Accelerating Implicit LS-DYNA® with GPU

Yih-Yih Lin
Hewlett-Packard Company

Abstract

A major hindrance to the widespread use of Implicit LS-DYNA is its high compute cost. This paper will show modern GPU, cooperating with CPU, can help remove this hindrance. Performance improvement for Implicit LS-DYNA with GPU relative to that without, as well as from recent GPU and X86 processor, will be studied. The study will cover GPU related hardware issues, including GPU Boost, memory and PCI Express Interface.

Introduction

Mathematically, Implicit LS-DYNA is a code to assemble and solve the linear system $\boldsymbol{Ku} = \boldsymbol{f}$ for the displacement $\boldsymbol{u}$. Assuming the banded stiff matrix $\boldsymbol{K}$ is associated with a configuration of N degrees of freedom, the theoretical floating operation count is then proportional to N^3 for a direct solver vis-á-vis N for the much less expensive explicit solution.

With the advent of the distributed memory parallel (called MPP) LS-DYNA more than a decade ago, Explicit LS-DYNA analysis has been able to achieve its excellent scalability with modern multicore multiprocessors. In contrast, the more compute intensive Implicit LS-DYNA analysis did not gain any scalability with the initial pure MPI implementation of MPP LS-DYNA due to its difficulty in meeting the memory requirement for each MPI process, even for problems of modest size. The hybrid implementation of MPP LS-DYNA, in which both distributed and shared memory parallelisms are adopted, alleviates the memory requirement and makes Implicit LS-DYNA gains respectable, but not spectacular, scalability. With the recent advance in the hardware GPU (graphic processing unit) and its programming software, LSTC has added GPU capability to the hybrid implementation to help its performance.

This paper will investigate how much the GPU can help the performance of the hybrid implementation. The memory requirement per MPI process for the hybrid implementation is proportional to N^2 but inversely proportional to the number of MPI processes (hence to the number of nodes). Given a fixed number of nodes, such memory requirement can exhaust the available in-core memory fast as the problem size increases. The only alternative is then to solve the problem out-of-core, by which the secondly memory, disk, is used. As the difference between speeds of main memory and disk is large, this paper will study the performance of GPU with the in-core and the out-of-core solutions separately. To provide a base for GPU's performance/cost estimate, how the number of GPU nodes affects performance will also be presented. The GPUs used are Nvidia Tesla K20X and the recently released K40. Their main difference lies in clock rate and memory transfer speed. GPU is a board attached to CPU via the PCI Express interface (PCIe), which is also shared by I/O devices such as disk. Consequently, the PCIe's configuration will have a much greater effect on performance of out-of-core solutions than in-core.

In this investigation, the Hybrid LS-DYNA 971 R6.1 and the GPU-enabled Hybrid LS-DYNA R6.1 versions are used. In the release note for the GPU-enabled version, it is documented that GPU cannot benefit all implicit problems, but can substantially benefit problems of solid elements, such as the AWE cylinder models, on a single 8-core node with 2 GPUs. This paper will further the same line of study with more CPU nodes and more GPUs. Three AWE cylinder models of 0.5, 1, and 2 million solid elements—known as CYL0P5E6, CYL1E6, and CYL2E—will be studied.

The hybrid method decomposes the stiff matrix into blocks for achieving both distributed and shared memory parallelisms. The degree of distributed parallelism is the product of the number of nodes times the number of MPI processes per node. The shared memory parallelism is embodied in a single MPI process, and its degree of parallelism depends on the number of threads. For optimal performance, one and only one of the CPU cores is associated with one thread. In the GPU-enabled hybrid method, the GPU processing is embodied with each MPI process so that the number of GPUs used is the same as the number of MPI processes per node.

Two HP ProLiant server configurations were used: One was a cluster of five HP SL250s nodes, each of which was equipped with three Tesla K20X GPUs and interconnected with FDR InfiniBand; the other was a single SL250s node, equipped with one Tesla K40 GPU. The hardware configuration of an HP SL250s node is as follows:

Node type	HP ProLiant SL250s
Processor model	Intel Xeon E5-2680 v2 (Ivy Bridge) (10 cores) (2.8GHz)
Processors/Cores per node	2 processors/20 cores per node
Memory per node	128 GB
Accelerators per node	Up to 3 Nvidia Tesla GPU or Intel Xeon Phi accelerators
Interconnect	InfiniBand FDR

