USING GRAPHIC PROCESSORS FOR HIGHSPEED SIMULATIONS AND OTHER HIGH PERFORMANCE COMPUTATIONS

Thomas Wiedemann [(a)]

[(a)] University of Applied Science Dresden

[(a)]wiedem@informatik.htw-dresden.de

ABSTRACT

Current hardware development is characterized by an increasing number of multi-core processors. The performance advantages of dual and quad core processors have already been applied in high-speed calculations of video streams and other multimedia tasks. New options arise from the increasing power of new graphic processors. They include up to 1600 shading processors, which can also be used for universal computations at present. The paper discusses possible applications of graphic processors in simulation and other areas of high computation needs, like FEM or flow-analysis. The implementation of parallel threads on more than one core requires substantial changes in the software structure, which are only possible inside the source code. The paper also introduces feasible architectures and compares the CUDA and OpenCL approach.

Keywords: Grid computing, CUDA, OpenCL

1. INTRODUCTION

Since 2005, we have observed a quiet revolution in hardware development – the performance of graphic processor units (GPU) has been developing at a speed leading to a ten times higher performance against standard central processors (CPU) (see Fig. 1/from (Kirk and Hwu 2010). But the development of CPU´s has been slowing down since 2003 due to energy consumption and heat-dissipation issues limiting a further increase of clock frequency. As demonstrated in Fig. 1, the performance of actual graphical processing units (GPU´s) achieves 1000 GFlops, a value lying in the range of older super computers. It is quite sure, that this revolution will continue also in the next years as a result of a very strong competition between the two major players – AMD and NVIDIA.

As a matter of fact, simulation science has already searched for the highest feasible performance (Wiedemann 2000). Otherwise, there exist completely new hardware architectures and requirements (see 2.2). The imple¬mentation of parallel threads on a large number of cores leads to substantial changes in the software structure. Changes like these are only feasible inside the source code and cannot be executed with COTS-simulation systems.

As a conclusion, we may expect a new era of high performance software development. This paper introduces not only options and constraints of the new hardware, but also the changes in the simulation software resulting from

.

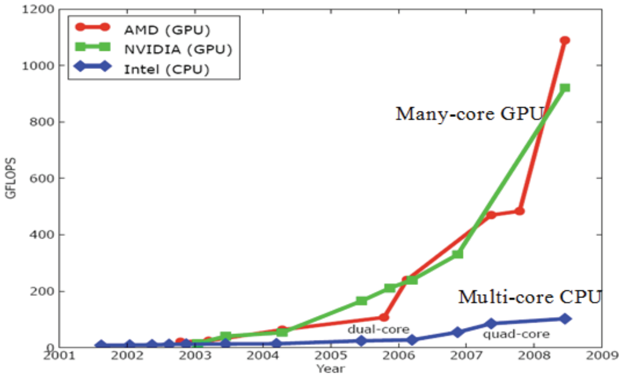

Figure 1: Performance chart ((Kirk and Hwu 2010))

The new hardware architecture of modern GPU´s was primarily designed for high-end 3D-computer games. In these games, the high quantity of processors is used for parallel computing of high-quality images with fine-grained textures and sophisticated rendering algorithms. The first versions of such GPU´s were tailored to special purposes and could not carry out universal computations (Kirk and Hwu 2010). The current versions are now capable of calculating common types of algorithms with double precision.

Resulting from the orientation on graphic algorithms, the hardware also follows a special architecture. First the host system and the graphic processor have separate memory and control areas. Programs for the graphic processor must be compiled in a special way and transferred to the graphic subsystem. The memory bandwidth of the GPU is up to ten-fold higher than the standard RAM memory of the host.

The GPU processor is divided into a number (16…128) of computing blocks, whereby each block consists of a grid of streaming core processors (cores). The number of cores inside a block is not fixed, but can be defined dynamically by the control program. High-end GPU processors, like the ATI Radeon™ HD 5870, are equipped with up to 1600 streaming cores.

