LS-DYNA® Scalability Analysis on Cray Supercomputers

Ting-Ting Zhu, Cray Inc.

Jason Wang, Livermore Software Technology Corp.

WP-LS-DYNA-062014

www.cray.com
www.lstc.com

Table of Contents

Abstract

For the automotive industry, car crash analysis by finite elements is crucial to shortening the design cycle and reducing costs. To increase the accuracy of analysis, in addition to the improvement in finite element analysis techniques, smaller cells of finite element meshes are used to better represent the car geometry. The use of finer mesh coupled with the need for fast turnaround has put increased demand on the scalability of finite element analysis.

In this paper, we will use the car2car model to measure LS-DYNA scalability on Cray® XC30™ supercomputers, an Intel ® Xeon® processor-based system using the Cray Aries network. We analyze the scalability of different functions in LS-DYNA at high core counts and the message-passing interface (MPI) communication pattern of LS-DYNA. We will also explore the performance difference between using one thread per core and two threads per core. Finally, we explore the performance impact of using Linux huge pages.

Introduction

Since May 2013, Cray Inc. and LSTC have been working with a customer on some very large (25 million to-100 million elements) explicit LS-DYNA analysis problems to evaluate LS-DYNA scalability on the Cray® XC30™ system. Because of the problems' size and the limited scalability of million packets per second (MPPS) LS-DYNA, the analysis takes nearly a month. To shorten their design cycle, the customer needs better hardware and software performance.

In the last decade, technological advances have doubled the number of cores per processor every couple of years, but CPU clock speed has not gotten much faster. To reduce the simulation turnaround time, we need highly scalable supercomputers and improved LS-DYNA parallel performance. At the same time, due to the nature of load imbalance in the LS-DYNA contact algorithm, MPP LS-DYNA scalability has been limited to 1,000 to 2,000 cores, depending on the size of the problem. To improve LS-DYNA scalability, we looked to hybrid LS-DYNA, which combines distributed memory parallelism using MPI with shared memory parallelism using OpenMP.

The advantage of hybrid LS-DYNA at low core counts has been demonstrated by Kendo and Makino[1] , and Meng and Wang[2]. In this paper, we will use the car2car model from topcrunch.org to understand the scalability of MPP and hybrid LS-DYNA at high core counts on the Cray XC30 system.

Scalability Analysis of Pure MPP LS-DYNA on the Cray XC30 System

[1] Kenshiro Kondo and Mitsuhiro Makino, "High Performance of Car Crash Simulation by LS-DYNA Hybrid Parallel Version on Fujitsu FX1," 11th International LS-DYNA Conference (2010).
[2] Nich Meng and Jason Wang, et al., "New Features in LS-DYNA HYBRID Version," 11th International LS-DYNA Conference (2010).

First, we use the car2car model (with 2.4 million elements) to test pure MPP LS-DYNA parallel performance at core counts of 256 to 4,096 on the Cray XC30 system.

Software and Hardware Configurations

The following hardware and software configurations are used for this evaluation:

Table 1: Summary of hardware and software configurations

System type	Cray® XC30™
Processor architecture	Intel® Xeon® processor E5 family (2.6 GHz Sandy Bridge)
Sockets/node	2 sockets per node
Number of cores per socket	8 cores per socket or 16 cores per node
Network	Aries Interconnect and Dragonfly Topology
Memory size per node	32 GB/node
OS	Cray Linux Environment (CLE) 5.1
LS-DYNA version	LS-DYNA R61_84148
Compiler version	Intel/13.1.3.192
MPI version	Cray-MPICH 6.0.2

Pure MPP LS-DYNA Scalability

Figure 1 shows the scalability of pure MPP LS-DYNA in car2car at core counts from 256 to 4,096 on an XC30 system. Pure MPP LS-DYNA scales reasonably well up to 2,048 cores, but beyond that the scaling levels off. In the next section we will look at LS-DYNA functionality profiles to understand the limiting factor in pure MPP LS-DYNA scaling.

