HUNGARIAN JOURNAL
OF INDUSTRIAL CHEMISTRY
VESZPRÉM
Vol. 39(2) pp. 215-218 (2011)

USING GRAPHICAL PROCESSING UNITS IN SCHEDULING PROBLEMS

K. MIHÁLY✉, O. HORNYÁK

University of Miskolc, Department of Information Engineering, 3535 Miskolc - Egyetemváros, HUNGARY
✉E-mail: krisztian.mihaly@sap.com

Scheduling problems exist everywhere in the so-called "real world". They are there in manufacturing, transportation and logistics as well. The main object of these problems is to find an optimal sequence of tasks to be able to fulfil predefined objectives. There are efficient methods to solve complex scheduling problems in science and industry, which methods can be divided into several classes, like heuristic algorithms, genetic algorithms, etc. Even if these methods allow reducing significantly the computational time of the solution search space exploration, this latter cost remains exorbitant when very large problem instances are to be solved. Some of these methods are not designed for parallel computing; they are using a CPU as an arithmetical unit. From this point of view the bottleneck is the number of processed commands. Meanwhile the capabilities of specialized Graphical Processing Units (GPUs) have been extremely increased and they can provide an efficient platform for developing graphical algorithms. Nowadays there are new programming languages and platforms, where these GPUs can be used for more generic problems, using its hardwired parallel processing resources. Our goal is to use this specialized graphical platform for solving scheduling problems. This paper is an initial research of the existing platforms and solutions in general and describes the existing solutions in fact of scheduling problems.

Keywords: GPU; CUDA; Scheduling

Introduction

This paper goes to give a short overview about the history of the age of parallel processing on graphical processing units and the available hardware and development platforms. This paper investigates the platform of main architecture and the feasibility of algorithms in general for parallel processing and special in area of scheduling problems. Finally the future researching areas will be outlined and some conclusions of the justification of this new approach will be given.

Short history of Graphical Processing Units (GPU)

Before the review of GPUperformance and computing capacity we have to point out the improvement problems with the currently used Central Processing Units (CPU). Over the last decades the main interest for improving the performance of a personal computer device was to increase the speed at the processor's clock operated. The best example is that the early 1980s, the consumer CPU ran with internal clock operating near 1 MHz. In 2010 most desktop processors have clock speeds between 1 GHz and 4 GHz. It means ~1000 times better solution in the same chip. So what is the problem?! This method hits the power and heat restrictions as a rapidly approaching physical limit of the transistor size. It is no longer feasible to rely on upward-spiralling processor clock speeds. The hardware manufactures changed their approach of creation of new CPUs and in the year 2005 they started to ship multicore central processors. This process is sometimes referred as multicore revolution.

"In the meantime the state of graphics processing underwent a dramatic revolution. In the late 1980s and 1990s, the growth in popularity of graphically driven operating systems such as Microsoft Windows helped create a market for a new type of processors. In the early 1990s, users began purchasing 2D display accelerators for their personal computer. These display accelerators offered hardware-assisted bitmap operations to assist in the display and usability of graphical operating systems" [1]. Around the same time another approach was developed for three-dimensional graphics. In 1992, the Silicon Graphics opened the programming interface to its hardware by releasing the OpenGL library. It was a standardized, platform-independent method for developing 3D applications. In this time the computing of rendering was running in the CPU. Because the success of first-person shooting games, as Doom, Duke Nukem 3D and Quake the market has created hardware support to create better, more-realistic applications with 3D display. The main companies were the NVIDIA, ATI Technologies and 3dfx Interactive.

First the graphical hardware took care of transform and lighting computations, so these operations could run in the hard-wired graphical processors and the CPU could use for other tasks. The next breakthrough in parallel-computing point of view was the first graphic card, which supported the Microsoft's DirectX 8.0 standard. This standard introduced the programmable vertex

and pixel shaders. This was the first point, when the developers were able to influence the control of GPUs program.

