Usage of GPU in LS-DYNA

Uli Göhner

DYNAmore GmbH, Stuttgart, Germany

1 Abstract

The increasing computing power of GPUs can be used to improve the performance of CAE systems.[1]. Within LS-DYNA an improved direct equation solver can be used, which accelerates the performance of implicit applications by use of a CUDA-based solver [2], [3], [4]. In this paper the performance improvements for different customer input decks for metal forming application (implicit springback calculations) are being compared on a Cluster system with actual GPU hardware nodes.

First an overview of the underlying hardware architecture of actual GPU systems is given. Different possibilities to use GPU for CAE computations are being introduced. The current implementation within LS-DYNA is explained and possible applications are decribed, which may benefit from the CUDA-version. For some customer applications the performance data of the LS-DYNA CUDA-version versus the LS-DYNA SMP-version is are compared.

2 GPU architecture versus CPU architecture

The GPU architecture differs from multicore CPUs considerably, as GPUs use a large number of so-called Stream Processors. GPUs have been developed to be used as SIMD-system (Flynn's classification [6]. Obviously the architecture is developed to satisfy the typical needs of graphical applications. Therefore processing the same operations (Single Instruction) for a very large number of elements/pixels (Multiple Data) perfectly fits to the GPU architecture. This type of operations typically appears within the standard rendering process, for which GPU of course is optimized for. The CPU architecture is better suited for a wider range of control structures and by this can work more efficiently on heterogeneous types of algorithms.

With the upcoming trend to use GPUs for general purpose computations ("GPGPU" - General purpose computing on graphics processing units) the GPU architecture was improved by e.g. introducing more capable Stream Multiprocessors (SM) and SFUs (Special Function Unit), which provide mathematical functions like sine, cosine, square root, etc. Also computations may be executed in double precision, which is important for a lot of CAE applications.

3 GPU for CAE applications

To improve the LS-DYNA performance by the help of GPU technology, a number of applications have been examined. One of the most promising applications was the implicit time stepping scheme, where large linear equation systems need to be solved. Linear equation solvers are well suited for the usage of GPUs. There are a couple of linear equation solvers available within LS-DYNA. As most applications involve nonlinear behaviour and thin-walled structures, iterative solvers are not ideally suited for the cases, where LS-DYNA is typically used for. In LS-DYNA implicit calculations mostly direct solver strategies are being used. As shown in [4] the direct solver implemented in LS-DYNA for implicit calculations is based on a multifrontal approach. This multifrontal algorithm has turned out to give satisfying speedup factors, if it is being ported to GPU hardware.

The actual LS-DYNA version allowing the usage of GPUs is based on the hybrid version [5], which can use MPI- and OpenMP-based parallelization at the same time. Within the work presented in this

© 2012 Copyright by DYNAmore GmbH

paper only one GPU was being tested on one node with a total of twelve cores. In future we intend to evaluate the performance of the LS-DYNA hybrid version on multiple GPUs.

4 GPU programming

There are a couple of methods to use the GPU for general programming work.

4.1 Direct shader programming model

This programming method is directly based on graphic libraries like e.g. OpenGL or Directx. The ides is, to use the texture map directly as an array, which contains the input values for the computations. After starting the rendering process, the results can be read from the gframe buffer memory. This way is the simplest way to use GPUs for computations, but it is of course not very transparent and easy to use and understand.

4.2 CUDA

CUDA (Computed Unified Device Architecture [7]) is a very well established way to use NVIDIA GPUs. It is developed by NVIDIA and only works for NVIDIA GPUs. The necessary libraries and drivers are being developed by NVIDA and provided without extra costs. Basically via the CUDA library interface functions are provided, so that the GPU can be used for computations directly. A typical CUDA application is performing the following steps:

- Memory allocation on host (CPU)
- Load Data to Memory (RAM)
- Memory allocation on device (GPU)
- Copy Data from host to device (GPU)
- Run kernel function on device (#blocks, #threads)
- Synchronise threads (barrier)
- Copy results from device to host
- Free memory on host and device

4.3 CUDA features

Similar to the SMP-Programming model on CPU also within CUDA there is a parallel thread model implemented, which takes care of the parallelization within the GPU. Modern GPUs typically consist of large number of simple processors (e.g. actual NVIDIA GPUs have several hundreds of Stream Processors). In order to use these processors in parallel a thread model is implemented in CUDA. Parallelization with threads is a very common methodology also for CPUs. One common standard for Multicore CPUs is e.g. the “Pthread” [9] or OpenMP Terminology [8].

