) Output of fully approximate SGMEPS execution. (b) SGMEPS difference image between fully approximate and fully accurate executions.

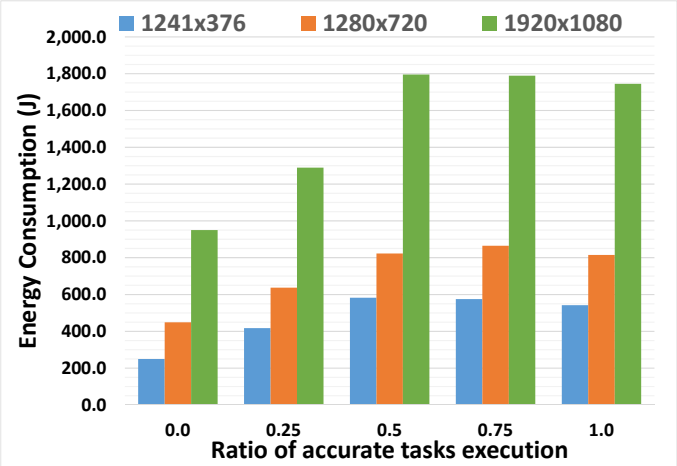


Figure 21: SGMEPS energy consumption vs ratio of accurate tasks.

Energy profile of SGMEPS stays the same with SGMRS for the same, already mentioned reason that SGMEPS doesn't show any time execution improvement.

4

MOLECULAR DYNAMICS SIMULATION

Molecular Dynamics (MD) is widely used to simulate many particle systems ranging from solids, liquids, gases, and biomolecules on Earth, to the motion of stars and galaxies in the Universe. The first successful Molecular Dynamics simulation was done in 1964 by A. Rahman [17] who studied dynamical properties such as heat transport of liquid Argon. Later his work was extended by Verlet whose improvements, especially the Verlet integration algorithm [18] and particle neighbor lists [19], are widely used today. In the sections to follow we will analyse the theoretical background of MD, our implementation, the approximations we exploited and their corresponding evaluation.

4.1 THEORETICAL ANALYSIS

MD is basically a system of N atoms, which interact between them under a kind of inter-particle force but also in some cases in addition with external forces. The baseline of such system is the equation of motion for particles which is basically the numerically integrated *Newton's equation of motion*. The resulting equation of motion for the particle system is defined as

$$m\frac{d^2\mathbf{r}_i}{dt^2} = \mathbf{F}_i(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N), \quad i = 1, 2, ..., N. \tag{12}$$

where:

- N is the number of atoms in the system.
- $\mathbf{r}_i$ are the position vectors of atoms.
- $\mathbf{F}_i$ are the forces acting upon N particles in the system.

In the absence of external forces, the system is in equilibrium state and static properties such as temperature and pressure are measured as averages over time. Moreover forces derives from potential functions $\mathbf{U}(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N)$ which can be presented as a sum of pairwise particles interactions

$$\mathrm{U} = \sum_{i=1}^{N}\sum_{j\neq i}^{N} u(r_{ij}) \tag{13}$$

where:

- $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ is the distance between particles.
- $r_{ij} \equiv |\mathbf{r_{ij}}|$ is the magnitude of distance vector.

In addition forces are correlated to potential accordingly to

$$\mathbf{F}_i(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N) = - \bigtriangledown_{\mathbf{r}_i} \mathbf{U}(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N) \tag{14}$$

where implies the conservation of total energy $E = E_{kin} + E_{pot}$.

One of the most famous pair potentials for van der Waals systems is the Lennard-Jones potential. Although Lennard-Jones potential is not the most faithful representation of the potential energy surface, it is used widespread due to its computational expediency. Lennard-Jones potential is given by the following equation:

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \tag{15}$$

where

- V is the intermolecular potential between the two atoms or molecules.
- ϵ is the well depth (material dependent).
- σ is the distance at which the intermolecular potential between the two particles is zero (material dependent).
- r is the distance between both particles.