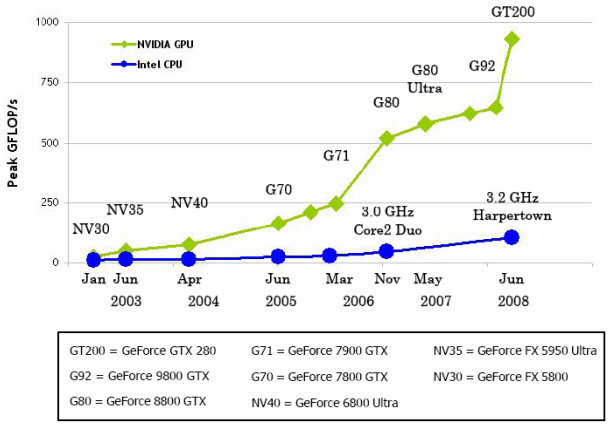

Figure 1: The difference between the CPU and GPU in measurement of GFLOP/s [2]

Early GPU programming; pain-points

The DirectX APIs was designed to use GPU to execute special graphical tasks. They were designed to get some data from "input colors" with some additional "color" or "texture" information. The programmer could use this information during the computation. The "color" can mean any number. A clever, but simple trick was to use these containers to put data into the GPU, do some calculations and get back the data.

It is quite simple to see, the pain points of this approach. The APIs were designed to support graphical APIs and the programmers needed very deep knowledge from the graphical platform. There were resource constraints; data could be loaded as picture and texture and retrieved data was another picture. So if the algorithm required accessing a memory area to write, it cannot run on GPU. It was nearly impossible to predict how your particular GPU can deal with floating-point data. They didn't exist any good method to debug implementations in GPU.

New GPU programming approach;OpenCL; CUDA and Stream as platforms

Graphical hardware manufacturer companies came out a new generation of GPUs, where the partitioned resources as vertex and pixel shaders where changed and a unified shader pipeline has been introduced. This new generation allows each and every arithmetic logical units (ALU) to be controlled by program intending to perform general-purpose computations. Usually, like in case of NVIDIA's solution, these ALUs were built to comply with IEEE requirements. Near IEEE these ALUs are able to read and write memory as well in software-managed ways. Unlike, each main supplier introduced their own architecture or standard. The NVIDIA has the "Computing Unified Device Architecture" (CUDA), the ATI has the Stream. Near the industry standards there are open standards as well, the main one is the OpenCL standard [3].

CUDA architecture

The CUDA architecture has two main parts, one is the hardware which supports the CUDA programming and it can be called as device and the programming language to be able create programs using the device capacity. The programming language is based on industry standard C and adds relatively small number of keywords in order to harness some of the special features of CUDA architecture. There is a public compiler for this language, the CUDA C. We are going to give a short overview about the architecture, but further information you can refer the CUDA Programming Guide by NVidia [4].

As mentioned before CUDA is an extension of C language and hides GPU relevant APIs and interfaces. There are 3 main tasks, which is the task of the developers, as thread hierarchy, memory access and synchronisation [5].

Thread hierarchy

To perform computation with CUDA, programmers have to define a special C function, the so-called kernel. This is function can be run in a thread using a specified number of lightweight threads in GPU. The kernel is loaded to the device from the host, where the "normal" code is running. Threads are grouped into blocks, and blocks forms a so-called grid. Threads can communicate through the memory area assigned to the block. The blocks are running independently and their behaviour cannot be affected by the programmer.