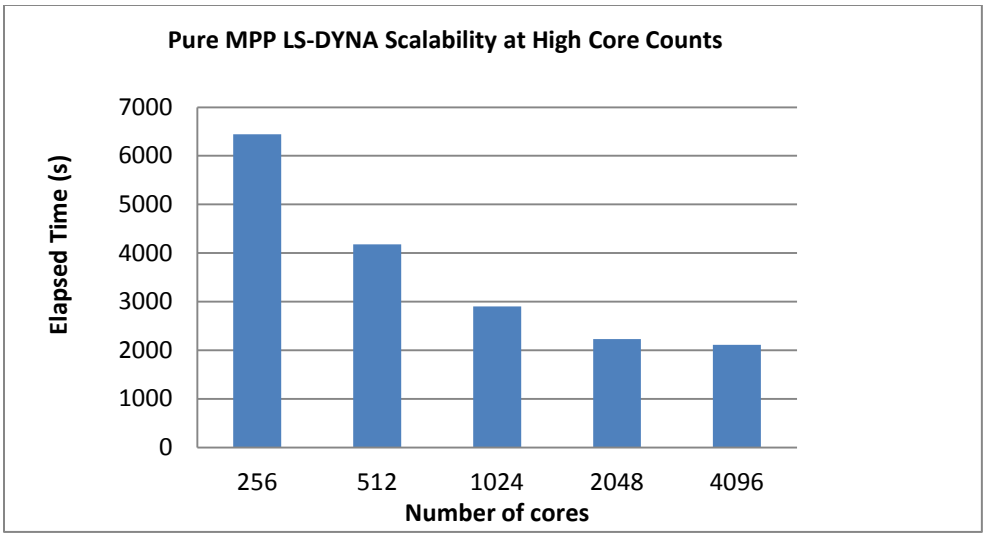

Figure 1. Pure MPP LS-DYNA scalability in car2car on a Cray® XC30™ system

Scalability of Element Processing and Contact

LS-DYNA provides detailed timing information of several major functions at the end of the log file. For car crash simulations, the functions of "element processing," "contact algorithm" and "rigid bodies" consume more than 85 percent of total wall clock time at high core counts. Since there is no MPI barrier call between "contact algorithm" and "rigid bodies," we will measure the combined time spent on both (called "contact + rigid bodies" in this paper).

Now let's take a look at how "element processing" and "contact + rigid bodies" of pure MPP LS-DYNA perform at high core counts. Figure 2 shows that "element processing" scales very well up to 4,096 cores; however, "contact + rigid bodies" scale poorly for core counts of 256 to 2,048 and hardly scales after 2,048 cores. This result indicates that "contact + rigid bodies" scalability is the limiting factor for pure MPP LS-DYNA scaling beyond 2,048 cores.

Next, we will use Cray MPI profiling tool called "profiler" to show the lack of scalability in "contact + rigid bodies."

