General Purpose Computing on the GPU

Characteristics of suitable problems

SIMON LJUNGSTRÖM
and VIKTOR LJUNGSTRÖM

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2012

General Purpose Computing on the GPU

Characteristics of suitable problems

S I M O N   L J U N G S T R Ö M
and   V I K T O R   L J U N G S T R Ö M

Abstract

In a society that grows more and more dependent on fast digital data processing, many developers have turned their attention toward performing general-purpose computations on the graphics processing unit. This thesis explores what types of problems might be, or might not be, suitable for implementation on the GPU by taking a look at both classical and modern GPU concepts. Two computational problems – matrix multiplication and maximum value of a matrix – are implemented for both multi-core CPU and GPU and a comparison is presented. We reach the conclusion that the GPU can be an extremely potent computation unit as long as the problem is highly parallelizable, has no or very few branches and is computationally intensive.

Referat

Generella beräkningar på GPU:n

I ett samhälle som blir allt mer beroende av snabb digital databehandling har utvecklare och forskare börjat rikta sitt intresse åt att utföra generella beräkningar på datorns grafikprocessor, GPU:n. I detta examensarbete undersöks vilken typ av beräkningar som är lämpade, eller inte lämpade, att behandlas av GPU:n genom att ta en titt på både klassiska och moderna GPU koncept. Utöver detta tar vi också en djupare titt på hur två problem, matrismultiplikation och att hitta maxima i en matris, presterar på flerkärnig CPU och GPU och jämför resultaten. Vi har kommit till slutsatsen att GPU:n kan vara en mycket kraftfull beräkningsenhet, så länge problemet i fråga är högeligen paralleliserbart, saknar eller har väldigt få villkorliga förgreningar samt är beräkningsintensivt.

Statement of Collaboration

This text and associated code is a collaboration between the two authors Simon Ljungström and Viktor Ljungström. Work was divided as follows. Any sections of text not explicitly mentioned below are considered to be written with equal, or close to equal, effort from both authors.

Author	Sections / Code
Simon Ljungström	1, 2.3, 2.5, 5.2, Matrix Maximum for CPU, Matrix Multiplication for GPU
Viktor Ljungström	2.1, 2.2, 4.1, 4.2, 5.1, Matrix Maximum for GPU, Matrix Multiplication for CPU

Definitions

Abbreviation	Term	Definition
AMD	Advanced Micro Devices	One of the worlds leading CPU and GPU manufacturers
API	Application Programming Interface	An interface used for application programming. Often consists of header-files in C, abstracting complex assmbler routines
CPU	Central Processing Unit	The processing unit that is normally used for computations
CUDA	C for CUDA	An API for C, used to program CUDA devices
	Compute Unified Device Architecture	An architecture implemented in recent NVIDIA devices
DLP	Data Level Parallelism	When a problem can be parallelized by running a function on different data in parallel
FLOPS	Floating Point Operations per Second	A common way to measure throughput
GPGPU	General Purpose Computing on the GPU	Performing non-graphics computations on the GPU
GPU	Graphics Processing Unit	A processing unit that is specialized on graphics-computations
Latency	Latency	The time you have to wait for something to finish
NVIDIA	NVIDIA	One of the worlds leading GPU manufacturers

Abbreviation	Term	Definition
OpenCL	Open Computing Language	An open programming language/API with a focus on portability between different systems and/or devices
OpenGL	Open Graphics Library	An API for performing graphics computations on the GPU
SIMD	Single Instruction Multiple Data	A processing model where the same instruction is applied to different data. Has historically been used mostly for image processing and graphics computations
SPMD	Single Program Multiple Data	Same as SIMD but with support for conditional branching
TLP	Task Level Parallelism	When a problem can be parallelized by dividing it into several sub-problems that can be performed independently
Throughput	Throughput	The total number of computations performed during a time interval. Usually measured in FLOPS

Contents

Chapter 1

Introduction

As society grows ever more dependent on digital systems for everything from banking to elections, expectations on these systems are constantly rising. The systems are expected to be able to process an increasing amount of data without any increase in latency. This means that there is a constantly increasing demand for faster processing for these systems, and as new more data intensive systems emerge this demand grows at an even higher pace.

In the early days of digital computing this demand was met by CPU manufacturers optimizing for low latency by dividing the pipeline into a larger amount of smaller steps as well as increasing the CPU clock speed and number of cache levels. But this development could not continue indefinitely due to the difficulty of dividing the pipeline further, and the correlation between high clock speeds and high temperatures.

Today, we instead see an increase in parallelism with multi-core processors becoming the norm. More and more software is written to utilize these multiple cores, often leading to a great boost in performance. As it is, this development can not continue unhindered either. Due to the large amount of logic and hardware used to reduce latency, processor cores become rather large and it is not physically possible to squeeze more cores into the same area without removing some of the latency reducing functionality or reducing transistor size.

There is, fortunately, already a type of processor that is inherently parallel and massively multi-core to begin with – the graphics processing unit. The GPU is, however, a specialized piece of hardware focused on graphics calculations and making specialized hardware do general computations is not trivial. Thus, utilizing this innate parallelism has been a subject of research for some time now and has also more recently awakened an interest throughout the general development community, mainly due to the emergence of higher level APIs targeting this audience such as CUDA and OpenCL [1].

Many developers and consumers are interested in the topic of General-Purpose computing on the GPU (GPGPU or GPU computing), but do not fully understand what types of software may be suitable for GPU acceleration. One of the reasons for

this is that in order to write efficient programs for the GPU, one needs to possess some basic knowledge of its architecture. This text aims to shed some light on this underlying architecture as well as identify some traits that make a problem or algorithm more or less suitable for GPU computing.

More explicitly, we are looking to answer the following two questions.

- **What are the main characteristics of problems that perform well on the GPU?**

- **Is the GPU a viable alternative to the CPU?**

To facilitate this, we shall also take a closer look at and implement two computational problems: matrix multiplication and finding the maximum element of a matrix. The first as an example of a problem that should see a significant performance boost on a GPU, and the second as a problem that should not. These implementations are tested and their performance evaluated in order to confirm whether or not they perform as anticipated.

We start off by explaining why the GPU is so parallel by taking a look at the graphics pipeline and its evolution including a brief overview of the Unified Shader Model, GPU latency hiding and the GPU memory model as well as a short introduction to OpenCL. This is followed by an explanation of the actual implementation details for our chosen problems. Finally, we present our results, discuss these and present our conclusions.

Chapter 2

Background

Before the graphics processing unit was invented, developers of graphical user interfaces and games struggled to make their creations run smoothly and without delay on the CPU. There was clearly a need to offload the CPU by performing these demanding computations elsewhere. This was the birth of the GPU. In contrast to the very general CPU the GPU only had to do one thing, compute what colors the pixels on the screen should have. This meant that the GPU could be very specialized for this purpose. The key ingredient in this specialization is the fact that the color of each pixel can be computed almost entirely independently from the other pixels. This resulted in a GPU design that was much more parallel than the CPU, but one that ran at a slower clock speed. The focus of this design was to maximize throughput for multiple tasks, rather than minimizing latency for a single task [2, 3].

In the following sections we present some basic background information that helps with the understanding of why the graphics processing unit is so parallel and what problems it may be good for. We take a look at the graphics pipeline and its evolution including the Unified Shader concept and graphics memory along with a short overview of some OpenCL concepts that will be used both in the problem implementations and when discussing the results.

2.1 The Classic Graphics Pipeline

The classic graphics pipeline is built upon one simple fact: almost all operations used for computing graphics have both task level parallelism (TLP) and data level parallelism (DLP) [4]. This means that (1) there are several independent stages in the computations of the image output, and (2) the data in one stage can be processed in parallel. Graphics manufacturers exploit this parallelism in many ways, which we shall now take a closer look at.

The pipeline is split into several stages, all of which can be computed in parallel. The first step is *vertex operations*, where a programmer-provided input stream of geometric primitives (points, lines and triangles) represented by vertices is normalized

into screen space and shaded, typically by calculating their interaction with the light sources in the virtual environment. A typical scene can have hundreds of thousands of vertices, all of which can be computed independently in parallel [1, 2, 3, 5].

The next stage is *primitive assembly*, where the vertices are assembled into triangles, the fundamental hardware-supported building block in the GPU. The following step, rasterization, determines which screen-space triangles are covered by which triangles. Every triangle generates a fragment at every pixel location that it covers. Many triangles may cover the same pixel location and it may therefore be affected by multiple fragments in the next step, *fragment operations*.

By using color information from the vertices and possibly fetching textures from global memory, the fragments are shaded to determine their color. This can be done in parallel and is generally considered to be the most computationally demanding stage of the GPU pipeline. When all the fragment colors have been computed, we move on to the last step, *composition*.

In this final step of the pipeline, the fragments are merged together in order to calculate the final color of the pixels in the output image that will be sent to the screen.

Many of the steps in the pipeline above are performed on shader processors. These are special processors with a very limited instruction set, specific to the task they perform. Two examples of shader processors are fragment shaders and vertex shaders.

Historically, the vertex and fragment shaders have not been programmable, only configurable. The programmer was only in control of the positions and color of the lights, not their interaction with the objects in the scene [1]. In the following sections we will have a closer look at the actual hardware that has evolved from the challenges presented by the very parallel nature of graphics calculations.

2.2 Shader Processors

The graphics processor has evolved over time from a completely fixed-function pipeline to a relatively programmable, fairly flexible ditto. Programmers needed more flexibility in order to implement more advanced graphical effects. This lead to a dire need for more programmable shader units, which was also a first step toward making GPGPU possible at all.

The shader units, or shader processors, are implemented using the Single Instruction Multiple Data (SIMD) processing model. This means that the shader units perform the same instructions on multiple data at the same time by sharing control structures between multiple shader processors. The number of shader processors sharing these control structures is called the SIMD-width and is usually a power of two. A larger SIMD-width means that a larger part of the chip-area can be used for actual computations rather than instruction decoding [1, 3, 6].

There is, however, a large disadvantage with the SIMD-model when doing general computations. Since the data differs between the cores, different paths may be

followed when a conditional branch is reached. This kind of behaviour is not defined in the standard SIMD-model. To handle this behaviour, a new, similar model is required. This is called the Single Program Multiple Data (SPMD) model. It is the same as SIMD, but with the addition of branch-support. This is however a modified truth. When a branch is detected in a SPMD-unit, all threads that diverge are put on hold to be computed later. This means that if half of the threads go one way and the other half another way, the execution will take twice as long as if all threads had taken the same path. You can imagine how much impact this would have on a program with multiple conditional branches. Because of this, it is not recommended to use conditional branches unless it is absolutely necessary [1, 3, 6].

In the next section we shall have a look at how the different kinds of shaders have been merged together into a more general and programmable shader unit.

2.3 Unified Shaders

One problem with a pipeline using fixed vertex and fragment processors is the hugely varying level of demand on these resources. Certain calculations may require vertex processing almost exclusively, while only utilizing a small amount of the fragment processors and vice versa [1, 2, 3, 5]. A solution was needed to be able to make use of all the provided hardware, all the time. This lead to the conception of the Unified Shader Model, a term which is often used to describe two separate, but nonetheless intertwined, concepts [1].

The actual Unified Shader Model is the concept of using a unified instruction set to communicate with all different shader processor types. This greatly simplifies the task of writing shader programs. It is also a necessary step towards the related Unified Shader Architecture concept [1].

A "Unified Shader Architecture" is a GPU architecture where the shader processors are physically unified, that is, every shader unit is a more general computation device, able to do all types of shader work. If a computation only needs vertex shading, all the processors can do vertex computations, leading to much better load balancing. This is also a move away from the task-parallel hardware-fixed pipeline from earlier, allowing for a single step in the pipeline to operate almost exclusively instead of all steps executing in parallel all the time.

To facilitate an easier SIMD implementation, the unified shaders are usually grouped together into what NVIDIA calls "streaming multiprocessors". These contain several shader processors, sharing resources such as instruction fetching and caches. One or more of these multiprocessors can then be grouped together to form a larger SIMD array where every processor executes the same instruction at the same time. The size of these arrays is equal to the SIMD-width of the GPU [1].

2.4   Graphics Memory

As any experienced programmer knows, a lot of the run time of a program is spent fetching data from memory. While data is being fetched, the CPU is blocked. This is of course not very productive and slows execution by an unacceptable amount. CPU-manufacturers have solved this delay by implementing several layers of cache-memory. The number and size of these caches is constantly increasing as new CPU models are introduced. The problem with this approach is that you will experience the full latency time the first time a memory block is accessed. The GPU has very little cache memory, often around 16k per stream multiprocessor, and thus handles the problem very differently.

When a thread starts fetching data from memory, the processing unit that handles that thread immediately switches to another thread. As long as the switch is done quickly, this behaviour allows for the processing unit to hide almost all latency. For this reason, modern GPUs support a huge amount of hardware threads. When the data fetch operation is finished for the waiting thread, it is queued for processing again. As long as there are more threads to switch to, most data fetching latency can be hidden [1, 2, 3, 4].

Lately, we have seen a large increase in the amount of on-board memory in high-end graphics cards. However, access time to this memory is often undesirably slow, even though GPUs generally do provide higher bandwidth to memory than CPUs do. As mentioned earlier, GPUs do have a bit of cache memory, even though they employ latency hiding. The cache memory is however different from the cache memory used by the CPU.

2.5   A Brief OpenCL Overview

OpenCL is an API for homogenous systems (systems with more than one type of computation device). In this thesis we will be using this API to implement the GPU versions of the chosen problems. More details on why this choice of API was made can be found in chapter 3.

At first glance, OpenCL programming may seem very daunting. As can be seen in Appendix A (specifically, the runOnGPU() methods), there is a fair amount of setup before you can actually use the GPU to perform calculations. But once you have overcome this hurdle, doing it again is not difficult since the setup is close to identical each time. At least when tackling the relatively simple problems we deal with in this text. We will not consider the setup further as it is outside the scope of this thesis.

In the following sections we will take a look at a few basic OpenCL concepts; kernels, the OpenCL memory model as well as work groups, work items and work sizes.

2.5.1 Kernels

In OpenCL (as well as in CUDA) the code which is run on the so called "OpenCL device" – in the case of this text, the GPU – is known as a kernel. Kernels are always declared using the keyword ___kernel and can be compiled either at runtime or not. To ensure correct compilation, the slightly performance-reducing runtime compilation should be used if the device the kernel will run on is not known beforehand [7, 8]. Note that code not run on the device, that is, the code that controls the device, is known as host code.

2.5.2 Memory Model

In the OpenCL memory model – which refers to the memory used by the device used to run kernels – there are four types of memory: global, constant, local and private [7, 8].

Global memory is the main memory used by the device; in the case of the GPU this refers to the on-board graphics memory [8].

Constant memory is the same as global memory, except it may be used more efficiently than global memory if the device has special hardware for handling constant memory caching. Most modern GPUs have such hardware.

Local memory is the shared memory on each compute unit [8]. On the GPU this corresponds to the shared memory within each stream multiprocessor, as discussed in sections 2.3 and 2.4.

Private memory is memory accesible only within the current unit of work [8]. For the GPU, this means the registers available to each stream processor.

2.5.3 Work items, work groups and work sizes

A work item is a unit of work that is to be executed on a single core.

The work items are then further grouped into work groups. All the work items in a work group can access memory shared within the work group, corresponding to the local memory discussed in the previous section.

How a work group is processed is not specified in OpenCL and thus depends on the device and its OpenCL drivers. On the GPU, however, the work group is usually represented as a group of threads (work items) executed on the same stream multiprocessor, using threads that cannot be processed straight away to hide latency as described in section 2.4.

The number of work items in a work group is called the local work size or simply work size. The maximum work size depends on the device to be used, and as we will se later on in the text, using a larger than maximum work size may lead to a system crash.

Chapter 3

Methodology

As discussed earlier, we implement and evaluate two problems both on the GPU using a GPGPU API and on the CPU. Below we consider some options for the implementation and performance evaluation of the problems.

There are not many options to choose from when deciding which GPGPU API to use. The two most well known and widely supported alternatives are CUDA and OpenCL. The C for CUDA API from NVIDIA is most likely more mature since it has been around for quite some time, but the fact that it is a closed standard only supported on NVIDIA hardware makes it a less attractive choice than the fully open OpenCL [7, 9]. As for programming languages, there are several alternatives available. There are wrappers for the OpenCL API for more or less all widely used programming languages [10, 11]. In the end C was chosen, mainly for two reasons. First, the available implementations of OpenCL which are used for all the wrappers are written in C or assembler, and second, for the performance gained by writing code that is so close to the hardware.

Due to the above decision, the alternatives for the CPU implementations are quite limited. Choosing a different language than C would make the comparison of the CPU and GPU results much more difficult as most other languages are not as close to the hardware layer. Thus, C was used to write the CPU implementations of the problems as well. Since most modern CPUs are in fact multi-core the CPU implementations have been made as parallel as possible in order to be able to make fair comparisons between CPU and GPU performance. There are two appealing choices of threading libraries to use for this parallelization: pthreads and OpenMP. Using the low level pthreads library gives a higher level of control over the CPU, compared to OpenMPs higher abstraction level. As loss of control could possibly affect the end result in a negative fashion, pthreads was used.