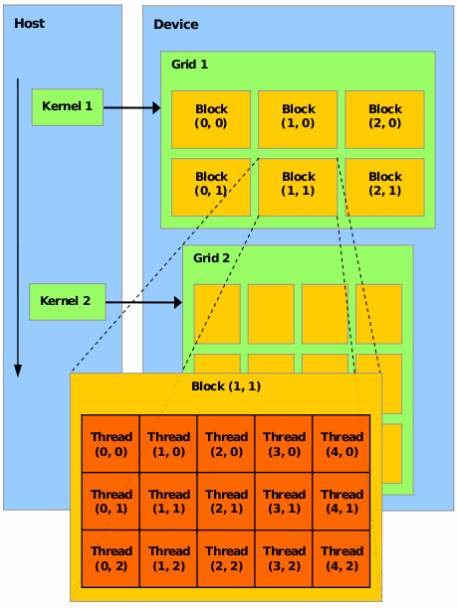

Figure 2: CUDA Thread hiearchy [6]

Memory hierarchy

There are a lot of types of memory areas, as Global memory, Shared memory, Constant memory, Registers and Local memory. The differences between the memory types are the size, the accessing time, the caching property. You can read more details in [4] or in [5].

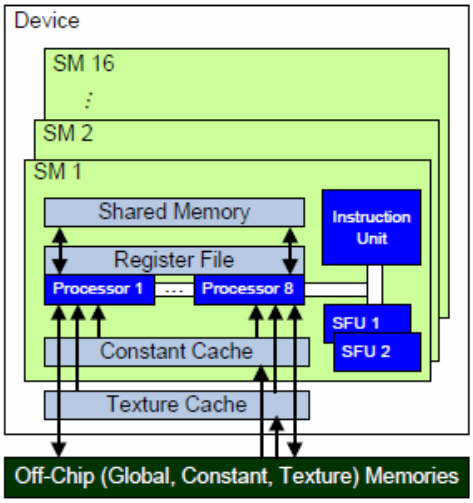

Figure 3: Basic Organization of the GeForce 8800 [7]

Synchronisation

It could be needed to force a central point in the kernel, where the threads within the same block are waiting for each other. For example when the shared memory is used by the threads, it could be necessary to have a consistent state in case after writing of a thread. The CUDA language provides mechanism, a __synctrheads( ) method, which defines a waiting point in the kernel code. Each thread executes his code and stops the processing, while other treads are executed this statement as well.

Show-cases of CUDA Applications

CUDA has many areas, where it could be used, from the astrophysics or physical modelling of fluids, through medical imaging or representing mathematical sets. It cannot be our goal to describe each one; you can read more in [8]. As an example one show-case is described here, from the medical imaging [1].

There are a lot of people, who have been affected by breast cancer in the past 20 years. Ultimately, every case of breast cancer should be caught early to prevent the ravaging side effects of chemotherapy and radiation. To achieve this, an effective, fast and minimally invasive way has to be developed. There is the mammogram for early detection, but it has limitations. During the process two or more image need to be taken and the images have to be analysed by skilled doctors. There is another technique, the ultrasound imaging, which is safer, then X-ray image made by mammogram, but it is not used in practice because of the computation limitations. With the GPU capacity this limitations are eliminated. In practice 35 GB of data can be generated by a scan. With Tesla C1060 a doctor can manipulate a highly detailed, three-dimensional image of the woman's breast within 20 minutes.

Scheduling algorithms in general

Generally, the purpose of scheduling is to find an optimal processing order of tasks (also called activities, operations or jobs) on a set of processors (machines) subject to several constraints imposed on the tasks, machines and their mutual relationships. In the literature there are a lot of types of scheduling models, like shop scheduling model, real-time scheduling, monoprocessor/dedicated processor/parallel process models, cyclic scheduling, etc. [9, 10, 11]

Show-cases of CUDA Applications and a scheduling algorithm

Parallel computing is considered as it is capable providing solutions for various complex optimization problems. [12] solves the traveling sales man problem by means of GPU. They also used generic operators, and tabu list. In a hybrid CPU-GPU environment the memory handling is also important.

