GPU COMPUTING WITH MSC NASTRAN 2013

Srinivas Kodiyalam, NVIDIA, Santa Clara, USA

THEME

Accelerated computing with GPUs

SUMMARY

Current trends in HPC (High Performance Computing) 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. This hybrid computing trend is led by GPUs (Graphics Processing Units), 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. In order to exploit this hybrid computing model and the massively parallel GPU architecture, application software will need to be redesigned. MSC Software and NVIDIA engineers have been working together on the use of GPUs to accelerate the sparse direct solvers in MSC Nastran for the last 2 years. This presentation will address the recent GPU computing developments including support of NVH solutions with MSC Nastran 2013. Representative industry examples will be presented to demonstrate the performance speedup resulting from GPU acceleration. A rapid CAE simulation capability from GPUs has the potential to transform current practices in engineering analysis and design optimization procedures.

KEYWORDS

High Performance Computing (HPC), Graphics Processing Units (GPUs), MSC Nastran, Structural analysis, Sparse matrix solvers, Noise Vibration and Harshness (NVH).

GPU COMPUTING WITH MSC NASTRAN 2013

1: Introduction

The power wall has introduced radical changes in computer architectures whereby increasing core counts and hence, increasing parallelism have replaced increasing clock speeds as the primary way of delivering greater hardware performance. A modern GPU consists of several hundreds or thousands of simple processing cores; this degree of parallelism on a single processor is typically referred to as 'many-core' relative to 'multi-core' that refers to processors with at most a few dozen cores.

Many-core GPUs often demand a high degree of fine-grained parallelism – the application program should create many threads so that while some threads are waiting for data to return from memory other threads can be executing – offering a different approach in terms of hiding memory latency because of their specialization to inherently parallel problems.

With the ever-increasing demand for more computing performance, the HPC industry is moving towards a hybrid computing model, where GPUs and CPUs work together to perform general purpose computing tasks. In this hybrid computing model, the GPU serves as an accelerator to the CPU, to offload the CPU and to increase computational efficiency. In order to exploit this hybrid computing model and the massively parallel GPU architecture, application software will need to be redesigned. MSC Software and NVIDIA engineers have been working together over the past 2 years on the use of GPUs to accelerate the sparse solvers in MSC Nastran.

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. 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 8 or 16 GB/s depending on whether it is of generation 2 or 3. When this number is compared to the bandwidth between GPUs on-board GDDR5 memory and the GPU multi-processors (up to 250 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: Sparse solver acceleration with MSC Nastran

A sparse direct solver is possibly the most important component in a finite element structural analysis program. Typically, a multi-frontal algorithm with out-of-core capability for solving extremely large problems and BLAS level 3 kernels for the highest compute efficiency is implemented. Elimination tree

and compute kernel level parallelism with dynamic scheduling is used to ensure the best scalability. The BLAS level 3 compute kernels in a sparse direct solver are the prime candidate for GPU computing due to their high floating point density and favourable compute to communication ratio.

The proprietary symmetric MSCLDL and asymmetric MSCLU sparse direct solvers in MSC Nastran employ a super-element analysis concept instead of dynamic tree level parallelism. In this super-element analysis, the structure/matrix is first decomposed into large sub-structures/sub-domains according to user input and load balance heuristics. The out-of-core multi-frontal algorithm is then used to compute the boundary stiffness, or the Schur compliment, followed by the transformation of the load vector, or the right hand side, to the boundary. The global solution is found after the boundary stiffness matrices are assembled into the residual structure and the residual structure is factorized and solved. The GPU is a natural fit for each sub-structure boundary stiffness/Schur compliment calculation.

Today's GPUs can provide memory bandwidth and floating-point performance that are several factors faster than the latest CPUs. In MSC Nastran, the most time consuming part is the BLAS level 3 operations in the multi-frontal factorization process. To date, most of solve and all of rank-N updates are done on the GPU implemented as CUDA kernels. Specifically:

- Factorization of diagonal block on CPU;
- Solve of panel (beneath diagonal block) largely on GPU; and,
- Trailing sub matrix update of the front factorization (rank N update) on GPU.

