Towards extending the SWITCH platform for time-critical, cloud-based CUDA applications: job scheduling parameters influencing performance

Louise Knight*, Polona Štefanič, Matej Cigale, Andrew C. Jones, Ian Taylor

School of Computer Science and Informatics, Cardiff University, Queen's Buildings, 5 The Parade, Cardiff, CF24 3AA

Abstract

SWITCH (Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications) allows for the development and deployment of real-time applications in the cloud, but it does not yet support instances backed by Graphics Processing Units (GPUs). Wanting to explore how SWITCH might support CUDA (a GPU architecture) in the future, we have undertaken a review of time-critical CUDA applications, discovering that run-time requirements (which we call 'wall time') are in many cases regarded as the most important. We have performed experiments to investigate which parameters have the greatest impact on wall time when running multiple Amazon Web Services GPU-backed instances. Although a maximum of 8 single-GPU instances can be launched in a single Amazon Region, launching just 2 instances rather than 1 gives a 42% decrease in wall time. Also, instances are often wasted doing nothing, and there is a moderately-strong relationship between how problems are distributed across instances and wall time. These findings can be used to enhance the SWITCH provision for specifying Non-Functional Requirements (NFRs); in the future, GPU-backed instances could be supported. These findings can also be used more generally, to optimise the balance between the computational resources needed and the resulting wall time to obtain results.

*Corresponding author

Email addresses: KnightL2@cardiff.ac.uk (Louise Knight), StefanicP@cardiff.ac.uk (Polona Štefanič), Matej.Cigale@ijs.si (Matej Cigale), JonesAC@cardiff.ac.uk (Andrew C. Jones), TaylorIJ1@cardiff.ac.uk (Ian Taylor)

Keywords: time-critical applications, CUDA, distributed cloud computing

1. Introduction

SWITCH (Software Workbench for Interactive, Time-critical and Highly self-adaptive applications) is a workbench for developing and deploying time-critical applications in the cloud[1]. SWITCH consists of three main subsystems: SIDE, DRIP, and ASAP. SIDE (SWITCH Interactive Development Environment) is the component that the software developer interacts with; it allows the developer to use a GUI to create an application's workflow and deploy the application, as well as manage applications once they are running. If an application has dependencies such that one node or service needs to be started before another, these can be specified graphically, and are then included in the deployment plan. DRIP (Dynamic Real-time Infrastructure Planner) is part of the back-end of the system; it takes all of the requirements specified by the developer, and uses them to plan the cloud-based infrastructure on which the application will run (further information is provided by Wang et al. [1]). ASAP (Autonomous System Adaptation Platform) is another subsystem of the back-end; it adapts the running application according to changes in the environment, with the aim of keeping the application's Quality of Service (QoS) at an acceptable level. ASAP has a load balancer, and allows for horizontal scaling.

As part of the SWITCH system, Qualitative Metadata Markers (QMMs) are generated and used to maximise the quality of an application. These QMMs allow us to see which constraints and Non-Functional Requirements (NFRs) have the most impact on the application's quality. In order to generate QMMs, an investigation (e.g. monitoring) must be done on the multi-cloud infrastructure we wish to deploy our application on, to discover which constraints, etc. have the most impact on an application's QoS. For more information on QMMs, see [2]. The investigation that is reported in the present paper provides information relevant to QMMs in the context of applications which make use of Graphics Processing Unit (GPU)-backed instances. For more detailed information about the various components within SWITCH, the reader is directed to [3].

[1]The SWITCH code repository can be found at https://github.com/switch-project/SWITCH-version-2

CUDA (Compute Unified Device Architecture) is an example of an architecture where GPUs can be used for general purposes. NVIDIA GPUs are programmed using the CUDA language, which is based on C with extensions for moving data to and from the GPU, running programs (kernels) on the GPU, etc. CUDA has a Single Instruction Multiple Data (SIMD) architecture. There are particular applications which suit the SIMD architecture; these are applications which run the same instruction on large quantities of data at the same time, before moving on to the next instruction. For example, the BEIA use case[2] of SWITCH performs many simulations at once in order to forecast possible natural disasters; as such, it would suit the SIMD architecture. SWITCH currently supports Amazon EC2 instances, but does not yet support EC2 GPU-backed instances.

Considering the multitude of time-critical applications already implemented in CUDA [4], it would be logical for SWITCH to support CUDA. CUDA is particularly well-suited for time-critical applications in that it is an architecture that can be utilised for High-Performance Computing (HPC), and therefore applications which suit its Single Instruction Multiple Data (SIMD) architecture can process more data, faster than using individual CPUs. The context of the present paper is to explore the implications of extending SWITCH to support CUDA (and GPU-backed instances). To understand how the SWITCH system, extended in this way, would be used in practice, we need to first investigate the constraints that impact on the performance of an application. Within this paper, we present an investigation of a representative CUDA application with a view to producing insights into how CUDA applications could be supported by SWITCH. Although every application is different, the insights generated are intended to be general enough to be applied (perhaps with some modification) to other applications.

The remaining sections of this paper are organised as follows: in Section 2 we present the related work, and in Section 3, we present an enhanced literature review, examining the QoS metrics which appear to be the most important to time-critical CUDA applications. In Section 4, we explain the experimental set-up and methods used to investigate how different parameters influence the QoS of an application, and in Section 5 we present the results of this investigation. Finally, in Section 6, we conclude the paper and in Section 7 discuss future work which could be undertaken.

[2]http://www.agile.ro/beia/

2. Related work

Most papers considered in this section compared specific applications' performances on different architectures. For example, Jackson et al. [5] ran seven different applications, as well as the High Performance Computing Challenge (HPCC) benchmark suite, on Amazon EC2, as well as two clusters and a supercomputer located in the Lawrence Berkeley National Lab. They used the m1.large instance type, which is a now-retired instance type for general-purpose computation [6]. Lenk et al. [7] developed a method to create a custom benchmark suite depending on the application that is needing to be deployed, and showed how this method could be used to compare Amazon EC2 to Flexiscale and Rackspace. There was a focus on CPU performance and monetary cost. Iosup et al. [8] compared Amazon EC2 with GoGrid, ElasticHosts, and Mosso on benchmarks which take into account resource and CPU usage, and communication time. The AWS instances used were of the general-purpose m1.* type, and the c1.* compute-optimised type. Mehrotra et al. [9] compared Cluster Compute instances on Amazon EC2 with a supercomputer, and a cluster, on several applications and benchmarks. Vöckler et al. [10] compared a scientific workflow on Amazon EC2 (without specifying the instance type), NERSC's Magellan cloud, and FutureGrid's sierra cloud. Other papers in this realm include those by Marathe et al. and by Berriman et al. [11, 12].

Bermudez et al. [13] passively gathered information on various AWS centres, and analysed this according to measurements such as response time, flow goodput, and network cost. Gohil et al. [14] inspected the performance of MapReduce applications on m1 (general-purpose) AWS instances, and recorded the execution time to finish each job (this is the most similar paper to ours, however it did use a different instance type).

Schad et al. [15] analysed the variance of wall clock times on Amazon EC2 general-purpose instances.

To the best of our knowledge, none of the previous papers considered AWS GPU-backed instances. Accordingly, the present paper investigates how such instances perform while varying certain job scheduling parameters.

3. Review

In this section, we present a literature review of the Quality of Service (QoS) metrics most important to time-critical CUDA applications. This

leads us on to the subsequent section, in which we describe the experimental set-up for an investigation into the constraints that impact the most on the QoS metrics that we deem most important in the present section.

In a previous paper [4], we performed a survey of time-critical CUDA applications with the aim of investigating which QoS parameters are most referred to in the papers describing these applications. Those parameters which are referred to most often are likely to be the most important to those specific applications. Four main fields of time-critical CUDA applications were investigated:

- Environment-related
- People/face detection
- Medical applications
- Materials-related

as well as a miscellaneous category for applications which did not fit into any of the above four categories. We now present an enhanced literature review by assessing more recent papers in these identified areas.

In the previous review, we searched for time-critical CUDA applications generally, and placed the papers found into categories based on what was discovered. For this paper, we inspected the papers within each category, and found newer papers which cite these ones, as some of the papers in the previous review were not very recent. We did not consider all papers found however; we would only consider those relevant to the category we were considering, e.g. environment-related applications.

In the current paper, we do not consider the miscellaneous category, but the terms referred to by papers in this category can be found in the previous review. For us to 'dig deeper' into a category with no defined theme would mean investigating the papers to the point where we would make new categories besides the four presented here.

3.1. Environment-related

First, we consider image processing-related applications within this field; following this we consider other environment-related applications which do not use image processing.

For those applications which involved image processing, we found several groups of methods concerning hyperspectral images. Wu et al. [16] and Wu et al. [17] both used algorithms for hyperspectral image classification. Sanchez et al. [18] examined the 'unmixing' of hyperspectral images. Ciznicki et al. [19] implemented hyperspectral image compression using the JPEG2000

algorithm. Goodman et al. [20] analysed marine imaging spectroscopy data using a method derived from that of Lee et al. [21, 22, 23]. A few papers concerned the implementation of methods for processing satellite images, namely Kurte and Durbha [24], Bhangale and Durbha [25], and Scott et al. [26]. Relating to non-image processing applications, Christgau et al. [27] implemented the tsunami simulation algorithm EasyWave [28] in parallel. Huang et al. [29] and Xu et al. [30] both used the Kalman filter method [31] to estimate hidden states in linear dynamic systems. Jiang et al. [32] implemented Multi-Scale Retinex with Colour Restoration (MSRCR) on the GPU with the aim of eventually developing a video processing system for NASA's Vision Enhancement for General Aviation (VEGA) project, which is meant to enhance pilot safety during times of poor visibility.