[13] applied a two level metaheuristic to solve the flexible job shop problem. The machine selection layer and the operation scheduling module are executed parallel. They implemented a code in C language (CUDA) for GPU. Also a tabu list was used. [14] suggested the scheduling problem to solve in aheterogeneousCPU-GPU environment. The CPU and GPU behave differently but both can execute the same task. So they implemented an appropriate load balancing algorithm between CPU and GPU. They achieved 30–40% speedup.

[14] investigates a flowshop problem to solve by GPU computing. They focused on an efficient memory mapping model in order to implement a multiobjective local searchtask on GPU. They described their speedup as promising.

[15] describes two approaches to parallel flowshop evaluation on CUDA platform. According to their measure the GPU based algorithmis faster up to 5%.

Conclusion

As the result of our first project task we get overview about the existing solutions and the main power of parallel computing on the GPU device. The show-cases have a very wide focus, from the medical to the abstract mathematic world. In this huge area the scheduling in manufacturing systems is a little stone. After the literature research we are going to implement the existing solutions in our new hardware, when it will be purchased and installed. We are going to do speed up implementations

in direction of floor-shop and job-shop models when the purchasing process of the new hardware will be finished.

ACKNOWLEDGEMENTS

This research was carried out as part of the TAMOP-4.2.1.B-10/2/KONV-2010-0001 project with support by the European Union, co-financed by the European Social Fund.

REFERENCES

1. J. SANDERS, E. KANDROT: CUDA by example, Addison-Wesley,(2010), p. 3
2. J. MASOOD: Nvidia Cuda http://www.hardwareinsight.com/nvidia-cuda/ (2011. 08. 25)
3. OpenCL standard, KronosGroup: http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf (2011. 08. 25)
4. NVidia Programming Guide 4.0, NVidia Developer Zone: http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf (2011. 08. 25)
5. M. CZAPISNKY, S. BARNES: Tabu Search with two approaches to parallel flowshop evaluation on CUDA platform, J. Parallel Distrib. Comput. 71(6), (2011), 802–811
6. http://mohamedfahmed.wordpress.com/2010/05/03/cuda-computer-unified-device-architecture/ (2011. 08. 25)
7. S. RYOO, C. I. RODRIGES, S. S. BAGHSORKHI, S. S. STONE: Optimization Princiles and Application Performance Evaulation of Multithread GPU Using CUDA
8. CUDA Showcases, NVidia webpage: http://www.nvidia.com/object/cuda_showcase_stage.html
9. J. BLAZEVICZ, K. H. ECKER, E. PESCH, G. SCHMIDT, J. WEGLARZ: Scheduling Computer and Manufacturing Processes, Springer, (2001), ISBN 3-540-41931-4
10. P. BRUCKNER, S. KNUST: Complex Scheduling, Springer, (2006), ISBN 978-3-540-29545-7
11. B. CHEN, C. N. POTTS, G. J. WOEGINER: A review of machine scheduling: Complexity, algorithms and approximability, Handbook of Combinatorial Optimization (Volume 3), (1998), 21–169.
12. J. ZHAO. Q. LIU, W. WANG, Z. WEI, P. SHI: A parallel immune algorithm for traveling salesman problem and its application on cold rolling scheduling. Information Scienced 181, (2011), 1212–1223
13. W. BOZEJKO, M. UCHRONSKI, M. WODECKI: Parallel hybrid metaheuristics for the flexible job shop problem. Computers and Industrial Engineering 59, (2010), 323–333
14. T. V LUONG, N. MELAB, E. G. TALBI: GPU-Based Approaches for Multiobjective Local Search Algorithms. A Case Study: The Flowshop Scheduling Problem. EvoCop 2011, 155–166
15. M. CZAPINSKI, S BARNES: Tabu Search with two approaches to parallel flowshop evaluation on CUDA platform. Journal of Parallel and Distributed Computing Archive, 71(6, June), 2011, 802–811
16. C. J. JIMENEZ, K. VILANOVA, I. GELADO, M. GIL: Predictive runtime code scheduling for heterogeneous architectures. HiPEAC 2009, 19–33