Furthermore at short distances the term proportional to r^{-12}, indicates a repulsive force between atoms due to nonbonded overlap of electronic orbitals. On the contrary, at long distances the dominant term is proportional to r^{-6} which indicates an attractive force and thus the pair of atoms experiences a small stabilizing force. When distance of pair atoms is approximately equal to σ then the potential energy reaches a minimum value and the force is approximately zero between the pair of atoms (Figure 22).

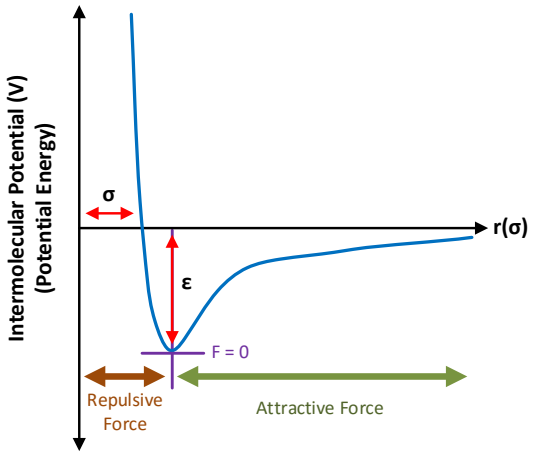


Figure 22: Lennard-Jones potential graph.

The inter-particle forces arising from the Lennard-Jones potential (15) in accordance with (14) have the form

$$F(r) = -\frac{dV(r)}{dr} = \frac{24\epsilon}{\sigma}\left[2\left(\frac{\sigma}{r}\right)^{13} - \left(\frac{\sigma}{r}\right)^{7}\right] \tag{16}$$

There are many interesting quantities that can be measured for a system in thermal equilibrium. For example the total energy which is given by

$$E_{total} = \frac{m}{2}\sum_{i=1}^{N}\mathbf{v}_i^2 + \sum_{\text{pairs } ij} V(|\mathbf{r}_i - \mathbf{r}_j|) \tag{17}$$

which is the sum of kinetic and potential energies. The conservation of total energy can be monitored by measuring it periodically between several simulation time steps. The averages of kinetic and potential energies can also be measured separately.

Another interesting observable quantity of the system is the pressure P. If the system is confined in a box with walls, the pressure can be estimated from the average force exerted by atoms bouncing off the walls. However, there is a more convenient way of measuring the pressure using the virial theorem [20] which states that

$$PV = Nk_BT - \frac{1}{3}\left\langle \sum_{i=1}^{N}\mathbf{r}_i \cdot \mathbf{F}_i \right\rangle$$

where

- k_B is the Boltzmann constant.

- T is the absolute temperature of the system in thermal equilibrium.
- $< \ldots >$ denotes a time (ensemble) average.

4.2 IMPLEMENTATION

In this section, the discussion will focus on the computational implementation of previous theoretical analysis. In order to be able to work with values proportional to units of length (e.g. σ, ϵ etc) instead of atomic scale values, which are typically very small and also to simplify the equations of motion, due to the elimination of atomic level factors, we choose to implement dimensionless units, also called MD units. Thus the Lennard-Jone potentials takes the simplified form

$$V(r) = 4\Big(r^{-12} - r^{-6}\Big) \tag{18}$$

The produced inter-particles forces are defined as

$$F(r) = 48\Big(r^{-14} - \frac{1}{2}r^{-8}\Big) \tag{19}$$

The total energy of the MD system is given by

$$E_{total} = \frac{1}{2}\sum_{i=1}^{N}\mathbf{v}_i^2 + \sum_{\text{pairs } ij} V(|\mathbf{r}_i - \mathbf{r}_j|) \tag{20}$$

The corresponding mapping of MD units to SI metric system is shown in the following table.

Table 1: System of units used in MD simulation.