The performance evaluation was performed as follows:

Hardware

A HP Pavilion dm4-2000eo laptop with:

- 6 GB RAM

- Intel Core i5 2410M @ 2.3 GHz (2 cores + hyperthreading)

- AMD Radeon HD 6470M Graphics (1 GB, 160 stream processors)

Note that the amount of stream processors is low compared to modern, high-end GPUs where the processor count can reach 2048 and above.

Method

1. Each problem was tested for both CPU and GPU using several different matrix sizes.

2. Each CPU test was run with 4 worker threads.

3. For the GPU, each matrix size was tested several times with different work sizes.

4. The maximum matrix and work sizes used in the tests were determined by trial and error, using "when the computer crashes due to the graphics card" as a cutoff. This cutoff was lower than expected due to a bug in the AMD OpenCL implementation leading to 75% of the graphics memory being inaccessible.

5. Each test was run ten times, taking the average runtime as the end result.

6. Any runtimes that were equal to or longer than twice the length of the median runtime were discarded and not used in the calculation of the average.

Chapter 4

Implementation

In this chapter we take a closer look at the two problems we have chosen to implement and test, matrix multiplication and finding the maximum value in a matrix. We explain in-depth which parts of the problem solution that make the problems more or less suited for a GPGPU implementation as well as have a look at some sample code.

4.1 Matrix Multiplication

When we chose an algorithm that should perform well on the GPU we were looking for a compution intensive and highly parallelizable algorithm, without conditional branching. Naive matrix multiplication seemed to do the trick. At a time complexity of $O(n^3)$, it is definitely not a quick algorithm. There is no doubt that it is highly parallelizable; every element in the result matrix can be calculated independently of the others. To top it off, there is no branching. The sequential version of this algorithm is a simple triple-loop, which you can see below.

Pseudo code: sequential matrix multiplication

```
1   void matrix_mult(float * a, float * b, float * c, int n){
2     int i, j, k;
3     for(i = 0; i < n; i++){
4       for(j = 0; j < n; j++){
5         for(k = 0; k < n; k++){
6           c[i][j] += a[i][k]*b[k][j];
7         }
8       }
9     }
10  }
```

The parallel version of this algorithm is quite simple, each thread asks for a row, and then computes all elements of that row. When this is finished, either compute a new row or terminate, depending on whether there are any rows left. See pseudo code below.

Pseudo code: parallel matrix multiplication

```
1   void matrix_mul(float * a, float * b, float * c, int n){
2     int i, j, k;
3     while(i=get_row()){
4       for(j = 0; j < n; j++){
5         for(k = 0; k < n; k++){
6           c[i][j] = a[i][k]*b[k][j];
7         }
8       }
9     }
10  }
```

The GPU version of the algorithm is actually the simplest of them all, since the OpenCL library hands out work to the cores, all we have to do is tell it how one element of the final matrix is calculated. See code below. The full code, including host code and comments, is available in appendix A.

OpenCL code: matrix multiplication

```
1   __kernel void matMul(__global float* a,
2       __global float* b,
3       __global float* c,
4       int width) {
5     int row = get_global_id(1);
6     int col = get_global_id(0);
7     float sum = 0;
8
9     for (int k = 0; k < width; k++) {
10      sum += a[row*width+k] * b[k*width+col];
11    }
12
13    c[row*width+col] = sum;
14  }
```

Now we need to clarify a few things. The code above is not optimized, in the sense that we have not made any variables local in the OpenCL version and have not spent time trying to polish the CPU version. The code is compiled with the -O2 flag using the g++ compiler, but that is all. Our intention is to make the different implementations as similar as possible. We believe that the results we have recorded speak a clear message even without optimization.

4.2 Matrix Maximum

The requirements for an algorithm that most likely would not perform well on the GPU are pretty much the opposite of the characteristics of matrix multiplication. We require an algorithm that performs few computations per data unit and has frequent conditional branching. Finding the maximum of a matrix fulfills these requirements. It is also easily parallelizable, making the comparison a bit more fair. The sequential algorithm is extremely simple, just look at all the elements in some order and save the current maximum. See below.

Pseudo code: sequential matrix maximum

```
1   float matrix_max(float * matrix, int n){
2     int i,j;
3     float max;
4     for(i = 0; i < n; i++){
5       for(j = 0; j < n; j++){
6         if(matrix[i][j] > max){
7           max = matrix[i][j];
8         }
9       }
10    }
11    return max;
12  }
```

This algorithm can be parallelized in many ways. We chose to calculate the maximum of each row, and then the maximum of that. In the CPU implementation, a thread keeps track of the largest value that it has encountered, and keeps calculating one row at a time until no rows remain. Each thread then compares their local maximum to a global maximum and changes it if necessary. See below.

Pseudo code: parallel matrix maximum

```
1   global float max = FLOAT.MIN;
2   global float * matrix;
3   thread Worker(){
4     int i,j;
5     float local_max;
6     while(i=get_row()){
7       for(j = 0; j < n; j++){
8         if(matrix[i][j] > local_max){
9           local_max = matrix[i][j]
10        }
11      }
12    }
13    lock(max);
14    if(local_max > max){
15      max = local_max;
16    }
17    unlock(max);
18  }
```

As with the matrix multiplication algorithm, we have tried to keep the GPU- and CPU-algorithms as similar as possible. However, when we are to merge together the maximums of the rows, we have to make a small adjustment in order to not cripple the GPU code. We perform the final maximum of maximums computation on the CPU, in order to avoid having to send data back and forth between the host and the device. See below.

OpenCL code: matrix maximum

```
1   __kernel void matrixMax( __global const float* matrix,
2                            __global float* out,
3                                     int width){
4       int row = get_global_id(0);
5       if(row < width){
```

```
 6                float max = matrix[row*width];
 7                float current;
 8
 9                for(int i = 0; i < width; i++){
10                        current = matrix[row*width + i];
11                        if(current > max){
12                                max = current;
13                        }
14                }
15                out[row] = max;
16        }
17  }
```

Chapter 5

Results and Discussion

In this section we present our findings and discuss them. The results from the GPU are those that were achieved with the optimal work size for the particular problem. The complete output of our tests can be seen in appendix B.

5.1 Matrix Multiplication

In this section we present and discuss the results from our matrix multiplication executions. We start off by comparing the execution times for different work sizes on our largest matrix to determine the optimal work size. As is depicted in fig. 5.1, the execution time decreases as we increase the work size. Since 16x16 – 256 work items – was the largest work size that did not induce a system crash, it is our optimum. All execution times mentioned from now on in this section will be from executions using work size 16x16.

A comparison of the runtime on the CPU and GPU is depicted in table 5.1 and fig. 5.2. Note that the scale of the horizontal axis is not uniform. As can be seen in the table, the GPU is only 2.44 times faster than the CPU on a 128x128 matrix, but as we increase the size of the matrix the achieved speed-up also increases. When we reach a matrix size of 1024x1024 the speed-up factor suddenly increases by 20. It is likely due to all the GPU's processors now being kept busy. At larger sizes the difference in performance slowly levels off to around 30-35. In section 4.1 we hypothesized that matrix multiplication would be very efficient on the GPU. It seems that we were correct. However, if a different matrix multiplication algorithm were to be chosen, there is room for an even larger performance gain from using the GPU. For example, there are algorithms with lower time complexity where the matrix is split up into squares that are calculated independently of each other and later multiplied together. An algorithm like that can utilize local memory in the GPU, something that our naive algorithm does not. This would likely lead to an even larger speed-up on the GPU compared to the same algorithm run on the CPU.

In the next section, we will have a look at how our implementation of finding the maximum of a matrix performed.

15
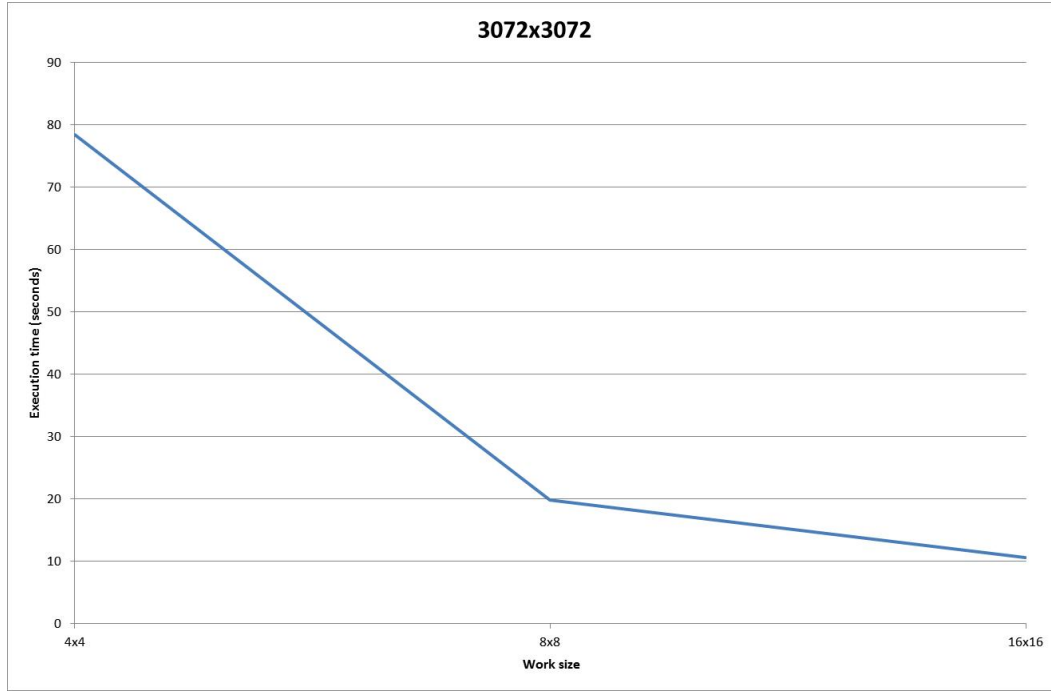
Figure 5.1. Matrix multiplication: Execution time in seconds for different work sizes on 3072x3072 matrices. Lower is better.

Matrix size	Execution time GPU	Execution time CPU	GPU speed-up
128x128	0.0036s	0.0088s	2.44
256x256	0.0085s	0.0637s	7.49
512x512	0.0535s	0.4738s	8.86
1024x1024	0.3998s	11.536s	28.9
1536x1536	1.3299s	39.417s	29.6
2048x2048	3.1459s	102.57s	32.6
3072x3072	10.572s	361.99s	34.2

Table 5.1. Matrix multiplication: Comparison of execution time in seconds on CPU and GPU for different matrix sizes with work size 16x16 on the GPU.
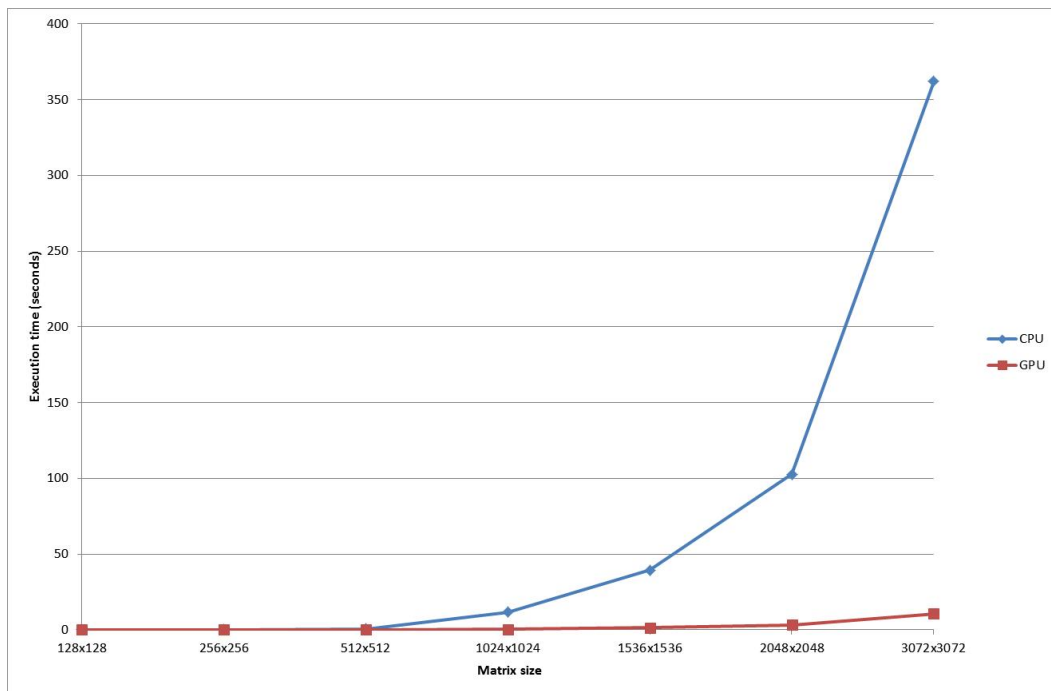
Figure 5.2. Matrix multiplication: Comparison of execution time in seconds on CPU and GPU for different matrix sizes with work size 16x16 on the GPU. Lower is better.

5.2 Matrix Maximum

In this section we take a look at the results of the performance evaluation of the maximum matrix value implementation. We begin, again, by taking a look at the GPU performance for different work sizes as shown in fig. 5.3. Recall from section 5.1 that the maximum work size for our hardware is 256 work items.

The figure depicts results quite different from those in the case of matrix multiplication. Instead of an execution time that always decreases, the faster execution only happens up to a certain point – work size 16 – followed by markedly diminishing performance. The initial performance increase is likely due to two reasons: at low work sizes not all shader processors in a work group can be put to use at the same time, and as a few more work items are added some latency hiding is possible. When the work size increases above 16, however, the negatives of branching under the SPMD model show themselves quite clearly. As noted in section 2.2, when a branch is detected in a SPMD unit, in this case the whole work group, some branches will have to be run later. This means that the work group will take at least twice as long to complete computation. Obviously, the impact gets larger as work size increases. Since the best work size for this problem on the specific GPU used for testing has been determined to be 16, the rest of the GPU results presented here use that work size.
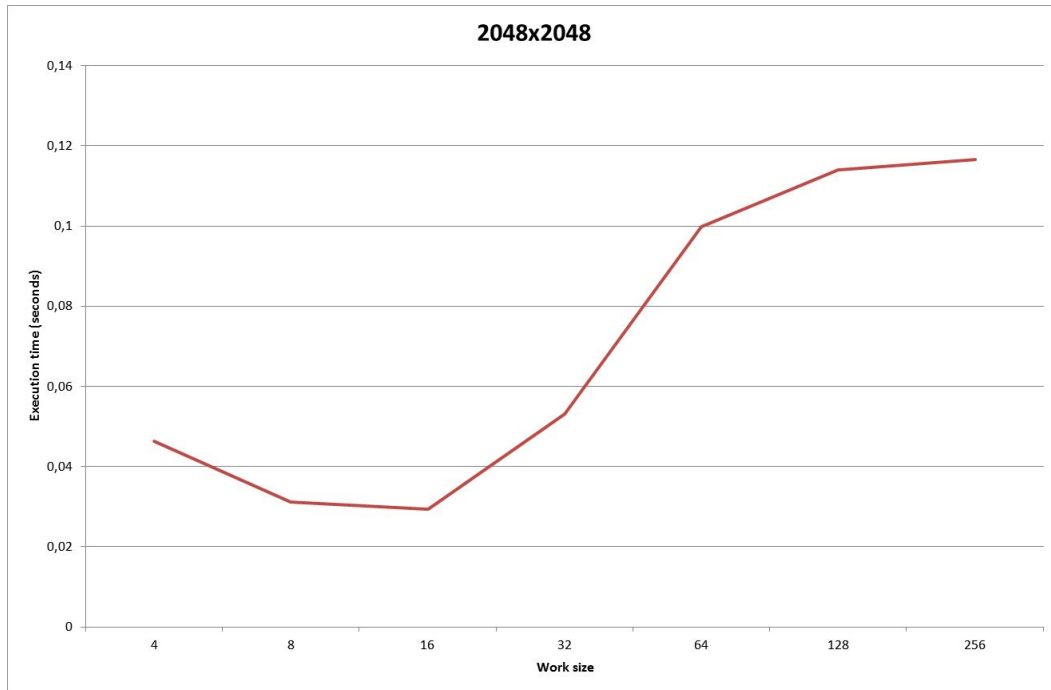
Figure 5.3. Matrix maximum: GPU execution time in seconds for different work sizes on a 2048x2048 matrix. Lower is better.

Let us now take a look at the differences in execution time between the CPU and GPU implementations, as seen in table 5.2 and fig. 5.4. Just as we hypothesized earlier in the text, the CPU wins over the GPU in terms of performance on a branch-intensive problem such as this. Since the resulting execution times are so low, they cannot give a definite answer due to differences in system load at the time of testing. We do, however, get a fair indication. For lower matrix sizes the CPU hovers around 4-5 times speed-up over the GPU after which the performance boost increases slightly more. This trend is expected to continue past the tested matrix sizes as larger matrices mean more potential branches as well as larger amounts of data to transfer to the GPU's global memory. Unfortunately, performance for larger matrices cannot be evaluated as the test-system crashes when trying to transfer the matrix to the graphics memory. A telling sign of the effects of branching on the GPU is the non-linear slowdown of execution as matrix size increases as depicted in fig. 5.4. Note that since the horizontal axis values increase in a quasi-exponential manner, linear slowdown will not be represented as a straight-line in the figure.

