# Performance of Implicit Solver Strategies on GPUs

Prof. Dr. Uli Göhner

DYNAmore GmbH

Stuttgart, Germany

**Abstract:**

The increasing power of GPUs can be used to solve systems of linear equations. This has already been shown for different application areas in CAE. In this paper an iterative solver has been ported to the GPU using the NVIDIA CUDA development system. The solver was integrated into the open source CFD system openFOAM. Performance observations comparing CPU with GPU are being given for this case.

**Keywords:**

GPU, CUDA, openFOAM, Iterative Solver.

 

## 1 GPU used in CAE applications

During recent years the demand to use GPUs for commercial CAE applications has increased. CAE vendors are starting to develop strategies to use GPUs for an increased speed of their applications [3], [5]. In many cases linear equation solvers have been ported from CPU to the GPU to speed up the calculation process.

In this paper an iterative solver has been chosen for GPU computations. The solver performance was tested within the open-source CFD-solver open-FOAM. It can also be integrated into other CAE software like LS-DYNA implicit. The programming environment was CUDA from NVIDIA [7]. The performance comparisons have been done on a workstation with standard Intel CPU and NVIDIA GPUs.

## 2 GPU architecture

Hardware architecture must be considered to determine an appropriate algorithm and method for satisfying performance. The hardware architecture of e.g. the NVIDIA G80 in figure 1 shows a couple of thread processors, which could do floating point operations in parallel. The thread execution manager can handle the deployment of the threads to the processors very efficiently

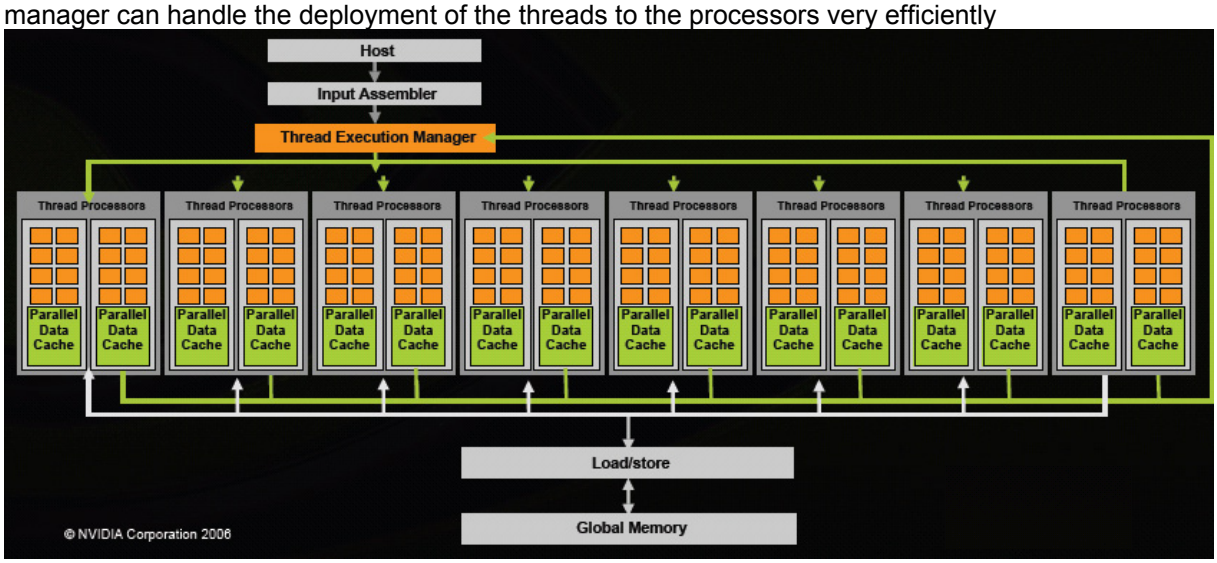


*Figure 1: NVIDIA G80 architecture [7]*

A very important point is the usage of “Load and Store” from Global Memory to GPU memory. It has been shown, that too many load and store operations may lead to bad performance data of the program on the GPU.

## 3 CUDA development environment

CUDA (=Compute Unified Device Architecture) was developed by NVIDIA in 2006 [4]. It consists of different extensions to the standard ANSI-C programming language. As shown in Figure 6 the NVIDIA graphics hardware is directly accessed by CUDA. Additionally standard libraries for linear algebra (BLAS) and digital signal processing (FFT) are contained in the CUDA framework.