Physical Quantity	Unit	Value for liq. Ar
length	σ	3.4×10^{-10} m
energy	ϵ	1.65×10^{-21} J
mass	m	6.69×10^{-26} kg
time	$\sigma(m/\epsilon)^{1/2}$	2.17×10^{-12} s
velocity	$(\epsilon/m)^{1/2}$	1.57×10^{2} m/s

As regards time integration we exploited the "Velocity" Verlet algorithm [18], because Euler time integration doesn't provide the numerical stability necessary to reproduce energy conservation phenomenon

in equilibrium state. The resulting equations for motion of an atom are:

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \mathbf{a}(t)\Delta t + O((\Delta t^2)) \tag{21}$$

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \frac{1}{2}\mathbf{a}(t)(\Delta t)^2 + O((\Delta t^3)) \tag{22}$$

And the expression of velocity at mid-interval $v(t+\Delta t/2)$ is:

$$\begin{cases} \mathbf{v}(t+\Delta t/2) = \mathbf{v}(t) + \mathbf{a}(t)\Delta t/2 \\ \mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \mathbf{v}(t+\Delta t/2)\Delta t \\ \mathbf{a}(t+\Delta t) = \mathbf{F}(\mathbf{r}(t+\Delta t))/m \\ \mathbf{v}(t+\Delta t) = \mathbf{v}(t+\Delta t/2) + \mathbf{a}(t+\Delta t)\Delta t/2 \end{cases} \tag{23}$$

Another important step in MD simulation is the initialization of physical quantities. Systems which are in thermal equilibrium like ours and are exposed to constant quantities such as number of particles, bound box volume and energy are called microcanonical ensembles. This type of simulation is called a microcanonical MD method [21]. According to the ergodic hypothesis, if the initial state of the system is chosen sufficiently carefully, then the successive states of the system as a function of time can be used to compute the thermal averages of various observables such as temperature and pressure. Thus for initialization we implemented a simple random generator filling particles positions vector within the range of box dimensions and initial particles velocity in a range $[-1.0, 1.0]$.
Finally the MD simulation is derived to the following algorithmic steps:

- **Initialization:** The volume box that bounds the atoms is set. Position and velocity vectors of the particles are being inflated.
- **Equilibration:** A time step h is chosen, and the equations of motion are solved iteratively for a sufficient number of steps to allow the system to come to equilibrium.
- **Simulation:** The iterations are continued. Physical quantities are measured at each time step, and their thermal averages are computed as time averages.

The implemented algorithm has both $O(N^2)$ time and space complexity and as Figure 23 depicts the sections of Initialization and Equilibration are done in host-scope. Furthermore in order to exploit parallel execution of tasks we divided Simulation section up to four tasks. Except the possibility of running to different devices, this implementation also give us the advantage of alleviating execution of multiple task command queues provided by Centaurus Framework. Thus before task invocation, the total position and velocity vectors of particles is broadcasted to each

device that is task execution binded. Next each device evaluates the necessary physical quantities of an individual portion of particles and at the end returns the corresponding updated values. These values are concatenated and are rebroadcasted in the next time step. Lastly for visual representation of the MD simulation we implemented an OpenGL graphics scene which visualizes the position vectors of atoms in each time-step.