The sparse solver does better when things are "blocky" as they tend to generate larger front sizes. Models using solid elements provide for more concentrated computational work in the sparse matrix factorization, which is highly desirable for the GPU; while with models using shell/plate elements the matrix is sparser. In other words, the computational work is less concentrated for a shell model compared to a solid model.

4: GPU Computing implementation & target analyses (Solution Sequences) in MSC Nastran:

NVIDIA's CUDA parallel programming architecture is used to implement the compute intensive sparse solver components on the GPU. CUDA is the hardware and software architecture that enables NVIDIA GPUs to execute programs written with C, C++, FORTRAN, OpenCL, and other languages.

The GPU computing functionality in MSC Nastran was first released in 2012.1, and updated and enhanced in 2013. The MSC Nastran GPU acceleration is delivered by a set of compute kernels for the symmetric (MSCLDL) and the non-symmetric (MSCLU) sparse direct solvers, for NVIDIA CUDA-capable GPUs.

A good GPU kernel implementation, such as the one in MSC Nastran as shown in Figure 1, overlaps compute on GPU, data transfer in the PCIe bus, and compute on CPU, with multiple CUDA streams. To have enough floating point computations, such that these overlaps can occur, the front size, i.e. nfront, has to be sufficiently large. In addition, to make the task more compute bound instead of PCIe communication bound, the rank update size, i.e. nrank also needs to be sufficiently large since PCIe limits the GPU performance for small ranks for Schur complement calculation with small inner dimension.

Therefore improved performance occurs for relatively large models. In particular, only matrices whose front sizes are larger than a certain threshold benefit. To get good performance the GPU capability works in conjunction with the Nastran system cells (sys653=1 and sys653=3), so that rank update sizes greater than 320 can be used effectively with MSCLDL and MSCLU. The three options, ie.sys653=0, 1 and 3, offer different levels of computational efficiency, numeric accuracy, and hardware resource requirement. In general, sys653=0 provides the most numerically stable solution and the least memory consumption, but also the lowest performance. For slightly more memory consumption, a positive definite, or diagonally dominant, model can be solved by sys653=1; since sys653=1 does not do numeric pivoting, the performance is typically the best among these three options. For SOL101 and SOL108 jobs, sys653=3 is set as the default in MSC Nastran; the user can expect improved performance at the possibility of larger memory requirement and less stable numeric pivoting results. The user is advised to select the one option that is most appropriate for the particular Nastran analysis model.

Vastly reduced use of pinned host memory and the ability to handle arbitrarily large fronts, for very large models (greater than 15M DOF) on a single GPU with 6GB device memory, are some strengths of the GPU implementation in MSC Nastran 2013. 'Staging' is a term that is used to describe how very large fronts are handled. If the trailing submatrix is too large to fit on the GPU device memory, then it is broken up into approximately equal-sized 'stages' and the stages are completed in order. Multiple streams are used within a stage. So, an arbitrarily large submatrix, say 40GB, can be solved in, say, 10 stages of 4GB each. The implementation in MSC Nastran is the lowest granularity GPU implementation of a sparse direct solver that solves very large sparse matrix sizes.

