Visit the Resource Center for more SIMULIA customer papers

GPU Computing with Abaqus 6.11 and its Application in Oil & Gas Industry

Goang-Ding Shyu, Guijun Deng, Baker Hughes Inc., Houston, Texas

Srinivas Kodiyalam, NVIDIA, Santa Clara, California

Abstract: This paper examines the performance characteristics of Abaqus 6.11 release for the latest GPU computing technology. The results are provided for finite element models with material, geometric and boundary nonlinearities relevant to current practice in the oil and gas industry, on workstation configurations, including a network of workstations. The motivation for these studies is to quantify the parallel performance benefits of Abaqus for the latest generation of GPUs from NVIDIA, the Tesla 20-series (codenamed Fermi) for implicit finite element models with multi-million DOFs involving plastic deformation, large strain and contact.

1. Introduction:

Today, in the oil and gas industry, well completions are going deeper and deeper into the earth to recover hydrocarbons. The market demands that oilfield tools and equipment be operated under much heavier loads and more severe environmental conditions. At the same time, the market is more competitive. Customers are demanding lower costs and shorter development cycle times from oilfield service companies.

Finite Element Analysis (FEA) has proven to be an excellent tool for evaluating equipment conceptual designs and structural capabilities prior to prototyping and testing. Parallel efficiency and turnaround times continue to be important factors behind engineering decisions to develop FEA models at higher fidelity. The need for High Performance Computing (HPC) for faster turnaround times; dealing with ever-growing model sizes; handling more complex physics such as with nonlinear FEA; and, enabling design of experiments (DOE) and formal design optimization methods in an effective manner cannot be overemphasized.

Current trends in HPC are moving towards the use of many core processor architectures in order to achieve speed-up through the extraction of a high degree of fine-grained parallelism from the applications. The trend is led by GPUs, which have been developed exclusively for computational tasks as massively-parallel co-processors to the CPU. Today's GPUs can provide memory bandwidth and floating-point performance that are several factors faster than the latest CPUs. A rapid CAE simulation capability from GPUs has the potential to transform current prctices in engineering analysis and design optimization procedures.

2. GPU Computing:

While parallel applications that use multiple cores are a well-established technology in engineering analysis, a recent trend towards the use of GPUs to accelerate CPU computations is now common. In this heterogeneous computing model the GPU serves as a co-processor to the CPU. Much work has recently been focused on GPUs as an accelerator that can produce a very high FLOPS (floating-point operations per second) rate if an algorithm is well-suited for the device. There have been several studies demonstrating the performance gains that are possible by using GPUs, but only a modest number of commercial structural mechanics software have made full use of GPUs. Independent Software Vendors (ISVs) have been able to demonstrate overall gains of 2x-3x over multi-core CPUs, a limit which is due to the current focus on linear equation solvers for GPUs vs. complete GPU implementations.

Linear solvers can be roughly ~50% of the total computation time of typical simulations, but more of the typical application software will be implemented on the GPU in progressive stages.

Shared memory is an important feature of GPUs and is used to avoid redundant global memory access among threads within a block. A GPU does not automatically make use of shared memory, and it is up to the software to explicitly specify how shared memory should be used. Thus, information must be made available to specify which global memory access can be shared by multiple threads within a block. Algorithm design for optimizing memory access is further complicated by the number of different memory locations the application must consider. Unlike a CPU, memory access is under the full and manual control of a software developer. There are several memory locations on the GPU which is closely related to the main CPU memory. Different memory spaces have different scope and access characteristics: some are read-only; some are optimized for particular access patterns. Significant gains (or losses) in performance are possible depending on the choice of memory utilization.

Another issue to be considered for GPU implementation is that of data transfers across the PCI-Express bus which bridges the CPU and GPU memory spaces. The PCI-Express bus has a theoretical maximum bandwidth of 4 or 8 GB/s depending on whether it is of generation 1 or 2. When this number is compared to the bandwidth between GPUs on-board GDDR5 memory and the GPU multi-processors (up to 150 GB/s), it becomes clear that an algorithm that requires a large amount of continuous data transfer between the CPU and GPU will unlikely achieve good performance.