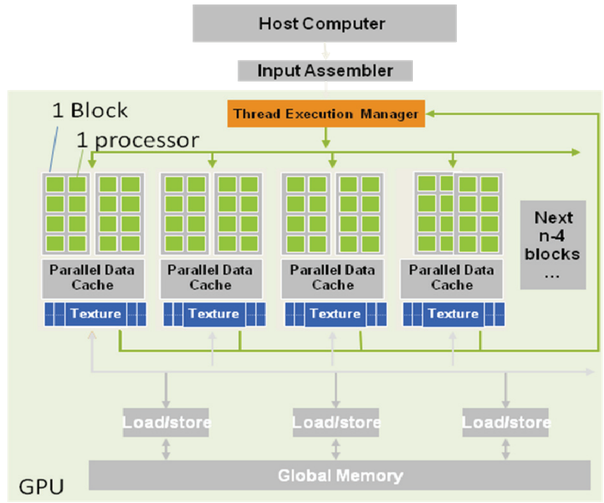

Figure 2: GPU architecture ((Kirk and Hwu 2010)

Memory organization is a significant limiting fact. The fastest memory is the shared memory inside the blocks. Only cores inside a block are able to communicate upon shared memory and to synchronize their work. Synchronization of cores between different blocks is slow and poorly supported! These aspects should be considered, when simulation scenarios are evaluated. A second, much more critical constraint is the single program multiple-data (SPMD) programming model of GPU´s. This means, that inside a block, only one program is executed over different areas of data. If a program like this includes a branch, then the alternative else-branch is performed after executing the then-branch, which slows down execution at all. For graphical applications with large data streams, like encoding of videos or rendering complex 3D scenes, this model is suitably adapted, since it reduces the necessary ratio of logics inside the small processors.

If a complex program is executed, this programming model must be considered carefully! An approach is demonstrated on the following pages.

2. GPU´S IN UNIVERSAL COMPUTATIONS

2.1. General discussion of multi core applications

The main algorithms and mathematical foundations of simulation systems are well defined and efficient (Heusmann and Wiedewitsch 1995). Although the software tools for continuous (CS) and discrete simulation (DS) are very different, there exist two general options for using parallel computing environments.

First, the model itself is divided in smaller sub-models and each sub-model is computed on one core. This Parallel Simulation approach has been known for about 25 years (Perumalla 2006). As a result of the necessary communication between the sub-models, this approach is very complex.

During the last decade the possible speedup degreed. The main reason is the nearly constant communication speed and increasing computation performance. The communication speed of standard parallel computers is limited by the simple pheno¬menon of distance between the computing cores. Let us assume a distance of 30 cm, only, than it takes the signal at light speed about 1ns (t=s/v=0.3m/3*10e-8 m/s), lying in the range of 3 periods of a 3 GHz processor. Additional delays occur by the electronics' latencies themselves. In summary, the resulting speedup of parallel computation can decrease (fall down?) to 2 or even below 1 on multi-core machines, when the models are not suitably distributed on the cores. However, the new hardware architecture of GPU´s may improve this situation again: First, by smaller distances of the core inside the chip die (<2mm) and second, by optimized synchronization hardware inside the same chip.

The second approach uses each core for computing exactly one simulation model, which is also known as Hyper Computing (Perumalla 2006). The larger number of cores is used for calculating the models n-time, e.g. by applying different random number seeds. Speeding up of such computations is nearly equal to the number of the cores and could be guaranteed in practice.

From a practice point of view, the Hyper computing approach is very interesting and to be used easily, if we leave the single simulation view and look on the whole simulation process. Nearly all larger simulation studies must consider random numbers inside the model or different input data scenarios. For statistical correctness, over 20 or more simulation runs must be executed for getting significant results. If there are different input data sets – Ndatasets, this number can be multiplied by the number of simulation runs Nruns, since all runs are independent one from each other and can be computed simultaneously. As a conclusion, running Ndatasets * Nruns , we obtain Ndatasets statis¬tical significant results over all data input sets.

If there are no different input scenarios, than Ndatasets runs could be used for making a sensitive test, which provides significant information about the quality of simulation variables used.

2.2. Performance considerations in Hyper Computing

Against the background of the new GPU architecture, it makes sense to subdivide calculations into two different classes. Typical graphical computations are also subdivided into different computation classes, like transforming, rendering and shading of 3D objects with textures. The resulting GPU architecture (see Figures 2 and 1.2) provides an adequate support to different computing groups.

Let us assume that the simulation models only differ in their specific random numbers, whereas input data and computation are always the same. This is true for a large number of continuous simulations, but only a small number of discrete event simulations.

Concludingly, a set of Ndatasets can be computed with Nruns each, whereby each block of Nruns computes one significant result for one of the datasets.