GPU Speedup

For a given model, GPU speedup is defined as the ratio of the elapsed time with the Hybrid LS-DYNA to that with the GPU-enabled Hybrid LS-DYNA. Six in-core solution runs with the following parallelism configurations were performed:

	Nodes	Ranks/GPUs per node	Threads per rank
CYL0P5E6	1	2	10
CYL1E6	4	2	10
CYL2E6	5	2	10

The result is recorded in Figure 1. It shows that with 2 GPUs per node, LS-DYNA has achieved 1.7 to 2 times speedup, with an average of 1.86 times. The GPU-enabled Hybrid LS-DYNA in fact only enables GPU computing in the matrix factorization in solving the linear system. Assuming a speedup of 1.86 with 2 GPUs per node and let P represents the runtime portion of

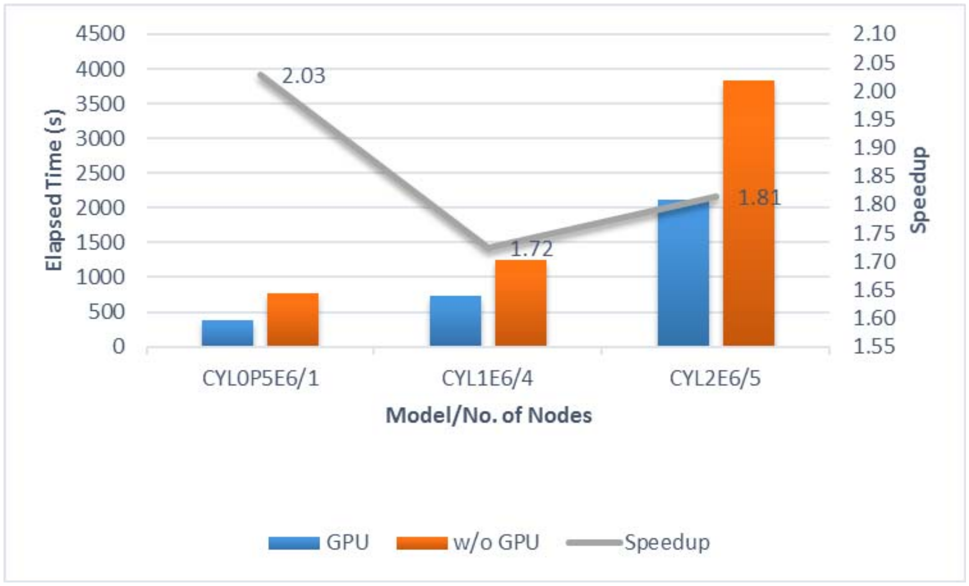

Figure 1 Elapsed times with vs. without GPU

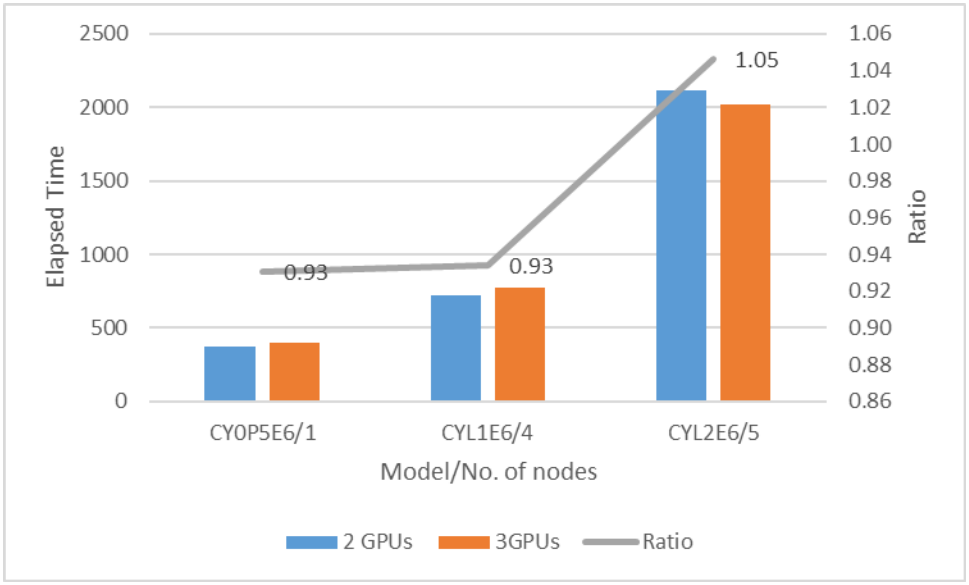

Figure 2 Elapsed times: 2 GPUs vs. 3 GPUs

the matrix factorization, Amdahl's Law then states the formula 1/((1-P) + P/2) = 1.86, which gives P=0.92. With P=0.92 and 3 GPUs per node, Amdahl's Law would predict a speedup of 1/((1-0.92) + 0.92/3) = 2.63, which is a speedup of 41% over 2 GPUs per node. However, as shown in Figure 2, results from actual runs cannot substantiate such a prediction. Figure 2 shows that for both the CYL0P5E6 and the CYL1E6 models, the 3 GPUs per node jobs are 7% slower than the 2 GPUs per node job, and that for the CYL2E6 nodes the 3 GPU job is only 5% faster. One reason for this discrepancy between theory and practice is that running a 3-GPU job requires the number of threads to be decreased to 6 from 10, so as to make the product 3 x 6 less than 20 (the number of available cores in a node), which in turn slows down the degree of thread parallelism.

GPU Boost

An Nvidia GPU board can be operated in two different clock modes: base clock and boost clock. Base clock is the default. When boost clock mode is turned on, the K20X will increase the clock frequency by 7%, and the K40 by 17%. In Figure 3 the elapsed times with and without GPU Boost are compared. It shows the following:

- K20X: 1% speedup was achieved with 7% increase in GPU clock.
- K40: 5% speedup was achieved with 17% increase in GPU clock.