Matrix size	Execution time GPU	Execution time CPU	CPU speed-up
128x128	0.00224s	0.00046s	4.87
256x256	0.00269s	0.00051s	5.27
512x512	0.00359s	0.00075s	4.79
1024x1024	0.00854s	0.00187s	4.57
2048x2048	0.02940s	0.00474s	6.20
4096x4096	0.10520s	0.01679s	6.26

Table 5.2. Matrix maximum: Comparison of execution time in seconds on CPU and GPU for different matrix sizes with work size 16 on the GPU
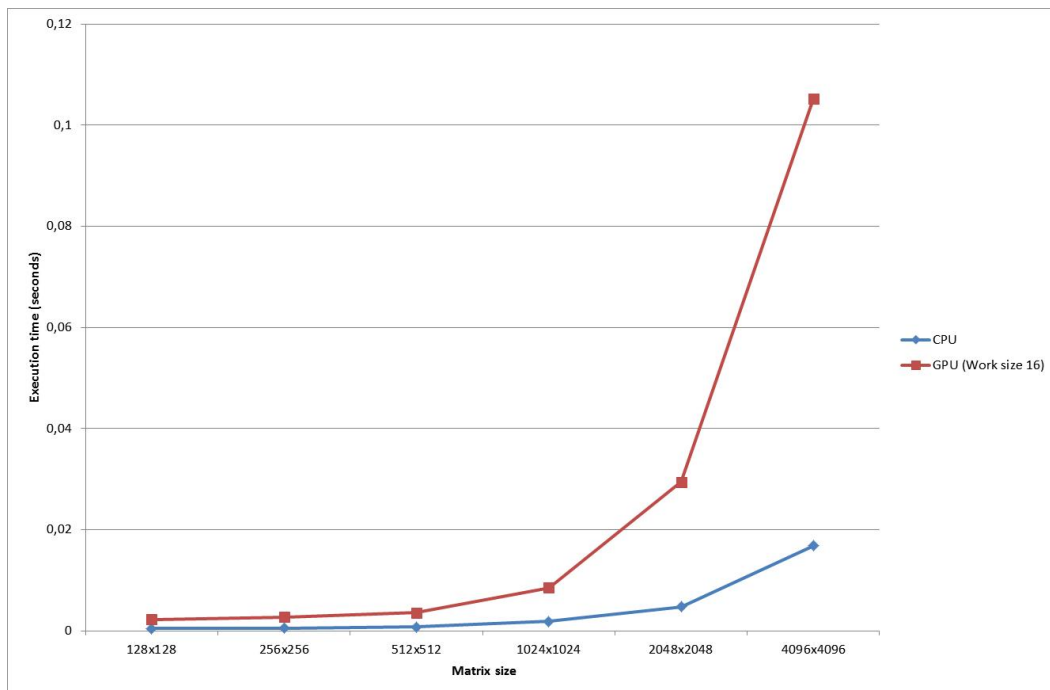
Figure 5.4. Matrix maximum: Comparison of execution time in seconds on CPU and GPU for different matrix sizes with work size 16 on the GPU. Lower is better.

19

Chapter 6

Conclusion

Throughout this text we have identified three important characteristics of problems that are suited for the GPU. The first and most important characteristic is that the problem needs to be highly parallel to begin with. There needs to be a high level of data parallelism in order for a programmer to even implement a sensible solution for the GPU. The second characteristic is that the problem needs to be computationally intensive (high amount of work per data), in order for the GPU not to be idle, waiting for more data from memory. The third and final characterstic is that conditional branching should be non-existent or be triggered very seldom – if every work item takes the same branch, there will be no impact on performance. The architectural details presented in chapter 2 support these claims and the results presented in chapter 5 back this up; both of our hypotheses seem to be correct.

We have chosen not to take pricing into account in previous parts of our text, since it is not relevant from a scientific perspective. However, to determine whether the GPU is a viable alternative to the CPU, we need to; we are already at a point in time where scientist are far from alone in being interested in GPU computing. In practice, pricing is one of the larger factors affecting whether GPGPU will be a success or not. And the matter of the fact is, that GPU performance to currency ratio is very high. High-end consumer graphics cards have, during recent years, never had a price tag greater than 7000 SEK. A high-end consumer CPU, on the other hand, can cost almost twice as much. To top it off, as of OpenCL 1.1 it is quite simple to program for multiple GPUs. This has opened up the possibility for cheap GPU clusters and these are already deployed in some of the worlds fastest super computers.

To summarize, the GPU is not the successor of the CPU. It is however a very potent processing unit that should be considered for all problems that work on large sets of data. Sadly enough, all problems are not suited for GPU computing. However, what it lacks in generality it makes up for in performance.

Bibliography

[1] J. D. Owens et al. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[2] D. Luebke and G. Humpreys. How GPUs Work. *IEEE Computer*, 40(2):96–100, February 2007.

[3] K. Fatahalian and M. Houston. A closer look at GPUs. *Communications of the ACM*, 51(10):50–57, October 2008.

[4] V. W. Lee et al. Debunking the 100x GPU vs CPU myth: an evaluation of throughput computing on CPU and GPU. *ACM SIGARCH Computer Architecture News - ISCA '10*, 38(3):451–460, June 2010.

[5] J. D. Owens et al. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):88–113, March 2007.

[6] H. Wong et al. Demystifying GPU microarchitecture through microbenchmarking. In *2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 235–246, March 2010.

[7] *Khronos OpenCL API Registry*. http://www.khronos.org/registry/cl/.

[8] Fixstars Corporation. *The OpenCL Programming Book*. Fixstars Corporation, March 2010.

[9] *NVIDIA GPU Computing Documentation*. http://developer.nvidia.com/nvidia-gpu-computing-documentation.

[10] *Java Bindings for OpenCL*. http://jocl.org/.

[11] *PyOpenCL - OpenCL for Python*. http://mathema.tician.de/software/pyopencl.

Appendix A

Code

A.1 Matrix Multiplication

matmul.cl

```
1   __kernel void matMul(__global float* a,
2                        __global float* b,
3                        __global float* c,
4                        int width) {
5
6           // get index of current row
7           int row = get_global_id(1);
8           // get index of current column
9           int col = get_global_id(0);
10
11          // keep track of element sum in a register
12          float sum = 0;
13
14          // calculate 1 element of the sub-matrix
15          for (int k = 0; k < width; k++) {
16                  sum += a[row*width+k] * b[k*width+col];
17          }
18
19          // write to output memory
20          c[row*width+col] = sum;
21  }
```

matmul.cpp

```
1   #ifndef __REENTRANT
2   #define __REENTRANT
3   #endif
4
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include <time.h>
8   #include <sys/time.h>
9   #include <CL/cl.h>
10  #include <pthread.h>
11
12  #define MAX_SOURCE_SIZE (0x10000)
13  #define LOCAL_TILE_SIZE (16)
14  #define DEFAULT_WIDTH (LOCAL_TILE_SIZE * 128)
15  #define NUM_REPETITIONS (10)
16
```

```
17   int row = 0; /* the bag of tasks */
18   unsigned int n = 1024; /* Width/Height of the matrix*/
19   unsigned int workGroupWidth = 16;
20
21   /* matrices in host memory */
22   float* h_mem_a;
23   float* h_mem_b;
24   float* h_mem_c;
25
26   pthread_mutex_t block; /* mutex lock for the bag */
27
28   /* timer */
29   double read_timer() {
30       static bool initialized = false;
31       static struct timeval start;
32       struct timeval end;
33
34       if(!initialized){
35           gettimeofday(&start, NULL);
36           initialized = true;
37       }
38       gettimeofday(&end, NULL);
39       return (end.tv_sec - start.tv_sec) + 1.0e-6 * (end.tv_usec - start.tv_usec);
40   }
41
42   /* Each worker calculates the values in one strip of the matrix.*/
43   void *Worker (void*) {
44           unsigned int sum, i, j, k;
45
46           while (true) {
47                   /* get a row number from the bag */
48                   pthread_mutex_lock (&block);
49                   i = row++;
50                   pthread_mutex_unlock (&block);
51
52                   if (i >= n) break;
53
54                   /* multiply the row */
55                   for (j = 0; j < n; j++) {
56                           sum = 0;
57                           for(k = 0; k < n; k++){
58                                   sum += h_mem_a[i*n+k]*h_mem_b[k*n+j];
59                           }
60                           h_mem_c[i*n+j] = sum;
61                   }
62           }
63           pthread_exit (NULL);
64           return NULL;
65   }
66
67   /* Multiplies two matrices and
68    * returns the time the calculation took */
69   double runOnCPU(int numWorkers) {
70           int i;
71           double start_time, end_time;
72           pthread_attr_t attr;
73           pthread_t workerid[numWorkers];
74
75           /* set global thread attributes */
76           pthread_attr_init (&attr);
77           pthread_attr_setscope (&attr, PTHREAD_SCOPE_SYSTEM);
78
79           pthread_mutex_init (&block, NULL);
80
81           /* do the parallel work: create the workers */
82           start_time = read_timer();
83
84           for (i = 0; i < numWorkers; i++)
85                   pthread_create (&workerid[i], &attr, Worker, NULL);
86
87           for (i = 0; i < numWorkers; i++)
```

A.1. MATRIX MULTIPLICATION

```
88                    pthread_join (workerid [i], NULL);
89
90          /* get end time */
91          end_time = read_timer ();
92
93          return end_time − start_time;
94   }
95
96
97   /* Matrix mutltiplication of n*n matrices on an opencl device,
98      assumes even numbered matrix width */
99   double runOnGPU(int buffer_time_included) {
100          /* OpenCL kernel related variables */
101          char *source_str;
102          size_t source_size;
103          cl_program program;
104          cl_kernel kernel;
105
106          /* Device and Platform related variables */
107          cl_platform_id platform_id;
108          cl_uint ret_num_platforms;
109          cl_device_id device_id;
110          cl_uint ret_num_devices;
111
112          /* context */
113          cl_context context;
114
115          /* command queue */
116          cl_command_queue command_queue;
117
118          /* memory buffers */
119          cl_mem d_mem_a, d_mem_b, d_mem_c;
120
121          /* error return value */
122          cl_int ret;
123
124          int size = n * n * sizeof(float);
125          double startTime, endTime;
126
127          /* OpenCL setup */
128
129          /* Load OpenCL kernel */
130
131          FILE *fp;
132          char fileName [] = "./matmul.cl";
133
134          /* Load kernel source */
135          fp = fopen(fileName, "r");
136          if (!fp) {
137                  fprintf(stderr, "Failed to load kernel.\n");
138                  exit (1);
139          }
140
141          source_str = (char*)malloc(MAX_SOURCE_SIZE);
142          source_size = fread(source_str, 1, MAX_SOURCE_SIZE, fp);
143          fclose(fp);
144
145
146          /* Get Platform and Device info */
147          ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
148          ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_GPU, 1, &device_id,
149                  &ret_num_devices);
150
151          /* Context creation */
152          context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);
153
154          /* Command queue creation */
155          command_queue = clCreateCommandQueue(context, device_id, 0, &ret);
156
157          /* Create memory buffers */
158          /* Write−buffers */
```

```
159            d_mem_a = clCreateBuffer(context, CL_MEM_READ_ONLY, size, NULL, &ret);
160            d_mem_b = clCreateBuffer(context, CL_MEM_READ_ONLY, size, NULL, &ret);
161
162            /* Read-buffer */
163            d_mem_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY,  size, NULL, &ret);
164
165            /* create kernel program */
166            program = clCreateProgramWithSource(context, 1,
167                    (const char **)&source_str, (const size_t *)&source_size, &ret);
168
169            /* build kernel program */
170            ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
171
172            /* create kernel */
173            kernel = clCreateKernel(program, "matMul", &ret);
174
175            if(buffer_time_included)
176                    startTime = read_timer();
177
178            /* Write to buffers */
179            ret = clEnqueueWriteBuffer(command_queue, d_mem_a, CL_TRUE, 0, size,
180                    (const void *)h_mem_a, 0, 0, NULL);
181            ret = clEnqueueWriteBuffer(command_queue, d_mem_b, CL_TRUE, 0, size,
182                    (const void *)h_mem_b, 0, 0, NULL);
183
184            if(!buffer_time_included)
185                    startTime = read_timer();
186
187            /* set kernel arguments */
188            clSetKernelArg(kernel, 0, sizeof(cl_mem), (void*)&d_mem_a);
189            clSetKernelArg(kernel, 1, sizeof(cl_mem), (void*)&d_mem_b);
190            clSetKernelArg(kernel, 2, sizeof(cl_mem), (void*)&d_mem_c);
191            clSetKernelArg(kernel, 3, sizeof(int), (void*)&n);
192
193            /* set work dimensions */
194            size_t globalWorkSize[2], localWorkSize[2];
195            globalWorkSize[0] = n;
196            globalWorkSize[1] = n;
197            localWorkSize[0] = workGroupWidth;
198            localWorkSize[1] = workGroupWidth;
199
200            /* Execute kernel */
201            clEnqueueNDRangeKernel(command_queue, kernel, 2, NULL, globalWorkSize,
202                    localWorkSize, 0, NULL, NULL);
203
204            /* read kernel result into C */
205            clEnqueueReadBuffer(command_queue, d_mem_c, CL_TRUE, 0, size, (void *)h_mem_c,
206                    0, 0, NULL);
207
208            endTime = read_timer();
209
210
211            /* free resources */
212            clFlush(command_queue);
213            clFinish(command_queue);
214            clReleaseMemObject(d_mem_a);
215            clReleaseMemObject(d_mem_b);
216            clReleaseMemObject(d_mem_c);
217            clReleaseCommandQueue(command_queue);
218            clReleaseContext(context);
219            clReleaseKernel(kernel);
220            clReleaseProgram(program);
221
222            free(source_str);
223
224            return endTime-startTime;
225    }
226
227    /*call with "matmul n workGroupWidth numWorkers" */
228    int main(int argc, char *argv[]) {
229            unsigned int i;
```

A.1. MATRIX MULTIPLICATION

```
230
231            n = (argc > 1) ? atoi(argv[1]) : n;
232            workGroupWidth = (argc > 2) ? atoi(argv[2]) : workGroupWidth;
233            int numWorkers = (argc > 3) ? atoi(argv[3]) : 2;
234
235            if(n%workGroupWidth != 0){
236                    printf("n needs to be a multiple of workGroupWidth\n");
237                    exit(1);
238            }
239
240            /* Initialize matrices */
241
242            h_mem_a = (float*) malloc(n*n*sizeof(float));
243            h_mem_b = (float*) malloc(n*n*sizeof(float));
244            h_mem_c = (float*) malloc(n*n*sizeof(float));
245
246            srand((unsigned) time(NULL));
247
248            for(i = 0; i < n*n; i++){
249                    h_mem_a[i] = (float) 1 + (float)rand()/(float)RAND_MAX;
250                    h_mem_b[i] = (float) 1 + (float)rand()/(float)RAND_MAX;
251            }
252
253            for(i = 0; i < n*n; i++){
254                    h_mem_c[i] = 0;
255            }
256
257            printf("Runnning matrix_multiplication on an %dx%d matrix\n", n, n);
258            printf("Local Work Size = %dx%d\n",workGroupWidth,workGroupWidth);
259            printf("Number of CPU workers = %d\n",numWorkers);
260            printf("ITER\tGPU(EX.BUF)\t\tGPU(INC.BUF)\t\tCPU\n");
261
262            /* Start performance testing */
263            double gpu_time_yes_buffer_sum = 0;
264            double gpu_time_no_buffer_sum = 0;
265            double cpu_time_sum = 0;
266            for(i = 0; i < NUM_REPETITIONS; i++){
267                    double tmp_no_buf = runOnGPU(0);
268                    double tmp_yes_buf = runOnGPU(1);
269                    double tmp_cpu = runOnCPU(numWorkers);
270
271                    gpu_time_no_buffer_sum += tmp_no_buf;
272                    gpu_time_yes_buffer_sum += tmp_yes_buf;
273                    cpu_time_sum += tmp_cpu;
274
275                    printf("%d\t%gs\t\t%gs\t\t%gs\n",
276                            i, tmp_no_buf, tmp_yes_buf, tmp_cpu);
277                    row = 0;
278            }
279
280            printf("————————————————————————————————\n");
281            printf("WHAT\t\t\t\t\tTIME(TOT)\tTIME(AVG)\n");
282            printf("GPU time (Transfer time excluded)\t %gs\t%gs\n",
283                    gpu_time_no_buffer_sum, gpu_time_no_buffer_sum/NUM_REPETITIONS);
284            printf("GPU time (Transfer time included)\t %gs\t%gs\n",
285                    gpu_time_yes_buffer_sum, gpu_time_yes_buffer_sum/NUM_REPETITIONS);
286            printf("CPU time\t\t\t %gs\t%gs\n",
287                    cpu_time_sum, cpu_time_sum/NUM_REPETITIONS);
288
289            free(h_mem_a);
290            free(h_mem_b);
291            free(h_mem_c);
292
293            return 0;
294    }
```