3.2. People/face detection

All of the applications within this category, unsurprisingly, use image processing. The Viola-Jones algorithm [33] (used in Adaboost [34]), or part thereof, was implemented in a large subset of the papers we collected in the field of people/face detection, specifically those of Sharma et al. [35], Chouchene et al. [36], Sun et al. [37], and Herout et al. [38]. Zhi et al. [39] used an ORB (Oriented FAST and Rotated BRIEF)-based algorithm for real-time image registration and target localisation. Iakovidou et al. [40] implemented a Colour and Edge Directivity Descriptor (CEDD) method for Content Based Image Retrieval (CBIR). Fauske et al. [41] and Weimer et al. [42] both implemented methods for background subtraction in video streams.

3.3. Medical applications

As in our previous review, because of the varied nature of medical applications, and the fact that there are many more than we would have the time and space to list, we have focused on the small number of specific applications that we could find. Most of these are image processing-related applications. Although there does not seem to be a great amount of variation in the methods/applications in this category, this is because, when we collected papers for our first review [4], there were simply so many papers which fit into this small number of applications; there is a lot of research being done on a small number of Medicine-related problems.

Several papers present algorithms to perform image registration, which involves aligning two images and is used in image-guided surgery: Muyan-Özçelik et al. [43], Ang Li et al. [44], Min Li et al. [45], Modat et al. [46],

Bhosale et al. [47], and Ikeda et al. [48]. In addition, we found several applications concerning image reconstruction: Keck et al. [49], Yu et al. [50], Sun et al. [51], Pang et al. [52], Zhao et al. [53], Riabkov et al. [54], Ziner and Keck [55], and Park et al. [56]. Regarding non-image processing applications within this field, there were several papers that we found, that of Wilson and Williams [57], Tam and Yang [58], and Juhasz [59], which all performed biological signal processing.

3.4. Materials-related

The first sub-category of materials-related applications covers ultrasonic imaging, namely those of Sutcliffe et al. [60] and Romero-Laorden et al. [61]. The second sub-category of materials-related applications was the use of Finite Element methods for the analysis of soft tissues; papers were from Strbac et al. [62] and Strbac et al. [63].

3.5. CUDA Metrics Analysis

In Table 1, we present an overview of the terms, synonyms, and units of measurement, found during our literature survey.

Table 1: QoS Parameters, Units Used, and Alternative Terms

Parameter	**Units**	**Other names**	**Sources**	**Percentage of total**
Time				
Runtime	Time units	Calculation speed, calculation time, computation burden, computational cost, computation time, computational time, computing time, convergence time, detection speed, detection time, elapsed time, execution speed, execution time, image reconstruction time, kernel execution time, kernel time, performance time, processing speed, processing time, reconstruction time, registration time, running time, scene time, simulation time, solution speed, solution time, time, time consumption, time cost	[25] [47] [36] [27] [19] [20] [38] [29] [48] [32] [59] [49] [24] [44] [45] [46] [43] [52] [56] [54] [61] [18] [26] [62] [63] [51] [37] [60] [58] [42] [57] [16] [17] [30] [50] [53] [55]	90

Runtime	Frames per second (fps)		[41] [60]	5
Per time-step runtime	Time units		[62]	2
Ratio of runtime to something else	Percentage	Performance ratio	[62]	2
Processing speed	Frames per second (fps), projections per second (pps), pixels per second, number of indexed images per second, Giga-(voxel)-updates per second (GUPS), voxels/s	Detection speed, frame rate, reconstruction speed, registration throughput, tracking rate	[41] [40] [32] [52] [61] [35] [55]	17
Processing rate	ms/pixel, ms/frame	Computation time	[20] [39]	5
Overall delay	Time units		[44]	2
Data transfer time	Time units	Memory transfer time	[48] [43] [54] [53]	10
Proportion of computational cost due to data transfer	Percentage		[61]	2
Pre-processing cost	Time units	Pre-processing time	[56] [50]	5
Time variation from reconstruction to reconstruction	Time units		[56]	2
Quality of results				
Accuracy	Percentage	Classification accuracy, detection accuracy, overall accuracy (OA)	[35] [16] [17]	7
Correlation coefficient (accuracy)			[20]	2
Average absolute difference (accuracy)	No units, percentage	Average absolute pixel value difference	[20] [56]	5
Average accuracy (AA)	Percentage		[16]	2
Sensitivity	Percentage	Correct detection rate, detection rate, true positive rate	[37] [58] [42]	7
False positives	Count, rate		[35] [58]	5
False detection rate	Percentage		[37]	2
Peak signal-to-noise ratio	dB		[52]	2

Mean-square-error	mm	Intensity standard deviation, root mean-square-error (RMSE)	[52] [62] [50] [39] [55]	12
Residual error	mm	Registration error, spatial error, target registration error (TRE)	[47] [44] [45]	7
Maximum absolute error	mm		[62]	2
Structural Similarity (SSIM) index			[50]	2
Contrast-to-noise ratio (CNR)			[56]	2
Spectral angle	degrees		[19] [18]	5
Standard deviation for different experimental runs			[40]	2
Maximum error of deformation vector	Voxels		[48]	2
Data				
Throughput	GB/s, GiB/s, voxels per time unit	Registration throughput	[47] [48] [44]	7
Throughput	Gflops/s, Gflops		[30]	2
Memory requirements	KB, MB, GB	Data size, memory footprint of compressed data format, space needed	[61] [50] [53]	7
Efficiency of memory throughput	Percentage		[48]	2
Costs (mentioned only briefly)				
Performance-per-watt			[54]	2
Performance-per-dollar			[54]	2
Implementation cost	Dollars	Development cost	[60]	2
Power				
Energy consumption	Watts, Joules		[36]	2

One interesting finding relating to the distinction between image processing and non-image processing applications (within the four categories above) was that the accuracy of results is referred to more in image processing applications than non-image processing applications. This may be due to the

exact nature of some specific image processing applications. For example, if the application is relating to safety, such as in applications which analyse video streams of street crossings, or in intra-operative applications, then having accurate results is obviously important to ensure people's safety.

Another finding of this literature review is that there are many different terms for runtime, and runtime is, unsurprisingly, the most-referred to QoS measure overall. Because of this, we focus on measuring the impact of different parameters on runtime in our experiments.

4. Experimental background and set-up

In the previous section, it was found that runtime is the QoS metric most referenced in the time-critical CUDA applications in our literature review. In this section, we describe the experimental set-up used, and the constraints varied, in order to find out which constraints have the most impact on runtime (which we call 'wall time'; the time taken to return the results for all problems set, in contrast to the 'runtime' to solve one problem). In the next section, we present the results of these experiments.

The focus of this experimentation is to provide a methodology and blueprint for time-critical CUDA applications. While many different types of CUDA applications exist, the results presented here do not depend on any specific application. We did not intend to generate all of the "rules" (e.g. "do not use more than x instances to run y problems") for all applications, or categories of application, because it is not practical, and somewhat redundant, to do so. Consequently, we hope that these results will provide insight and a blueprint for those who wish to perform experiments using varying constraints, and measuring results for any type of time-critical CUDA application. Our insights are presented in Section 5.

4.1. Instances

Amazon Web Services (AWS) has an Elastic Compute Cloud (EC2)[3] which provides users with many different kinds of instance types to deploy and use. For general purposes (i.e. not specifically graphics processing), there are two classes of GPU-backed instances available: P2 and P3 instances. The main difference between these is that the P3 instances use more recent graphics cards. For this paper, we performed experiments using P2 instances, so

[3]https://aws.amazon.com/ec2/

our description here focuses on those, although the description can easily be mapped to P3 instances. p2.xlarge, p2.8xlarge, and p2.16xlarge contain 1, 8, and 16 GPUs respectively. We chose to use p2.xlarge instances, i.e. single-GPU instances, as we could then vary the number of GPUs and show how applications can scale as this is changed.

The number of instances of the type p2.xlarge launched in any one AWS Region is limited to 8, so we varied the number of instances between 1 and 8 (1, 2, 3, 4, 5, 6, 7, 8 instances).

4.2. Application

Even though many different time-critical CUDA applications were described in the previous section, in the case of varying constraints and measuring results, the exact application does not matter; the results can be relevant to most applications. As such, we used a CUDA implementation of a method for solving a Bioinformatics problem [64, 65, 66]. We chose this specifically as the sizes of the problems, and the number of concurrent problems given to the application can be scaled up or down quite easily.

Although the specific application chosen does not seem to have QoS requirements in the traditional sense, the results from experiments conducted using this application can be related to other CUDA applications in that, in certain contexts, some loss of quality can be acceptable if the system cannot maintain a particular QoS. A user could specify that they want their results to be accurate to within 0.01%, or they may wish to trade-off between accuracy and resource utilisation. The Bioinformatics application chosen is related to similar applications where a preference needs to be made between the quality of results and the resources available. Accordingly, we have a means of varying the number of concurrent tasks, the resource requirements of each task, and the deadlines within which the tasks must be performed. This allows us to derive results which are of interest in the deployment of other time-critical applications. Also, with reference to CUDA, the choice of experimenting with a single application is sensible, since the underlying architecture of applications based on CUDA is fairly homogeneous since the CUDA hardware is quite specific. This means that most problems differentiate themselves by the type of input they receive, which we simulated with the chosen application (see below).

4.3. Number of problems

We decided to double the number of problems given to the instances to compute, starting from 10, to yield 10, 20, 40, 80, and 160 problems. As a note, each problem is simply a different input file into our CUDA program, so when we give a set of instances 10 problems to work on, say, what we are really doing is giving them a list of 10 filenames, and telling them to compute the output for each of those files.

4.4. Problem size

The problem size was varied to be either constant or random (without varying the size within the "constant" category) because if the problems are all the same size, then theoretically the time to perform one problem on one instance should be the same as the time to perform another problem on another instance, as all the instances in our experiments have identical architectures, etc.