Since the maximum number of Ndatasets * Nruns is 512, a definition of Nrun =32 runs and Ndatasets = 16 data sets is a good practical combination.

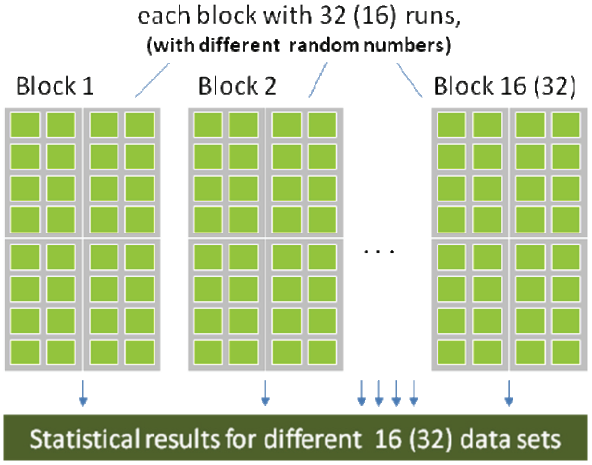
each block with 32 (16) runs,
(with different random numbers)
Block 1
Block 2
Block 16 (32)
. . .
Statistical results for different 16 (32) data sets

Figure 2: Hyper computing scenarios

If such a combination of variance and sensitive computation is realized, than the possible speedup can be the number of parallel running cores, in this case up to 512! This speedup does not depend on special methods of disaggregation of complex simulation models. The method can be adapted easily to new hardware characteristics, e.g. if the limit of 512 computing units in a block is extended (up to 1024 or more?) in the next years.

2.3. Typical scenarios for independent and equal simulation runs in a Hyper Computing context

In the field of **continuous simulation**, all formulae must be equal and only differ in the vector index of the input data and the index of the random number generator:

```
dv/dt = f( a(idx), vstart(idx), rand(idx) )
```

The value of the idx-variable is equal to the blockidx-value of the core inside the block. The blockidx is automatically determined by the host program at the start of the parallel runs and counts all the used cores from 1 to N.

In the example, each run may have different values for acceleration *a*, the initial speed *vstart* and the random values of the motion, like wind or engine characteristics. Like mentioned before in chapter 2.2, additional cores can be used for calculating sensitive tests or different data input sets.

A similar formula *may* be used for **Monte-Carlo-simulations** (MC) (e.g. for determining ∏) :

```
for (int i=1; i<= experiments; i++)
{  x = rand1(blockidx);  y = rand2(blockidx);
   r_testPI =  x * x  +  y * y;
   if (  fabs(r_testPI) <  Radius)      hit++;
}  PI = hit / experiments * 4; // get PI by MC
```

The rand1() and rand2() are typical random number generators, where the seed and current value are stored in a vector, referenced by the blockidx-value again. For both applications, speedup may grow up to the number of cores used in parallel.

2.4. Parallel execution in discrete simulation

Application of GPU´s in the discrete event simulation is much more difficult. In general, the objects are very different in their characteristics, and thus code execution is not equal, which, in turn, slows down execution speed in the context of the single program multiple-data (SPMD) programming model.

One special option is possible, if the simulation model consists of objects with nearly identical characteristics and an equal schedule sequence (e.g. each minute a customer is served or nothing is done)! Such a code for one object of some hundreds of objects could be described by the following expression:

```
while (  running )
{ if  ( mynext_time >  simtime)
     //  do nothing
     else  {    /* do  actions  */
      p= getnextproduct(blockidx);
     optime = workon_product(p);
     mynext_time = simtime +optime;
   }  _syncthreads(); // wait for the other …
}
```

In any case, the two branch sections *then { }* and *else {}* are executed in sequence and not in parallel, but the first section does nothing and the loss of speed is minimal. The functions getnextproduct() and workonproduct() should execute the same code, only depending from the blockidx-value of the core, which corresponds to the number of the machine in the simulation model.

Of course, this approach is limited by the restrictions of the single program multiple-data (SPMD) programming model. Much more complex discrete simulation models must be executed on different cores in result of their heterogeneous code, but the number of such cores is not so high. Future work on the hardware will give new opportunities also for discrete simulation in this area.