A.2 Matrix Max

matrixmax.cl

```
1  __kernel void matrixMax( __global const float* matrix,
2                                   __global float* out,
3                                   int width){
4          int row = get_global_id(0);
5          if(row < width){
6                  float max = matrix[row*width];
7                  float current;
8
9                  for(int i = 0; i < width; i++){
10                         current = matrix[row*width + i];
11                         if(current > max){
12                                 max = current;
13                         }
14                 }
15                 out[row] = max;
16         }
17 }
```

matrixmax.cpp

```
1  #ifndef _REENTRANT
2  #define _REENTRANT
3  #endif
4
5  #include <pthread.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <math.h>
9  #include <stdbool.h>
10 #include <time.h>
11 #include <sys/time.h>
12
13 #ifdef __APPLE__
14 #include <OpenCL/opencl.h>
15 #else
16 #include <CL/cl.h>
17 #endif
18
19 #define MAX_SOURCE_SIZE (0x100000)
20
21 size_t matrixSize, rowSize;
22
23 /* default size of local work on gpu */
24 size_t localSize = 16;
25
26 /* default number of workers in cpu-implementation */
27 unsigned int numWorkers = 4;
28
29 /* default size of the n x n matrix */
30 unsigned int n = 2048;
31
32 /* default value for including matrix-to-gpu transfer time */
33 bool gpu_buffer_time_included = true;
34
35 unsigned int NUM_REPETITIONS = 10;
36 float *h_mem_matrix;
37
38 int gmax;
39 int row = 0; /* the bag of tasks */
40
41 pthread_mutex_t lmax; /* mutex lock for result */
42 pthread_mutex_t block; /* mutex lock for the bag */
43
```

A.2. MATRIX MAX

```
44  void *Worker (void *);
45
46  void getFileContent(const char* filename, char** source_str, size_t* source_size){
47          FILE* fp;
48          fp = fopen(filename, "r");
49          if(!fp){
50                  fprintf(stdout, "Failed to load file\n");
51                  exit(1);
52          }
53          *source_str = (char*) malloc(MAX_SOURCE_SIZE);
54          *source_size = fread(*source_str, 1, MAX_SOURCE_SIZE, fp);
55          fclose(fp);
56  }
57
58  /* timer */
59  double read_timer() {
60          static bool initialized = false;
61          static struct timeval start;
62          struct timeval end;
63          if( !initialized )
64          {
65                  gettimeofday( &start, NULL );
66                  initialized = true;
67          }
68          gettimeofday( &end, NULL );
69          return (end.tv_sec - start.tv_sec) + 1.0e-6 * (end.tv_usec - start.tv_usec);
70  }
71
72  /* Calculates the maximum value of a matrix and
73   * returns the time the calculation took.
74   * Uses multiple worker threads and a bag of tasks.*/
75  double runOnCPU(int numWorkers) {
76          int i;
77          double start_time, end_time;
78          pthread_attr_t attr;
79          pthread_t workerid[numWorkers];
80
81          /* set global thread attributes */
82          pthread_attr_init (&attr);
83          pthread_attr_setscope (&attr, PTHREAD_SCOPE_SYSTEM);
84
85          /* initialize mutexes */
86          pthread_mutex_init(&lmax, NULL);
87          pthread_mutex_init (&block, NULL);
88
89          gmax = h_mem_matrix[0];
90
91          /* do the parallel work: create the workers */
92          start_time = read_timer();
93
94          for (i = 0; i < numWorkers; i++)
95            pthread_create (&workerid[i], &attr, Worker, NULL);
96
97          for (i = 0; i < numWorkers; i++)
98            pthread_join (workerid[i], NULL);
99
100         /* get end time */
101         end_time = read_timer();
102
103         return end_time - start_time;
104 }
105
106 /* Each worker determines the max value in one strip of the matrix.
107  * After each updates the global max if the local is lager */
108 void *Worker (void*) {
109         int max, worked = 0;
110         unsigned int i, j;
111         max = h_mem_matrix[0];
112
113         while (true) {
114                 /* get a row number from the bag */
```

```
115                        pthread_mutex_lock (&block);
116                        i = row++;
117                        pthread_mutex_unlock (&block);
118
119                        if (i >= n) break;
120                        if (!worked) worked = 1;
121
122                        /* update local max with elements in the row */
123                        for (j = 0; j < n; j++) {
124                                if (max < h_mem_matrix[i*n+j]) {
125                                        max = h_mem_matrix[i*n+j];
126                                }
127                        }
128                }
129                if (worked) {
130                        /* update global max */
131                        if (gmax < max) {
132                                pthread_mutex_lock(&lmax);
133                                if (gmax < max) {
134                                        gmax = max;
135                                }
136                                pthread_mutex_unlock(&lmax);
137                        }
138                }
139                pthread_exit (NULL);
140                return 0; /* avoid compiler warning */
141        }
142
143        /* return the runtime on the GPU */
144
145        double runOnGPU(){
146                const char file_name[] = "./matrix_max.cl";
147                char * source;
148                size_t source_size;
149
150                double start_time, end_time;
151                start_time = end_time = 0.0;
152
153                float *h_mem_result;
154
155                cl_mem d_mem_matrix;
156                cl_mem d_mem_result;
157
158                cl_platform_id platform;
159                cl_device_id device;
160                cl_context context;
161                cl_command_queue queue;
162                cl_program program;
163                cl_kernel kernel;
164
165                /* Load OpenCL kernel source code */
166                getFileContent(file_name, &source, &source_size);
167
168                h_mem_result = (float*) malloc(rowSize);
169
170                size_t globalSize;
171                cl_int err;
172
173                globalSize = ceil(n/(float)localSize)*localSize;
174
175                /* Get platform and device info */
176                err = clGetPlatformIDs(1, &platform, NULL);
177                err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
178
179                context = clCreateContext(0, 1, &device, NULL, NULL, &err);
180
181                queue = clCreateCommandQueue(context, device, 0, &err);
182
183                program = clCreateProgramWithSource(context, 1, (const char **) &source,
184                                (const size_t*) &source_size, &err);
185
```

A.2. MATRIX MAX

```
186          /* Compile kernel source */
187          clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
188
189          /* Create kernel */
190          kernel = clCreateKernel(program, "matrixMax", &err);
191
192          /* Create memory buffers */
193          d_mem_matrix = clCreateBuffer(context, CL_MEM_READ_ONLY, matrixSize, NULL,
194                      NULL);
195          d_mem_result = clCreateBuffer(context, CL_MEM_WRITE_ONLY, rowSize, NULL,
196                      NULL);
197
198          /* if we want to include time to copy data to GPU,
199           * start timer now */
200          if(gpu_buffer_time_included)
201                  start_time = read_timer();
202
203          /* Copy matrix to GPU */
204          err = clEnqueueWriteBuffer(queue, d_mem_matrix, CL_TRUE, 0, matrixSize,
205                      h_mem_matrix, 0, NULL, NULL);
206
207          /* if we don't want to include time to copy data to GPU,
208           * start timer now */
209          if(!gpu_buffer_time_included)
210                  start_time = read_timer();
211
212          /* Set kernel arguments */
213          err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &d_mem_matrix);
214          err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_mem_result);
215          err |= clSetKernelArg(kernel, 2, sizeof(unsigned int), &n);
216
217          /* Execute kernel */
218          err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &globalSize, &localSize,
219                      0, NULL, NULL);
220
221          clFinish(queue);
222
223          /* Read kernel result */
224          clEnqueueReadBuffer(queue, d_mem_result, CL_TRUE, 0, rowSize, h_mem_result, 0,
225                      NULL, NULL);
226
227          /* Calculate global max */
228          float max = 0;
229          unsigned int i;
230          for(i = 0; i < n; i++){
231                  if(h_mem_result[i] > max){
232                          max = h_mem_result[i];
233                  }
234          }
235
236          end_time = read_timer();
237
238          /* Free resources */
239          clFlush(queue);
240          clFinish(queue);
241          clReleaseMemObject(d_mem_matrix);
242          clReleaseMemObject(d_mem_result);
243          clReleaseCommandQueue(queue);
244          clReleaseContext(context);
245          clReleaseKernel(kernel);
246          clReleaseProgram(program);
247
248          free(source);
249          free(h_mem_result);
250
251          return end_time−start_time;
252  }
253
254  /* Arg1: n (for an n x n matrix)
255   * Arg2: local work size when running on GPU
256   * Arg3: numWorkers for CPU calculation */
```

```
257    int main(int argc, char* argv[]){
258
259            /* read command line */
260            n = (argc > 1) ? atoi(argv[1]) : n;
261            localSize = (argc > 2) ? atoi(argv[2]) : localSize;
262            numWorkers = (argc > 3) ? atoi(argv[3]) : numWorkers;
263
264            matrixSize = n*n*sizeof(float);
265            rowSize = n*sizeof(float);
266
267            h_mem_matrix = (float*) malloc(matrixSize);
268
269            srand((unsigned) time(NULL));
270
271            /* initialize matrix with random floats */
272            unsigned int i;
273            unsigned int j;
274            for(i = 0; i < n; i++){
275                    for(j = 0; j < n; j++){
276                            h_mem_matrix[i*n+j] = ((float)rand()/(float)RAND_MAX)*2000;
277                    }
278            }
279
280            /* Total times */
281            double gpu_time_buffers_included_sum = 0.0;
282            double gpu_time_buffers_excluded_sum = 0.0;
283            double cpu_time_sum = 0.0;
284
285            printf("Running matrix_max on an %d x %d matrix\n", n, n);
286            printf("Local Work Size = %d\n", localSize);
287            printf("Number of CPU workers = %d\n", numWorkers);
288            printf("ITER\tGPU(EX.BUF)\t\tGPU(INC.BUF)\t\tCPU\n");
289
290            /* run calculations */
291            for(i = 0; i < NUM_REPETITIONS; i++){
292
293                    gpu_buffer_time_included = true;
294                    double tmp_gpu_included = runOnGPU();
295
296                    gpu_buffer_time_included = false;
297                    double tmp_gpu_excluded = runOnGPU();
298
299                    double tmp_cpu = runOnCPU(numWorkers);
300
301                    gpu_time_buffers_included_sum += tmp_gpu_included;
302                    gpu_time_buffers_excluded_sum += tmp_gpu_excluded;
303                    cpu_time_sum += tmp_cpu;
304
305                    printf("%d\t%gs\t\t%gs\t\t%gs\n",
306                                    i, tmp_gpu_excluded, tmp_gpu_included, tmp_cpu);
307                    row = 0;
308            }
309
310            printf("——————————————————————————————————————\n");
311            printf("WHAT\t\t\t\t\tTIME(TOT)\tTIME(AVG)\n");
312            printf("GPU time (Transfer time excluded)\t %gs\t%gs\n",
313                            gpu_time_buffers_excluded_sum,
314                            gpu_time_buffers_excluded_sum/NUM_REPETITIONS);
315            printf("GPU time (Transfer time included)\t %gs\t%gs\n",
316                            gpu_time_buffers_included_sum,
317                            gpu_time_buffers_included_sum/NUM_REPETITIONS);
318            printf("CPU time\t\t\t\t %gs\t%gs\n",
319                            cpu_time_sum, cpu_time_sum/NUM_REPETITIONS);
320    }
```

Appendix B

Execution Output

```
 1   Runnning matrix_multiplication on an 128x128 matrix
 2   Local Work Size = 4x4
 3   Number of CPU workers = 4
 4   ITER    GPU(EX.BUF)           GPU(INC.BUF)          CPU
 5   0       0.014544s             0.006908s             0.027684s
 6   1       0.006921s             0.008005s             0.007776s
 7   2       0.006097s             0.008721s             0.007641s
 8   3       0.007118s             0.008943s             0.007415s
 9   4       0.006907s             0.008053s             0.008947s
10   5       0.006178s             0.008495s             0.010691s
11   6       0.006963s             0.006974s             0.007507s
12   7       0.007048s             0.008876s             0.007663s
13   8       0.006978s             0.008522s             0.00744s
14   9       0.006996s             0.008615s             0.007399s
15   ─────────────────────────────────────────
16   WHAT                                  TIME(TOT)      TIME(AVG)
17   GPU time (Transfer time excluded)     0.07575s       0.007575s
18   GPU time (Transfer time included)     0.082112s      0.0082112s
19   CPU time                              0.100163s      0.0100163s
20   Runnning matrix_multiplication on an 128x128 matrix
21   Local Work Size = 8x8
22   Number of CPU workers = 4
23   ITER    GPU(EX.BUF)           GPU(INC.BUF)          CPU
24   0       0.002142s             0.004609s             0.012336s
25   1       0.002447s             0.003991s             0.00741s
26   2       0.00295s              0.004738s             0.007363s
27   3       0.00289s              0.004231s             0.007529s
28   4       0.002992s             0.004124s             0.00739s
29   5       0.002995s             0.004022s             0.007769s
30   6       0.002773s             0.00312s              0.007511s
31   7       0.002745s             0.003945s             0.007384s
32   8       0.00287s              0.003956s             0.007417s
33   9       0.00209s              0.004641s             0.007871s
34   ─────────────────────────────────────────
35   WHAT                                  TIME(TOT)      TIME(AVG)
36   GPU time (Transfer time excluded)     0.026894s      0.0026894s
37   GPU time (Transfer time included)     0.041377s      0.0041377s
38   CPU time                              0.07998s       0.007998s
39   Runnning matrix_multiplication on an 128x128 matrix
40   Local Work Size = 16x16
41   Number of CPU workers = 4
42   ITER    GPU(EX.BUF)           GPU(INC.BUF)          CPU
43   0       0.002454s             0.003739s             0.010249s
44   1       0.002217s             0.003702s             0.007481s
45   2       0.00138s              0.003782s             0.00745s
46   3       0.002295s             0.00393s              0.007502s
47   4       0.001965s             0.002118s             0.007514s
48   5       0.002067s             0.003862s             0.010289s
```

```
49  6          0.002362s              0.003275s              0.007396s
50  7          0.002318s              0.003925s              0.007549s
51  8          0.002377s              0.004092s              0.010567s
52  9          0.002124s              0.003987s              0.008268s
53  ──────────────────────────────────────────
54  WHAT                                     TIME(TOT)      TIME(AVG)
55  GPU time (Transfer time excluded)        0.021559s      0.0021559s
56  GPU time (Transfer time included)        0.036412s      0.0036412s
57  CPU time                                 0.084265s      0.0084265s
58  Runnning matrix_multiplication on an 256x256 matrix
59  Local Work Size = 4x4
60  Number of CPU workers = 4
61  ITER   GPU(EX.BUF)            GPU(INC.BUF)           CPU
62  0      0.047152s              0.048985s              0.061802s
63  1      0.047153s              0.048146s              0.05825s
64  2      0.047097s              0.048916s              0.06081s
65  3      0.047138s              0.048837s              0.059228s
66  4      0.047322s              0.048462s              0.058271s
67  5      0.046546s              0.049s          0.058146s
68  6      0.047019s              0.048893s              0.05831s
69  7      0.047099s              0.049944s              0.058315s
70  8      0.046271s              0.048926s              0.058278s
71  9      0.046692s              0.048705s              0.060958s
72  ──────────────────────────────────────────
73  WHAT                                     TIME(TOT)      TIME(AVG)
74  GPU time (Transfer time excluded)        0.469489s      0.0469489s
75  GPU time (Transfer time included)        0.488814s      0.0488814s
76  CPU time                                 0.592368s      0.0592368s
77  Runnning matrix_multiplication on an 256x256 matrix
78  Local Work Size = 8x8
79  Number of CPU workers = 4
80  ITER   GPU(EX.BUF)            GPU(INC.BUF)           CPU
81  0      0.013063s              0.013069s              0.062792s
82  1      0.012985s              0.0144s         0.058684s
83  2      0.012305s              0.015225s              0.06699s
84  3      0.013208s              0.014056s              0.062544s
85  4      0.012204s              0.013156s              0.060832s
86  5      0.012187s              0.01369s               0.069126s
87  6      0.012257s              0.014892s              0.05856s
88  7      0.012228s              0.013297s              0.065233s
89  8      0.012079s              0.013533s              0.060112s
90  9      0.0123s         0.013678s                     0.059022s
91  ──────────────────────────────────────────
92  WHAT                                     TIME(TOT)      TIME(AVG)
93  GPU time (Transfer time excluded)        0.124816s      0.0124816s
94  GPU time (Transfer time included)        0.138996s      0.0138996s
95  CPU time                                 0.623895s      0.0623895s
96  Runnning matrix_multiplication on an 256x256 matrix
97  Local Work Size = 16x16
98  Number of CPU workers = 4
99  ITER   GPU(EX.BUF)            GPU(INC.BUF)           CPU
100 0      0.006952s              0.00807s               0.082145s
101 1      0.006935s              0.007612s              0.078144s
102 2      0.006991s              0.007632s              0.081516s
103 3      0.006937s              0.007787s              0.077798s
104 4      0.006924s              0.007701s              0.077695s
105 5      0.006826s              0.008497s              0.05988s
106 6      0.007877s              0.009768s              0.058996s
107 7      0.007949s              0.009742s              0.058145s
108 8      0.007892s              0.009748s              0.058245s
109 9      0.007751s              0.008269s              0.061847s
110 ──────────────────────────────────────────
111 WHAT                                     TIME(TOT)      TIME(AVG)
112 GPU time (Transfer time excluded)        0.073034s      0.0073034s
113 GPU time (Transfer time included)        0.084826s      0.0084826s
114 CPU time                                 0.694411s      0.0694411s
115 Runnning matrix_multiplication on an 512x512 matrix
116 Local Work Size = 4x4
117 Number of CPU workers = 4
118 ITER   GPU(EX.BUF)            GPU(INC.BUF)           CPU
119 0      0.36394s               0.369404s              0.475085s
```

```
120  1       0.365542 s                0.368877 s                0.472146 s
121  2       0.364422 s                0.368729 s                0.475139 s
122  3       0.364867 s                0.368139 s                0.471157 s
123  4       0.365211 s                0.367603 s                0.473709 s
124  5       0.364226 s                0.367755 s                0.473935 s
125  6       0.364707 s                0.367439 s                0.467896 s
126  7       0.365334 s                0.368103 s                0.469914 s
127  8       0.36468 s                 0.367493 s                0.469593 s
128  9       0.365196 s                0.367271 s                0.46983 s
129  ——————————————————————————————————
130  WHAT                             TIME(TOT)      TIME(AVG)
131  GPU time ( Transfer time excluded )   3.64812 s      0.364812 s
132  GPU time ( Transfer time included )   3.68081 s      0.368081 s
133  CPU time                         4.7184 s       0.47184 s
134  Runnning matrix_multiplication on an 512x512 matrix
135  Local Work Size = 8x8
136  Number of CPU workers = 4
137  ITER   GPU(EX.BUF)               GPU(INC.BUF)              CPU
138  0       0.093214 s                0.101403 s                0.479275 s
139  1       0.09406 s                 0.09623 s                 0.479455 s
140  2       0.094295 s                0.097246 s                0.475197 s
141  3       0.092446 s                0.096514 s                0.466775 s
142  4       0.093139 s                0.096261 s                0.467282 s
143  5       0.093233 s                0.096633 s                0.472997 s
144  6       0.093517 s                0.096412 s                0.467624 s
145  7       0.093264 s                0.095925 s                0.479448 s
146  8       0.093687 s                0.095954 s                0.46823 s
147  9       0.093502 s                0.095945 s                0.478312 s
148  ——————————————————————————————————
149  WHAT                             TIME(TOT)      TIME(AVG)
150  GPU time ( Transfer time excluded )   0.934357 s     0.0934357 s
151  GPU time ( Transfer time included )   0.968523 s     0.0968523 s
152  CPU time                         4.73459 s      0.473459 s
153  Runnning matrix_multiplication on an 512x512 matrix
154  Local Work Size = 16x16
155  Number of CPU workers = 4
156  ITER   GPU(EX.BUF)               GPU(INC.BUF)              CPU
157  0       0.051152 s                0.054221 s                0.480828 s
158  1       0.051684 s                0.053914 s                0.478244 s
159  2       0.050385 s                0.051905 s                0.474266 s
160  3       0.050783 s                0.053953 s                0.491792 s
161  4       0.050828 s                0.053437 s                0.46962 s
162  5       0.050096 s                0.053974 s                0.470617 s
163  6       0.050905 s                0.053441 s                0.4863 s
164  7       0.050841 s                0.053495 s                0.470826 s
165  8       0.050821 s                0.053476 s                0.470212 s
166  9       0.050831 s                0.053445 s                0.469887 s
167  ——————————————————————————————————
168  WHAT                             TIME(TOT)      TIME(AVG)
169  GPU time ( Transfer time excluded )   0.508326 s     0.0508326 s
170  GPU time ( Transfer time included )   0.535261 s     0.0535261 s
171  CPU time                         4.76259 s      0.476259 s
172  Runnning matrix_multiplication on an 1024x1024 matrix
173  Local Work Size = 4x4
174  Number of CPU workers = 4
175  ITER   GPU(EX.BUF)               GPU(INC.BUF)              CPU
176  0       2.9067 s        2.91092 s                11.4591 s
177  1       2.90139 s                 2.90811 s                11.3965 s
178  2       2.90752 s                 2.90925 s                11.4244 s
179  3       2.89842 s                 2.91144 s                11.4002 s
180  4       2.90246 s                 2.90893 s                11.5065 s
181  5       2.90252 s                 2.90519 s                11.3982 s
182  6       2.90845 s                 2.91526 s                11.2742 s
183  7       2.9083 s        2.9067 s         11.3131 s
184  8       2.90104 s                 2.90797 s                11.3955 s
185  9       2.90034 s                 2.91257 s                11.2565 s
186  ——————————————————————————————————
187  WHAT                             TIME(TOT)      TIME(AVG)
188  GPU time ( Transfer time excluded )   29.0371 s      2.90371 s
189  GPU time ( Transfer time included )   29.0963 s      2.90963 s
190  CPU time                         113.824 s      11.3824 s
```