4.5. Scheduling

We varied the order the problems were scheduled in because the order can have an impact on how long it takes for the problems to be solved (i.e. for all the results to be returned). We chose to first schedule by randomly ordering the problems in a list, and then ordering them in increasing size from smallest to largest (using file size as a measure of this) (this is commonly called Shortest Job First (SJF) scheduling [67]).

4.6. Summary of constraints varied

To summarise, the constraints varied were:

- Number of instances (1 – 8)
- Number of problems (10, 20, 40, 80, 160)
- Problem size (constant, random)
- Order that problems are scheduled (randomly, by size increasing)

While we varied these constraints, we collected information on the amount of time each instance took to complete, and how the problems were distributed amongst instances (which depends on the order that problems are scheduled in, as well as the size of the problems, as this will influence when certain results will be ready relative to others).

4.7. Technical details

Only a subset of Amazon Regions supports the P2 GPU-backed instances (which have NVIDIA K80 GPUs); we chose to use Ohio (us-east-2). The Amazon Machine Image (AMI) used was Ubuntu Server 16.04 LTS (HVM), SSD Volume Type (ami-916f59f4). The version of CUDA installed on these instances was 9.0. The instances were connected to a single (shared) Elastic File System (EFS) storage volume, on which the input files and results were stored. A Simple Queue Service (SQS) First-In-First-Out (FIFO) queue was used to store the list of input files; each instance was running a program such that once an instance had finished working on one problem, it would access the queue and retrieve (and delete) the next filename from the queue. Using a FIFO queue guarantees that each message (filename) is delivered exactly once, and preserves the ordering of the messages.

5. Results

In this section, we shall investigate the impact that different constraints have on wall time as our main measure of performance.

5.1. Relationship between the number of instances and wall time

It is intuitive to assume that as we increase the number of instances working on a set of problems, the wall time to retrieve the results for this set of problems should decrease. Indeed, plotting the number of instances against the wall time for all experiments run on that number of instances (see Figure 1), we find such a relationship. What does this mean when setting up a cloud-based architecture for time-critical applications? Depending on the nature of the applications and the necessary QoS, it may not be necessary to use the maximum allowed number of instances; indeed, averaging over all experiments, one can find that the biggest benefit comes from increasing the number of instances working on a set of problems from 1 to 2 (this corresponds to a (average time for 1 instance 88.11 - average time for 2 instances 51.46)/88.11 = 42% decrease in average wall time. Increasing the number of instances further results in smaller and smaller benefits, although going from 1 to 8 instances (the maximum number of p2.xlarge instances one can run in any one Amazon Region) results in an overall (average time for 1 instance 88.11 - average time for 8 instances 32.93)/88.11 = 63% decrease in wall time.

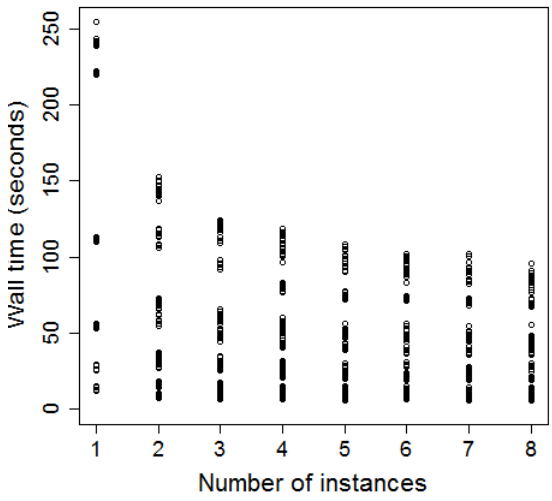

Figure 1: Graph showing the number of instances plotted against wall time for all experiments.

Considering each set number of problems individually, we find that the relationship looks slightly different each time. Unsurprisingly, overall, the wall time increases as the number of problems being worked on increases (this can be seen looking at the scales on the graphs in Figure 2).

Looking at all of the different numbers of problems collectively, it can be plainly seen that the benefit to wall time is greater the bigger the number of problems, which is intuitive; depending on just how 'time-critical' an application is and also how many resources (money) there is available, it may or may not be worth increasing the number of instances working on a problem depending on the number of problems that need results.

5.2. Relationship between problem size and wall time

We now consider the impact of having different-sized problems. We varied the problem size by having all problems be either a constant problem size, or varied in size. Overall, the wall time increases as the number of problems being worked on increases (this can be seen looking at the scales on the graphs in Figure 3). As the number of problems goes up, we can see that more and more often, varying the problem size gives on average, higher wall times than keeping the problem size constant. This does make some sense

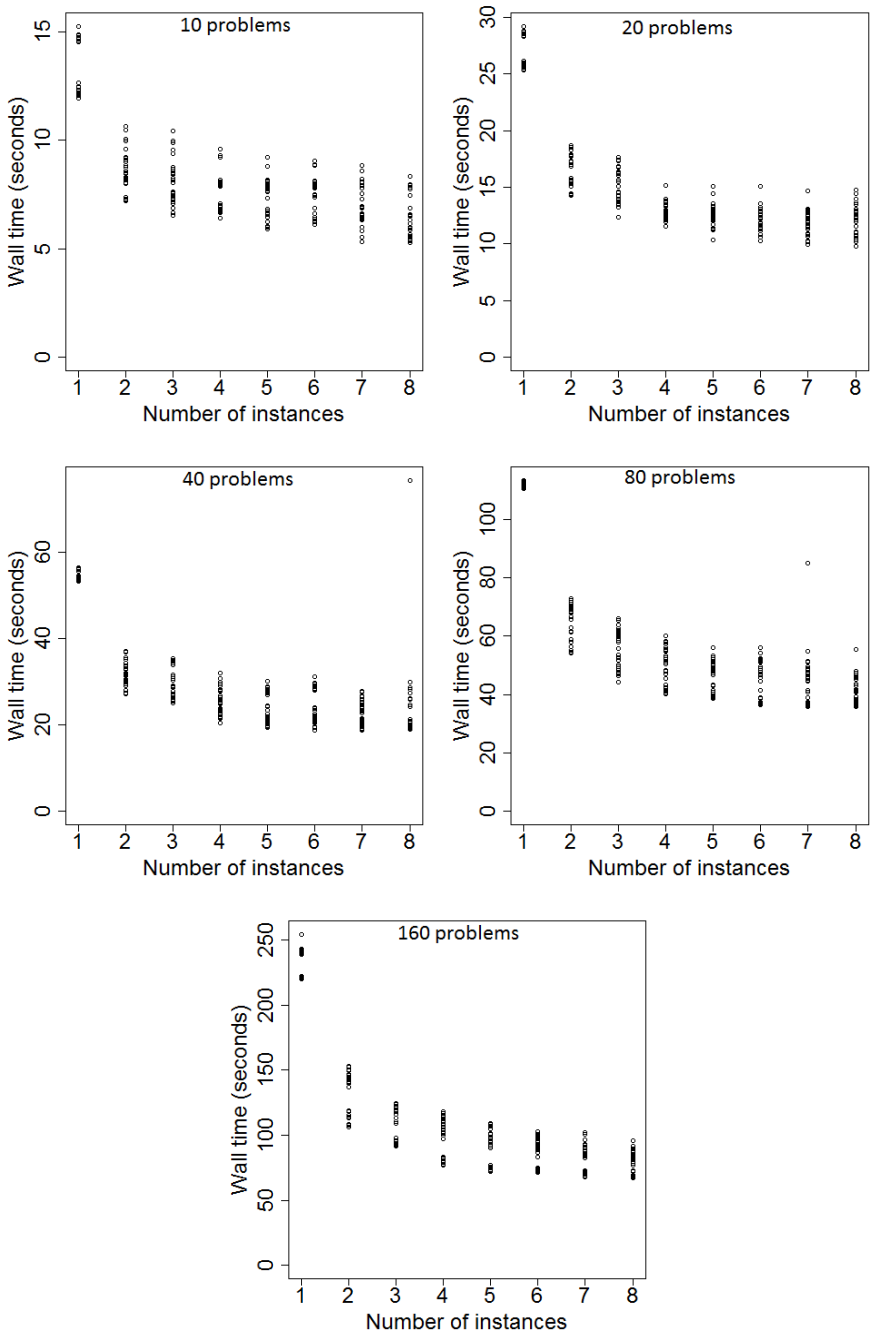

Figure 2: Graphs showing the number of instances plotted against wall time for experiments with 10, 20, 40, 80, and 160 problems, respectively.