2.5. Using GPU´s for high-performance applications

All the discussed scenarios for the different simulation methods are valid also for other computations in the CAD/CAM area. Like in simulation, the GPU can be used in two different modes:

- If the data or computation model is distributed in space and the computation algorithms are the same with different parameters, the GPU cores can run in parallel over a distributed model.
- If the data or model is not distributed, the cores could be used for equal runs with changing experiment parameters. For example, a complex CAD or CAM calculation could be done for different levels of external temperature or mechanical stress.

Mixed modes of the two options are also possible!

2.6. Optimization with GPU´s

A third level of parallelization is possible by using optimization techniques. The approach from chapter 1.3 -1.5 can be extended by using the results of the basic runs in an optimization method with independent points, like the Monte-Carlo optimization method or genetic algorithms.

On the GPU stream cores, a method like this would be distributed in the following way:

- One single optimization point is calculated by 16 or 32 cores inside one block.
- 32 or 16 blocks are used for getting the points for 32 or 16 individuals in the search space.
- One additional block works as an optimization control block and collects all the points and calculates the next generation of individuals.

If there are more blocks available, the whole optimization run can be started for a second or third time with different starting values for using all computing cores.

Let us assume that there are 32 runs executed for each of the 16 individuals. With this assumption, we provide the maximum number of 512 cores on a computing unit. But if there are 1600 cores in a high performance GPU like the ATI HD 5870, we can perform three runs of these optimizations, providing a total speedup of 1500.

2.7. Conclusion

Consequently, it is easier and much more flexible to use a Hyper computing approach. The major limiting factor is the single program multiple-data (SPMD) programming model inside the blocks, which defines some constraints on the bandwidth of code.

3. PREREQUISITES AND FUTURE DEVELOPMENT

The GPU hardware is supported by special API´s and C-style programming libraries. Both companies, AMD and Nvidia, provide special software drivers and programming environments (Nvidia 2012) (OpenCL 2012) (OpenAcc 2012).

3.1. CUDA, OpenCL and OpenAcc

In 1999, Nvidia invented the GPU-multicore architecture and supported/ supports the hardware with its proprietary CUDA technology (Nvidia 2012). AMD and its subdivision ATI assist the own hardware and also the Nvidia hardware with the open and non-proprietary OpenCL technology (OpenCL 2012). A third option is available since 2012 – the OpenAcc-interface (OpenAcc 2012) .

Until summer 2012, the final result of this competition has still been open:

- Nvidia´s CUDA is more efficient and easier to use on the Nvidia GPU´s.
- The OpenCL is much more flexible, but requires more development efforts. OpenCl can be executed also on multi- core CPU´s.
- OpenAcc is only an extension of C-compilers and tries to generate automatic code for the GPU from standard C-code. The performance will be lower in most cases, but the ease of use will be higher compared to CUDA and OpenCL.

From the author’s point of view, the final result will mainly depend on the OpenCL and OpenAcc development.

3.2. New hardware opportunities

New hardware from Nvidia, based on the next-generation CUDA architecture codenamed “Fermi” brings the performance of a small supercomputing cluster to the desktop. Compared to a Cray-1 from 1980 with 150 MFlops and a price of about 8 Mio $ one card now offers 480 GFlops for a price of a desktop PC. Up to 4 cards can be combined in a PC, which offers a peak performance of nearly 2 Terraflops. The future development of the hardware will continue and the results will be very interesting for all areas of high speed computing !

3.3. Final summary

The new GPU architectures are very promising for applications with a high demand of computation for a low price. Practical results are feasible and will show speedup´s of some hundreds at a very interesting price, compared with traditional parallel computers.

REFERENCES

Kirk D., Hwu W. 2010. Programming Massively Parallel Processors: A Hands-On Approach

Heusmann J., Wiedewitsch J. 1995. "Future Directions of Modeling and Simulation in the Department of Defense", Proc. of the SCSC'95, Ottawa, Canada

Wiedemann, T., 2000. VisualSLX – an open user shell for high-performance modeling and simulation, *Proceedings of the 2000 Winter Simulation Conference*, Orlando Florida

Perumalla K., 2006 “Parallel and Distributed Simulation: Traditional Techniques and Recent Advances”. *Proceedings of the 2006 Winter Simulation Conference*

Nvidia 2012. available from developer.nvidia.com/ [Accessed 15.7.12]

OpenCL 2012 available from www.khronos.org/opencl/ [Accessed 15.7.12]

OpenAcc 2012 available from www.openacc.org/ [Accessed 15.7.12]