37

```
191  Runnning matrix_multiplication on an 1024x1024 matrix
192  Local Work Size = 8x8
193  Number of CPU workers = 4
194  ITER    GPU(EX.BUF)           GPU(INC.BUF)           CPU
195  0       0.732165s             0.745469s              11.8158s
196  1       0.729733s             0.736106s              11.7965s
197  2       0.731051s             0.735587s              11.7032s
198  3       0.730153s             0.736223s              11.7488s
199  4       0.730235s             0.736733s              11.7934s
200  5       0.73018s              0.736837s              11.6443s
201  6       0.729615s             0.735408s              11.7064s
202  7       0.729015s             0.738169s              11.8337s
203  8       0.728292s             0.735792s              11.7321s
204  9       0.728999s             0.733398s              11.8367s
205  ─────────────────────────────────────────
206  WHAT                                TIME(TOT)    TIME(AVG)
207  GPU time (Transfer time excluded)    7.29944s     0.729944s
208  GPU time (Transfer time included)    7.36972s     0.736972s
209  CPU time                             117.611s     11.7611s
210  Runnning matrix_multiplication on an 1024x1024 matrix
211  Local Work Size = 16x16
212  Number of CPU workers = 4
213  ITER    GPU(EX.BUF)           GPU(INC.BUF)           CPU
214  0       0.394965s             0.409668s              11.5972s
215  1       0.392015s             0.398419s              11.5878s
216  2       0.391848s             0.39935s               11.4787s
217  3       0.391784s             0.398754s              11.3879s
218  4       0.391632s             0.398469s              11.3492s
219  5       0.391253s             0.399194s              11.4741s
220  6       0.391657s             0.39804s               11.5484s
221  7       0.391634s             0.399436s              11.324s
222  8       0.392128s             0.398106s              11.4123s
223  9       0.39186s              0.399134s              11.4966s
224  ─────────────────────────────────────────
225  WHAT                                TIME(TOT)    TIME(AVG)
226  GPU time (Transfer time excluded)    3.92078s     0.392078s
227  GPU time (Transfer time included)    3.99857s     0.399857s
228  CPU time                             114.656s     11.4656s
229  Runnning matrix_multiplication on an 1536x1536 matrix
230  Local Work Size = 4x4
231  Number of CPU workers = 4
232  ITER    GPU(EX.BUF)           GPU(INC.BUF)           CPU
233  0       9.79704s              9.80289s               39.6244s
234  1       9.79296s              9.80364s               39.5768s
235  2       9.79746s              9.80418s               40.0286s
236  3       9.78764s              9.80714s               39.7419s
237  4       9.80338s              9.81078s               39.4886s
238  5       9.80744s              9.82036s               39.4701s
239  6       9.8015s        9.8157s        39.3647s
240  7       9.81195s              9.81367s               39.4671s
241  8       9.80586s              9.81067s               39.349s
242  9       9.7926s        9.80951s              39.2807s
243  ─────────────────────────────────────────
244  WHAT                                TIME(TOT)    TIME(AVG)
245  GPU time (Transfer time excluded)    97.9978s     9.79978s
246  GPU time (Transfer time included)    98.0985s     9.80985s
247  CPU time                             395.392s     39.5392s
248  Runnning matrix_multiplication on an 1536x1536 matrix
249  Local Work Size = 8x8
250  Number of CPU workers = 4
251  ITER    GPU(EX.BUF)           GPU(INC.BUF)           CPU
252  0       2.46612s              2.48582s               40.0789s
253  1       2.45816s              2.4707s        40.2424s
254  2       2.45953s              2.47398s               40.2495s
255  3       2.46116s              2.47415s               40.0477s
256  4       2.46024s              2.47284s               39.5299s
257  5       2.45948s              2.4725s        39.9242s
258  6       2.45944s              2.47295s               39.8136s
259  7       2.45958s              2.47353s               39.5433s
260  8       2.45824s              2.47239s               39.643s
261  9       2.46103s              2.47358s               40.1815s
```

```
262  ─────────────────────────────────────────
263  WHAT                                        TIME(TOT)        TIME(AVG)
264  GPU time (Transfer time excluded)           24.603s          2.4603s
265  GPU time (Transfer time included)           24.7424s         2.47424s
266  CPU time                                    399.254s         39.9254s
267  Runnning matrix_multiplication on an 1536x1536 matrix
268  Local Work Size = 16x16
269  Number of CPU workers = 4
270  ITER    GPU(EX.BUF)          GPU(INC.BUF)            CPU
271  0       1.3248s         1.33917s                39.0588s
272  1       1.31587s             1.32876s                39.1052s
273  2       1.31722s             1.3285s         38.893s
274  3       1.31604s             1.32837s                38.6614s
275  4       1.31618s             1.32476s                39.1071s
276  5       1.31581s             1.32906s                38.9687s
277  6       1.31627s             1.3292s         38.6317s
278  7       1.31639s             1.3309s         38.6012s
279  8       1.31632s             1.33138s                38.2786s
280  9       1.31621s             1.32875s                38.5687s
281  ─────────────────────────────────────────
282  WHAT                                        TIME(TOT)        TIME(AVG)
283  GPU time (Transfer time excluded)           13.1711s         1.31711s
284  GPU time (Transfer time included)           13.2988s         1.32988s
285  CPU time                                    387.874s         38.7874s
286  Runnning matrix_multiplication on an 2048x2048 matrix
287  Local Work Size = 4x4
288  Number of CPU workers = 4
289  ITER    GPU(EX.BUF)          GPU(INC.BUF)            CPU
290  0       23.209s         23.2351s                104.623s
291  1       23.2174s             23.2117s                106.212s
292  2       23.1939s             23.2181s                104.377s
293  3       23.1905s             23.2172s                104.21s
294  4       23.226s         23.2317s                104.133s
295  5       23.1873s             23.2741s                104.622s
296  6       23.1944s             23.2185s                103.404s
297  7       23.2106s             23.2143s                102.911s
298  8       23.2202s             23.2483s                104.384s
299  9       23.2119s             23.2179s                105.226s
300  ─────────────────────────────────────────
301  WHAT                                        TIME(TOT)        TIME(AVG)
302  GPU time (Transfer time excluded)           232.061s         23.2061s
303  GPU time (Transfer time included)           232.287s         23.2287s
304  CPU time                                    1044.1s          104.41s
305  Runnning matrix_multiplication on an 2048x2048 matrix
306  Local Work Size = 8x8
307  Number of CPU workers = 4
308  ITER    GPU(EX.BUF)          GPU(INC.BUF)            CPU
309  0       5.82678s             5.85154s                102.643s
310  1       5.81994s             5.84253s                100.954s
311  2       5.81964s             5.84165s                102.538s
312  3       5.82112s             5.83922s                101.686s
313  4       5.8215s         5.84715s                101.826s
314  5       5.82161s             5.84122s                102.546s
315  6       5.81516s             5.84573s                100.865s
316  7       5.81573s             5.83851s                101.014s
317  8       5.8212s         5.84242s                101.28s
318  9       5.82119s             5.84431s                101.131s
319  ─────────────────────────────────────────
320  WHAT                                        TIME(TOT)        TIME(AVG)
321  GPU time (Transfer time excluded)           58.2039s         5.82039s
322  GPU time (Transfer time included)           58.4343s         5.84343s
323  CPU time                                    1016.48s         101.648s
324  Runnning matrix_multiplication on an 2048x2048 matrix
325  Local Work Size = 16x16
326  Number of CPU workers = 4
327  ITER    GPU(EX.BUF)          GPU(INC.BUF)            CPU
328  0       3.13105s             3.15585s                100.52s
329  1       3.12284s             3.14552s                101.578s
330  2       3.12185s             3.14306s                100.062s
331  3       3.12214s             3.14661s                100.995s
332  4       3.12221s             3.14241s                102.571s
```