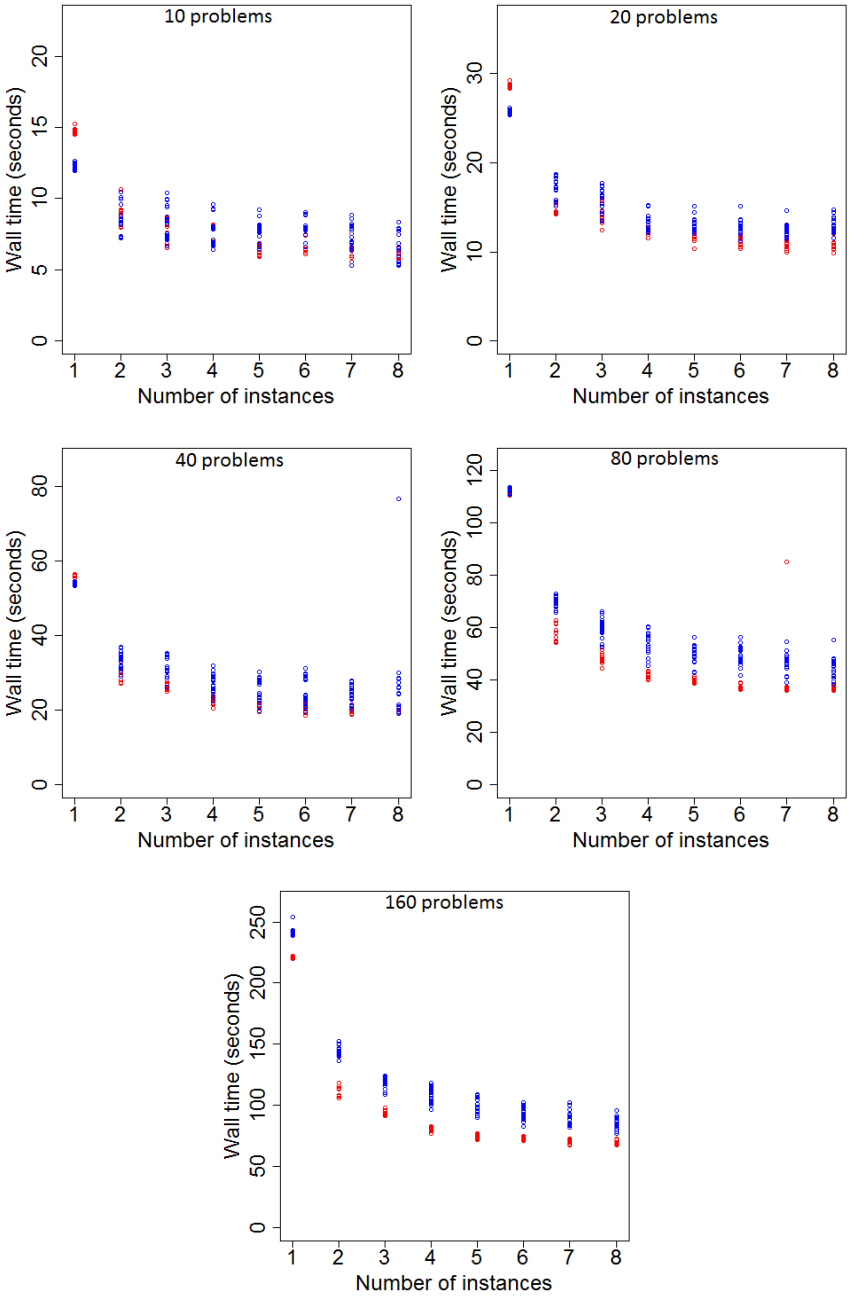

Figure 3: Graphs showing the number of instances plotted against wall time for experiments with 10, 20, 40, 80, and 160 problems, respectively. Red points show constant problem size, while blue points show varying problem size.

Number of problems	Variance	Minimum	Maximum
10	386388	972	3160
20	982801	972	4752
40	4288649	621	10182
80	4239192	274	10182
160	6694622	267	19390

Table 2: The variance of problem sizes for each number of problems, as well as the minimum and maximum values (in terms of number of amino acids). See text for further explanation.

intuitively, as the constant-sized problem can be viewed as being of 'medium' size, whereas when varying the size, we can have much larger problem sizes included (the variances and ranges in problem size for different numbers of problems can be seen in Table 2). (The problem size is measured in total number of amino acids, as each 'problem' can be represented as a matrix of amino acids, with an associated length and width.) However, in some cases, constant problem size gives higher wall times than varying the problem size. It is uncertain why this could be, but it is worth noting that with the varied problem size experiments, we also varied the ordering of the problems in the queue (if the problem sizes are constant, we cannot vary the ordering in any meaningful way). The size of the problems we are working on is not really something we can control (past the fact we could decide to use a different architecture), but investigating this parameter is interesting nonetheless.

5.3. Relationship between problem ordering and wall time

To see what effect the ordering of the problems in the queue has on wall time, we look to the graphs in Figure 4. We order the problems randomly, and by size (from smallest problem size to largest). We can first see that for the smaller numbers of problems, there is less of a pattern in which kind of ordering gives better wall time performance, which is to be expected as with such a small number of problems, it is inevitable that we shall see more variation. For larger numbers of problems, we often see that ordering by size gives greater wall times than ordering the problems randomly. This may be because when ordering by size, adjacent problems in the queue will be close in size, and so instances taking problems from the queue may finish their work (and attempt to access the queue again) around the same time, which may cause some blocking for queue access. From this, we can see that we

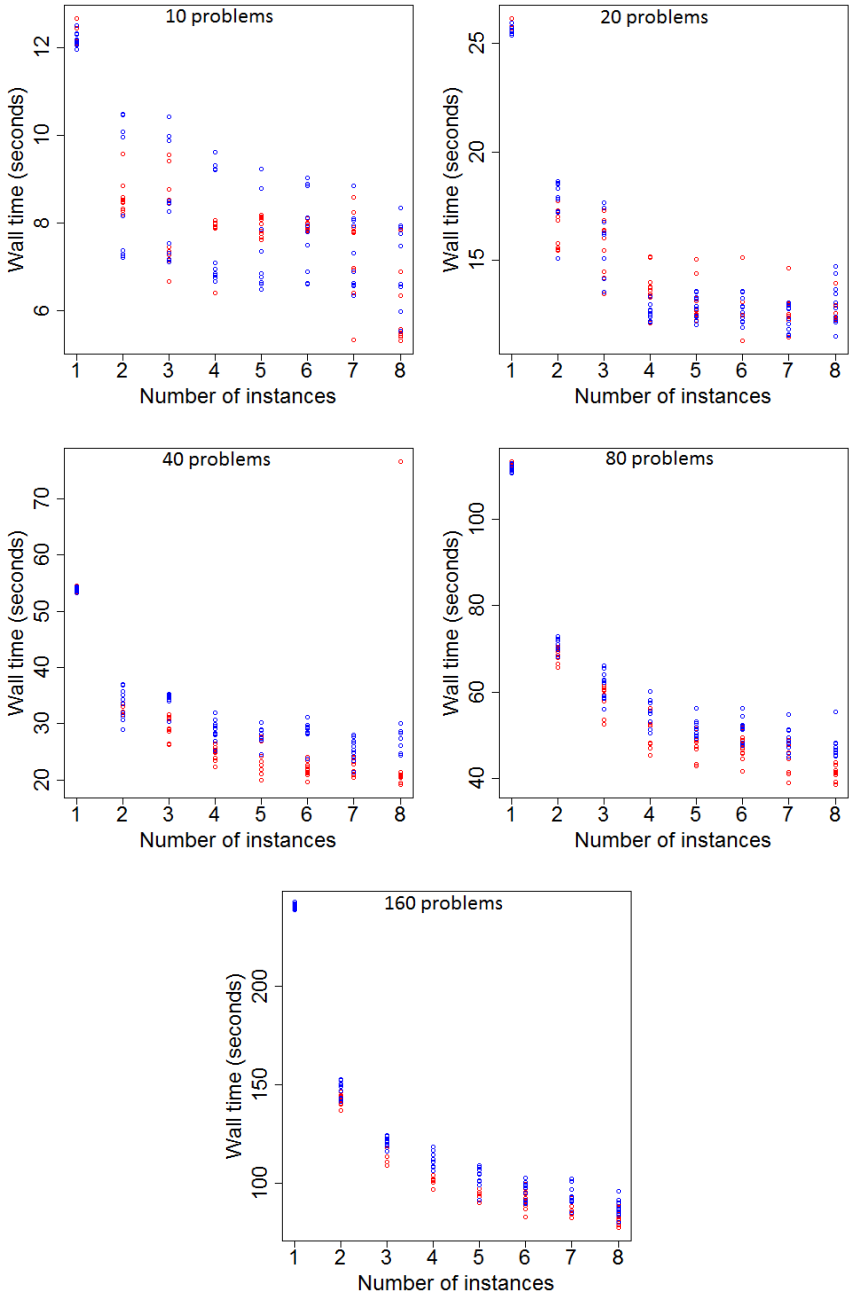

Figure 4: Graphs showing the number of instances plotted against wall time for different numbers of problems. Red points show experiments where the problems are randomly ordered, blue where the problems are ordered from smallest problem size to largest.

Table 3: Instances which are wasting time doing nothing. We only list numbers of instances where at least one experiment has at least 1 instance which did not process any problems. No instances were wasted for 80 or 160 problems for any of the numbers of instances, so these columns have been omitted for brevity. The percentage of experiments for that number of instances where at least one instance wasted time is given in brackets in the column 'No. of experiments with at least one wasted instance'. The following columns show the counts of how many problems were in those experiments (the maximum in any of the cells in these three columns is 30).

No. of instances	**No. of experiments with at least one wasted instance**	**10 problems**	**20 problems**	**40 problems**
4	1 (0.7%)	1	0	0
5	17 (11%)	16	1	0
6	27 (18%)	20	6	1
7	42 (28%)	30	12	0
8	46 (31%)	30	15	1

should order the problems in the queue randomly when using larger numbers of instances, but that it does not make as much of a difference for smaller numbers of instances.

5.4. Are there instances which are doing nothing?

Table 3 shows how often instances waste time doing nothing, i.e. how often an instance is launched to solve some problems, and ends up doing no work at all. We can see that as the number of instances increases, we have more cases where at least one instance is wasting time doing nothing, and more often than not, it happens for smaller numbers of problems. This is unsurprising; we would not expect all instances to do work if we have 10 problems spread across 8 instances (if each instance picks the next problem to work on from a queue, it may happen that some instances are not able to obtain access to the queue, and if some instances have more success accessing the queue than others, those instances will inevitably do more work than others).

In Table 4, we can see exactly how many instances are being wasted. As the number of instances increases, the number of instances being wasted also

Table 4: The number of experiments for each number of instances, where we are wasting a particular number of instances.

Number of instances	**1 instance wasted**	**2 instances wasted**	**3 instances wasted**
4	1	0	0
5	16	1	0
6	22	5	0
7	23	13	6
8	21	15	10

increases. This could result in a non-trivial amount of money/resources being wasted as sometimes we are not using almost half of the launched instances.

