# TDTS08: Lab Report

Multiprocessing and Multi-Computer Systems


| **Name** | **PIN** | **Email** |
|---|---|---|
| Alexander Yngve | 930320-6651 | aleyn573@student.liu.se |
| Pål Kastman | 851212-7575 | palka285@student.liu.se |

# Contents

# 1 Introduction

The article we have choosen is *Numerical Parallel Processing Based on GPU with CUDA Architecture* written by *Chengming Zou, Chunfen Xia, Guanghui Zhao* at the *College of Computer Science and Technology Wuhan University of Technology Wuhan, China*, which is number 15 in the list of articles.

We choose this article because we didn't have a lab about multicore processors and GPUs and we wanted to explore this area further, but also because we think this area is very interesting and that we will se a lot of development in this area over the next few years.

# 2 Overview

The article compares CPUs and GPUs, and looks at how a GPU can be used in high density parallel computing because it has lots of simple processors which can operate independently and concurrently at high speeds.

The article is about the differences between CPUs and GPUs and how GPUs can be used for calculations in a faster way than in the CPU. The major advantages with GPUs is that is it able to compute a lot of calculations in parallel independent of each other, this means that the CPU is able to only handle branches and control instructions that needs to be controlled allowing the hardware of GPUs to be simplified a lot because their controlling unit can be smaller. However, it should be noted that it needs to be made sure that the data sent to the GPU contains as few branches and control instructions as possible leaving them to the CPU.

The main area of applicability is scientific computing, financial risk modeling and image processing.

# 3 Architecture

The architecture presented in the paper is NVIDIAs CUDA (Compute Unified Device Architecture) architecture, used in NVIDIA GPUs since the G80 which was released in 2006. The CUDA architecture consists of both hardware (graphics card) and software (compiler, libraries and drivers).

The main control flow of the program is executed by the CPU and the GPU is used as a coprocessor. This way the low latency CPU executes the serial instructions and deals with branching and control and the GPU is used for heavy parallel computations. The CPU copies instructions and data from main memory into memory on the graphics card and then starts execution on the GPU. When the computations on the GPU are finished the result is written back into main memory.

The GPU is structured into many smaller *Streaming Multiprocessors* which in turn consists of lots of computational cores (*CUDA cores*). One thread runs on each CUDA core, however only copies of the same thread may run simultaneously, making CUDA a SIMD architecture. A block of threads is run on each Streaming Multiprocessor. A block of blocks, called a grid or kernel, is run on the whole CUDA device.

The division into Streaming Multiprocessors (SMs) and CUDA cores is done to ensure scalability and also to provide a way for the threads to share data between each other. There are three levels in the memory hierarchy. Each CUDA core has access to its registers and a small amount of local memory. Each Streaming Multiprocessor, and therefore all its

CUDA cores, has access to shared memory, enabling data sharing between the threads within the SM. All SMs also have access to global memory which enables data sharing between all threads in the kernel.

# 4 Experiments

Two experiments are performed which correspond to two use cases, scientific computing and image processing/computer vision. The first experiment is to calculate the inverse of a matrix, a very common mathematical operation. The second experiment is to perform binarization of a greyscale image, which means that each pixel is set to either black or white depending on some threshold value.

The results clearly show that parallel computation is faster on GPUs, by at least an order of magnitude. It was also found that the performance increase of the GPU over the CPU increases as the problem size increases, further strengthening the authors argument.

# 5 Review

We would like to have a better explanation of how the code is run on the GPU, as we think that this isn't sufficiently explained.

We also believe that a higher level of linguistics could be desired from a report written by a team of researchers. Not only should it have been proof-read, there is also other issues on the matter such as no separation between mathematical formulas and no italic format on the formulas. This is also a issue with the code parts of the report as it is very hard to distinguish what is code and what is text sometimes. We think that the code parts could, and probably should have been placed in an appendix which could have been cited.

# 6 Conclusion

The advantages of the CUDA architecture is the very large performance increase for parallel computations. There is simply no way for a normal CPU to match the throughput of the GPU when dealing with lots of similar calculations.

The disadvantages to CUDA is that it leads to more complicated code, since it introduces another large API into the application codebase and also relies heavily on lots of memory copying between main memory and graphics memory. The build process is also more complicated since CUDA uses its own compiler.

However the largest disadvantage is that CUDA is proprietary and only works with GPUs from NVIDIA. Since the time of the article was published OpenCL has been released, which is an open standard and toolkit similar to CUDA which works on GPUs from many vendors.

We believe the advantages outweigh the disadvantages and that the GPGPU trend will continue.