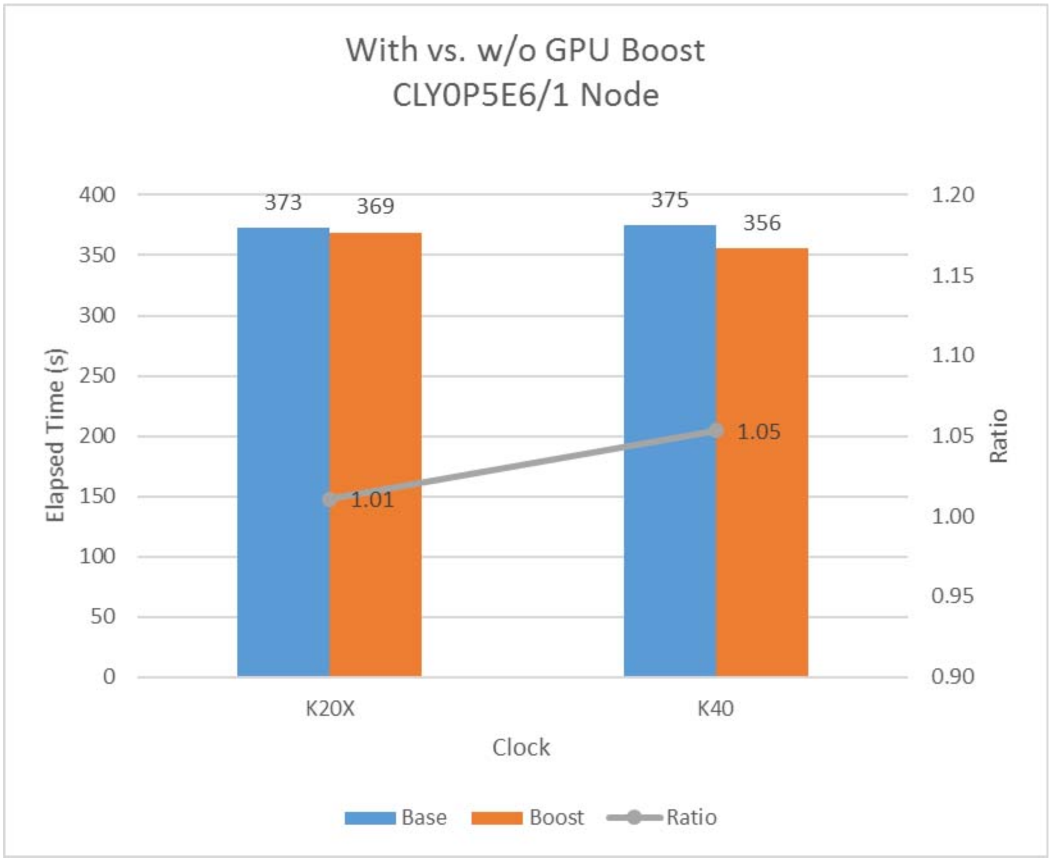

Figure 3 CYL0P5E6 elapsed times with and without GPU Boost

Out-of-core solutions

As mentioned before, the memory requirement can grow very fast with increasing problem sizes and force LS-DYNA to use out-of-core solutions. The benefit of GPUs to in-core solutions was shown in Section **GPU Speedup** earlier. Can a similar benefit be extended to out-of-core solutions? It has been known that the elapsed times of out-of-core solutions, for the same problem, vary with the value specified in LS-DYNA's "memory" command parameter. In Figure 4, two memory settings that causes the problem to be solved out-of-core were used: one with memory=2000m and one with memory=3000m. From Figure 4, the following observations can be drawn:

- Unlike an in-core solution, for a given problem the elapsed time for an out-of-core solution varies with the value specified in the "memory" parameter.
- The relationship between the elapsed time and the "memory" parameter is difficult to explain. One would expect that with more memory per rank, the 3000m case would be faster than the 2000m. But it was not.
- Out-of-core solutions are always slower than in-core solutions. Nevertheless, GPU still speeded up both out-of-core solutions by 50 percent or more.

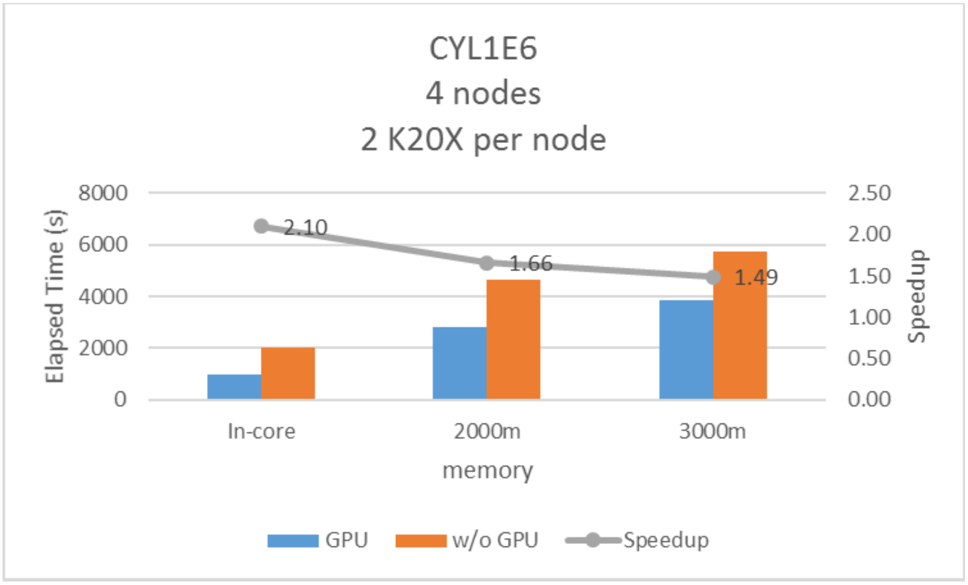

Figure 4 Elapsed Times for CYL1E6 with various memory settings

Effects of SL250s I/O Architecture on GPU Performance

Each GPU communicates to a CPU over PCIe x16. In SL250s, two GPUs can be attached to one CPU, and the other GPU and the disks are attached to the other CPU. Consequently, if two GPUs are used in a run, either they are attached to the same CPU or on different CPUs. Since each CPU has a separate memory subsystem, the two different attachments will cause two different PCIe communication paths and thus affect GPU performance differently.

How the two PCIe communication paths affect the performance of in-core solutions is shown in Figure 5, from which the following conclusions can be drawn:

- On K20X, the performance for 1 GPU attached to each CPU was 2% slower than for both GPUs attached to a single CPU.
- On K40, the performance for 1 GPU attached to each CPU was 9% slower than for both GPUs attached to a single CPU.

Figure 6 studies the effect of the two different PCIe communication paths on out-of-core solutions by using one GPU, and from it we can conclude:

- The out-of-core solution is much slower on K40 than on K20X.
- On the K40, it mattered little to which CPU the GPU attached.
- On the K20X, attaching the GPU to the CPU which did not support the disks was more advantageous than to the other by 13% in performance.