From what we have found, we can derive the following rules:

- Do not use more then 4 instances to run 10 problems
- Do not use more than 5 instances to run 20 problems

If we could run more than 8 instances at once within a Region, we may find that we should not use more than 8 to run 40 problems. This is intuitive but important to bear in mind. Considering what we found earlier concerning the potential benefit to average wall time from increasing our number of instances from 1 to a certain number, we can say that depending on how the workload may change over time, we could get by with only 2 or 3 instances if we know for certain that we will never have to process more than 10 or 20 problems at once. Launching more instances than this is only worthwhile if we have enough work to do.

5.5. Relationship between workload distribution across instances and wall time

Next we consider whether there is a correlation between how the problems (files) are distributed across instances (represented by the standard deviation of the number of files per instance), and the wall time. We did not consider the results for 1 instance only here, as the standard deviation will always be 0 for the file distribution. Figure 5 shows these two variables plotted. The simple correlation between the two variables is 0.63, suggesting a moderately-strong relationship. This makes sense intuitively, since if the standard deviation of the files is high, this means some instances will be

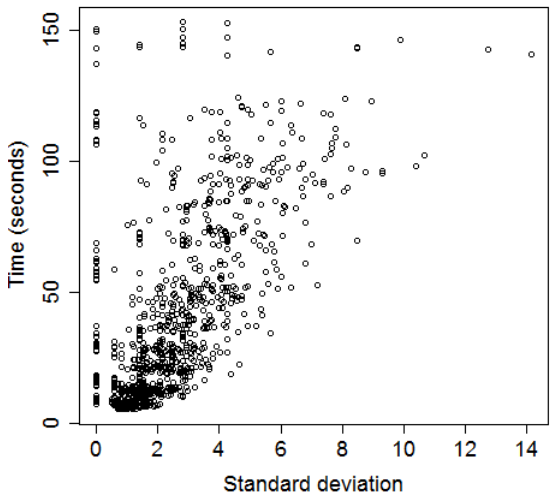

Figure 5: Graph showing the standard deviation of the number of files per instance against the wall time, for all experiments except those for 1 instance only.

doing more work than others (since there is overhead in starting a CUDA program), and this will take longer than if the problems were distributed evenly.

Inspecting each possible number of instances (except 1) individually, we can see that this relationship (and correlation) changes slightly depending on the number of instances (see Figure 6 a and b). The correlation between the standard deviation of the number of files per instance and the associated wall times is 0.46 for 2 instances, 0.77 for 3, 0.71 for 4, 0.82 for 5, 0.81 for 6, 0.81 for 7, and 0.81 for 8 instances. As the number of instances is increased, the possible ways one could spread problems across instances increases (with the possibility for larger standard deviations), so this makes sense. All of this contributes to the overall message that one should only launch as many instances as one can feasibly provide enough work for.

This relationship cannot be seen when breaking down the results according to number of problems only.

We can see some correlation between the standard deviation of the file counts across instances and the wall time when considering the two file size options individually (Figure 7). Also worth noting is that the correlation is larger for randomly-sized files than for constant file sizes (0.68 and 0.48

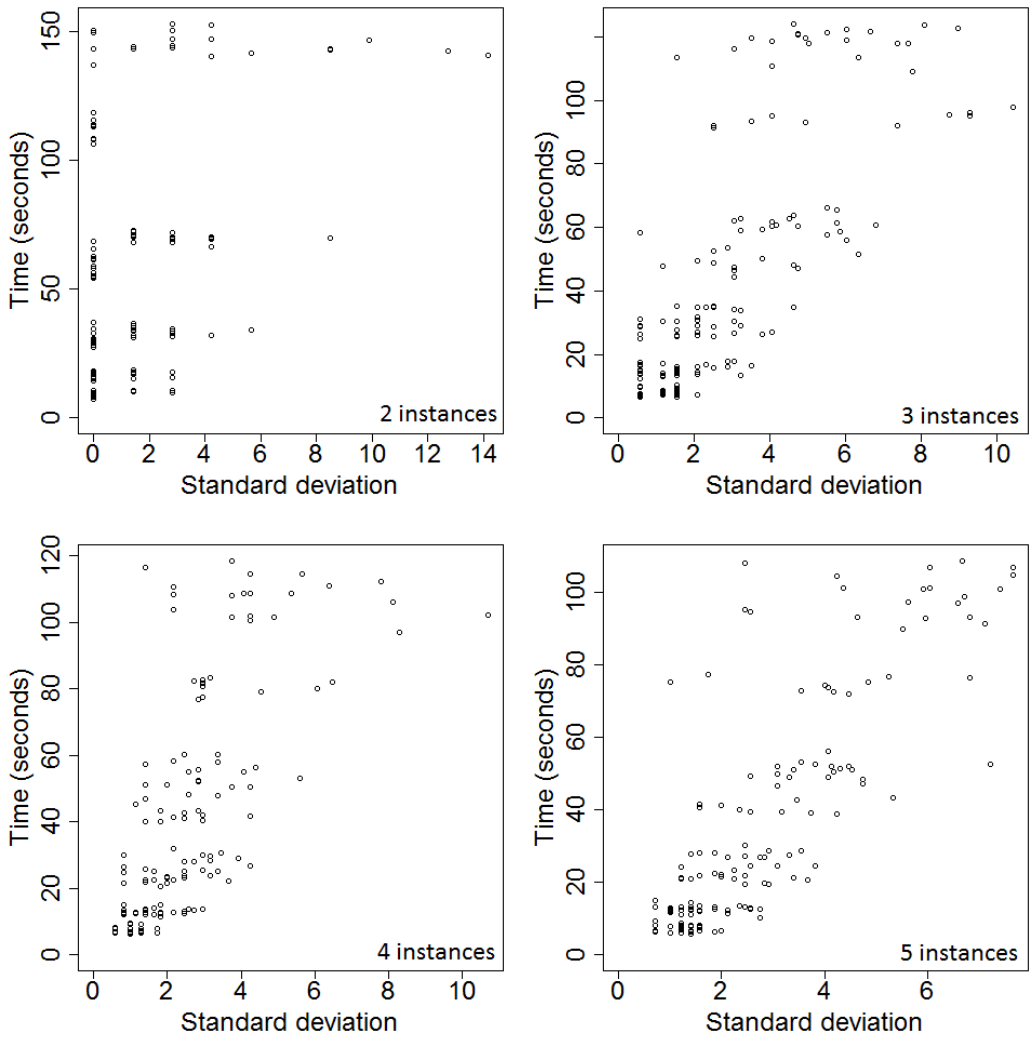

Figure 6: (a) Graphs showing the standard deviation of the number of files per instance against the wall time, for different numbers of instances (2 – 5).

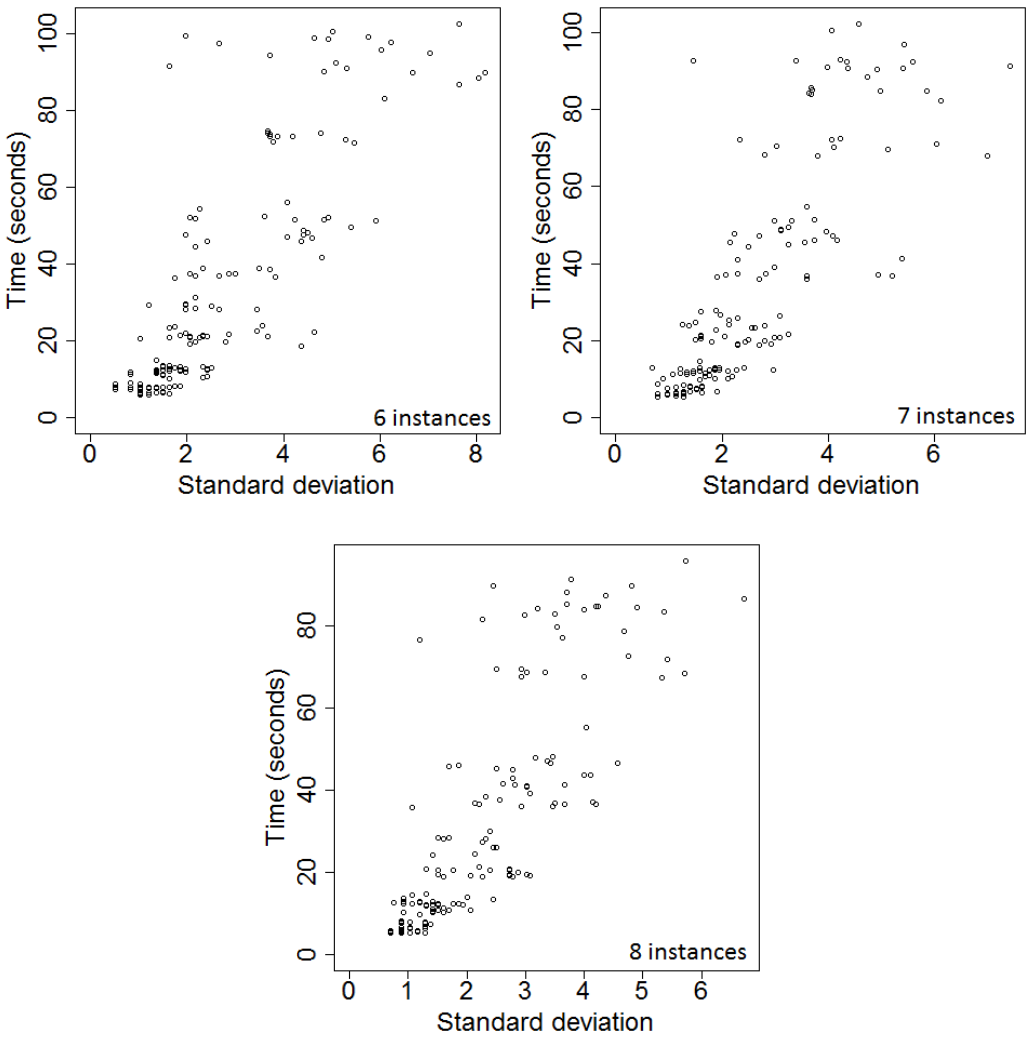

Figure 6: (b) Graphs showing the standard deviation of the number of files per instance against the wall time, for different numbers of instances (6 – 8).