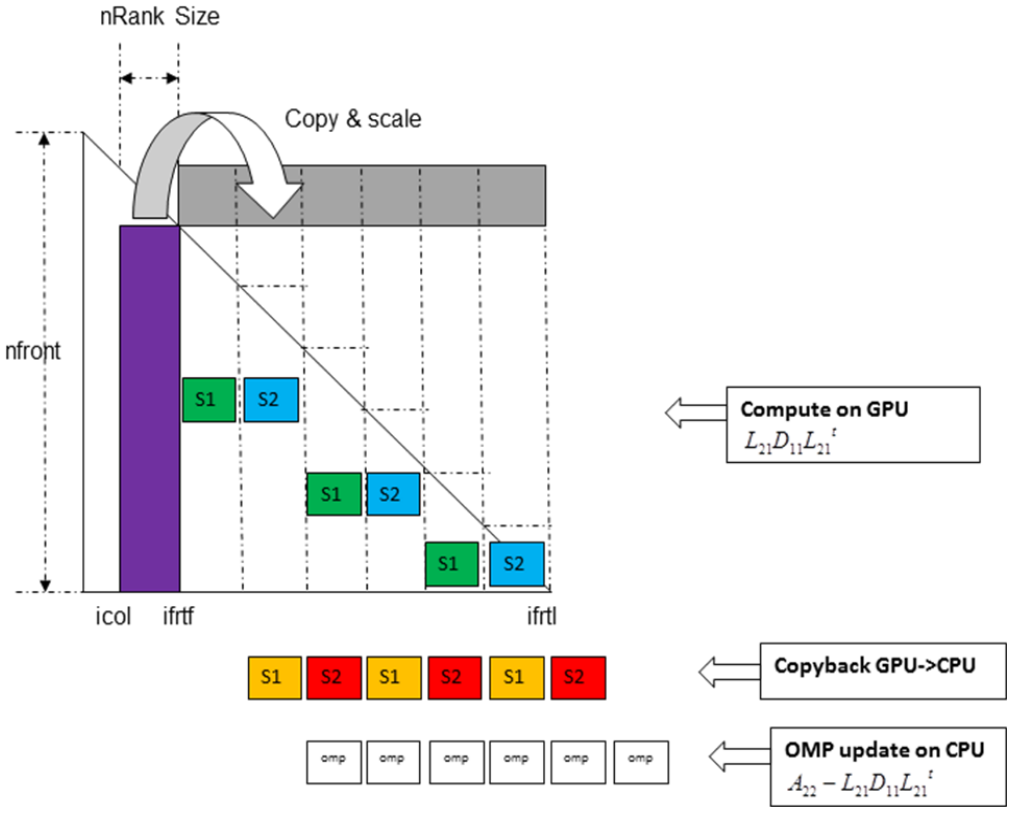

Figure 1: Sparse direct solver GPU kernel implementation

The MSC Nastran implementation also supports multiple GPU computing capability for DMP (Distributed Memory Parallel) runs. In such cases of DMP>1, multiple fronts are factorized concurrently on multiple GPUs. The matrix is decomposed into two domains, and each domain is computed by a MPI process.

Typical MSC Nastran job submission commands with single and multiple GPUs are shown below:

1) SMP (Shared Memory Parallel) + GPU acceleration of SOL101 analysis:

nast20130 jid=job101.dat mode=i8 mem=24gb parallel=4 *gpuid=0* sys655=32 sys656=1024 bat=no scr=yes sdir=/tmp sys205=320

2) DMP + SMP + GPU acceleration of SOL108 analysis:

nast20130 jid=job108.dat mem=8gb dmp=2 parallel=4 *gpuid=0:1* sys655=16 sys656=1024 bat=no scr=yes sdir=/tmp sys219=384

gpuid is the ID of a licensed GPU device to be used in the analysis. Multiple IDs may be assigned to MSC Nastran DMP runs;

sys655 represent fronts with rank sizes equal to or greater than the specified value are computed on GPU: 16 (for complex data types) or 32 (for real);

sys656 represents fronts with front sizes equal to or greater than the specified value are computed on GPU;

sys205 is real symmetric sparse decomposition rank update size;

sys219 is complex symmetric sparse decomposition rank update size.

Any 'fat' BLAS3 code path would be potential candidate for GPU computing. In MSC Nastran, several solution sequences are impacted to varying levels by the sparse solver acceleration using GPUs - SOL101 (linear statics), SOL103 (normal modes), SOL108 (direct frequency), SOL111 (modal frequency), SOL200 (design sensitivity and optimization) and SOL400 (nonlinear) fall into this category.

It is noteworthy that in SOL101 and SOL108, the sparse matrix factorization and solver times are a higher percentage of the overall job time (> 50%) and hence higher speed-ups can be realized with GPU acceleration while with SOL103 and SOL111 it is lesser. With a SOL103 normal modes analysis, say, using a Lanczos algroithm, there are multiple compute intensive parts including sparse matrix factorization, iteration on a block of vectors (solve) and orthogonalization of vectors. Only the sparse matrix factorization is performed on the GPU and hence lesser speedup from GPU acceleration.