For a given simulation, one obvious approach is to limit the size of the domain that can be calculated so that all of the necessary data can be stored in the GPU's main memory. Using this approach, it is only necessary to perform large transfers across the PCI-Express bus at the start of the computation and at the end (final solution). High-end NVIDIA GPUs offer up to 6 GB of main memory, sufficient to store a large portion of the data needed by most engineering software, so this restriction is not a significant limitation.

3. Direct Sparse Solvers:

Direct sparse solvers are widely used in the discipline of computational structural mechanics (CSM) for simulations that deploy implicit schemes. Multi-frontal direct sparse solver techniques are the most common algorithms for commercial software. In a multi-frontal solver, an assembly tree is built for the sparse matrix, whose nodes are dense matrices. The factorization of the sparse matrix is based on the factorization of those dense matrices (fronts) and their assembly into super-nodes. Most of the runtime of a direct sparse solver is spent in the factorization of the frontal matrices and their assembly.

The typical strategy of developing a direct sparse solver for the GPU is to offload those dense matrix operations to the GPU, which typically can be very efficient for a massively parallel GPU architecture. Other solver operations such as the matrix assembly, tree transversal, and forward/backward solve can all stay on CPU, since they are difficult to parallelize on the GPU and typically require only a small fraction of the total solution time.

Since matrix fronts are copied to the GPU one-by-one for computation and then copied back to the CPU afterwards, there are two potential performance bottlenecks. First, for finite element models with many "holes" and/or thin shells, the fraction of runtime for small fronts will increase. Generally, a GPU requires a dense matrix to be larger than ~1K to reach peak performance. For many small matrices, this could require a significant fraction of the factorization phase and the overall performance could be well below the peak GPU performance.

The solution to a more general direct solver GPU approach that could benefit a broader range of model geometries is to use the concurrent kernel feature introduced in the Fermi architecture. Since the matrix fronts on the same assembly tree level can be processed independently, their GPU kernels can run concurrently on the GPU, and thereby fully utilizing the hardware for peak performance. Another potential problem is overhead associated with

CPU-GPU data transfers. Knowing that fronts on the same assembly tree level can be processed independently, one could overlap CPU-GPU data transfer of one front with the kernel computation of another front, which is a supported feature on Fermi GPUs.

4. Oil and Gas Industry Applications:

4.1 Pump shaft analysis:

Three pump shaft models (shown in Figure 1), each with increasing level of fidelity, are used with Abaqus 6.11 for performing plastic deformation and large strain analysis. The analysis includes material and geometric nonlinearities. The objective of the pump shaft analyses is to evaluate the maximum torque that can be applied to the three different models.

The Abaqus analysis were performed on a Dell workstation with 2x hex-core Intel Xeon processor X5680, 3.33GHz, 48GB memory, 1x Tesla C2070 GPU, 1x Quadro FX 3800, and, Red Hat Enterprise Linux 5.3.

Model	Elements*	DOF	CPU 4 core	CPU 4 core + 1 GPU	Speedup
PumpShaft 1	172949	745545	22606	11497	1.97
PumpShaft2	314888	1325301	28647	14926	1.92
PumpShaft3	513524	2146926	102138	52875	1.93

- C3D10M (3D 10-node tetrahedral solid elements)