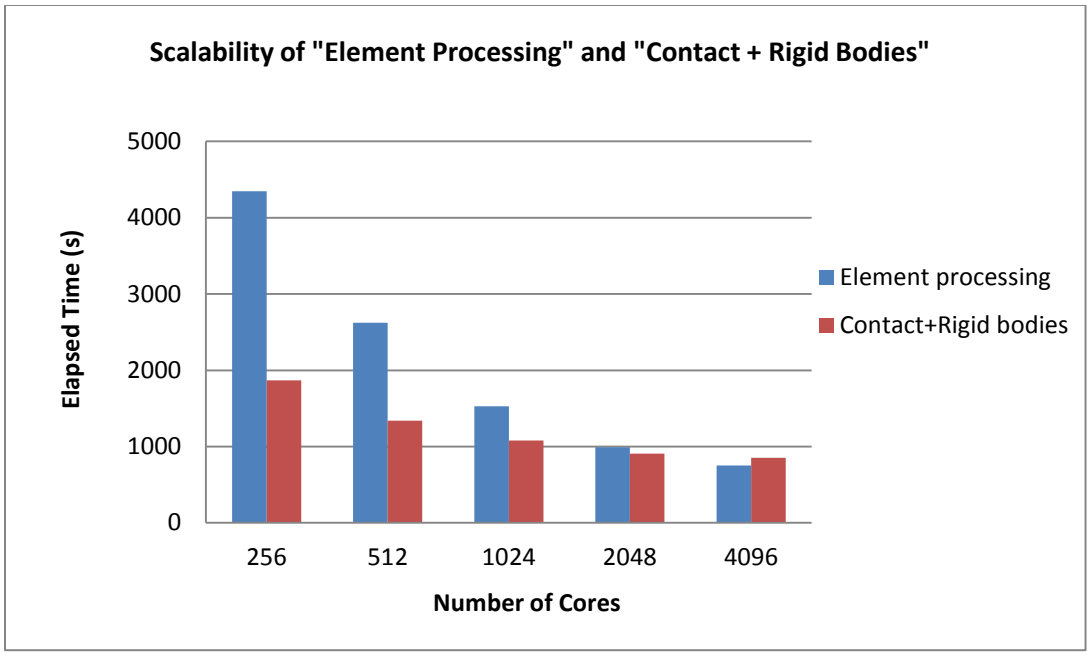

Figure 2. Scalability of "element processing" and "contact + rigid bodies" in Car2car on a Cray® XC30™ system

MPI Profile of Pure MPP LS-DYNA

Cray MPI "profiler" reports the total elapsed time, computational time and MPI time as well as MPI communication patterns. For collective calls, it further breaks down MPI time to MPI synchronization time (time that a process spends synchronizing with other processes) and MPI communication time (time that a process spends communicating with other processes). MPI synchronization time is proportional to the amount of the load imbalance in a program, and MPI communication time is disproportional to the speed of interconnect. For MPI time in point-to-point communications, such as MPI_send and MPI_recv, however, MPI synchronization time and communication time cannot be measured separately. Rather, measurement of MPI communication time in point-to-point MPI communication calls is the sum of MPI synchronization time and communication time.

Figure 3 shows the scalability of MPI time of pure MPP LS-DYNA in car2car on the Cray XC30 system. The total MPI time drops from 2,309s at a core count of 256 to 1,478s at a core count of 2,048, but increases to 1,570s at a core count of 4,096. The percentage of total MPI time to total elapsed time increases from 36 percent at 256 cores to 74 percent at 4,096 cores. Additionally, MPI synchronization time is about 65 percent of total MPI time. This result further indicates that load imbalance in contact algorithm of pure MPP LS-DYNA is the main cause of the poor scalability at high core counts.