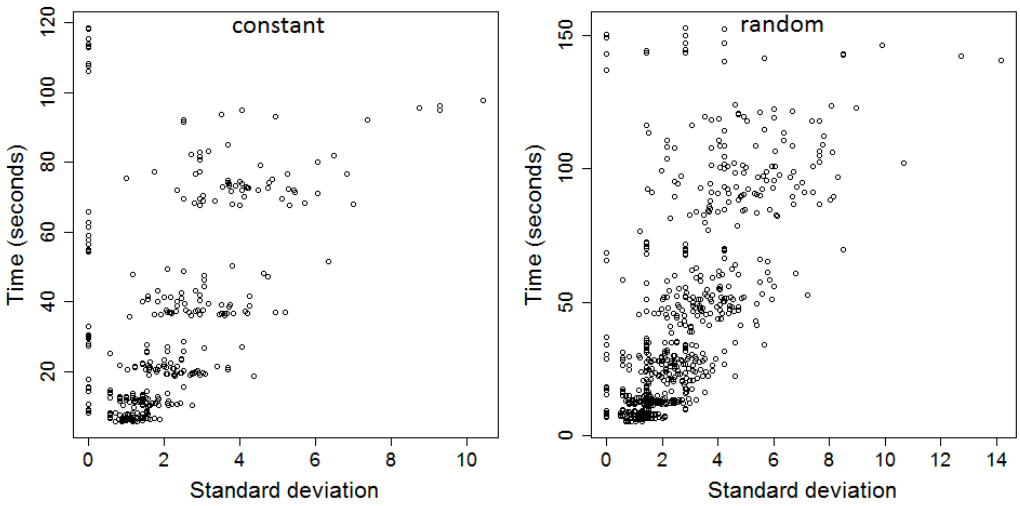

Figure 7: Graphs showing the standard deviation of the number of files per instance against the wall time, for experiments with constant sized problems and randomly-sized problems.

respectively). We believe this is due simply to the fact that we can obtain more variation in wall time for individual randomly-sized problems.

Considering the two possible orderings of randomly-sized problems, ordering the problems randomly gives a higher correlation to wall time than ordering by size (correlations are 0.71 and 0.66), although not by much (see Figure 8).

Because a lot of results have been presented in this section, here we synthesise these comments and show which of the results were exactly as expected, which were not as expected but that we can explain, and what we were unable to explain (as they may be due to factors beyond the average AWS customer's knowledge or ability to influence):

- Exactly as expected:
  - As the number of instances working on a set of problems is increased, the wall time to return results for all problems decreases
  - As the number of problems being worked on increases, the wall time increases

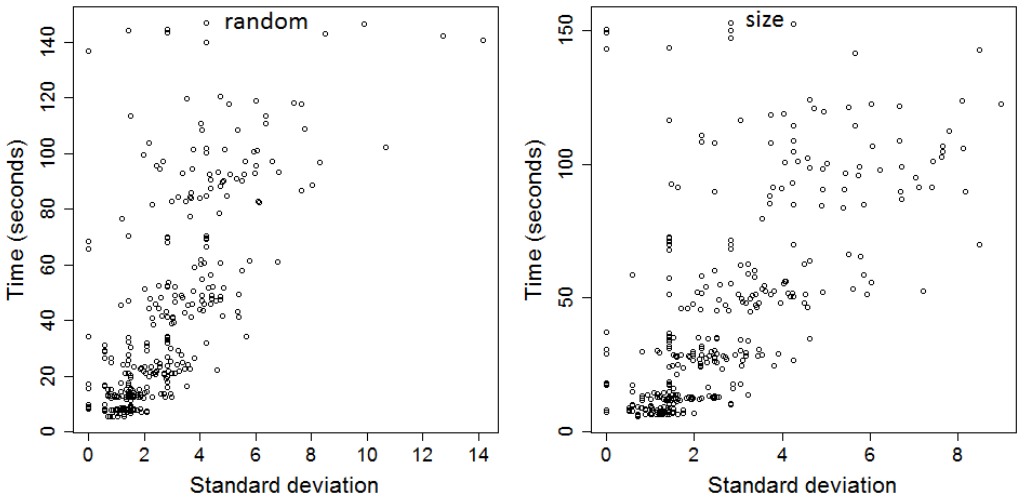

Figure 8: Graphs showing the standard deviation of the number of files per instance against the wall time, for experiments where the problems are ordered in the queue randomly, and by size from smallest to largest.

  - The benefit to wall time (from increasing the number of instances) is greater the bigger the number of problems that are being processed
  - As the number of problems increases, varying the problem size gives higher wall times than keeping a constant problem size
  - As the number of instances increases, we have more cases where at least one instance is wasting time doing nothing; more often than not, this happens for smaller numbers of problems. Also, as the number of instances increases, the number of instances being wasted increases
  - There is a moderately-strong correlation relationship between workload distribution across instances and wall time. This relationship is stronger for larger numbers of instances, when we inspect these individually

- Not as expected but can be explained:

  - In some cases, constant problem size gives higher wall times than varied problem size; possibly due to the fact that if we vary the

problem size, we can also vary the ordering that the problems are completed in (in the queue)
  - For larger numbers of problems, ordering by size gives greater wall times than ordering the problems randomly; when ordering by size, adjacent problems in the queue will be close in size, and so instances taking problems from the queue may finish their work (and attempt to access the queue again) around the same time, which may cause some blocking for queue access
  - Correlation between SD of file counts across instances and the wall time is larger for randomly-sized files than for constant file sizes, possibly because we can obtain more variation in wall time for individual randomly-sized problems

- Unable to explain:

  - The relationship between workload distribution across instances and wall time cannot be seen when breaking down the results according to number of problems

In this section, we investigated the impact of different parameters on the overall wall time, and used the results of this investigation to derive some recommendations that could be used to ensure a good balance between wall time achieved for a problem set and the amount of resources needed to achieve that performance.

6. Conclusions

In this paper, we extended our previous review of time-critical CUDA applications [4], finding that unsurprisingly, runtime is the most important QoS metric for these applications. With this in mind, we conducted experiments on Amazon Web Services GPU-backed instances, varying the number of problems, number of instances, the ordering and size of the problems, to investigate the effect these variables have on the wall time to retrieve results for the set problems. From these experiments, we found that it is not necessary to use the maximum allowed number of instances in order to see a benefit to wall time; indeed, increasing the number of P2 single-GPU instances working on a set of problems from one instance to two gives a 42% decrease in average wall time. We also found that as the number of problems increases, varying the problem size tends to give higher wall times than keeping the

problem size constant. For larger numbers of instances, ordering the problems in the queue by size (from smallest to largest) gives greater wall times than ordering problems randomly. There are also times where instances can be wasted doing nothing, and there is a moderately-strong relationship between how problems are distributed among instances and the resulting wall time. Taking all of these findings into account, it is imperative that we think about how many resources are actually needed, given the problem size we have, in order to not waste time and money. This information will be useful to develop SWITCH in the future to support GPU-based instances.

Even though we only performed experiments on one application, the insights listed in Section 5 are generally-applicable to all applications, as the insights are not coupled to the application used. However, it should be emphasised that the rules derived, also in the previous section, are strongly dependent on the application being run; it is recommended that developers profile their applications on sufficiently large test configurations in order to determine what the parameter values would be (x and y for "Do not use more than x instances to run y problems", for example) for their scenarios.

7. Future work

Relating to the fact that the rules generated are linked to the application being run, a useful area of future work would be to derive a set of application-independent rules, such as those rules defined in Section 5, but which are more generally-applicable. This would possibly require using machine learning, or a similar approach.

In this paper, we did not consider network factors, such as latency, and the start-up time when loading a problem onto a GPU. In the case of AWS, there is the ability to use the Elastic Network Adapter to provide enhanced networking to multiple instances. We did not consider this, as this was not related to the other parameters we wished to vary; in this paper, we primarily focused on the problems set, and numbers of instances. We also did not consider the start-up time when loading a problem onto a GPU, because each P2 instance used is architecturally the same, and the start-up time would be more to do with the GPU on that instance than anything we could control. Clearly another possible area of future work would be to consider these factors too.

In the enhanced literature review, we found that runtime was the most-referred to QoS measure, and measured different parameters' impacts on this

accordingly. Accuracy was also found to be important to image processing-related applications specifically, so we could investigate this in the future. Given certain time constraints, we could find out the most accurate results we can obtain. Bearing in mind that the application we used for our experiments was non-image processing-related, we could either find an appropriate image processing CUDA application to use as an example, or find some other application where, given more and more time, we can obtain better and better results. In the application we used, if we were to stop the application at some point, we would not obtain results at all; the program needs to run to completion in order for us to obtain results.

In addition, in the future, the impact of placing instances in different Amazon Regions could be investigated. At the time of writing, the number of P2 single-GPU instances one can launch in a single Amazon Region is set at 8, and so if more computing power were necessary, using different Regions would also be necessary. One would need to research how to connect different instances together across Regions, which would require a different experimental set-up than that used in this paper.

Acknowledgements

The research reported in this paper was funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 643963 (SWITCH project).

[1] J. Wang, A. Taal, P. Martin, Y. Hu, H. Zhou, J. Pang, C. de Laat, Z. Zhao, Planning virtual infrastructures for time critical applications with multiple deadline constraints, Future Generation Computer Systems 75 (2017) 365–375.

[2] P. Stefanic, M. Cigale, A. Jones, V. Stankovski, Quality of Service Models for Microservices and Their Integration into the SWITCH IDE, Proceedings - 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2017 (2017) 215–218doi:10.1109/FAS-W.2017.150.