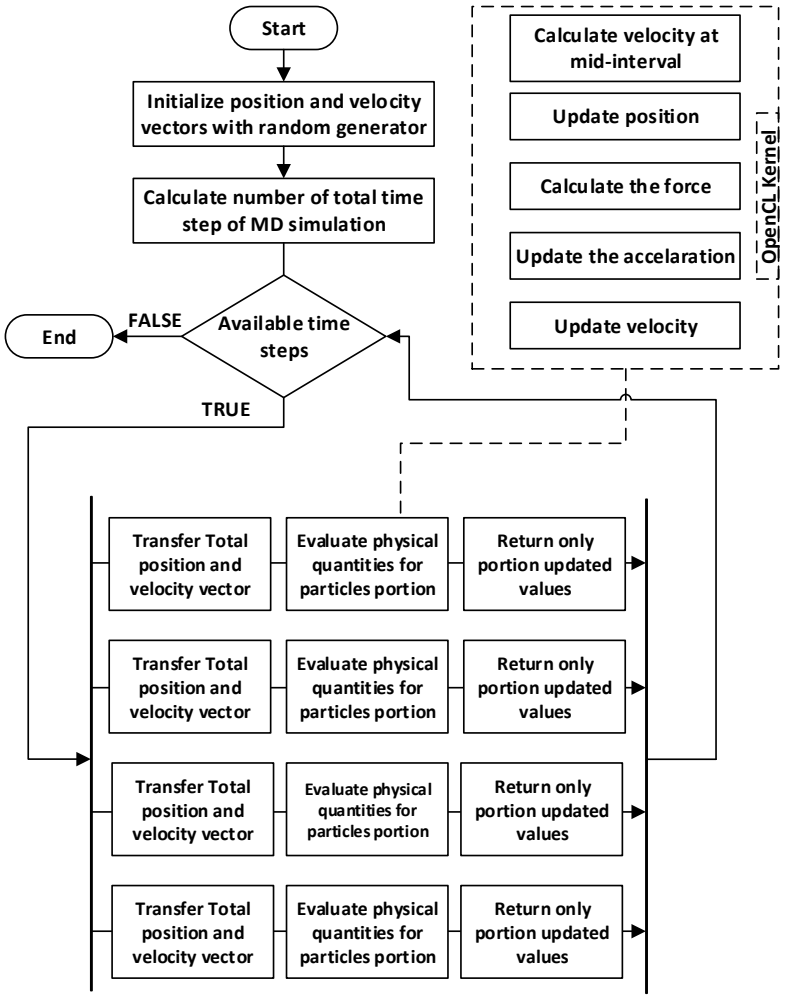


Figure 23: MD Centaurus Framework implementation flowchart.

4.3 APPROXIMATIONS

In order to reduce as much as possible the computational intensiveness of the implemented simulation, we set a cut-off region (MDCT). This distance acts as a threshold which creates a surrounding region to a particle and enables it only to interact with any particle that submits to this region and ignore everything out of it. This kind of approximation

is viable as the dominant term in equation 15 for large distances, is negligible and has small magnitude on the resulting total force that acts on the desired particle. After experimentation, we found that an acceptable distance in terms of quality is 40 Å. Another approximation we exploited is propagating the particles in time with double time step (MDDT) in comparison with accurate version. Although we skip time steps the implemented "Velocity" Verlet integration method provides sufficient motion stability in order for energy conservation phonemenon to be observed, without particular loss of quality.

4.4 EXPERIMENTAL EVALUTION

In this section both approximations will be evaluated in aspect of time execution, quality and energy consumption. The quality is being evaluated by calculating the average total energy of the system every 100 time steps of simulation. The pressure is sampled with the same interval and at the end of simulation is averaged to result into the final average pressure per particle. Both approximations will be evaluated on a MD simulation parameterised with 32768 particles bounded in a $[-300, 300]$ Å Box per dimension, for total simulation 1 ps with time steps of 1 fs (1000 steps). For better representing in the following Figures (26, 30) the corresponding position vectors are calculated for 1024 particles.

4.4.1 *MDCT*

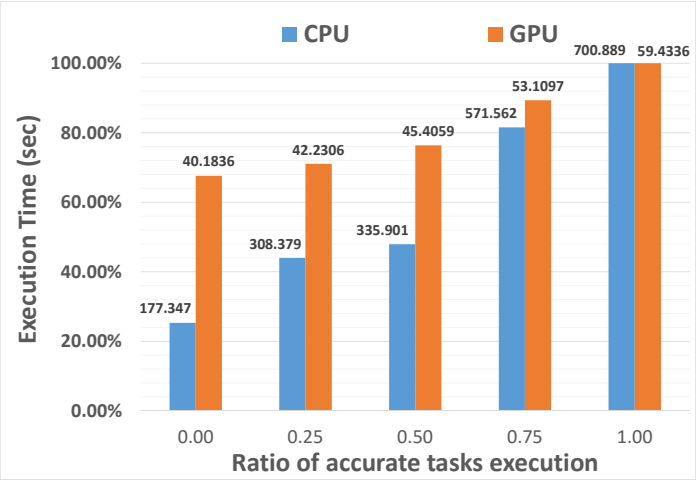


Figure 24: MDCT time execution vs ratio of accurate tasks

The time execution reduction between fully accurate and fully approximate executions (ratio 1.0 and 0.0 respectively) is 37% for GPU and 75% for CPU.