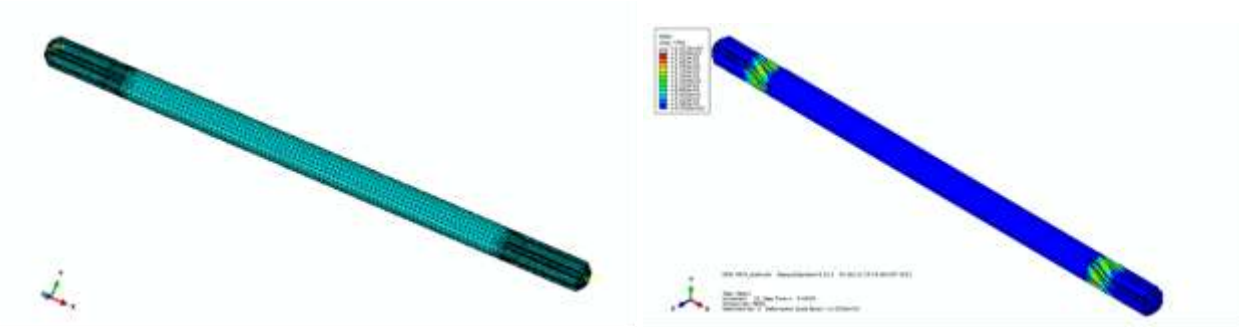

Figure 1. Pump shaft finite element model

4.2 Lower Lock Cone Buckling Analysis:

The buckling analysis of the lower lock cone (shown in Figure 2) with Abaqus 6.11 includes material, geometric as well as boundary nonlinearities. In other words, plastic deformation, large strain and contact all occur in this application. The objective is to determine the load-displacement curve and the maximum load that can be applied to the cone.

The Abaqus analysis were performed on a Dell workstation with 2x hex-core Intel Xeon processor X5680, 3.33GHz, 48GB memory, 1x Tesla C2070 GPU, 1x Quadro FX 3800, and, Red Hat Enterprise Linux 5.3.

Model	Elements*	DOF	CPU 4 core	CPU 4 core + 1 GPU	Speedup
LowerLockCone	538284	1910400	29223	18031	1.62

- C3D8 (3D 8-node hexahedral solid elements)

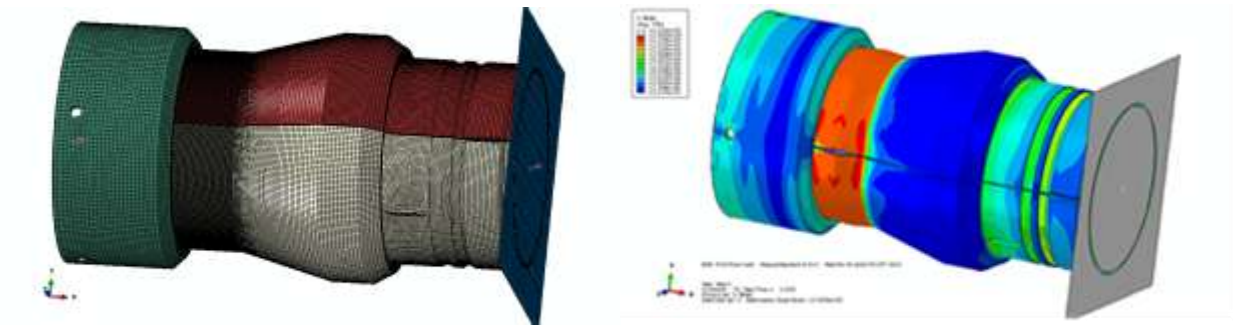

Figure 2. Lower lock cone finite element model

4.3 Chemical Injection Side Pocket Mandrel:

The analysis objective with this application is to check the state of deflection, stresses and plastic strains of the tool for loading conditions on the operation envelope. The analysis itself is a plastic deformation and large strain analysis involving material and geometric nonlinearity. The model consists of six parts welded together (shown in Figure 3).