[3] P. Stefanic, M. Cigale, A. C. Jones, L. Knight, I. Taylor, C. Istrate, G. Suciu, A. Ulisses, V. Stankovski, S. Taherizadeh, G. Flores Salado, S. Koulouzis, P. Martin, Z. Zhao, SWITCH workbench: A novel approach for the development and deployment of time-critical

microservice-based cloud-native applications, Submitted to Future Generation Computer Systems.

[4] L. Knight, P. Stefanic, M. Cigale, A. C. Jones, I. J. Taylor, Towards a methodology for creating time-critical, cloud-based CUDA applications, in: Proceedings of IT4RIs 18: Interoperable infrastructures for interdisciplinary big data sciences- Time critical applications and infrastructure optimization, Amsterdam, 2018. doi:10.5281/zenodo.1162877.
URL https://doi.org/10.5281/zenodo.1162877

[5] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, N. J. Wright, Performance analysis of high performance computing applications on the Amazon Web Services cloud, Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010 (2010) 159–168doi:10.1109/CloudCom.2010.69.

[6] Amazon Web Services, Previous Generation Instances (2018).
URL https://aws.amazon.com/ec2/previous-generation/

[7] A. Lenk, M. Menzel, J. Lipsky, S. Tai, P. Offermann, What are you paying for? Performance benchmarking for Infrastructure-as-a- Service offerings, Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011 (2011) 484–491doi:10.1109/CLOUD.2011.80.

[8] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, Performance analysis of cloud computing services for many-tasks scientific computing, IEEE Transactions on Parallel and Distributed Systems 22 (6) (2011) 931–945. arXiv:1003.4074, doi:10.1109/TPDS.2011.66.

[9] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, R. Biswas, Performance evaluation of Amazon Elastic Compute Cloud for NASA high-performance computing applications, Concurrency Computation 28 (4) (2016) 1041–1055. arXiv:arXiv:1302.5679v1, doi:10.1002/cpe.3029.

[10] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, G. B. Berriman, Experiences Using Cloud Computing for a Scientific Workflow Application, in: Proceedings of ScienceCloud '11, San Jose, California, 2011, pp. 15–24.

[11] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, X. Yuan, A comparative study of high-performance computing on the cloud, Proceedings of the 22nd international symposium on High-performance parallel and distributed computing - HPDC '13 (2013) 239doi:10.1145/2493123.2462919.
URL http://dl.acm.org/citation.cfm?doid=2493123.2462919

[12] G. B. Berriman, G. Juve, E. Deelman, M. Regelson, P. Plavchan, The application of cloud computing to astronomy: A study of cost and performance, Proceedings - 6th IEEE International Conference on e-Science Workshops, e-ScienceW 2010 (2010) 1–7arXiv:1010.4813, doi:10.1109/eScienceW.2010.10.

[13] I. Bermudez, S. Traverso, M. Mellia, M. Munafo, Exploring the cloud from passive measurements: The Amazon AWS case, Proceedings - IEEE INFOCOM (2013) 230–234doi:10.1109/INFCOM.2013.6566769.

[14] P. Gohil, D. Garg, B. Panchal, A performance analysis of MapReduce applications on big data in cloud based Hadoop, 2014 International Conference on Information Communication and Embedded Systems, ICICES 2014 (978) (2015) 2–7. doi:10.1109/ICICES.2014.7033791.

[15] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, Proceedings of the VLDB Endowment 3 (1) (2010) 460–471. doi:10.14778/1920841.1920902.
URL http://portal.acm.org/citation.cfm?id=1920841.1920902{%}5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=5810EC7C84DF87718CDFE563503DCB92?doi=10.1.1.174.5672{&}rep=rep1{&}type=pdf{%}5Cnhttp://portal.acm.org/citation.cfm?id=1920902

[16] Z. Wu, Q. Wang, A. Plaza, J. Li, L. Sun, Z. Wei, Real-Time Implementation of the Sparse Multinomial Logistic Regression for Hyperspectral Image Classification on GPUs, IEEE Geoscience and Remote Sensing Letters 12 (7) (2015) 1456–1460.

[17] Z. Wu, Q. Wang, A. Plaza, J. Li, J. Wei, Z. Wei, GPU Implementation of Hyperspectral Image Classification based on Weighted Markov Random Fields, in: Proceedings of the 2016 8th Workshop

on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), Los Angeles, CA, USA, 2016, pp. 1–4. doi:10.1109/WHISPERS.2016.8071791.
URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp={&}arnumber=8071791{&}isnumber=8071655

[18] S. Sánchez, R. Ramalho, L. Sousa, A. Plaza, Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs, Journal of Real-Time Image Processing 10 (3) (2015) 469–483. doi:10.1007/s11554-012-0269-2.

[19] M. Ciznicki, K. Kurowski, A. Plaza, Graphics processing unit implementation of JPEG2000 for hyperspectral image compression, Journal of Applied Remote Sensing 6 (1) (2012) 061507. doi:10.1117/1.JRS.6.061507.
URL http://remotesensing.spiedigitallibrary.org/article.aspx?doi=10.1117/1.JRS.6.061507

[20] J. A. Goodman, D. Kaeli, D. Schaa, Accelerating an imaging spectroscopy algorithm for submerged marine environments using graphics processing units, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 4 (3) (2011) 669–676. doi:10.1109/JSTARS.2011.2108269.

[21] Z. P. Lee, K. L. Carder, T. G. Peacock, C. O. Davis, J. L. Mueller, Method to derive ocean absorption coefficients from remote-sensing reflectance, Applied Optics 35 (3) (1996) 453. doi:10.1364/AO.35.000453.
URL https://www.osapublishing.org/abstract.cfm?URI=ao-35-3-453

[22] Z. Lee, K. L. Carder, C. D. Mobley, R. G. Steward, J. S. Patch, Hyperspectral remote sensing for shallow waters. I. A semianalytical model., Applied optics 37 (27) (1998) 6329–38. doi:10.1364/AO.37.006329.
URL http://www.ncbi.nlm.nih.gov/pubmed/18286131

[23] Z. Lee, K. L. Carder, C. D. Mobley, R. G. Steward, J. S. Patch, Hyperspectral remote sensing for shallow waters: 2 Deriving bottom depths and water properties by optimization, Applied Optics 38 (18) (1999) 3831. doi:10.1364/AO.38.003831.

URL https://www.osapublishing.org/abstract.cfm?URI=ao-38-18-3831

[24] K. R. Kurte, S. S. Durbha, High resolution disaster data clustering using Graphics Processing Units, 2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS (2013) 1696–1699doi:10.1109/IGARSS.2013.6723121.
URL http://ieeexplore.ieee.org/document/6723121/

[25] U. M. Bhangale, S. S. Durbha, High performance SIFT feature classification of VHR satellite imagery for disaster management, ACM International Conference Proceeding Series (2015) 324–329doi:10.1145/2791405.2791460.
URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84960962995{&}partnerID=40{&}md5=3cc624143217dd61823799f976d0beee{%}5Cnhttps://www.scopus.com/inward/record.uri?eid=2-s2.0-84960962995{%}7B{&}{%}7DpartnerID=40{%}7B{&}{%}7Dmd5=3cc624143217dd61823799f976d0beee

[26] G. J. Scott, G. A. Angelov, M. L. J. Reinig, E. C. Gaudiello, M. R. England, cvTile: Multilevel parallel geospatial data processing with OpenCV and CUDA, in: Proceedings of the 2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Milan, Italy, 2015, pp. 139–142.

[27] S. Christgau, J. Spazier, B. Schnor, M. Hammitzsch, A. Babeyko, J. Wächter, A comparison of CUDA and OpenACC: Accelerating the Tsunami Simulation EasyWave, Workshop Proceedings of ARCS 2014.

[28] M. Waechter, K. Jaeger, S. Weissgraeber, S. Widmer, M. Goesele, K. Hamacher, Information-Theoretic Analysis of Molecular (Co)Evolution Using Graphics Processing Units, in: ECMLS '12 Proceedings of the 3rd international workshop on Emerging computational methods for the life sciences, 2012, pp. 49–58.

[29] M.-Y. Huang, S.-C. Wei, B. Huang, Y.-L. Chang, Accelerating the Kalman filter on a GPU, Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS (2011) 1016–1020doi:10.1109/ICPADS.2011.153.

[30] D. Xu, Z. Xiao, D. Li, F. Wu, Optimization of Parallel Algorithm for Kalman Filter on CPU-GPU Heterogeous System, 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) (2016) 2165–2172doi:10.1109/FSKD.2016.7603516.

[31] R. E. Kalman, A New Approach to Linear Filtering and Prediction Problems, Journal of Basic Engineering 82 (1) (1960) 35. doi:10.1115/1.3662552.
URL http://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402

[32] B. Jiang, G. A. Woodell, D. J. Jobson, Novel multi-scale retinex with color restoration on graphics processing unit, Journal of Real-Time Image Processing 10 (2) (2015) 239–253. doi:10.1007/s11554-014-0399-9.
URL http://dx.doi.org/10.1007/s11554-014-0399-9

[33] P. Viola, M. Jones, Robust Real-time Object Detection, in: Proceedings of Second International Workshop on Statistical and Computational Theories of Vision - Modeling, Learning, Computing, and Scaling, Vancouver, 2001.

[34] R. E. Schapire, A Brief Introduction to Boosting, in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999. arXiv:arXiv:1508.01136v1, doi:citeulike-article-id:765005.