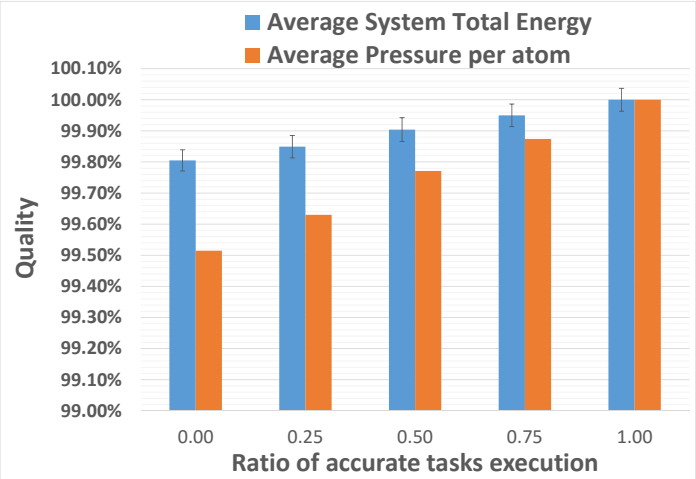


Figure 25: MDCT quality vs ratio of accurate tasks

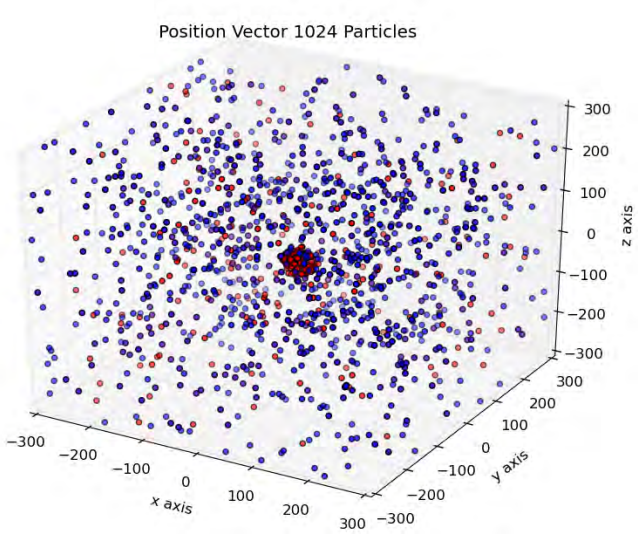


Figure 26: MDCT position vector of particles between approximate and accurate execution at half simulation

At ratio 0.0 configuration relative error is 0.2% for average system total energy quality reduction and 0.5% for average pressure per atom. This order of magnitude for relative error is almost negligible. Figure 26 shows that distribution of atoms for both fully accurate (blue) and fully approximate (red) execution is homogeneous another factor that leads to so small quality drops and makes the MDCT a viable approximation. The energy consumption is respectively reduced 38% and 72% for GPU and CPU device executions.