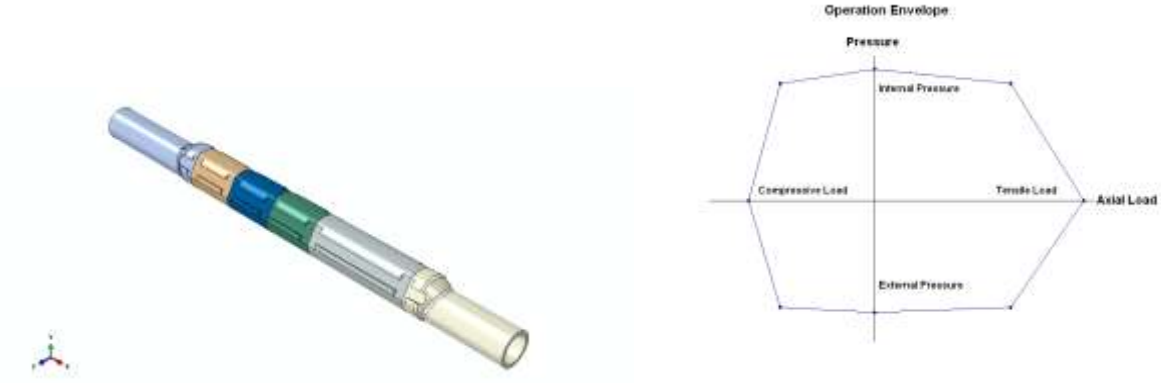

Figure 3. Chemical injection side pocket mandrel finite element model and operation envelope

The Abaqus analysis were performed on a Dell workstation with 2x hex-core Intel Xeon processor X5680, 3.33GHz, 48GB memory, 1x Tesla C2070 GPU, 1x Quadro FX 3800, and, Red Hat Enterprise Linux 5.3.

Model	Elements*	DOF	CPU 4 core	CPU 4 core + 1 GPU	Speedup
LowerLockCone	955058	3584631	6721	3897	1.73

* C3D10I and C3D8

In addition, this analysis was also performed using Abaqus direct sparse on a network of 2 workstations connected with a Gigabit network switch. A hybrid MPI and thread based parallelization is used with the direct sparse solver solution in Abaqus 6.11 on a compute cluster of server nodes or workstations. MPI is used for communication between the nodes or workstations.

The following shows the realized speed-up in analysis using 4 cores on a network of 2 workstations versus all 4 cores on a single workstation with single and multiple GPUs.

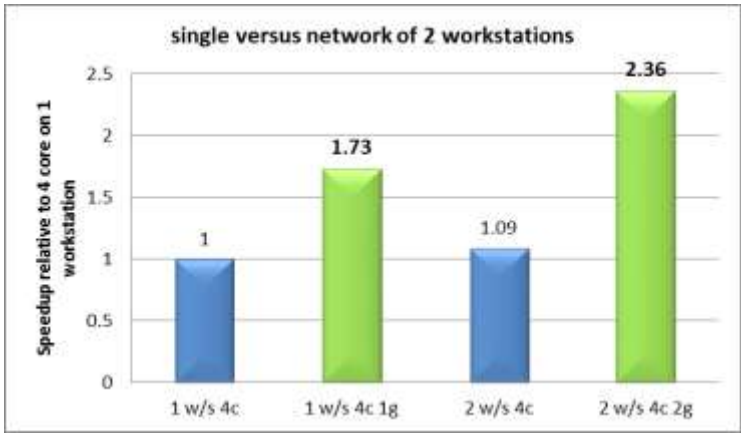

Figure 4. Performance speed-up with single and two GPUs

In the above chart:
1 w/s 4c = 4 CPU core on a single workstation
1 w/s 4c 1g = 4 CPU core + 1 Tesla GPU on a single workstation
2 w/s 4c = 2 workstations with 2 CPU core per workstation (total of 4 core)
2 w/s 4c 2g = 2 workstations with 2 CPU core and 1 Tesla GPU per workstation (total of 4 core and 2 GPUs)

The single GPU acceleration of the 4 CPU core run provides a speedup of 1.73x and the speedup improves to 2.36x using 2 GPUs (1 GPU/workstation).

4.4 Cylinder Sub and Spring Housing of a Safety Valve:

The analysis objective in this is to check if the tool can meet one design criterion, which is, the maximum ductile damage to be less than 1 at a given load factor. The model is shown below and it consists of 1.03M nodes and 0.92M elements modeled using C3D8 and C3D10 elements.