39

```
333   5          3.12263s                    3.14389s                    101.934s
334   6          3.12108s                    3.14248s                    101.878s
335   7          3.11968s                    3.15114s                    101.923s
336   8          3.12159s                    3.14272s                    102.572s
337   9          3.12126s                    3.14502s                    102.403s
338  ─────────────────────────────────────
339   WHAT                                    TIME(TOT)      TIME(AVG)
340   GPU time (Transfer time excluded)       31.2263s       3.12263s
341   GPU time (Transfer time included)       31.4587s       3.14587s
342   CPU time
343   Runnning matrix_multiplication on an 3072x3072 matrix
344   Local Work Size = 4x4
345   Number of CPU workers = 4
346   ITER    GPU(EX.BUF)                 GPU(INC.BUF)               CPU
347   0          78.3929s                    78.4727s                    362.76s
348   1          78.3602s                    78.4396s                    361.861s
349   2          78.3588s                    78.4062s                    360.334s
350   3          78.4473s                    78.397s         362.749s
351   4          78.4221s                    78.4819s                    356.477s
352   5          78.4357s                    78.4805s                    363.152s
353   6          78.3524s                    78.4014s                    362.459s
354   7          78.3476s                    78.4139s                    360.539s
355   8          78.3749s                    78.4088s                    360.499s
356   9          78.4087s                    78.4945s                    359.463s
357  ─────────────────────────────────────
358   WHAT                                    TIME(TOT)      TIME(AVG)
359   GPU time (Transfer time excluded)       783.901s       78.3901s
360   GPU time (Transfer time included)       784.397s       78.4397s
361   CPU time                                3610.29s       361.029s
362   Runnning matrix_multiplication on an 3072x3072 matrix
363   Local Work Size = 8x8
364   Number of CPU workers = 4
365   ITER    GPU(EX.BUF)                 GPU(INC.BUF)               CPU
366   0          19.8566s                    19.8859s                    365.059s
367   1          19.8333s                    19.8493s                    363.537s
368   2          19.8469s                    19.8502s                    363.345s
369   3          19.8299s                    19.8811s                    364.895s
370   4          19.8087s                    19.8395s                    364.145s
371   5          19.8425s                    19.8577s                    365.016s
372   6          19.8396s                    19.8495s                    362.937s
373   7          19.8175s                    19.846s         361.469s
374   8          19.8262s                    19.8789s                    363.374s
375   9          19.8246s                    19.8713s                    362.91s
376  ─────────────────────────────────────
377   WHAT                                    TIME(TOT)      TIME(AVG)
378   GPU time (Transfer time excluded)       198.326s       19.8326s
379   GPU time (Transfer time included)       198.609s       19.8609s
380   CPU time                                3636.69s       363.669s
381   Runnning matrix_multiplication on an 3072x3072 matrix
382   Local Work Size = 16x16
383   Number of CPU workers = 4
384   ITER    GPU(EX.BUF)                 GPU(INC.BUF)               CPU
385   0          10.5373s                    10.5816s                    361.763s
386   1          10.5306s                    10.5658s                    361.001s
387   2          10.5282s                    10.574s         362.444s
388   3          10.5277s                    10.572s         361.273s
389   4          10.5269s                    10.5765s                    362.639s
390   5          10.5272s                    10.5745s                    360.781s
391   6          10.5328s                    10.5674s                    362.028s
392   7          10.5274s                    10.5645s                    360.654s
393   8          10.528s          10.5761s                    360.777s
394   9          10.5249s                    10.5667s                    359.035s
395  ─────────────────────────────────────
396   WHAT                                    TIME(TOT)      TIME(AVG)
397   GPU time (Transfer time excluded)       105.291s       10.5291s
398   GPU time (Transfer time included)       105.719s       10.5719s
399   CPU time                                3612.4s        361.24s
      1016.44s        101.644s
400   Running matrix_max on an 128 x 128 matrix
401   Local Work Size = 4
402   Number of CPU workers = 4
```

```
ITER      GPU(EX.BUF)              GPU(INC.BUF)            CPU
0         0.000897s               0.002459s               0.002947s
1         0.001386s               0.002301s               0.000126s
2         0.001552s               0.002576s               0.000204s
3         0.001649s               0.002389s               0.000162s
4         0.0014s         0.00245s                0.000142s
5         0.001004s               0.002377s               0.00013s
6         0.001415s               0.000991s               0.000243s
7         0.001462s               0.002622s               0.000157s
8         0.001685s               0.00261s                0.000187s
9         0.001537s               0.002275s               0.000162s
————————————————————————————————
WHAT                                     TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)        0.013987s      0.0013987s
GPU time (Transfer time included)        0.02305s       0.002305s
CPU time                                 0.00446s       0.000446s
Runnning matrix_multiplication on an 3072x3072 matrix
Local Work Size = 4x4
Number of CPU workers = 4
ITER      GPU(EX.BUF)              GPU(INC.BUF)            CPU
0         78.3929s                78.4727s                362.76s
1         78.3602s                78.4396s                361.861s
2         78.3588s                78.4062s                360.334s
3         78.4473s                78.397s          362.749s
4         78.4221s                78.4819s                356.477s
5         78.4357s                78.4805s                363.152s
6         78.3524s                78.4014s                362.459s
7         78.3476s                78.4139s                360.539s
8         78.3749s                78.4088s                360.499s
9         78.4087s                78.4945s                359.463s
————————————————————————————————
WHAT                                     TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)        783.901s       78.3901s
GPU time (Transfer time included)        784.397s       78.4397s
CPU time                                 3610.29s       361.029s
Runnning matrix_multiplication on an 3072x3072 matrix
Local Work Size = 8x8
Number of CPU workers = 4
ITER      GPU(EX.BUF)              GPU(INC.BUF)            CPU
0         19.8566s                19.8859s                365.059s
1         19.8333s                19.8493s                363.537s
2         19.8469s                19.8502s                363.345s
3         19.8299s                19.8811s                364.895s
4         19.8087s                19.8395s                364.145s
5         19.8425s                19.8577s                365.016s
6         19.8396s                19.8495s                362.937s
7         19.8175s                19.846s          361.469s
8         19.8262s                19.8789s                363.374s
9         19.8246s                19.8713s                362.91s
————————————————————————————————
WHAT                                     TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)        198.326s       19.8326s
GPU time (Transfer time included)        198.609s       19.8609s
CPU time                                 3636.69s       363.669s
Runnning matrix_multiplication on an 3072x3072 matrix
Local Work Size = 16x16
Number of CPU workers = 4
ITER      GPU(EX.BUF)              GPU(INC.BUF)            CPU
0         10.5373s                10.5816s                361.763s
1         10.5306s                10.5658s                361.001s
2         10.5282s                10.574s          362.444s
3         10.5277s                10.572s          361.273s
4         10.5269s                10.5765s                362.639s
5         10.5272s                10.5745s                360.781s
6         10.5328s                10.5674s                362.028s
7         10.5274s                10.5645s                360.654s
8         10.528s         10.5761s                360.777s
9         10.5249s                10.5667s                359.035s
————————————————————————————————
WHAT                                     TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)        105.291s       10.5291s
```

```
474  GPU time (Transfer time included)        105.719s          10.5719s
475  CPU time                                 3612.4s           361.24s
476  Running matrix_max on an 128 x 128 matrix
477  Local Work Size = 8
478  Number of CPU workers = 4
479  ITER    GPU(EX.BUF)            GPU(INC.BUF)          CPU
480  0       0.000652s             0.001052s             0.00304s
481  1       0.002327s             0.001952s             0.000138s
482  2       0.001269s             0.002122s             0.00017s
483  3       0.001411s             0.001187s             0.000215s
484  4       0.000872s             0.001357s             0.000149s
485  5       0.001387s             0.002209s             0.000151s
486  6       0.001072s             0.001346s             0.000173s
487  7       0.001291s             0.00224s              0.000131s
488  8       0.00136s              0.001077s             0.000236s
489  9       0.001343s             0.001522s             0.000167s
490  ─────────────────────────────────────
491  WHAT                                     TIME(TOT)         TIME(AVG)
492  GPU time (Transfer time excluded)        0.012984s         0.0012984s
493  GPU time (Transfer time included)        0.016064s         0.0016064s
494  CPU time                                 0.00457s          0.000457s
495  Running matrix_max on an 128 x 128 matrix
496  Local Work Size = 16
497  Number of CPU workers = 4
498  ITER    GPU(EX.BUF)            GPU(INC.BUF)          CPU
499  0       0.001214s             0.002315s             0.003017s
500  1       0.001161s             0.001305s             0.000147s
501  2       0.001136s             0.002218s             0.000152s
502  3       0.001169s             0.002483s             0.000146s
503  4       0.001069s             0.002606s             0.000147s
504  5       0.001219s             0.002912s             0.000159s
505  6       0.001446s             0.002332s             0.000162s
506  7       0.001459s             0.002196s             0.000122s
507  8       0.001116s             0.002303s             0.000129s
508  9       0.000798s             0.001805s             0.00015s
509  ─────────────────────────────────────
510  WHAT                                     TIME(TOT)         TIME(AVG)
511  GPU time (Transfer time excluded)        0.011787s         0.0011787s
512  GPU time (Transfer time included)        0.022475s         0.0022475s
513  CPU time                                 0.004331s         0.0004331s
514  Running matrix_max on an 128 x 128 matrix
515  Local Work Size = 32
516  Number of CPU workers = 4
517  ITER    GPU(EX.BUF)            GPU(INC.BUF)          CPU
518  0       0.002256s             0.002659s             0.003209s
519  1       0.001104s             0.002447s             0.000149s
520  2       0.001607s             0.001493s             0.000245s
521  3       0.0011s        0.0022s        0.000137s
522  4       0.001496s             0.001978s             0.000137s
523  5       0.001525s             0.003187s             0.000171s
524  6       0.001503s             0.002482s             0.000122s
525  7       0.001508s             0.00247s              0.000119s
526  8       0.001451s             0.002497s             0.000155s
527  9       0.001335s             0.002292s             0.000166s
528  ─────────────────────────────────────
529  WHAT                                     TIME(TOT)         TIME(AVG)
530  GPU time (Transfer time excluded)        0.014885s         0.0014885s
531  GPU time (Transfer time included)        0.023705s         0.0023705s
532  CPU time                                 0.00461s          0.000461s
533  Running matrix_max on an 128 x 128 matrix
534  Local Work Size = 64
535  Number of CPU workers = 4
536  ITER    GPU(EX.BUF)            GPU(INC.BUF)          CPU
537  0       0.001582s             0.002355s             0.003233s
538  1       0.001441s             0.002108s             0.000122s
539  2       0.001541s             0.002106s             0.000159s
540  3       0.001085s             0.002548s             0.000146s
541  4       0.001577s             0.002477s             0.000166s
542  5       0.001313s             0.002331s             0.000191s
543  6       0.000906s             0.001385s             0.000158s
544  7       0.001042s             0.002219s             0.000145s
```

```
545  8          0.001406 s              0.001531 s              0.000152 s
546  9          0.001475 s              0.002322 s              0.000273 s
547  ─────────────────────────────────────
548  WHAT                                     TIME(TOT)        TIME(AVG)
549  GPU time ( Transfer time excluded )      0.013368 s       0.0013368 s
550  GPU time ( Transfer time included )      0.021382 s       0.0021382 s
551  CPU time                                 0.004745 s       0.0004745 s
552  Running matrix_max on an 128 x 128 matrix
553  Local Work Size = 128
554  Number of CPU workers = 4
555  ITER     GPU(EX.BUF)              GPU(INC.BUF)            CPU
556  0          0.001166 s              0.002359 s              0.010251 s
557  1          0.001147 s              0.000991 s              0.000135 s
558  2          0.001552 s              0.002307 s              0.000128 s
559  3          0.001647 s              0.00249 s               0.000126 s
560  4          0.000923 s              0.002384 s              0.000142 s
561  5          0.001337 s              0.001148 s              0.000165 s
562  6          0.000822 s              0.001856 s              0.000126 s
563  7          0.00089 s               0.002228 s              0.000127 s
564  8          0.000692 s              0.001153 s              0.000123 s
565  9          0.001345 s              0.001294 s              0.000288 s
566  ─────────────────────────────────────
567  WHAT                                     TIME(TOT)        TIME(AVG)
568  GPU time ( Transfer time excluded )      0.011521 s       0.0011521 s
569  GPU time ( Transfer time included )      0.01821 s        0.001821 s
570  CPU time                                 0.011611 s       0.0011611 s
571  Running matrix_max on an 128 x 128 matrix
572  Local Work Size = 256
573  Number of CPU workers = 4
574  ITER     GPU(EX.BUF)              GPU(INC.BUF)            CPU
575  0          0.000592 s              0.002583 s              0.003145 s
576  1          0.000597 s              0.001216 s              0.000133 s
577  2          0.001406 s              0.002086 s              0.000151 s
578  3          0.000706 s              0.00217 s               0.000152 s
579  4          0.000928 s              0.002013 s              0.00016 s
580  5          0.001447 s              0.001383 s              0.000169 s
581  6          0.000926 s              0.002435 s              0.000256 s
582  7          0.001596 s              0.002584 s              0.00015 s
583  8          0.000921 s              0.002177 s              0.000386 s
584  9          0.0014 s          0.001418 s              0.000186 s
585  ─────────────────────────────────────
586  WHAT                                     TIME(TOT)        TIME(AVG)
587  GPU time ( Transfer time excluded )      0.010519 s       0.0010519 s
588  GPU time ( Transfer time included )      0.020065 s       0.0020065 s
589  CPU time                                 0.004888 s       0.0004888 s
590  Running matrix_max on an 256 x 256 matrix
591  Local Work Size = 4
592  Number of CPU workers = 4
593  ITER     GPU(EX.BUF)              GPU(INC.BUF)            CPU
594  0          0.002058 s              0.002926 s              0.003533 s
595  1          0.001675 s              0.002633 s              0.000193 s
596  2          0.001843 s              0.002728 s              0.000257 s
597  3          0.001829 s              0.002599 s              0.000196 s
598  4          0.001959 s              0.003095 s              0.000207 s
599  5          0.002033 s              0.002957 s              0.000193 s
600  6          0.002038 s              0.002937 s              0.000195 s
601  7          0.001865 s              0.001716 s              0.000193 s
602  8          0.001564 s              0.00257 s               0.000196 s
603  9          0.002022 s              0.003117 s              0.000188 s
604  ─────────────────────────────────────
605  WHAT                                     TIME(TOT)        TIME(AVG)
606  GPU time ( Transfer time excluded )      0.018886 s       0.0018886 s
607  GPU time ( Transfer time included )      0.027278 s       0.0027278 s
608  CPU time                                 0.005351 s       0.0005351 s
609  Running matrix_max on an 256 x 256 matrix
610  Local Work Size = 8
611  Number of CPU workers = 4
612  ITER     GPU(EX.BUF)              GPU(INC.BUF)            CPU
613  0          0.001733 s              0.002496 s              0.003082 s
614  1          0.001563 s              0.001583 s              0.000292 s
615  2          0.000974 s              0.002618 s              0.000197 s
```

```
616  3          0.001788s               0.002666s                    0.000205s
617  4          0.001691s               0.002968s                    0.000206s
618  5          0.001525s               0.002397s                    0.000253s
619  6          0.001605s               0.002426s                    0.000235s
620  7          0.001568s               0.002352s                    0.000216s
621  8          0.001411s               0.002353s                    0.000205s
622  9          0.001358s               0.002808s                    0.000195s
623  ————————————————————————
624  WHAT                               TIME(TOT)        TIME(AVG)
625  GPU time (Transfer time excluded)   0.015216s        0.0015216s
626  GPU time (Transfer time included)   0.024667s        0.0024667s
627  CPU time                            0.005086s        0.0005086s
628  Running matrix_max on an 256 x 256 matrix
629  Local Work Size = 16
630  Number of CPU workers = 4
631  ITER    GPU(EX.BUF)         GPU(INC.BUF)            CPU
632  0        0.00172s            0.002676s               0.00316s
633  1        0.002563s           0.002492s               0.000188s
634  2        0.001315s           0.002299s               0.000207s
635  3        0.001518s           0.002548s               0.0002s
636  4        0.001532s           0.002389s               0.000279s
637  5        0.001725s           0.002478s               0.000199s
638  6        0.001715s           0.002639s               0.000239s
639  7        0.001725s           0.002646s               0.000185s
640  8        0.001766s           0.002621s               0.000204s
641  9        0.001716s           0.004157s               0.000208s
642  ————————————————————————
643  WHAT                               TIME(TOT)        TIME(AVG)
644  GPU time (Transfer time excluded)   0.017295s        0.0017295s
645  GPU time (Transfer time included)   0.026945s        0.0026945s
646  CPU time                            0.005069s        0.0005069s
647  Running matrix_max on an 256 x 256 matrix
648  Local Work Size = 32
649  Number of CPU workers = 4
650  ITER    GPU(EX.BUF)         GPU(INC.BUF)            CPU
651  0        0.001528s           0.002729s               0.010113s
652  1        0.001557s           0.002734s               0.000177s
653  2        0.001804s           0.002561s               0.000209s
654  3        0.00175s            0.002776s               0.000227s
655  4        0.001835s           0.002278s               0.000216s
656  5        0.001528s           0.002406s               0.000194s
657  6        0.001716s           0.00247s                0.000208s
658  7        0.001519s           0.0023s           0.000205s
659  8        0.001529s           0.002291s               0.000192s
660  9        0.001776s           0.002629s               0.000286s
661  ————————————————————————
662  WHAT                               TIME(TOT)        TIME(AVG)
663  GPU time (Transfer time excluded)   0.016542s        0.0016542s
664  GPU time (Transfer time included)   0.025174s        0.0025174s
665  CPU time                            0.012027s        0.0012027s
666  Running matrix_max on an 256 x 256 matrix
667  Local Work Size = 64
668  Number of CPU workers = 4
669  ITER    GPU(EX.BUF)         GPU(INC.BUF)            CPU
670  0        0.00159s            0.002474s               0.003036s
671  1        0.001488s           0.0022s           0.00029s
672  2        0.00157s            0.002674s               0.000191s
673  3        0.001762s           0.002774s               0.000193s
674  4        0.001338s           0.002666s               0.00018s
675  5        0.000919s           0.002831s               0.000205s
676  6        0.001885s           0.00233s                0.000356s