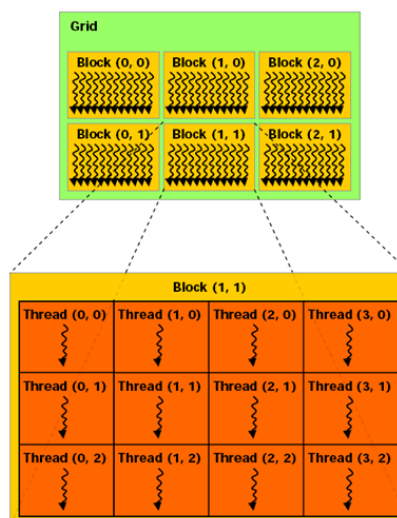

Figure 1 [7]

In comparison to the usage of CPU threads, the threads handled by CUDA are very light-weight. The system could handle several thousands of threads in a very effective manner. This is not the case on CPUs working with OpenMP or Pthreads, as the overhead necessary to initiate CPU threads is much

© 2012 Copyright by DYNAmore GmbH

higher than on the GPU system. GPU threads are also structured in a hierarchical manner (Figure 1). Up to 512 threads form a block. All blocks are organized in a 2-dimensional grid.

4.4 OpenCL

OpenCL (=Open Compute Language) is a new standard for the usage of GPU devices. Its goal is to be hardware-independent. Although initially focussing on GPUs, it is now possible also to work on CPUs. This allows one consistent programming interface to use both CPU and GPU. A large number of GPU and CPU vendors provide toolsets for programming their devices with OpenCL.

5 LS-DYNA CUDA version

For LS-DYNA a CUDA accelerated version exists, which is to be tested in this paper. The CUDA version is at the moment restricted to the implicit LS-DYNA capabilities. The reasons to first concentrate the CUDA porting efforts LS-DYNA implicit were:

- Implicit solver was most promising
- Prototype GPU accelerated solver gave satisfying speedups

The only solver which is ported to CUDA is the multifrontal solver. As the GPU usage was being developed with the help of CUDA, the version is restricted to NVIDIA hardware. To get the LS-DYNA CUDA version running, the appropriate CUDA library version is necessary. I our tests we used the cudatoolkit-4.1.28, which is available at no costs.

5.1 LS-DYNA parallelization and CUDA

For LS-DYNA there are SMP parallel and MPP parallel versions available, which are using OpenMP and MPI respectively. Recently also a hybrid version is being developed, which makes use of OpenMP parallel threads and MPI-parallel processes at the same time [10]. This gives more flexibility when using cluster architectures with a large number of cores. Also as the number of cores per processor is increasing, OpenMP can be used for 2,4, 8 Cores within one node whereas MPI-processes can distribute the work to the different nodes. By this the number of MPI-processes can be reduced, which leads to less communication and therefore to a better speedup for large number of Cores.
The LS-DYNA CUDA version is available for the LS-DYNA SMP and LS-DYNA Hybrid. In the case of the Hybrid version of LS-DYNA each LS-DYNA MPI-process must have one GPU. So multiple GPUs can be used. To make use of the MPI-parallel version one needs one GPU per node. As still the number of Cores could be varied per node via the OpenMP-parallelization this is not a big restriction in terms of flexibility [10]

5.2 Multifrontal solver strategy

The idea of the multifrontal algorithm makes use of the sparsity of the underlying matrices. The algorithm tries to minimize storage requirements and the number of necessary operations by making use of sparsity. The “fill-in” is minimized by a proper ordering of the factorization process. There is a hierarchy of dense submatrices established in form of an elimination tree. The front matrices (these are smaller dense submatrices) need to be factored directly. By a proper grouping of the frontal matrices to “supernodes” the overall cost for the factorization of the multifrontal matrices is minimzed.

5.3 Parallelization of the multifrontal solver strategy

The multifrontal algorithm can be parallelized by making use of either MPI or OpenMP. The GPU will be used to cover the “larger matrices”. OpenMP on multiple CPU-Cores will cover the smaller matrices. Multiple GPUs can be used (one GPU per MPI-process) in the LS-DYNA Hybrid Version

© 2012 Copyright by DYNAmore GmbH

6 Benchmark results

The hardware we were using for the benchmark results consists of one node with an Intel Xeon X5675 CPU and a Nvidia Tesla M2050 GPU. We are using the LS-DYNA SMP-version with 8 Cores

6.1 Model with solid elements

First we show some benchmark comparisons for a model consisting of 500.000 solid elements. Figure 2 shows, that by using the GPU could result in a speedup factor of 3,3 or in other words, the calculation time can be reduced by 70%.

