TDTS08 LABRATORY ASSIGNMENT 5 - MULTIPROCESSOR AND MULTI-COMPUTER SYSTEMS

M.E. Söderén[*], LiU, Linköping, Sweden,

Abstract

This report will review the computational model described in [1] which has number 25 in the list of papers available. The system in question is a GPU (*Graphics Processing Unit*) combined with a CPU (*Central Processing Unit*). The article presents a new model to balance the load between the GPU and the CPU in game engines, the model is called *multi-thread uncoupled with GPGPU (General-Purpose computing on Graphics Processing Units)*. This article is interesting and was chosen because it is old (released 2007) and the field of GPGPU has expanded quite a lot since then and become more mainstream.

OVERVIEW OF PAPER

The system is a GPU combined with a CPU, so it is a SIMD/MIMD system. They focus on how some computations cannot be directly ported from a CPU to a GPU since the GPU is best suited for vector operations. It explains relevant research regarding doing rigid/elastic body simulations, collision detection, FEM (*Finite Element Method*) and data mining can be carried out on GPUs using different available libraries such as Havok FX. It discusses that none of the available libraries performers load-balancing between the CPU and the GPU which [1] wants to achieve. It achieves this by implementing a game loop with two threads, one that is running on the CPU and the other one that handles the GPGPU. By being able to dispatch events to both of the threads, the solution can achieve load balancing.

CLASSIFICATION OF THE PRESENTED ARCHITECTURE

When a new event is created, either from user input or by the game engine, a new task is created which must be computed. This could be collision detect for example, and the main game loop adds this task to a task manager which assigns this task to either the CPU or the GPU. To control the load-balancing and making sure that the task can run on the specific hardware, a Lua script is used when describes which tasks can run on which architecture and the priority different task has.

ADVANTAGES AND DISADVANTAGES WITH THIS ARCHITECTURE

There are clear advantages since the GPU is a powerful resource and SIMD problems fit the model very well and these problems arise in games frequently it makes sense to offload computation to the GPU. There are some disadvantages as well, as mentioned in the paper, not all problems can be executed on the GPU at all and some does not fit the model very well. Also if a certain task is scheduled to run both on the GPU and the CPU, the data and the task has to be transferred between the graphical memory and the main memory for optimal performance. The GPU can access the main memory but it will limit the performance compared to using the GPU memory. Another problem is mutual exclusion, assume that one task is running on the GPU and another one is running on the CPU and they must access the same memory, memory synchronization between GPU and CPU is not trivial and can add extensive overhead compared with if both tasks were running on the same unit.

IMPRESSION OF THE ARTICLE

The paper itself is interesting and discusses an important question regarding how to utilize the powerful GPUs available today. It does not however clearly describe how this should be done. It describes a simple load-balancer while not giving any information regarding implementation, testing or results. Some more information is required on how this is implemented, for example how do you write efficient code that can run on both the CPU and the GPU. This can be done by using OpenCL for example but it does not describe how it is done in the new architecture. Another problem with GPU/CPU load balancing is how to fully utilize the hardware specific performance options. A program must be specifically programmed and compiled for a certain hardware to run optimally on that platform. Previous research in this area [3] solves this problem with skeleton programming.

The subject that is discussed here is important but there is very lite information on how it should be implemented or if any test has been carried out and any potential performance increase. The only data that is presented is time of computation for rigid body collision detection which was carried out on CPU and GPU separately. The data shows an impressive speed gain when using GPUs and it is already known that GPUs are faster for certain problems. It would be interesting to have a framework in which it is possible to code normal C or C++ and the framework would take care of optimization for both architectures, figure out which computation is best suited for a specific architecture and also do the load-balancing.

EXPERIENCE FROM THE ARTICLE

The idea of load-balancing between the GPU and CPU is pretty straight-forward and also how they plan to achieve

[*] marso329@student.liu.se

this. It would be more interesting to read how they plan to overcome the problem with compatibility between GPU and CPU code. Their solution is a static load-balancer where the user has to define an initial schedule for the task manager and take in consideration task priority, which hardware and mutual exclusion which is a lot of work for the user and can probably be error-prone. Personally, the best way to fully utilize the hardware for a game engine would be to have dedicated tasks which run on the CPU and the same for the GPU. Those problems which can't run on a GPU should, of course, run on the CPU and those problems which are more suited for a SIMD architecture should run on a GPU. But if there were a framework that could achieve this transparently, it would be great. For a newer article which describes a dynamic load-balancer and more information on the actual implementation and test, see [2].

REFERENCES

[1] M. P. d. M. Zamith, E. W. G. Clua, A. Conci, A. Montenegro, P. A. Pagliosa and L. Valente, "Parallel processing between GPU and CPU: Concepts in a game architecture," Computer Graphics, Imaging and Visualisation (CGIV 2007), Bangkok, 2007, pp. 115-120.

[2] Dollinger, Jean-François and Loechner, Vincent, "CPU+GPU Load Balance Guided by Execution Time Prediction",Fifth International Workshop on Polyhedral Compilation Techniques (IMPACT 2015), Amsterdam, 2015

[3] J. Enmyren, 'A Skeleton Programming Library for Multicore CPU and Multi-GPU Systems', Dissertation, 2010.