The GPUs supported with this implementation are the Tesla Fermi GPUs (C2050, C2070, C2075, M2090), Quadro 6000, Tesla Kepler GPUs (K20, K20X). Any GPU supporting CUDA compute level 2.0 or better will run. Memory will be the most limiting factor for other. Linux and Windows 64-bit platforms are supported.

5: Performance analysis on industry standard models and solution sequences

Structural analysis solutions that use the sparse direct solvers are the target applications for illustrating the use of GPU computing in MSC Nastran 2013. Industry standard models for SOL101, SOL103 and SOL108 are used to demonstrate the benefits of GPU computing. A range of models with varying fidelity, from solids only to a mix of shells, solids and bars is considered. Performance acceleration are reported relative to a serial Nastran run, which is still widely adopted within the customer community, as well as relative to multi-core CPU runs.

The hardware configuration used with the performance analysis consists of NVIDIA PSG cluster with each server node having 2x 8-core Sandy Bridge CPUs, 2.6GHz, Tesla 2x K20X GPU, 128GB memory, QDR IB interconnect.

Case 1: Stress analysis of automotive engine block (SOL101):

Number of elements: ~786K (CHEXA)

Number of nodes: ~811K

Number of degrees of freedom: ~2.4M

Estimated Maximum Front Size: ~41K

Table 1 summarizes the speed-up observed from using a single GPU.

	Elapsed Time in minutes for full analysis	Speed-up
Serial (CPU only)	150	1
4-core (CPU only)	55	2.7
4-core + **1 GPU**	25	6

Table 1: Single GPU speedup of SOL101 engine block

Case 2: Normal modes analysis of aerospace piston design (SOL103):

Number of elements: ~561K (CTETRA)

Number of nodes: ~881K

Number of degrees of freedom: ~2.6M

Estimated Maximum Front Size: ~19K

Table 2 summarizes the speed-up observed from using a single GPU.

	Elapsed Time in minutes for full analysis	Speed-up
Serial (CPU only)	128	1
4-core (CPU only)	67	1.9
4-core + **1 GPU**	45	2.8

Table 2: Single GPU speedup of SOL103 piston model

As noted in Section 4, with a SOL103 normal modes analysis using a Lanczos algroithm, there are multiple compute intensive parts including sparse matrix factorization, iteration on a block of vectors (solve) and orthogonalization of vectors. Only the sparse matrix factorization is performed on the GPU and hence lesser speedup from GPU acceleration.

Case 3: Trimmed Car body frequency response analysis (SOL108 - NVH) with multiple GPUs:

Number of elements: ~1.04M (CQUAD4); ~109K (CTETRA); ~41K (CTRIA3), CBARs, CELAS1s, CBUSHs, CONM2, …

Number of nodes: ~1.2M

Number of degrees of freedom: ~7.5M

Estimated Maximum Front Size: ~14K

There are a total of 100 frequency increments in the FREQ1 card and the Lanczos algorithm (EIGRL) is used for computing the modes.

Figure 2 shows the speedup obtained from using a single GPU per node across multiple nodes of the cluster.