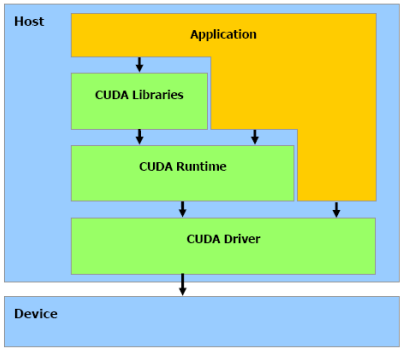


*Figure 2: CUDA architecture [7]*

# 4 Solver technology

Most CAE software systems use either direct or iterative equation solvers. The advantage of direct solver technology is, that the performance stays independent of the condition number of the stiffness matrix. In the case of nonlinear mechanics of thin shell structures the equation system could be ill-conditioned or even rank deficient. Unless special preconditioning techniques are being used, iterative solvers cannot be used for such problems efficiently, as the number of iterations needed for convergence increases considerably with the condition number of the stiffness matrix [6]. Even non-convergence can happen in case of very ill-conditioned equation systems. Therefore most equation solvers used for the analysis of nonlinear structural problems are based on direct solution techniques.

## 4.1 Equation solvers inside LS-DYNA

In LS-DYNA different solvers are accessible, if the user chooses the implicit time integration method. The direct solvers included in LS-DYNA are Multi-Frontal sparse solvers and a sparse direct solver from BCSLIB. The iterative solvers use the Jacoby method or an Incomplete Cholesky factorization as preconditioning. [3]

## 4.2 Preconditioning techniques for iterative solvers

The idea of preconditioning is as follows. Instead of solving the linear system:

$$Au = b \tag{1}$$

The system of equations will be multiplied by the inverse of the preconditioning Matrix C,

$$C^{-1}Au = C^{-1}b \tag{2}$$

where the condition number of system (2) is presumably better than the condition number of system (1). Thus the number of iterations can be reduced considerably. The closer Matrix C is to A, the better is the condition number of system (2). In an extreme case, if A=C, only 1 iteration is needed, but of course the computation of $C^{-1}$ must be performed, so all efforts go into the calculation of the preconditioning matrix.

## 4.3 Jacoby preconditioner

The simplest way of preconditioning is the method of Jacoby. Here the main diagonal of Matrix A is chosen as preconditioning Matrix C:

$$C = diag(A) \tag{3}$$

The computational effort to calculate the inverse of C is very low in this case, as it is a diagonal matrix, but the reduction effect on the number of iterations needed by an iterative solver like the conjugate gradient solver is not as strong as it is the case of other preconditioning techniques. [2]

# 5 OpenFOAM

As the main intent of this work was to investigate the relative performance of iterative solvers on CPU versus GPU, the simplest iterative solver has been chosen, namely the PCG with Jacoby preconditioning technique. As also the integration into a complete CAE system needs to be considered to achieve realistic performance numbers, the solver was integrated into an open-source CFD solver, OpenFOAM. OpenFOAM contains besides many other CFD solver options a discretization of the incompressible Navier-Stokes-Equations based on the PISO approach [1]. The incompressible Flow Solver inside openFOAM is called IcoFOAM and was used as a testbench for the Jacoby PCG on GPU.

## 5.1 IcoFOAM

IcoFoam solves the incompressible laminar Navier-Stokes equations using the PISO algorithm. The code is inherently transient, requiring an initial condition and boundary conditions. An example from the OpenFoam tutorial [7] is shown in figures 2, 3 and 4:

 