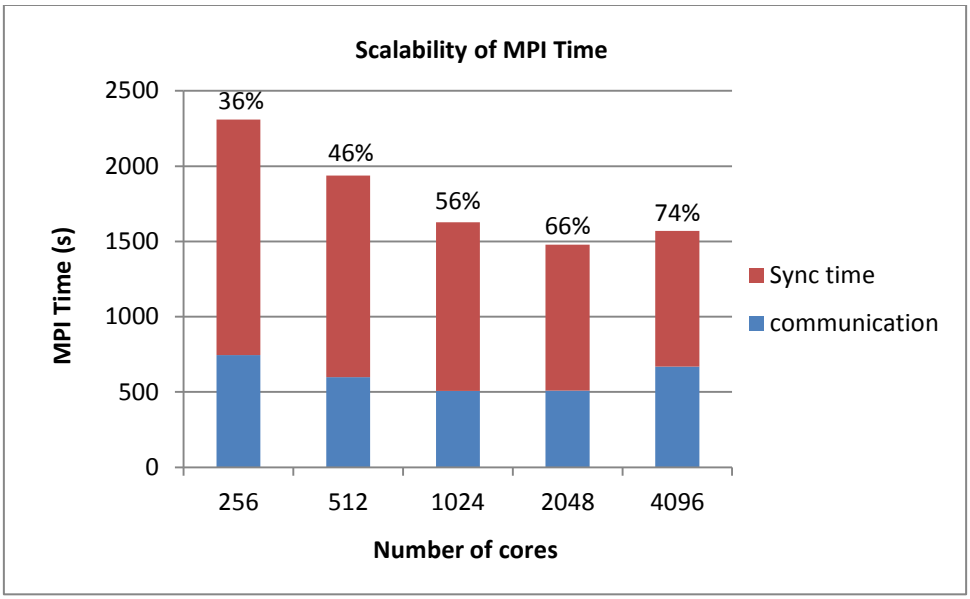

Figure 3. Scalability of MPI time in car2car on a Cray® XC30™ system

From Cray "profiler" reports, we find that 80 percent of MPI time in pure MPP LS-DYNA is spent on MPI_bcast, MPI_allreduce and MPI_recv. Figure 4 depicts the MPI time, synchronization time and communication time on these three MPI calls. It shows that MPI_bcast communication time is less than 2.5 percent of MPI_bcast synchronization time, and overall MPI_bcast time decreases as core count increases. MPI_allreduce communication time is less than 5 percent of MPI_allreduce synchronization time, and overall MPI_allreduce time increases with core count. MPI_recv time, which is about 26 percent to 38 percent of total MPI time, reduces as core count increases from 256 to 1,024 cores. But MPI_recv time increases as core counts increases from 1,024 to 4,096 cores.

As we mentioned earlier, MPI_recv synchronization and communication time cannot be measured separately. Based on the amount of synchronization time in MPI_bcast and MPI_allreduce, we believe that larger portion of measured MPI_recv time is probably spent on synchronization. This hypothesis implies that most of MPI time is spent on MPI synchronization, meaning load imbalance in LS-DYNA is what prevents it from scaling beyond 2,048 cores. This also means that without the load imbalance issue, the speed of MPI communication in an XC30 system would be fast enough for pure MPP LS-DYNA to potentially scale to 4,096 and beyond.

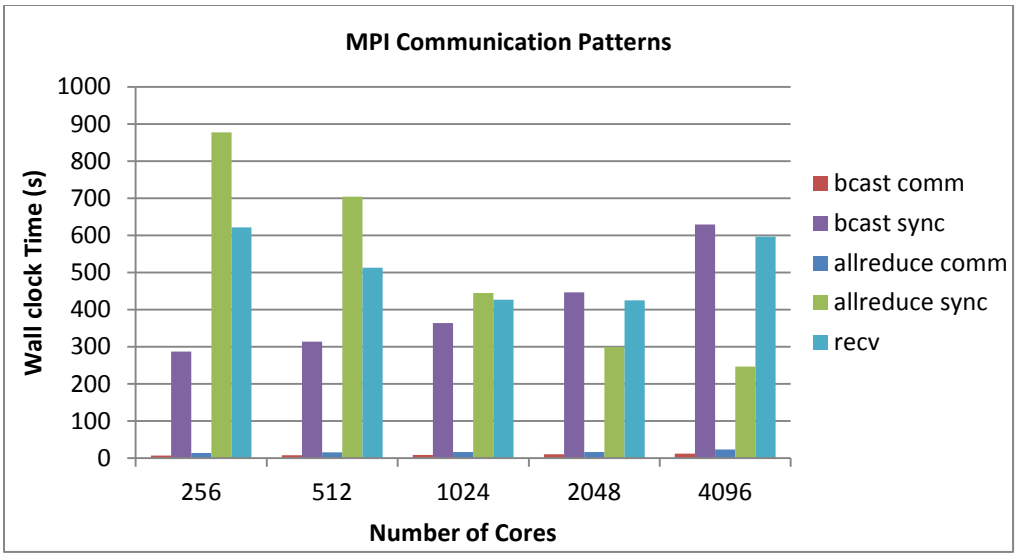

Figure 4. MPI communication patterns of pure MPP LS-DYNA in car2car on a Cray® XC30™ system

One effective way to reduce the impact of load imbalance on LS-DYNA parallel performance is to use hybrid LS-DYNA, which combines MPI processes with OpenMP threads. With hybrid LS-DYNA, we can push LS-DYNA scaling to beyond 2,048 cores. In the next section, we will discuss hybrid LS-DYNA parallel performance in car2car, as well as its comparison with pure MPP LS-DYNA on an XC30 system.