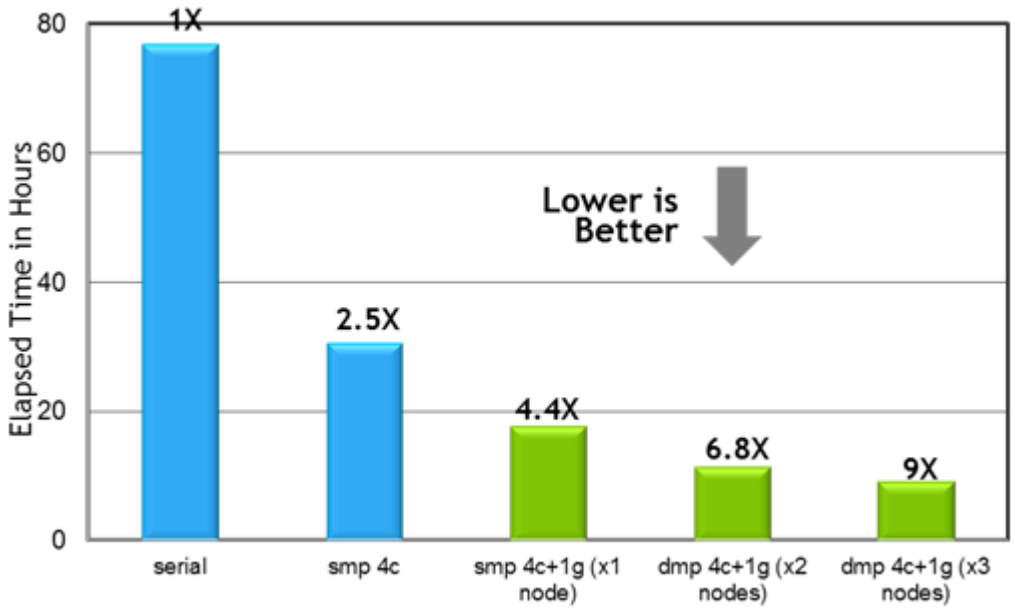

Figure 2: Multi-GPU acceleration of trimmed car body NVH analysis

Case 4: Coupled Structural-Acoustics simulation (SOL108 NVH) with multiple GPUs:

Number of elements: ~3.8M (CTETRA)

Number of nodes: ~710K

Number of degrees of freedom: ~710K

Estimated Maximum Front Size: ~8.9K

The Nastran analysis is a coupled structural-acoustics analysis using SOL108. Within the internal cavity, the typical responses are the sound pressures at the ear locations of driver and passengers. There are a total of 100 frequency increments in the FREQ1 card and the Lanczos algorithm (EIGRL) is used for computing the modes.

Figure 3 shows the speedup obtained from using single and multiple GPUs. We observe around 5X with a single GPU acceleration over a serial run; about 2x with a single GPU over a 4-core SMP run, and comparing the last 2 lines another 2x speedup with 2 GPUs over a 8 core DMP run.

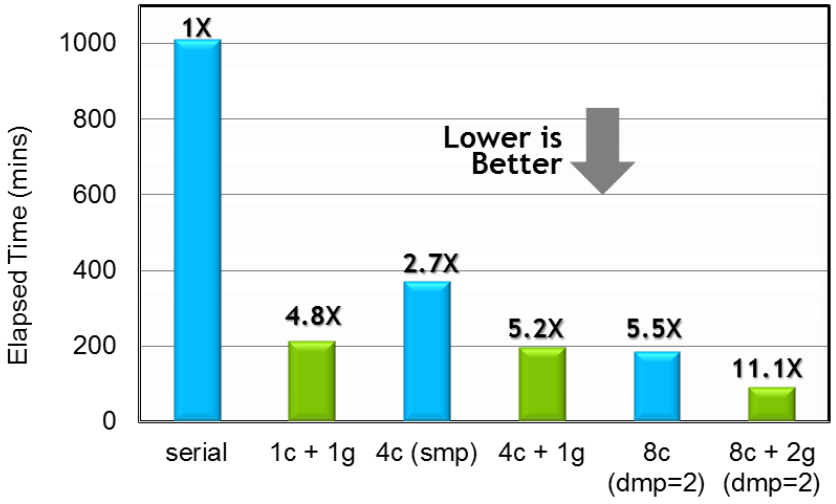

Figure 3: Multi-GPU acceleration of a vehicle system SOL108 analysis

6: Summary

GPU computing is implemented in MSC Nastran to significantly lower the simulation times for industry standard analysis models. Vastly reduced use of pinned memory; the ability to handle arbitrarily large front sizes for very large models and support for the various data types (real, complex, symmetric, unsymmetric) are some of the strengths of this implementation. Further, multiple GPUs can be used with Nastran DMP analysis. The performance speed-ups enabled by GPU computing will facilitate MSC Nastran users to add more realism to their models thus improving the quality of the simulations. A rapid CAE simulation capability from GPUs has the potential to transform current practices in engineering analysis and design optimization procedures.

REFERENCES

1. 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.

2. C. Liao, MSC Nastran Sparse Direct Solvers for Tesla GPUs, GPU Technology Conference, GTC2012, San Jose, CA, May 2012.

3. CUDA Toolkit 5.0 Performance Report, NVIDIA, January 2013.