677  7        0.001746s           0.002618s               0.000192s
678  8        0.00154s            0.002643s               0.000203s
679  9        0.001595s           0.001548s               0.000283s
680  ————————————————————————
681  WHAT                               TIME(TOT)        TIME(AVG)
682  GPU time (Transfer time excluded)   0.015433s        0.0015433s
683  GPU time (Transfer time included)   0.024758s        0.0024758s
684  CPU time                            0.005129s        0.0005129s
685  Running matrix_max on an 256 x 256 matrix
686  Local Work Size = 128
```

```
687  Number of CPU workers = 4
688  ITER    GPU(EX.BUF)           GPU(INC.BUF)          CPU
689  0       0.001456s             0.002627s             0.003016s
690  1       0.001544s             0.003878s             0.000215s
691  2       0.00148s              0.002359s             0.000207s
692  3       0.001736s             0.002755s             0.000207s
693  4       0.001806s             0.002383s             0.000213s
694  5       0.001759s             0.002853s             0.000185s
695  6       0.001758s             0.002761s             0.000207s
696  7       0.001748s             0.002698s             0.000193s
697  8       0.00144s              0.002319s             0.000211s
698  9       0.001519s             0.002384s             0.000267s
699  ─────────────────────────────────────────
700  WHAT                              TIME(TOT)     TIME(AVG)
701  GPU time (Transfer time excluded)    0.016246s    0.0016246s
702  GPU time (Transfer time included)    0.027017s    0.0027017s
703  CPU time                             0.004921s    0.0004921s
704  Running matrix_max on an 256 x 256 matrix
705  Local Work Size = 256
706  Number of CPU workers = 4
707  ITER    GPU(EX.BUF)           GPU(INC.BUF)          CPU
708  0       0.000763s             0.003019s             0.002969s
709  1       0.001494s             0.001471s             0.000187s
710  2       0.00149s              0.002564s             0.000203s
711  3       0.001676s             0.002821s             0.000196s
712  4       0.001794s             0.001528s             0.000215s
713  5       0.001692s             0.002221s             0.00023s
714  6       0.001723s             0.003359s             0.000223s
715  7       0.001805s             0.002571s             0.000202s
716  8       0.001471s             0.001451s             0.000255s
717  9       0.00136s              0.002619s             0.000285s
718  ─────────────────────────────────────────
719  WHAT                              TIME(TOT)     TIME(AVG)
720  GPU time (Transfer time excluded)    0.015268s    0.0015268s
721  GPU time (Transfer time included)    0.023624s    0.0023624s
722  CPU time                             0.004965s    0.0004965s
723  Running matrix_max on an 512 x 512 matrix
724  Local Work Size = 4
725  Number of CPU workers = 4
726  ITER    GPU(EX.BUF)           GPU(INC.BUF)          CPU
727  0       0.003581s             0.005311s             0.006013s
728  1       0.003582s             0.004912s             0.000372s
729  2       0.003549s             0.004037s             0.00039s
730  3       0.003546s             0.005449s             0.000385s
731  4       0.003427s             0.005251s             0.000399s
732  5       0.003328s             0.004118s             0.000489s
733  6       0.003635s             0.005577s             0.000389s
734  7       0.002633s             0.005227s             0.000379s
735  8       0.003376s             0.004945s             0.000402s
736  9       0.003091s             0.00494s              0.000373s
737  ─────────────────────────────────────────
738  WHAT                              TIME(TOT)     TIME(AVG)
739  GPU time (Transfer time excluded)    0.033748s    0.0033748s
740  GPU time (Transfer time included)    0.049767s    0.0049767s
741  CPU time                             0.009591s    0.0009591s
742  Running matrix_max on an 512 x 512 matrix
743  Local Work Size = 8
744  Number of CPU workers = 4
745  ITER    GPU(EX.BUF)           GPU(INC.BUF)          CPU
746  0       0.002372s             0.002607s             0.003433s
747  1       0.002362s             0.003144s             0.00039s
748  2       0.002784s             0.002736s             0.000377s
749  3       0.002342s             0.003731s             0.000516s
750  4       0.002386s             0.003967s             0.000393s
751  5       0.001958s             0.00393s              0.000396s
752  6       0.002367s             0.003991s             0.000385s
753  7       0.002134s             0.003115s             0.00039s
754  8       0.002645s             0.00306s              0.00036s
755  9       0.002637s             0.004239s             0.000376s
756  ─────────────────────────────────────────
757  WHAT                              TIME(TOT)     TIME(AVG)
```

```
758  GPU time (Transfer time excluded)      0.023987s        0.0023987s
759  GPU time (Transfer time included)      0.03452s         0.003452s
760  CPU time                               0.007016s        0.0007016s
761  Running matrix_max on an 512 x 512 matrix
762  Local Work Size = 16
763  Number of CPU workers = 4
764  ITER    GPU(EX.BUF)            GPU(INC.BUF)         CPU
765  0       0.001427s             0.006169s            0.014051s
766  1       0.002124s             0.00367s             0.000494s
767  2       0.002405s             0.00406s             0.000411s
768  3       0.001305s             0.003754s            0.000395s
769  4       0.002041s             0.002682s            0.000506s
770  5       0.002204s             0.004033s            0.000398s
771  6       0.001409s             0.002312s            0.000386s
772  7       0.002095s             0.003267s            0.000383s
773  8       0.00203s              0.003635s            0.000504s
774  9       0.002061s             0.002369s            0.000417s
775  ————————————————————————————————————————
776  WHAT                          TIME(TOT)            TIME(AVG)
777  GPU time (Transfer time excluded)      0.019101s        0.0019101s
778  GPU time (Transfer time included)      0.035951s        0.0035951s
779  CPU time                               0.017945s        0.0017945s
780  Running matrix_max on an 512 x 512 matrix
781  Local Work Size = 32
782  Number of CPU workers = 4
783  ITER    GPU(EX.BUF)            GPU(INC.BUF)         CPU
784  0       0.00462s              0.005915s            0.010312s
785  1       0.004092s             0.006272s            0.0004s
786  2       0.004315s             0.006022s            0.000394s
787  3       0.00464s              0.006097s            0.000398s
788  4       0.004555s             0.006319s            0.000371s
789  5       0.004618s             0.005079s            0.000427s
790  6       0.004564s             0.006543s            0.000397s
791  7       0.004732s             0.00629s             0.000383s
792  8       0.004601s             0.005026s            0.000438s
793  9       0.004628s             0.006035s            0.000395s
794  ————————————————————————————————————————
795  WHAT                          TIME(TOT)            TIME(AVG)
796  GPU time (Transfer time excluded)      0.045365s        0.0045365s
797  GPU time (Transfer time included)      0.059598s        0.0059598s
798  CPU time                               0.013915s        0.0013915s
799  Running matrix_max on an 512 x 512 matrix
800  Local Work Size = 64
801  Number of CPU workers = 4
802  ITER    GPU(EX.BUF)            GPU(INC.BUF)         CPU
803  0       0.004532s             0.00656s             0.003189s
804  1       0.004591s             0.006119s            0.000386s
805  2       0.004628s             0.004923s            0.000406s
806  3       0.004503s             0.005256s            0.000388s
807  4       0.004758s             0.006546s            0.000373s
808  5       0.004707s             0.006483s            0.000368s
809  6       0.0048s        0.006491s              0.000397s
810  7       0.004751s             0.006485s            0.000363s
811  8       0.004783s             0.00559s             0.0004s
812  9       0.004844s             0.006159s            0.000368s
813  ————————————————————————————————————————
814  WHAT                          TIME(TOT)            TIME(AVG)
815  GPU time (Transfer time excluded)      0.046897s        0.0046897s
816  GPU time (Transfer time included)      0.060612s        0.0060612s
817  CPU time                               0.006638s        0.0006638s
818  Running matrix_max on an 512 x 512 matrix
819  Local Work Size = 128
820  Number of CPU workers = 4
821  ITER    GPU(EX.BUF)            GPU(INC.BUF)         CPU
822  0       0.004497s             0.005994s            0.003219s
823  1       0.003917s             0.005922s            0.000391s
824  2       0.004545s             0.006247s            0.000376s
825  3       0.004814s             0.00636s             0.000387s
826  4       0.004244s             0.005759s            0.000393s
827  5       0.004457s             0.004633s            0.000391s
828  6       0.005942s             0.006509s            0.000373s
```

```
7        0.004811s              0.006163s              0.000383s
8        0.004805s              0.00611s               0.00038s
9        0.004453s              0.005198s              0.00041s
───────────────────────────────────
WHAT                                TIME(TOT)          TIME(AVG)
GPU time ( Transfer time excluded )  0.046485s          0.0046485s
GPU time ( Transfer time included )  0.058895s          0.0058895s
CPU time                             0.006703s          0.0006703s
Running matrix_max on an 512 x 512 matrix
Local Work Size = 256
Number of CPU workers = 4
ITER    GPU(EX.BUF)             GPU(INC.BUF)           CPU
0        0.004612s              0.005496s              0.008763s
1        0.004806s              0.006357s              0.000387s
2        0.004594s              0.005249s              0.00041s
3        0.004532s              0.005147s              0.000528s
4        0.004701s              0.006346s              0.000393s
5        0.004534s              0.006227s              0.000379s
6        0.004788s              0.00643s               0.000467s
7        0.004566s              0.006088s              0.000462s
8        0.004744s              0.005411s              0.000455s
9        0.00381s               0.005289s              0.000462s
───────────────────────────────────
WHAT                                TIME(TOT)          TIME(AVG)
GPU time ( Transfer time excluded )  0.045687s          0.0045687s
GPU time ( Transfer time included )  0.05804s           0.005804s
CPU time                             0.012706s          0.0012706s
Running matrix_max on an 1024 x 1024 matrix
Local Work Size = 4
Number of CPU workers = 4
ITER    GPU(EX.BUF)             GPU(INC.BUF)           CPU
0        0.009783s              0.013521s              0.004329s
1        0.009828s              0.016197s              0.001169s
2        0.009617s              0.0138s        0.00118s
3        0.009765s              0.011175s              0.00116s
4        0.009567s              0.013369s              0.001163s
5        0.009699s              0.013311s              0.001209s
6        0.009629s              0.01356s               0.00118s
7        0.00982s               0.012597s              0.001182s
8        0.008899s              0.013642s              0.001167s
9        0.009707s              0.012726s              0.001168s
───────────────────────────────────
WHAT                                TIME(TOT)          TIME(AVG)
GPU time ( Transfer time excluded )  0.096314s          0.0096314s
GPU time ( Transfer time included )  0.133898s          0.0133898s
CPU time                             0.014907s          0.0014907s
Running matrix_max on an 1024 x 1024 matrix
Local Work Size = 8
Number of CPU workers = 4
ITER    GPU(EX.BUF)             GPU(INC.BUF)           CPU
0        0.00602s               0.007891s              0.011243s
1        0.006157s              0.009102s              0.001177s
2        0.00606s               0.011089s              0.001151s
3        0.006023s              0.009611s              0.001272s
4        0.006284s              0.009754s              0.001159s
5        0.006116s              0.009796s              0.001167s
6        0.006089s              0.00886s               0.002135s
7        0.005985s              0.009824s              0.00126s
8        0.006149s              0.010218s              0.001168s
9        0.00585s               0.009659s              0.001175s
───────────────────────────────────
WHAT                                TIME(TOT)          TIME(AVG)
GPU time ( Transfer time excluded )  0.060733s          0.0060733s
GPU time ( Transfer time included )  0.095804s          0.0095804s
CPU time                             0.022907s          0.0022907s
Running matrix_max on an 1024 x 1024 matrix
Local Work Size = 16
Number of CPU workers = 4
ITER    GPU(EX.BUF)             GPU(INC.BUF)           CPU
0        0.004852s              0.007888s              0.003961s
1        0.004668s              0.008571s              0.00272s
```

```
2        0.005048 s              0.008662 s              0.001186 s
3        0.00498 s               0.008991 s              0.001211 s
4        0.005059 s              0.008952 s              0.001186 s
5        0.004067 s              0.008756 s              0.001177 s
6        0.004773 s              0.008607 s              0.001454 s
7        0.004751 s              0.008759 s              0.001201 s
8        0.004799 s              0.008623 s              0.001187 s
9        0.004764 s              0.007621 s              0.001269 s
————————————————————————
WHAT                                  TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)     0.047761 s     0.0047761 s
GPU time (Transfer time included)     0.08543 s      0.008543 s
CPU time                              0.016552 s     0.0016552 s
Running matrix_max on an 1024 x 1024 matrix
Local Work Size = 32
Number of CPU workers = 4
ITER     GPU(EX.BUF)             GPU(INC.BUF)            CPU
0        0.015142 s              0.017043 s              0.003967 s
1        0.021241 s              0.025413 s              0.001209 s
2        0.015065 s              0.025127 s              0.001166 s
3        0.015205 s              0.018223 s              0.001158 s
4        0.020599 s              0.017911 s              0.001195 s
5        0.015305 s              0.018813 s              0.001155 s
6        0.020905 s              0.019487 s              0.001162 s
7        0.0156 s        0.01823 s               0.001307 s
8        0.015499 s              0.024815 s              0.001143 s
9        0.021106 s              0.018921 s              0.001163 s
————————————————————————
WHAT                                  TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)     0.175667 s     0.0175667 s
GPU time (Transfer time included)     0.203983 s     0.0203983 s
CPU time                              0.014625 s     0.0014625 s
Running matrix_max on an 1024 x 1024 matrix
Local Work Size = 64
Number of CPU workers = 4
ITER     GPU(EX.BUF)             GPU(INC.BUF)            CPU
0        0.027427 s              0.030744 s              0.007786 s
1        0.028529 s              0.02936 s               0.001164 s
2        0.029176 s              0.029449 s              0.001277 s
3        0.029704 s              0.032546 s              0.001145 s
4        0.028593 s              0.030487 s              0.00116 s
5        0.028324 s              0.033259 s              0.001152 s
6        0.027266 s              0.033412 s              0.001177 s
7        0.028815 s              0.029342 s              0.001182 s
8        0.027337 s              0.033338 s              0.001286 s
9        0.027946 s              0.030395 s              0.001274 s
————————————————————————
WHAT                                  TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)     0.283117 s     0.0283117 s
GPU time (Transfer time included)     0.312332 s     0.0312332 s
CPU time                              0.018603 s     0.0018603 s
Running matrix_max on an 1024 x 1024 matrix
Local Work Size = 128
Number of CPU workers = 4
ITER     GPU(EX.BUF)             GPU(INC.BUF)            CPU
0        0.027443 s              0.031353 s              0.010925 s
1        0.028296 s              0.032279 s              0.001181 s
2        0.029408 s              0.031936 s              0.001178 s
3        0.029217 s              0.029927 s              0.001168 s
4        0.027704 s              0.032497 s              0.001176 s
5        0.027188 s              0.031925 s              0.001178 s
6        0.027231 s              0.030137 s              0.001185 s
7        0.028792 s              0.032743 s              0.001168 s
8        0.027958 s              0.030945 s              0.00126 s
9        0.027412 s              0.033447 s              0.001167 s
————————————————————————
WHAT                                  TIME(TOT)      TIME(AVG)
GPU time (Transfer time excluded)     0.280649 s     0.0280649 s
GPU time (Transfer time included)     0.317189 s     0.0317189 s
CPU time                              0.021586 s     0.0021586 s
Running matrix_max on an 1024 x 1024 matrix
```

```
Local Work Size = 256
Number of CPU workers = 4
ITER    GPU(EX.BUF)            GPU(INC.BUF)           CPU
0       0.026769s              0.029023s              0.010639s
1       0.027628s              0.030637s              0.001155s
2       0.02746s               0.030373s              0.001168s
3       0.027631s              0.031535s              0.0015s
4       0.027323s              0.030836s              0.001161s
5       0.027196s              0.026719s              0.001158s
6       0.027129s              0.031038s              0.001165s
7       0.027836s              0.030373s              0.001154s
8       0.026442s              0.030461s              0.001177s
9       0.027806s              0.032078s              0.001159s
————————————————————————————————————
WHAT                                      TIME(TOT)    TIME(AVG)
GPU time (Transfer time excluded)         0.27322s     0.027322s
GPU time (Transfer time included)         0.303073s    0.0303073s
CPU time                                  0.021436s    0.0021436s
Running matrix_max on an 2048 x 2048 matrix
Local Work Size = 4
Number of CPU workers = 4
ITER    GPU(EX.BUF)            GPU(INC.BUF)           CPU
0       0.037975s              0.044523s              0.007314s
1       0.034144s              0.046313s              0.004208s
2       0.034182s              0.047215s              0.004219s
3       0.034537s              0.047141s              0.004189s
4       0.034357s              0.046892s              0.004176s
5       0.033332s              0.04485s               0.004295s
6       0.034521s              0.047309s              0.004217s
7       0.034373s              0.04499s               0.004199s
8       0.034246s              0.046476s              0.004546s
9       0.034176s              0.046729s              0.004248s
————————————————————————————————————
WHAT                                      TIME(TOT)    TIME(AVG)
GPU time (Transfer time excluded)         0.345843s    0.0345843s
GPU time (Transfer time included)         0.462438s    0.0462438s
CPU time                                  0.045611s    0.0045611s
Running matrix_max on an 2048 x 2048 matrix
Local Work Size = 8
Number of CPU workers = 4
ITER    GPU(EX.BUF)            GPU(INC.BUF)           CPU
0       0.019552s              0.031673s              0.01173s
1       0.019347s              0.031547s              0.004211s
2       0.019734s              0.03188s               0.004216s
3       0.019609s              0.029883s              0.004774s
4       0.019546s              0.031881s              0.004177s
5       0.019449s              0.032877s              0.004188s
6       0.019939s              0.03216s               0.004882s
7       0.019309s              0.031995s              0.004182s
8       0.019084s              0.030489s              0.004291s
9       0.019495s              0.026772s              0.005307s
————————————————————————————————————
WHAT                                      TIME(TOT)    TIME(AVG)
GPU time (Transfer time excluded)         0.195064s    0.0195064s
GPU time (Transfer time included)         0.311157s    0.0311157s
CPU time                                  0.051958s    0.0051958s
Running matrix_max on an 2048 x 2048 matrix
Local Work Size = 16
Number of CPU workers = 4
ITER    GPU(EX.BUF)            GPU(INC.BUF)           CPU
0       0.016829s              0.028146s              0.007165s
1       0.017173s              0.027926s              0.004239s
2       0.016533s              0.030183s              0.004226s
3       0.017411s              0.030866s              0.004184s
4       0.01724s               0.029749s              0.004207s
5       0.017385s              0.030135s              0.004214s
6       0.017242s              0.027609s              0.004327s
7       0.016919s              0.030108s              0.004211s
8       0.017344s              0.029764s              0.004185s
9       0.016754s              0.02951s               0.004325s
————————————————————————————————————
```

```
1042  WHAT                                        TIME(TOT)        TIME(AVG)
1043  GPU time (Transfer time excluded)          0.17083s         0.017083s
1044  GPU time (Transfer time included)          0.293996s        0.0293996s
1045  CPU time                                   0.045283s        0.0045283s
1046  Running matrix_max on an 2048 x 2048 matrix
1047  Local Work Size = 32