Scalability Analysis of Hybrid LS-DYNA on a Cray XC30 System

MPI Profile of Pure MPP LS-DYNA

The shared memory parallelism of explicit LS-DYNA scales well to eight threads. In the first evaluation, we run hybrid LS-DYNA using four OpenMP threads with core counts from 256 to 4,096. Figure 5 shows that hybrid LS-DYNA is 23 percent to 60 percent faster than pure MPP LS-DYNA for the same core counts.

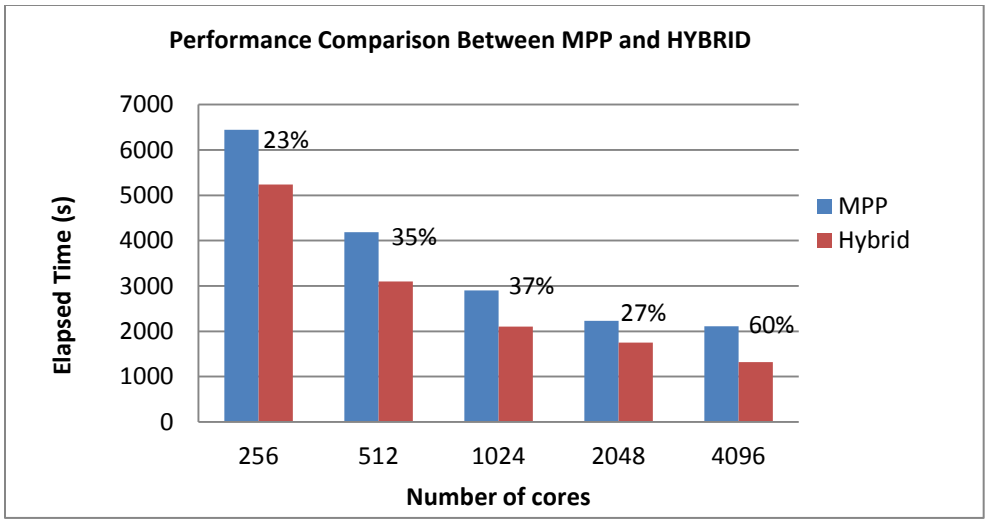

Figure 5. Performance comparison between pure MPP and HYBRID LS-DYNA in car2car on Cray® XC30™

Figure 6 and Figure 7 show the performance comparisons of "element processing" and "contact + rigid bodies" between pure MPP and hybrid LS-DYNA on an XC30 system. "Element processing" of hybrid LS-DYNA is 35 percent to 58 percent faster than that of pure MPP LS-DYNA, while "contact + rigid bodies" of hybrid LS-DYNA is 2 percent slower than that of MPP LS-DYNA at 256 cores, becomes 10 percent faster at 512 cores, and is 43 percent faster at 4,096 cores. So for core counts of 256 to 512, hybrid LS-DYNA gains its performance over MPP LS-DYNA mainly from "element processing." For cores counts of 1,024 to 4,096, it gains its performance from both "element processing" and "contact + rigid bodies."