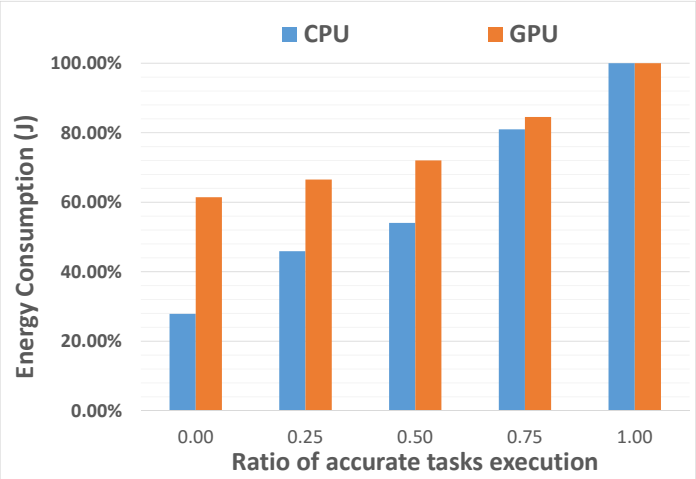


Figure 27: MDCT energy consumption vs ratio of accurate tasks

4.4.2 *MDDT*

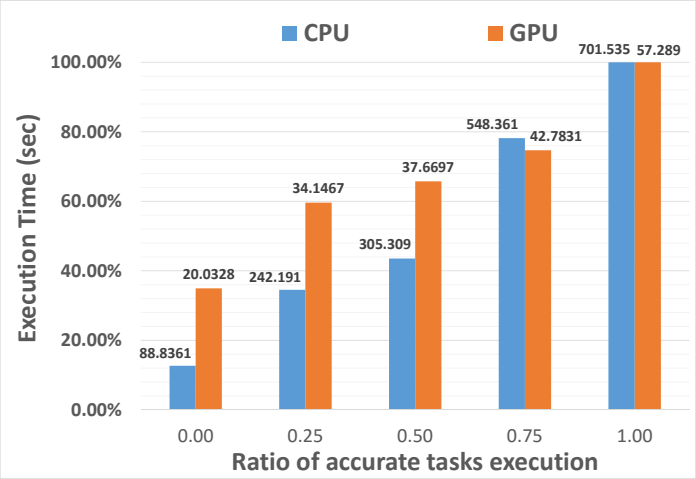


Figure 28: MDDT time execution vs ratio of accurate tasks

The time execution meets the gains of the already MDCT approximation with further reduction reaching of 65% for GPU and 87% for CPU between fully accurate and fully approximate executions (ratio 1.0 and 0.0 respectively). The double propagation of atoms in time is causing falsely containment of some atoms to the box edges (Figure 30) causing higher order of relative error in comparison with MDCT. Specifically at ratio 0.0 configuration relative error is 2.8% for average system total energy quality reduction and 3.1% for average pressure per atom. This order of magnitude for relative error is still acceptable. The energy consumption is respectively reduced 67% and 86% for GPU and CPU device executions.