1048  Number of CPU workers = 4
1049  ITER     GPU(EX.BUF)            GPU(INC.BUF)           CPU
1050  0        0.03854s               0.059666s              0.011355s
1051  1        0.04447s               0.053674s              0.004299s
1052  2        0.041393s              0.051144s              0.004589s
1053  3        0.037739s              0.056971s              0.0042s
1054  4        0.063651s              0.052132s              0.004199s
1055  5        0.038427s              0.049212s              0.00431s
1056  6        0.04236s               0.051234s              0.004207s
1057  7        0.038562s              0.058057s              0.004187s
1058  8        0.053482s              0.048596s              0.004185s
1059  9        0.044194s              0.051001s              0.00419s
1060  ─────────────────────────────────────
1061  WHAT                                        TIME(TOT)        TIME(AVG)
1062  GPU time (Transfer time excluded)          0.442818s        0.0442818s
1063  GPU time (Transfer time included)          0.531687s        0.0531687s
1064  CPU time                                   0.049721s        0.0049721s
1065  Running matrix_max on an 2048 x 2048 matrix
1066  Local Work Size = 64
1067  Number of CPU workers = 4
1068  ITER     GPU(EX.BUF)            GPU(INC.BUF)           CPU
1069  0        0.09886s               0.112406s              0.007138s
1070  1        0.07567s               0.111075s              0.004304s
1071  2        0.073248s              0.117349s              0.004199s
1072  3        0.095706s              0.086189s              0.004213s
1073  4        0.10048s               0.110765s              0.004206s
1074  5        0.10083s               0.084656s              0.004214s
1075  6        0.096937s              0.089828s              0.00423s
1076  7        0.09979s               0.086881s              0.004231s
1077  8        0.096096s              0.085355s              0.004208s
1078  9        0.098991s              0.113271s              0.004213s
1079  ─────────────────────────────────────
1080  WHAT                                        TIME(TOT)        TIME(AVG)
1081  GPU time (Transfer time excluded)          0.936608s        0.0936608s
1082  GPU time (Transfer time included)          0.997775s        0.0997775s
1083  CPU time                                   0.045156s        0.0045156s
1084  Running matrix_max on an 2048 x 2048 matrix
1085  Local Work Size = 128
1086  Number of CPU workers = 4
1087  ITER     GPU(EX.BUF)            GPU(INC.BUF)           CPU
1088  0        0.104191s              0.112773s              0.01054s
1089  1        0.102969s              0.112369s              0.004416s
1090  2        0.103673s              0.112885s              0.005257s
1091  3        0.099153s              0.115232s              0.004208s
1092  4        0.105941s              0.114272s              0.004288s
1093  5        0.105845s              0.11226s               0.00418s
1094  6        0.105462s              0.113682s              0.00418s
1095  7        0.100953s              0.114217s              0.004221s
1096  8        0.102782s              0.114758s              0.004191s
1097  9        0.104498s              0.11809s               0.004178s
1098  ─────────────────────────────────────
1099  WHAT                                        TIME(TOT)        TIME(AVG)
1100  GPU time (Transfer time excluded)          1.03547s         0.103547s
1101  GPU time (Transfer time included)          1.14054s         0.114054s
1102  CPU time                                   0.049659s        0.0049659s
1103  Running matrix_max on an 2048 x 2048 matrix
1104  Local Work Size = 256
1105  Number of CPU workers = 4
1106  ITER     GPU(EX.BUF)            GPU(INC.BUF)           CPU
1107  0        0.107878s              0.114359s              0.006965s
1108  1        0.107007s              0.112568s              0.004204s
1109  2        0.109405s              0.118287s              0.004159s
1110  3        0.103465s              0.11491s               0.004184s
1111  4        0.10545s               0.116487s              0.004216s
1112  5        0.101241s              0.123807s              0.00422s
```

```
1113  6          0.102558 s                   0.112428 s                    0.004209 s
1114  7          0.105321 s                   0.120157 s                    0.004217 s
1115  8          0.103111 s                   0.114123 s                    0.004181 s
1116  9          0.102518 s                   0.117968 s                    0.004182 s
1117  ——————————————————————————————————
1118  WHAT                                    TIME(TOT)          TIME(AVG)
1119  GPU time ( Transfer time excluded )     1.04795 s          0.104795 s
1120  GPU time ( Transfer time included )     1.16509 s          0.116509 s
1121  CPU time                                0.044737 s         0.0044737 s
1122  Running matrix_max on an 4096 x 4096 matrix
1123  Local Work Size = 4
1124  Number of CPU workers = 4
1125  ITER     GPU(EX.BUF)                GPU(INC.BUF)               CPU
1126  0          0.131658 s                   0.179302 s                    0.019246 s
1127  1          0.131686 s                   0.17934 s                     0.016279 s
1128  2          0.133496 s                   0.172332 s                    0.016338 s
1129  3          0.131699 s                   0.168124 s                    0.016302 s
1130  4          0.131565 s                   0.169192 s                    0.017017 s
1131  5          0.131663 s                   0.179068 s                    0.016488 s
1132  6          0.133576 s                   0.176722 s                    0.016252 s
1133  7          0.133376 s                   0.176414 s                    0.016498 s
1134  8          0.133366 s                   0.166915 s                    0.01624 s
1135  9          0.13288 s                    0.174505 s                    0.016471 s
1136  ——————————————————————————————————
1137  WHAT                                    TIME(TOT)          TIME(AVG)
1138  GPU time ( Transfer time excluded )     1.32497 s          0.132497 s
1139  GPU time ( Transfer time included )     1.74191 s          0.174191 s
1140  CPU time                                0.167131 s         0.0167131 s
1141  Running matrix_max on an 4096 x 4096 matrix
1142  Local Work Size = 8
1143  Number of CPU workers = 4
1144  ITER     GPU(EX.BUF)                GPU(INC.BUF)               CPU
1145  0          0.076053 s                   0.115518 s                    0.019122 s
1146  1          0.077227 s                   0.118312 s                    0.01623 s
1147  2          0.075462 s                   0.118964 s                    0.016383 s
1148  3          0.077935 s                   0.120758 s                    0.016228 s
1149  4          0.086644 s                   0.115309 s                    0.016245 s
1150  5          0.080869 s                   0.118394 s                    0.016238 s
1151  6          0.076934 s                   0.117439 s                    0.016219 s
1152  7          0.075285 s                   0.116531 s                    0.016251 s
1153  8          0.074601 s                   0.113045 s                    0.016265 s
1154  9          0.075773 s                   0.117732 s                    0.016238 s
1155  ——————————————————————————————————
1156  WHAT                                    TIME(TOT)          TIME(AVG)
1157  GPU time ( Transfer time excluded )     0.776783 s         0.0776783 s
1158  GPU time ( Transfer time included )     1.172 s 0.1172 s
1159  CPU time                                0.165419 s         0.0165419 s
1160  Running matrix_max on an 4096 x 4096 matrix
1161  Local Work Size = 16
1162  Number of CPU workers = 4
1163  ITER     GPU(EX.BUF)                GPU(INC.BUF)               CPU
1164  0          0.057033 s                   0.114978 s                    0.019586 s
1165  1          0.061941 s                   0.09736 s                     0.01624 s
1166  2          0.05934 s                    0.10736 s                     0.016233 s
1167  3          0.066152 s                   0.10568 s                     0.016265 s
1168  4          0.067301 s                   0.103986 s                    0.016206 s
1169  5          0.081248 s                   0.10584 s                     0.016221 s
1170  6          0.071491 s                   0.112342 s                    0.016224 s
1171  7          0.070696 s                   0.104142 s                    0.016285 s
1172  8          0.066575 s                   0.098806 s                    0.016206 s
1173  9          0.069086 s                   0.101545 s                    0.016407 s
1174  ——————————————————————————————————
1175  WHAT                                    TIME(TOT)          TIME(AVG)
1176  GPU time ( Transfer time excluded )     0.670863 s         0.0670863 s
1177  GPU time ( Transfer time included )     1.05204 s          0.105204 s
1178  CPU time                                0.165873 s         0.0165873 s
1179  Running matrix_max on an 4096 x 4096 matrix
1180  Local Work Size = 32
1181  Number of CPU workers = 4
1182  ITER     GPU(EX.BUF)                GPU(INC.BUF)               CPU
1183  0          0.137356 s                   0.161587 s                    0.019011 s
```

```
1184  1        0.130635s              0.192045s              0.016236s
1185  2        0.145129s              0.17879s               0.016311s
1186  3        0.114893s              0.169601s              0.0163s
1187  4        0.153264s              0.169223s              0.016209s
1188  5        0.151884s              0.184035s              0.016241s
1189  6        0.144728s              0.211291s              0.017821s
1190  7        0.116898s              0.185365s              0.016305s
1191  8        0.115248s              0.189688s              0.016274s
1192  9        0.15323s               0.16913s               0.016325s
1193  ——————————————————————————————
1194  WHAT                           TIME(TOT)    TIME(AVG)
1195  GPU time (Transfer time excluded)    1.36327s     0.136327s
1196  GPU time (Transfer time included)    1.81076s     0.181076s
1197  CPU time                       0.167033s    0.0167033s
1198  Running matrix_max on an 4096 x 4096 matrix
1199  Local Work Size = 64
1200  Number of CPU workers = 4
1201  ITER   GPU(EX.BUF)            GPU(INC.BUF)           CPU
1202  0        0.237582s              0.271394s              0.019043s
1203  1        0.248587s              0.341792s              0.016213s
1204  2        0.317244s              0.357047s              0.016227s
1205  3        0.304983s              0.361604s              0.016221s
1206  4        0.332338s              0.278672s              0.016305s
1207  5        0.305837s              0.344257s              0.016466s
1208  6        0.33021s               0.355925s              0.017171s
1209  7        0.318338s              0.348765s              0.017051s
1210  8        0.331313s              0.35637s               0.017363s
1211  9        0.33192s               0.335312s              0.017246s
1212  ——————————————————————————————
1213  WHAT                           TIME(TOT)    TIME(AVG)
1214  GPU time (Transfer time excluded)    3.05835s     0.305835s
1215  GPU time (Transfer time included)    3.35114s     0.335114s
1216  CPU time                       0.169306s    0.0169306s
1217  Running matrix_max on an 4096 x 4096 matrix
1218  Local Work Size = 128
1219  Number of CPU workers = 4
1220  ITER   GPU(EX.BUF)            GPU(INC.BUF)           CPU
1221  0        0.331774s              0.32765s               0.019931s
1222  1        0.285925s              0.394738s              0.018028s
1223  2        0.279375s              0.395744s              0.017022s
1224  3        0.33099s               0.397785s              0.016458s
1225  4        0.281085s              0.404075s              0.016573s
1226  5        0.296238s              0.33581s               0.016262s
1227  6        0.331469s              0.392683s              0.017332s
1228  7        0.279669s              0.402449s              0.016792s
1229  8        0.281285s              0.398493s              0.016691s
1230  9        0.328756s              0.39021s               0.016373s
1231  ——————————————————————————————
1232  WHAT                           TIME(TOT)    TIME(AVG)
1233  GPU time (Transfer time excluded)    3.02657s     0.302657s
1234  GPU time (Transfer time included)    3.83964s     0.383964s
1235  CPU time                       0.171462s    0.0171462s
1236  Running matrix_max on an 4096 x 4096 matrix
1237  Local Work Size = 256
1238  Number of CPU workers = 4
1239  ITER   GPU(EX.BUF)            GPU(INC.BUF)           CPU
1240  0        0.153446s              0.187273s              0.019068s
1241  1        0.147271s              0.188242s              0.016242s
1242  2        0.143291s              0.19893s               0.01631s
1243  3        0.155284s              0.188041s              0.016246s
1244  4        0.148455s              0.191923s              0.016182s
1245  5        0.146254s              0.194932s              0.016243s
1246  6        0.149059s              0.190919s              0.02001s
1247  7        0.147974s              0.191543s              0.016247s
1248  8        0.154037s              0.182802s              0.016589s
1249  9        0.153081s              0.197119s              0.016233s
1250  ——————————————————————————————
1251  WHAT                           TIME(TOT)    TIME(AVG)
1252  GPU time (Transfer time excluded)    1.49815s     0.149815s
1253  GPU time (Transfer time included)    1.91172s     0.191172s
1254  CPU time                       0.16937s     0.016937s
```

```
Running matrix_max on an 8192 x 8192 matrix
Local Work Size = 4
Number of CPU workers = 4
ITER    GPU(EX.BUF)         GPU(INC.BUF)             CPU
0       0.486677s           0.481426s               0.067399s
1       0.478481s           0.486936s               0.064348s
2       0.478843s           0.475862s               0.06434s
3       0.47912s            0.476938s               0.06448s
4       0.477748s           0.475909s               0.06429s
5       0.474912s           0.478002s               0.064853s
6       0.474519s           0.477661s               0.064267s
7       0.477599s           0.475624s               0.064315s
8       0.475162s           0.476265s               0.064265s
9       0.475774s           0.475413s               0.064363s
—————————————————————————————————————————————————
WHAT                            TIME(TOT)       TIME(AVG)
GPU time (Transfer time excluded)    4.77883s    0.477883s
GPU time (Transfer time included)    4.78004s    0.478004s
CPU time                             0.64692s    0.064692s
Running matrix_max on an 8192 x 8192 matrix
Local Work Size = 8
Number of CPU workers = 4
ITER    GPU(EX.BUF)         GPU(INC.BUF)             CPU
0       0.245722s           0.244436s               0.067372s
1       0.245343s           0.2448s       0.065246s
2       0.245571s           0.245364s               0.064321s
3       0.245069s           0.244648s               0.066329s
4       0.24552s            0.245168s               0.06469s
5       0.243336s           0.244409s               0.064297s
6       0.245751s           0.246419s               0.064986s
7       0.254729s           0.243795s               0.064239s
8       0.245093s           0.244538s               0.06425s
9       0.244242s           0.243915s               0.065069s
—————————————————————————————————————————————————
WHAT                            TIME(TOT)       TIME(AVG)
GPU time (Transfer time excluded)    2.46038s    0.246038s
GPU time (Transfer time included)    2.44749s    0.244749s
CPU time                             0.650799s   0.0650799s
Running matrix_max on an 8192 x 8192 matrix
Local Work Size = 16
Number of CPU workers = 4
ITER    GPU(EX.BUF)         GPU(INC.BUF)             CPU
0       0.138015s           0.138181s               0.067772s
1       0.137435s           0.138433s               0.064527s
2       0.13731s            0.138422s               0.064354s
3       0.137315s           0.138138s               0.064329s
4       0.137493s           0.137672s               0.06508s
5       0.137433s           0.138425s               0.066035s
6       0.136428s           0.137979s               0.064333s
7       0.137609s           0.137346s               0.064326s
8       0.138453s           0.137923s               0.064256s
9       0.137136s           0.136742s               0.064212s
—————————————————————————————————————————————————
WHAT                            TIME(TOT)       TIME(AVG)
GPU time (Transfer time excluded)    1.37463s    0.137463s
GPU time (Transfer time included)    1.37926s    0.137926s
CPU time                             0.649224s   0.0649224s
Running matrix_max on an 8192 x 8192 matrix
Local Work Size = 32
Number of CPU workers = 4
ITER    GPU(EX.BUF)         GPU(INC.BUF)             CPU
0       0.10669s            0.115407s               0.067092s
1       0.108893s           0.115574s               0.064317s
2       0.115982s           0.120901s               0.06571s
3       0.115868s           0.116144s               0.064247s
4       0.115581s           0.114323s               0.066162s
5       0.115336s           0.11988s                0.064228s
6       0.115913s           0.115885s               0.064254s
7       0.114442s           0.108985s               0.064281s
8       0.106479s           0.121078s               0.06445s
9       0.105123s           0.115441s               0.064192s
```

```
1326  ————————————————————————————————
1327  WHAT                                      TIME(TOT)        TIME(AVG)
1328  GPU time ( Transfer time excluded )       1.12031 s        0.112031 s
1329  GPU time ( Transfer time included )       1.16362 s        0.116362 s
1330  CPU time                                  0.648933 s       0.0648933 s
1331  Running matrix_max on an 8192 x 8192 matrix
1332  Local Work Size = 64
1333  Number of CPU workers = 4
1334  ITER    GPU(EX. BUF)             GPU(INC. BUF)           CPU
1335  0       0.084819 s               0.084835 s              0.067364 s
1336  1       0.084627 s               0.086459 s              0.064226 s
1337  2       0.086188 s               0.086259 s              0.064337 s
1338  3       0.084569 s               0.085386 s              0.06468 s
1339  4       0.08664 s                0.087318 s              0.064252 s
1340  5       0.083892 s               0.085589 s              0.064227 s
1341  6       0.084584 s               0.084735 s              0.064354 s
1342  7       0.085227 s               0.083969 s              0.064289 s
1343  8       0.086087 s               0.085147 s              0.064296 s
1344  9       0.086052 s               0.085495 s              0.064145 s
1345  ————————————————————————————————
1346  WHAT                                      TIME(TOT)        TIME(AVG)
1347  GPU time ( Transfer time excluded )       0.852685 s       0.0852685 s
1348  GPU time ( Transfer time included )       0.855192 s       0.0855192 s
1349  CPU time                                  0.64617 s        0.064617 s
1350  Running matrix_max on an 8192 x 8192 matrix
1351  Local Work Size = 128
1352  Number of CPU workers = 4
1353  ITER    GPU(EX. BUF)             GPU(INC. BUF)           CPU
1354  0       0.106948 s               0.109526 s              0.067848 s
1355  1       0.107187 s               0.113418 s              0.064364 s
1356  2       0.105224 s               0.105687 s              0.064626 s
1357  3       0.123325 s               0.112672 s              0.064766 s
1358  4       0.117601 s               0.105831 s              0.064882 s
1359  5       0.092825 s               0.109842 s              0.064635 s
1360  6       0.117525 s               0.111299 s              0.064337 s
1361  7       0.105265 s               0.106353 s              0.064383 s
1362  8       0.113094 s               0.115145 s              0.06435 s
1363  9       0.108792 s               0.107833 s              0.064297 s
1364  ————————————————————————————————
1365  WHAT                                      TIME(TOT)        TIME(AVG)
1366  GPU time ( Transfer time excluded )       1.09779 s        0.109779 s
1367  GPU time ( Transfer time included )       1.09761 s        0.109761 s
1368  CPU time                                  0.648488 s       0.0648488 s
1369  Running matrix_max on an 8192 x 8192 matrix
1370  Local Work Size = 256
1371  Number of CPU workers = 4
1372  ITER    GPU(EX. BUF)             GPU(INC. BUF)           CPU
1373  0       0.114115 s               0.106475 s              0.068122 s
1374  1       0.076723 s               0.118151 s              0.064366 s
1375  2       0.112423 s               0.108933 s              0.065309 s
1376  3       0.111586 s               0.099482 s              0.064281 s
1377  4       0.114877 s               0.111822 s              0.064221 s
1378  5       0.108627 s               0.112013 s              0.064338 s
1379  6       0.112327 s               0.113743 s              0.064301 s
1380  7       0.105834 s               0.113864 s              0.064213 s
1381  8       0.110282 s               0.090779 s              0.064328 s
1382  9       0.107973 s               0.110524 s              0.064308 s
1383  ————————————————————————————————
1384  WHAT                                      TIME(TOT)        TIME(AVG)
1385  GPU time ( Transfer time excluded )       1.07477 s        0.107477 s
1386  GPU time ( Transfer time included )       1.08579 s        0.108579 s
1387  CPU time                                  0.647787 s       0.0647787 s
```

54