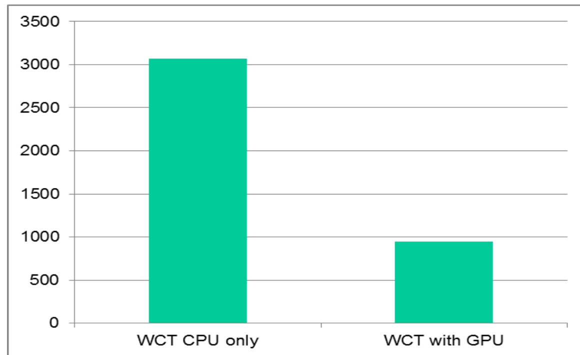

Figure 2

6.2 Model with shell elements

The second application we were investigating was a linear static analysis for a typical car model. Figure 3 shows, that in this case the gain was much smaller. For a higher number of elements, the efficiency of the GPU version was increasing. Figure 3 shows the result for a Four million element model. The calculation time could be reduced by 20%.

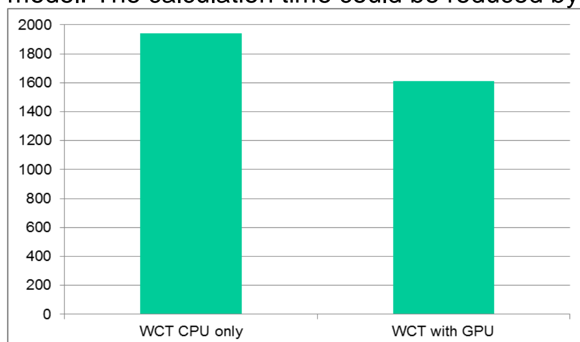

Figure 3

6.3 Springback analysis

In metal forming application springback analysis is typically done with LS-DYNA implicit. A typical customer model with 500.000 elements was being analyzed with the LS-DYNA SMP CUDA-version. In this case the calculation time could only be reduced by 6%, as shown in Figure 4. This is in line with the results in 6.2, as the model was much smaller than the car model analyzed in 6.2

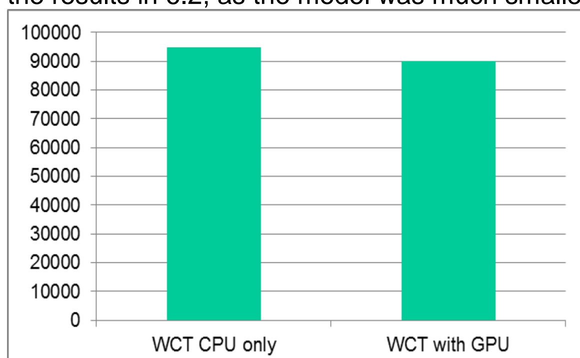

Figure 4

© 2012 Copyright by DYNAmore GmbH

6.4 Comparison of different benchmark results

There is a big difference in the speedup factors, one could achieve with the LS-DYNA CUDA version. The reason for this is the basic concept of the GPU architecture. As described in chapter 4.2 for any computation on the GPU first the data must be transferred from RAM to the device. After performing the calculation the results need to be transferred back to RAM. The amount of work which is devoted to the GPU must be large enough to compensate for the additional time needed for the data transfer between CPU and GPU. Otherwise, the time for the data transfer dominates the acceleration of the computations by the GPU leading to only small speedup factors. The amount of work the GPU gets is dependent of the size of the multifrontal matrices. In the case of thin-shelled structures the underlying element topology leads to smaller-sized matrices. Therefore the GPU power cannot be used to the full extend. The same argument explains why the GPU version gets better speedup factors the larger the model is.

7 Other applications

A multi-body simulation was being ported from CPU to GPU. Figure 5 shows some applications of the so-called physics engine developed by F. Gross in his bachelor thesis at university Kempten. [11] The simulation can be done in real-time and the results can be visualized as the simulation is being performed. In this application AMD CPU and ATI GPU hardware was used. OpenCL was chosen as IDE (integrated development environment) for the complete C++ programming work. In this case the speedup factor gained by the GPU without data transfer between GPU and CPU was measured to be a factor of 4 or 75% and with data transfer between CPU and GPU it reduced to a factor of 1,4 or 30%.