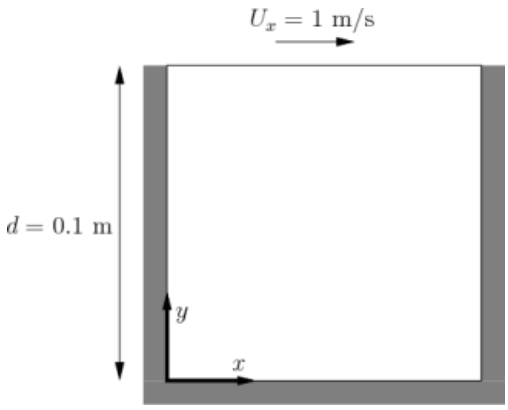


*Figure 3: Lid driven cavity*

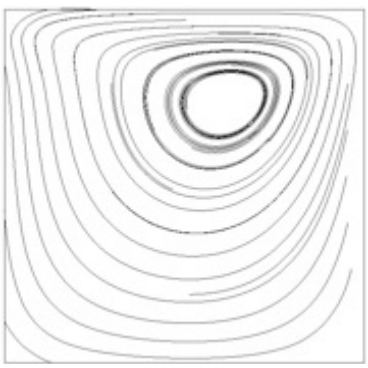

*Figure 4: Streamlines*

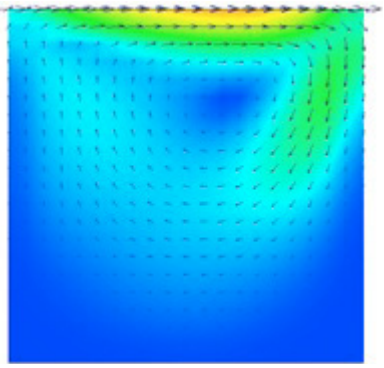

*Figure 5: Velocity field and temperature*

### 5.2 Pressure Equation

Most of the computational effort in IcoFoam is used to solve the pressure equation, which is a discretized version of the Poisson equation [1]. Discretizations of the Poisson equation are well-conditioned and therefore no convergence difficulties are expected for the iterative solvers. The resulting linear equation system for the pressure equation can be solved with the Jacoby PCG described in section 4. The existing solvers inside icoFOAM can be replaced by the GPU-based solver. (Figure 6)

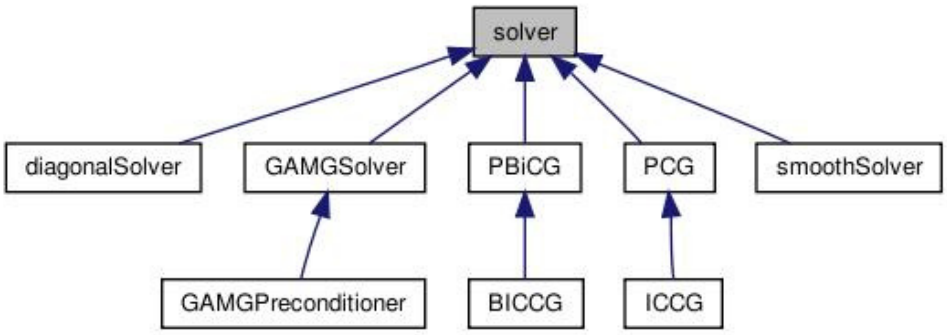


*Figure 6: IcoFOAM solver hierarchy*

### 5.3 Performance

Different mesh sizes have been used to investigate the dependency of the computing time on the number of cells used. The Hardware used was a standard Workstation with Windows XP. The performance gain for 1 GPU (NVIDIA 9600GT) against CPU (Intel Core 2 Duo) was over 75% for a large problem (CPU: 8 hours and 28 minutes versus GPU: 2 hours and 10 minutes), which is a speedup of 3,9. Single precision has been used for all computations. All programming and benchmark work was being done as part of a students' team project at the University of Applied Sciences Kempten.

### 5.4 Next steps

It has been shown, that codes using linear algebraic operations could get a large performance gain if ported from CPU to GPU through CUDA. Further performance improvements are being considered by using two or more GPUs and coupling the GPUs using the SLI technology [8]. One major problem when coupling to LS-DYNA will be the contact algorithm, which needs to be addressed for GPU-based computing. This will be investigated in another project. Also the more general programming language OpenCL will be investigated.

## 6 Summary

An iterative linear equation solver was succesfully ported to GPU through the NVIDIA CUDA development framework. The performance data are very promising, so that the usage of GPUs to speed up simulation systems could be recommended.

## 7 Literature

[1] Ferziger, J. H., Peric, M.: "Computational Methods for Fluid Mechanics" Springer, 2002.

[2] Golub, G. H., van der Vorst, H. A. "Closer to the solution: Iterative linear solvers", STATE OF THE ART IN NUMERICAL ANALYSIS, 1996, pp. 63--92, University Press.

[3] Grimes, R., Lucas, R. , Wagenbreth, G.: "The potential impact of GPUs on LS-DYNA implicit", LS-DYNA international conference, 2010.

[4] Livermore Software Technology Corporation, LS-DYNA User's Manual, Version 971 / Rev5, May 2010.

[5] Lucas, R., Wagenbreth, G., Davis, D.: „Implementing a GPU-Enhanced Cluster for Large-Scale Simulations".

[6] Rottner, Th., Lenhardt, I., Alefeld, G., Schweizerhof, K.: „Nonlinear Finite Element Analysis using the Preconditioned Lanczos Method on Serial and Parallel Computers", BIT Mathematics, Volume 37, Number 3, pp. 759-769.

[7] "The NVIDIA Cuda Zone", www.nvidia.com, 2010.

[8] "The NVIDIA SLI Zone", www.slizone.com, 2010.

 

 