[35] B. Sharma, R. Thota, N. Vydyanathan, A. Kale, Towards a robust, real-time face processing system using CUDA-enabled GPUs, 2009 International Conference on High Performance Computing (HiPC) (2009) 368–377doi:10.1109/HIPC.2009.5433189.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5433189

[36] M. Chouchene, F. E. Sayadi, H. Bahri, J. Dubois, J. Miteran, M. Atri, Optimized parallel implementation of face detection based on GPU component, Microprocessors and Microsystems 39 (6) (2015) 393–404. doi:10.1016/j.micpro.2015.04.009.
URL http://dx.doi.org/10.1016/j.micpro.2015.04.009

[37] L.-c. Sun, S.-b. Zhang, X.-t. Cheng, M. Zhang, Acceleration Algorithm for CUDA-based Face Detection, in: 2013 IEEE International Conference on Signal Processing, Communications and Computing, ICSPCC 2013, KunMing, 2013, pp. 1–5. doi:10.1109/ICSPCC.2013.6664139.

[38] A. Herout, R. Jošth, R. Juránek, J. Havel, M. Hradiš, P. Zemčík, Real-time object detection on CUDA, Journal of Real-Time Image Processing 6 (3) (2011) 159–170. doi:10.1007/s11554-010-0179-0.
URL http://link.springer.com/10.1007/s11554-010-0179-0

[39] X. Zhi, J. Yan, Y. Hang, S. Wang, Realization of CUDA-based real-time registration and target localization for high-resolution video images, Journal of Real-Time Image Processing (2016) 1–12doi:10.1007/s11554-016-0594-y.

[40] C. Iakovidou, L. Bampis, S. A. Chatzichristofis, Y. S. Boutalis, A. Amanatiadis, Color and Edge Directivity Descriptor on GPGPU, Proceedings - 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015 (2015) 301–308doi:10.1109/PDP.2015.105.

[41] E. Fauske, L. M. Eliassen, R. H. Bakken, A Comparison of Learning Based Background Subtraction Techniques Implemented in CUDA, Proceedings of the First Norwegian Artificial Intelligence Symposium (2009) 181–192.

[42] D. Weimer, S. Köhler, C. Hellert, K. Doll, U. Brunsmann, R. Krzikalla, GPU architecture for stationary multisensor pedestrian detection at smart intersections, IEEE Intelligent Vehicles Symposium, Proceedings (Iv) (2011) 89–94. doi:10.1109/IVS.2011.5940411.

[43] P. Muyan-Özçelik, J. D. Owens, J. Xia, S. S. Samant, Fast deformable registration on the GPU: A CUDA implementation of demons, Proceedings - The International Conference on Computational Sciences and its Applications, ICCSA 2008 (2008) 223–233doi:10.1109/ICCSA.2008.22.

[44] A. Li, A. Kumar, Y. Ha, H. Corporaal, Correlation ratio based volume image registration on GPUs, Microprocessors and Microsystems 39 (8) (2015) 998–1011. doi:10.1016/j.micpro.2015.04.002.
URL http://dx.doi.org/10.1016/j.micpro.2015.04.002

[45] M. Li, Z. Xiang, L. Xiao, E. Castillo, R. Castillo, T. Guerrero, GPU-accelerated Block Matching Algorithm for Deformable Registration of Lung CT Images, 2015 IEEE International Conference on Progress in Informatics and Computing (PIC) (2015) 292–295arXiv:15334406, doi:10.1109/PIC.2015.7489856.
URL http://ieeexplore.ieee.org/document/7489856/

[46] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, S. Ourselin, Fast free-form deformation using graphics processing units, Computer Methods and Programs in Biomedicine 98 (3) (2010) 278–284. doi:10.1016/j.cmpb.2009.09.002.

[47] P. Bhosale, M. Staring, Z. Al-Ars, F. F. Berendsen, GPU-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications (March). doi:10.1117/12.2293098.
URL http://elastix.isi.uu.nl/marius/downloads/2018{_}c{_}SPIEMI.pdf

[48] K. Ikeda, F. Ino, K. Hagihara, Efficient Acceleration of Mutual Information Computation for Nonrigid Registration Using CUDA, IEEE Journal of Biomedical and Health Informatics 18 (3) (2014) 956–968. doi:10.1109/JBHI.2014.2310745.

[49] B. Keck, H. Hofmann, H. Scherl, M. Kowarschik, J. Hornegger, GPU-accelerated SART reconstruction using the CUDA programming environment, Proc. SPIE Medical Imaging (2009) 72582Bdoi:10.1117/12.811559.
URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=813806

[50] X. Yu, H. Wang, W.-c. Feng, H. Gong, G. Cao, GPU-Based Iterative Medical CT Image Reconstructions, Journal of Signal Processing Systems (2018) 1–18.

[51] Y. Sun, X. Sun, H. Zhang, Modern GPU-based forward-projection algorithm with a new sampling method, 2010 International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2010 2 (1) (2010) 691–694. doi:10.1109/ICMTMA.2010.818.

[52] W.-M. Pang, J. Qin, Y. Lu, Y. Xie, C.-K. Chui, P.-A. Heng, Accelerating simultaneous algebraic reconstruction technique with motion compensation using CUDA-enabled GPU, International Journal of Computer Assisted Radiology and Surgery 6 (2) (2011) 187–199. doi:10.1007/s11548-010-0499-3.

[53] X. Zhao, J.-J. Hu, T. Yang, GPU based iterative cone-beam CT reconstruction using empty space skipping technique, Journal of X-ray science and technology 21 (1) (2013) 53–69. doi:10.3233/XST-130366.
URL http://www.ncbi.nlm.nih.gov/pubmed/23507852

[54] D. Riabkov, X. Xue, D. Tubbs, A. Cheryauka, Accelerated cone-beam backprojection using GPU-CPU hardware, Proceedings of the 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine (2007) 4–7.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.2564{&}rep=rep1{&}type=pdf

[55] T. Zinßer, B. Keck, Systematic Performance Optimization of Cone-Beam Back-Projection on the Kepler Architecture, Fully Three-Dimensional - Image Reconstruction in Radiology and Nuclear Medicine (2013) 225–228.
URL http://www5.informatik.uni-erlangen.de/Forschung/Publikationen/2013/Zinsser13-SPO.pdf

[56] J. C. Park, S. H. Park, J. S. Kim, Y. Han, M. K. Cho, H. K. Kim, Z. Liu, S. B. Jiang, B. Song, W. Y. Song, Ultra-Fast Digital Tomosynthesis Reconstruction Using General-Purpose GPU Programming for Image-Guided Radiation Therapy, Technology in cancer research & treatment 10 (4) (2011) 295–306. doi:c4315/Ultra-Fast-Digital-Tomosynthesis-Reconstruction-Using-General-Purpose-GPU-Programming-for-Image-Guided-Radiation-Therapy-295-306-p17868.html[pii].
URL http://www.ncbi.nlm.nih.gov/pubmed/21728386

[57] J. A. Wilson, J. C. Williams, Massively Parallel Signal Processing Using the Graphics Processing Unit for Real-Time Brain-Computer Interface Feature Extraction, Frontiers in Neuroengineering 2 (July). doi:10.3389/neuro.16.011.2009.

URL http://journal.frontiersin.org/article/10.3389/neuro.16.011.2009/abstract

[58] W.-k. Tam, Z. Yang, Neural Parallel Engine: A toolbox for massively parallel neural signal processing, Journal of Neuroscience Methods 301 (2018) 18–33. doi:10.1016/j.jneumeth.2018.03.004.
URL https://doi.org/10.1016/j.jneumeth.2018.03.004

[59] Z. Juhasz, Highly parallel online bioelectrical signal processing on GPU architecture, 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017 - Proceedings (2017) 340–346doi:10.23919/MIPRO.2017.7973446.

[60] M. Sutcliffe, M. Weston, P. Charlton, K. Donne, B. Wright, I. Cooper, Full matrix capture with time-efficient auto-focusing of unknown geometry through dual-layered media, Insight: Non-Destructive Testing and Condition Monitoring 55 (6) (2013) 297–301. doi:10.1784/insi.2012.55.6.297.

[61] D. Romero-Laorden, J. Villazón-Terrazas, O. Martínez-Graullera, A. Ibáñez, M. Parrilla, M. S. Peñas, Analysis of Parallel Computing Strategies to Accelerate Ultrasound Imaging Processes, IEEE Transactions on Parallel and Distributed Systems 27 (12) (2016) 3429–3440. doi:10.1109/TPDS.2016.2544312.

[62] V. Strbac, J. Vander Sloten, N. Famaey, Analyzing the potential of GPGPUs for real-time explicit finite element analysis of soft tissue deformation using CUDA, Finite Elements in Analysis and Design 105 (2015) 79–89. doi:10.1016/j.finel.2015.07.005.
URL http://dx.doi.org/10.1016/j.finel.2015.07.005

[63] V. Strbac, D. M. Pierce, J. Vander Sloten, N. Famaey, GPGPU-based explicit finite element computations for applications in biomechanics: the performance of material models, element technologies, and hardware generations, Computer Methods in Biomechanics and Biomedical Engineering 20 (16) (2017) 1643–1657. doi:10.1080/10255842.2017.1404586.
URL https://doi.org/10.1080/10255842.2017.1404586

[64] U. Göbel, C. Sander, R. Schneider, A. Valencia, Correlated Mutations and Residue Contacts in Proteins, Proteins: Structure, Function, and Genetics 18 (4) (1994) 309–317.

[65] O. Olmea, A. Valencia, Improving contact predictions by the combination of correlated mutations and other sources of sequence information, Folding & Design 2 (3) (1997) S25–S32.
URL http://www.ncbi.nlm.nih.gov/pubmed/9218963

[66] L. Knight, Co-evolving protein sites: their identification using novel, highly-parallel algorithms, and their use in classifying hazardous genetic mutations, Ph.D. thesis, Cardiff University (2017).
URL http://orca.cf.ac.uk/108951/

[67] P. Krzyzanowski, Process Scheduling (2015).
URL https://www.cs.rutgers.edu/{~}pxk/416/notes/07-scheduling.html