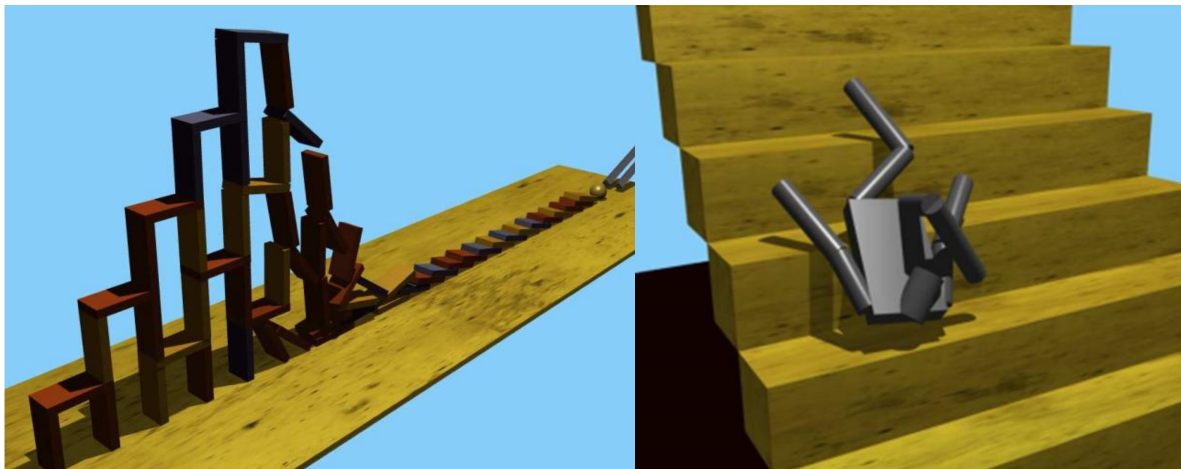

Figure 5

8 Summary

A GPU-accelerated LS-DYNA-version has been tested for LS-DYNA SMP and LS-DYNA Hybrid. The tested versions are based on CUDA, therefore restricted to NVIDIA GPUs. Only LS-DYNA Implicit makes use of the GPU. Good performance can be achieved for large solid element models. In our example an improvement of 70 percent could be achieved. Due to Matrix structure for thin-walled structures (shell elements) the performance improvements in these cases have been limited to 5-20 percent. The bottleneck is the relatively slow data transfer between CPU and GPU. As future hardware devices will improve the speed of the data transfer between CPU and GPU, the total performance can be improved considerably also in the case of thin walled shell structures.

9 Further steps

We will continue to optimize the necessary parameters in order to get optimal performance of the hardware systems. More configurations can be tested and compared, e.g. Multiple GPUs could be used. LS-DYNA HYBRID GPU quite now only has the possibility to use one GPU per MPI-process, but still multiple OpenMP-processes can be initiated.

© 2012 Copyright by DYNAmore GmbH

10 Literature

[1] Göhner, U.: “Performance of Implicit Solver Strategies on GPUs”, 9. LS-DYNA Forum, 2010.
[2] Grimes, R., Lucas, R. , Wagenbreth, G.: “The potential impact of GPUs on LS-DYNA implicit”, 11th LS-DYNA international conference, 2010.
[3] Grimes, R., Lucas, R. , Wagenbreth, G.: “The Progress on GPU Implementation for LS-DYNA Implicit Mechanics”, 8th European LS-DYNA Users Conference, 2011
[4] Lucas, R., Wagenbreth, G., Davis, D.: „Implementing a GPU-Enhanced Cluster for Large-Scale Simulations” I/ITSC conference, 2007, paper 7437.
[5] Meng, N., Wang, J., Pathy, S.: „New Features in LS-DYNA HYBRID Version” 11th LS-DYNA international conference, 2010.
[6] Flynn, M. "Some Computer Organizations and Their Effectiveness". *IEEE Trans. Comput.*, 1972
[7] “CUDA Programming Model Overview”, NVIDIA Corporation, 2008.
[8] “OpenMP Application Program Interface, Version 3.1”, OpenMP Architecture Board, July 2011.
[9] “Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language]”, ISO/IEC 9945-1:1996.
[10] Wang, J: “Hybrid (OpenMP+MPI) Version of LS-DYNA”, LS-DYNA Forum 2011.
[11] Groß, F.: “Entwicklung einer Physik-Engine zur Simulation von Starrkörpern in Echtzeit“, Bachelor Thesis, University Kempten, 2011.

© 2012 Copyright by DYNAmore GmbH