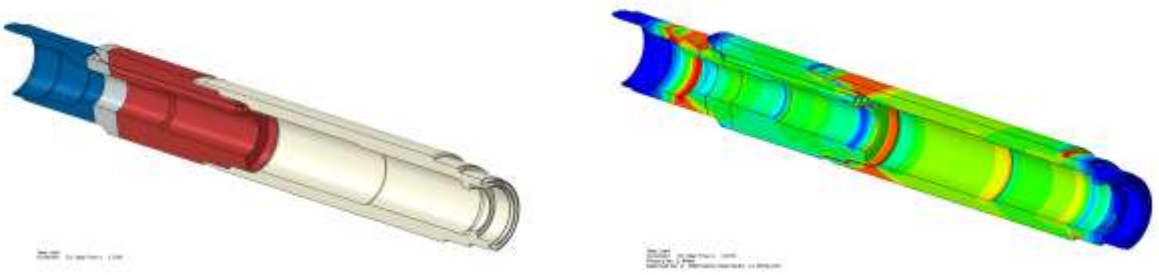

Figure 5. Cylinder sub and spring housing model

Since the model has around 3M DOF, it requires more than a single workstation with 48GB memory for the Abaqus analysis. We present timing results below (see Figure 6) using a network of 2 workstations to perform structural and coupled thermal-structural analyses.

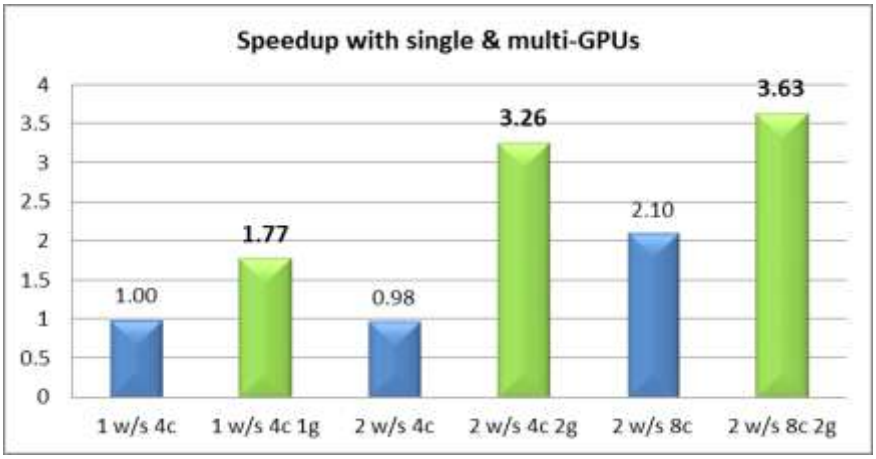

Figure 6. Performance speed-up with single and two GPUs

In the above chart:
1 w/s 4c = 4 CPU cores on a single workstation
1 w/s 4c 1g = 4 CPU cores + 1 Tesla GPU on a single workstation
2 w/s 4c = 2 workstations with 2 CPU cores per workstation (total of 4 cores)
2 w/s 4c 2g = 2 workstations with 2 CPU cores and 1 Tesla GPU per workstation (total of 4 cores and 2 GPUs)
2 w/s 8c = 2 workstations with 4 CPU cores per workstation (total of 8 cores)
2 w/s 8c 2g = 2 workstations with 4 CPU cores and 1 Tesla GPU per workstation (total of 8 cores and 2 GPUs)

The single GPU acceleration of the 4 CPU core run provides a speedup of 1.77x and the speedup improves to 3.26x using 2 GPUs (1 GPU/workstation).

5. Summary:

GPU computing is supported in Abaqus 6.11 release to significantly lower the simulation times for industry standard analysis models. The performance speed-ups enabled by GPU computing facilitates Abaqus users to add more realism to their models thus improving the quality of the simulations.

In this paper, we have discussed the application of GPU computing in Abaqus to a range of highly nonlinear analysis models associated with the oil and gas industry. The speed up from GPU computing is significant and up to 2X over 4 core CPU only solutions. Higher speedups are also observed when using multiple GPUs on a network of two workstations.

6. References:

1. V. Belsky, et. al., Accelerating Commercial Linear Dynamic And Nonlinear Implicit FEA Software Through High-Performance Computing, Proceeding of NAFEMS 2011, Boston, USA, May 2011.

2. S. Posey and P. Wang, GPU Progress in Sparse Matrix Solvers for Applications in Computational Mechanics, Proceedings of 50th Aerospace Sciences Meeting, AIAA, Nashville, TN, January 2012.

Visit the Resource Center for more SIMULIA customer papers