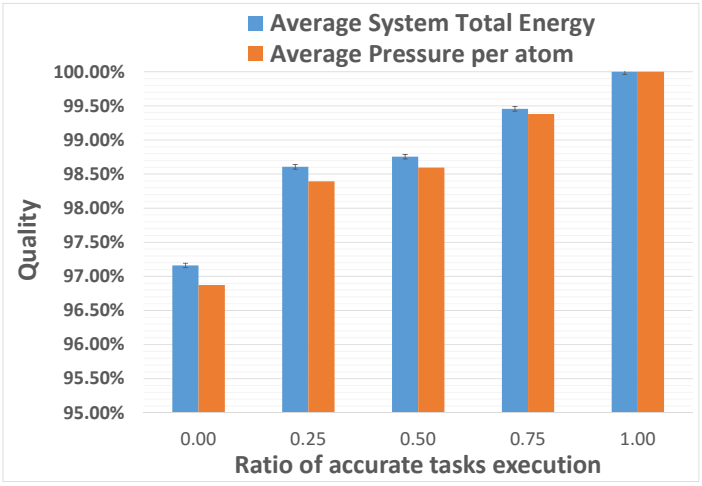


Figure 29: MDDT quality vs ratio of accurate tasks

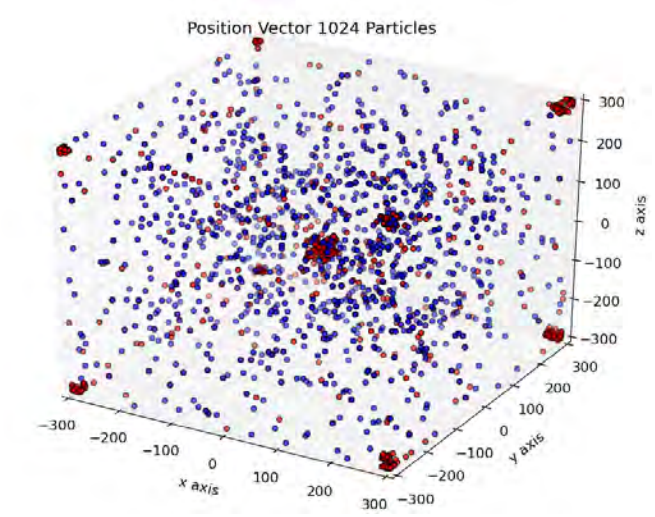


Figure 30: MDDT position vector of particles between approximate and accurate execution

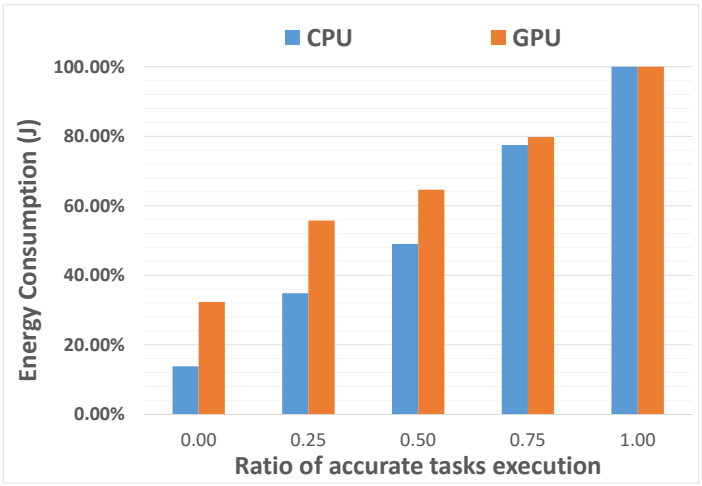


Figure 31: MDDT energy consumption vs ratio of accurate tasks

5

RELATED WORK

As this Msc thesis consists of two applications, in this section we expatiate related work which are some also consisted of many applications and often are a part of a bigger framework or a stand-alone benchmark suite. We classify related work in heterogeneity and approximate computing exploitation and further analysis.
All these approaches focus either on heterogeneity or approximate computing and some of them are performance-wise focused rather than energy consumption. Also most of them contain small size mostly kernel-scale applications which prohibit or distort the real impact on an application energy footprint and loss of output quality by exploiting heterogeneity and approximate computing.

5.1 APPROXIMATE COMPUTING

AxBench [22] is a benchmark suite combined with the NPU compiler, and is written in C++ and CUDA. It consists of seven pre-made benchmarks with the necessary annotations to work with the NPU compilation workflow. The set of benchmarks covers both CPU and GPU applications. ApproxIt [23] is an approximate computing framework for iterative methods with quality guarantees. Specifically is a lightweight quality estimator that is able to capture the solution quality of each iteration and use it to guide the selection of approximate computing mode in the next iteration. Thus, ApproxIt is able to gain application energy efficiency under quality guarantees. EnerJ [24] is an extension to Java that gives the ability to use type qualifiers that distinguish between approximate and precise data types. Each computation that follow, regarding the aforementioned data types is done either approximately or accurately according to these type qualifier.

5.2 HETEROGENEITY

Rodinia [25], is a benchmark suite that provides several applications that support execution on diverse accelerators. By this, it tries to capture the execution patterns and match the best accelerator to the consisting computational tasks of an application performance-wise. Opendwarfs [26] and Shoc [27] as Rodinia offers a varity of application that

support heterogeneous execution and tries to evaluate the efficacy of such parallel architectures and ultimately identify the architectural innovations that benefit the performance of an application.

6

CONCLUSION

In this MSc thesis we introduce the combination of heterogeneity and approximate computing to minimize energy consumption on non-trivial applications for energy footprint minimization. Experimental results indicate that it is realistic and beneficial to exploit the trade-off between energy consumption and output quality, by providing different approximation levels for each application.
As part of future work is to evaluate, the trade-off between power efficiency and error resilience on heterogeneous reliable / unreliable computing environments. Also another key direction is to expose the significance analysis of each application computations to an automatic –machine learning– technique.