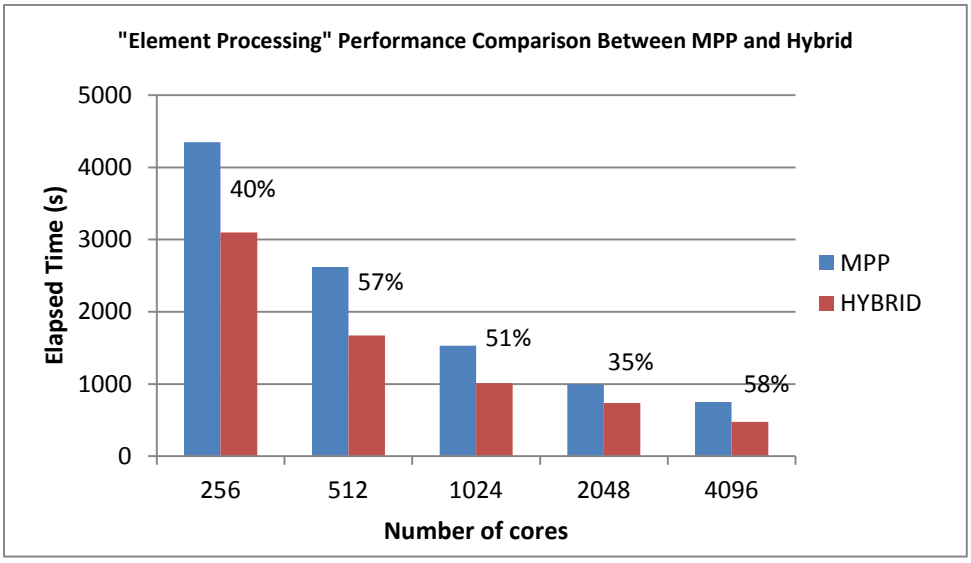

Figure 6. "Element processing" performance comparison between MPP and hybrid LS-DYNA in car2car on an XC30 system

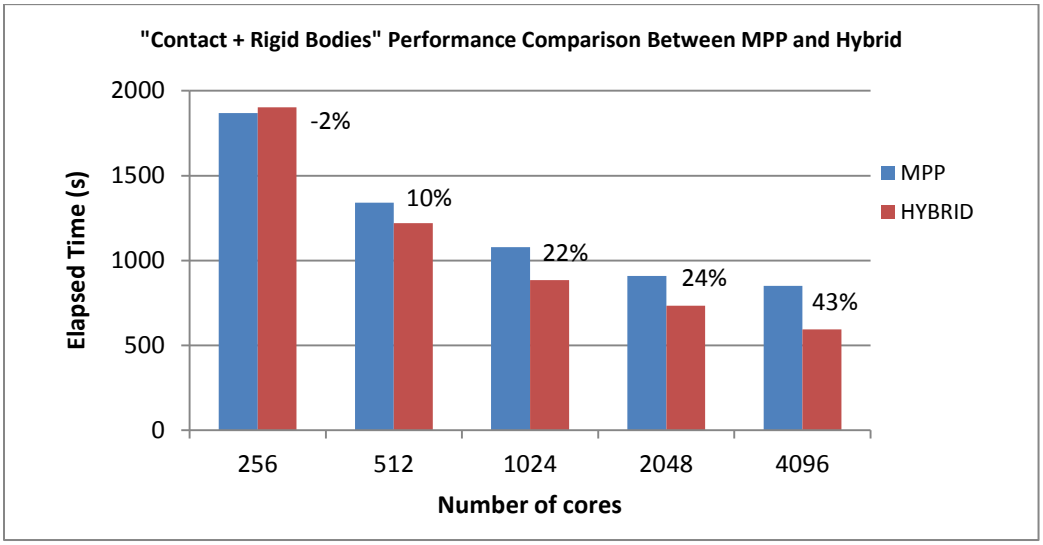

Figure 7. "Contact and rigid bodies" performance comparison between MPP and hybrid in car2car on an XC30 system

In the next section, we will explore the effect of the number of OpenMP threads on hybrid LS-DYNA performance at high core counts.