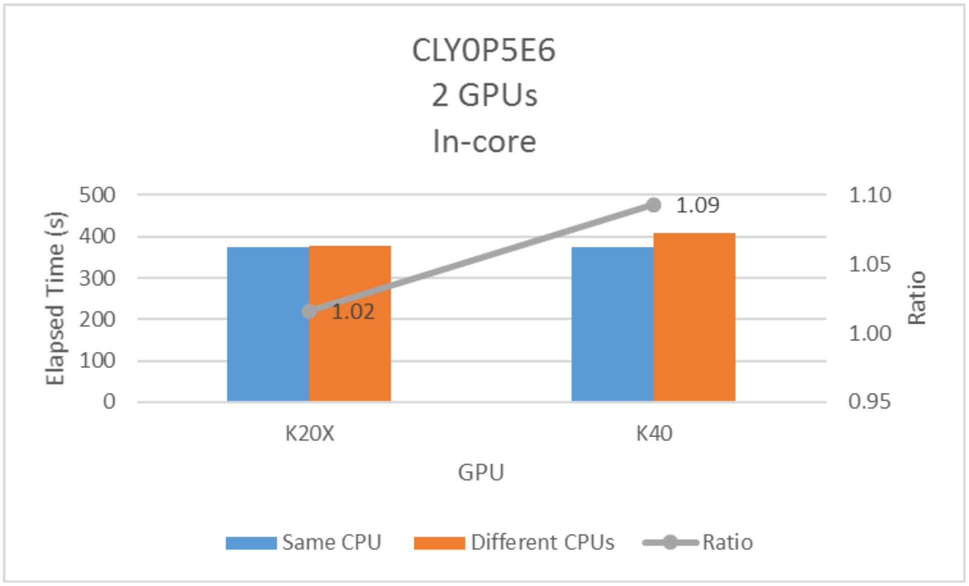

Figure 5 Elapsed times with the two GPUs attached to same CPU and to different CPUs

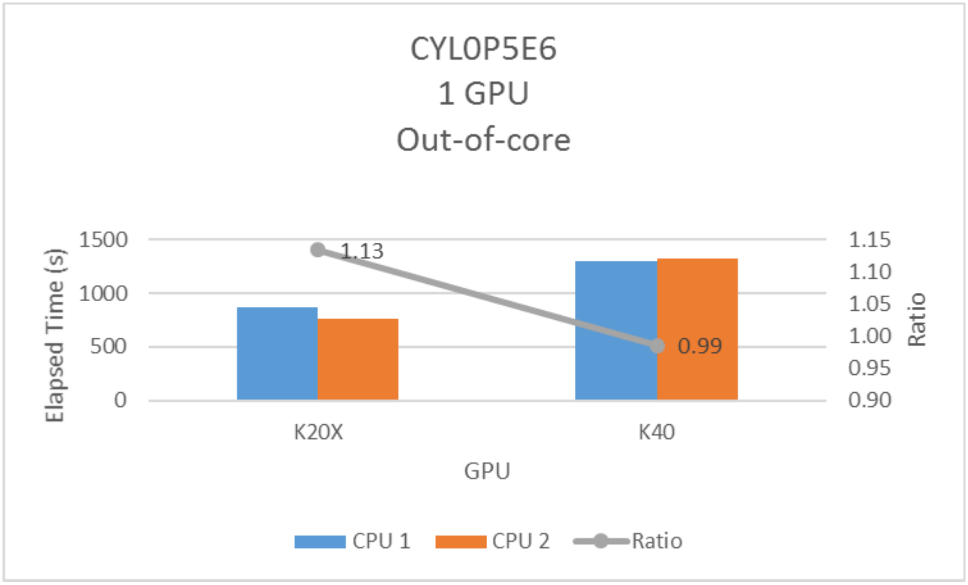

Figure 6 Elapsed times with the GPU attached one of the two CPUs

Conclusion

The following summarize the findings of this study:

- The GPU can speed up an in-core Implicit LS-DYNA solution by a factor of 1.5 or more, and it can speed up an out-of-core Implicit LS-DYNA solution by 50% or more.
- The optimal number of GPUs is two per node.
- GPU Boost helps performance a little for in-core Implicit LS-DYNA solution, and it helps the K40 more than the K20X.
- With two GPUs per node, attaching the two GPUs to the same CPU is more advantageous for both K20X and K40's performances than attaching them to different CPUs.