BIBLIOGRAPHY

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European*, pp. 1–6, May 2013.

[2] K. Yamaguchi, D. McAllester, and R. Urtasun, "Efficient joint segmentation, occlusion labeling, stereo and flow estimation," in *ECCV*, 2014.

[3] J. E. Jones, "On the determination of molecular fields. i. from the variation of the viscosity of a gas with temperature," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 106, no. 738, pp. 441–462, 1924.

[4] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full chip leakage estimation considering power supply and temperature variations," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, ISLPED '03, (New York, NY, USA), pp. 78–83, ACM, 2003.

[5] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pp. 148–157, 2002.

[6] W. M. Arden, "The international technology roadmap for semiconductors—perspectives and challenges for the next 15 years," *Current Opinion in Solid State and Materials Science*, vol. 6, no. 5, pp. 371 – 377, 2002.

[7] A. Asenov, G. Slavcheva, A. Brown, J. Davies, and S. Saini, "Increase in the random dopant induced threshold fluctuations and lowering in sub-100 nm mosfets due to quantum effects: a 3-d density-gradient simulation study," *Electron Devices, IEEE Transactions on*, vol. 48, pp. 722–729, Apr 2001.

[8] H. Esmaeilzadeh, E. Blem, R. St.Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pp. 365–376, June 2011.

[9] P. Gepner and M. Kowalik, "Multi-core processors: New way to achieve high system performance," in *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, pp. 9–13, Sept 2006.

[10] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[11] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore gpus," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–10, May 2009.

[12] A. Munshi *et al.*, "The opencl specification," *Khronos OpenCL Working Group*, vol. 1, pp. l1–15, 2009.

[13] C. Lattner and V. Adve, "The llvm compiler framework and infrastructure tutorial," in *Languages and Compilers for High Performance Computing*, pp. 15–16, Springer, 2005.

[14] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 2, pp. 328–341, 2008.

[15] K. Yamaguchi, T. Hazan, D. McAllester, and R. Urtasun, "Continuous markov random fields for robust stereo estimation," in *Computer Vision–ECCV 2012*, pp. 45–58, Springer, 2012.

[16] K. Yamaguchi, D. McAllester, and R. Urtasun, "Robust monocular epipolar flow estimation," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 1862–1869, IEEE, 2013.

[17] A. Rahman, "Correlations in the motion of atoms in liquid argon," *Phys. Rev.*, vol. 136, pp. A405–A411, Oct 1964.

[18] L. Verlet, "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules," *Physical Review Online Archive (Prola)*, vol. 159, pp. 98–103, July 1967.

[19] L. Verlet, "Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules," *Phys. Rev.*, vol. 159, pp. 98–103, Jul 1967.

[20] R. Wolfe and J. R. Sams, "The virial theory of adsorption and the surface areas of solids," *The Journal of Physical Chemistry*, vol. 69, no. 4, pp. 1129–1135, 1965.

[21] S. Nosé, "A molecular dynamics method for simulations in the canonical ensemble," *Molecular physics*, vol. 52, no. 2, pp. 255–268, 1984.

[22] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 449–460, IEEE Computer Society, 2012.

[23] Q. Zhang, F. Yuan, R. Ye, and Q. Xu, "Approxit: An approximate computing framework for iterative methods," in *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, (New York, NY, USA), pp. 97:1–97:6, ACM, 2014.

[24] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *ACM SIGPLAN Notices*, vol. 46, pp. 164–174, ACM, 2011.

[25] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 44–54, IEEE, 2009.

[26] K. Krommydas, W.-c. Feng, M. Owaida, C. D. Antonopoulos, and N. Bellas, "On the characterization of opencl dwarfs on fixed and reconfigurable platforms," in *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, pp. 153–160, IEEE, 2014.

[27] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pp. 63–74, ACM, 2010.