The effect of the Number of OpenMP Threads on Hybrid LS-DYNA Performance

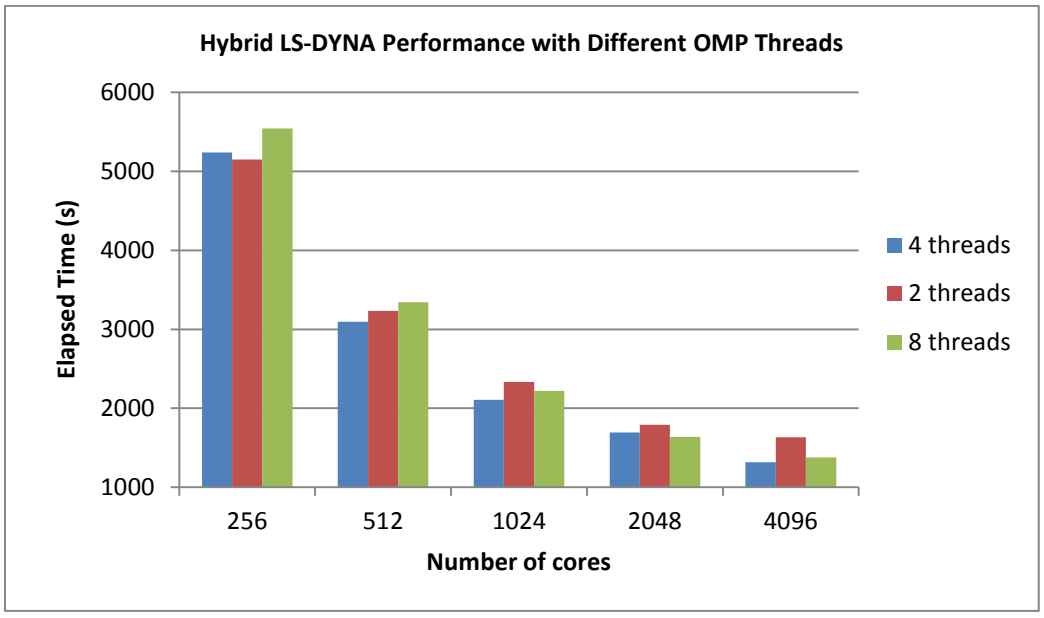

Figure 8. Hybrid LS-DYNA performance comparisons among different OpenMP threads in car2car on an XC30 system

In this study, we will keep the total number of cores, which is equal to MPI ranks × OpenMP threads, the same while varying MPI ranks and OpenMPI threads. For example, for 256 cores, we use three different combinations: 128 MPI ranks × 2 OpenMP threads, 64 MPI ranks × 4 OpenMP threads, and 32 MPI ranks × 8 OpenMP threads. Figure 8 depicts the performance difference among the three different OpenMP threads. At 256 cores, using two threads is 1.5 percent faster than using four threads and it is also 7 percent faster than using eight threads. At 2,048 cores, using eight threads is about 3 percent faster than using four threads, and it is also 9 percent faster than using two threads. At other core counts, using four threads is 5 percent to 23 percent faster than using two threads and it is also 5 percent to 8 percent faster than using eight threads. So we recommend the use of four OpenMP threads for hybrid LS-DYNA at high core counts.

The Effect of Intel Hyper-Threading Technology and Linux Huge Pages on LS-DYNA Performance

The Effect of Intel Hyper-Threading Technology on LS-DYNA Performance

Intel Hyper-Threading Technology is enabled on the Cray XC30 system's BIOS. We run pure MPP LS-DYNA with two threads per core in car2car for core counts from 256 to 2,048. We find that using two threads per core is about 5 percent to 29 percent slower than using only one thread per core for the same core counts. So we do not recommend using two threads per core for pure MPP LS-DYNA at high core counts.

The Effect of Linux Huge Pages on LS-DYNA Performance

Cray XC30 software allows the use of different sizes of Linux huge pages. The default Linux huge page size is 4 KB. We test pure MPP LS-DYNA in car2car using 2 MB Linux huge pages at core counts of 256 to 4,096, but observe no noticeable performance advantage. This is because LS-DYNA is a cache-friendly code.

Conclusion

Pure MPP LS-DYNA scaling stops at 2,048 cores on the Cray XC30 supercomputer. The limiting factor is load imbalance in the LS-DYNA contact algorithm. Hybrid LS-DYNA extends LS-DYNA scaling to 4,096 cores and perhaps beyond on the XC30 system. Using four OpenMP threads in general yields close to

the best or the best performance. Using two threads per core for LS-DYNA is not recommended on the CRAY XC30 system. Using large Linux huge pages has no effect on LS-DYNA performance.

Acknowledgement

The authors would like to thank Cray Inc. for their support on this work.