

Increasing LS-DYNA[®] Productivity on SGI Systems: A Step-by-Step Approach

Olivier Schreiber, Tony DeVarco, Scott Shaw, Aaron Altman

SGI

Abstract

SGI delivers a unified compute, storage and remote visualization solution to our manufacturing customers that reduces overall system management requirements and costs. LSTC has now integrated Explicit, Implicit solver technologies into a single hybrid code base allowing seamless switching from large time steps transient dynamics to linear statics and normal modes analysis. There are multiple computer architectures available from SGI to run LS-DYNA. They can all run LSTC solvers using Shared Memory Parallelism (SMP), Distributed Memory Parallelism (DMP) and their combination (Hybrid Mode) as supported by LS-DYNA. Because computer resources requirements are different for Explicit and Implicit solvers, this paper will study how advanced SGI computer systems, ranging from multi-node Distributed Memory Processor clusters to Shared Memory Processor servers address the computer resources used and what tradeoffs are involved. This paper will also outline the SGI hardware and software components for running LS-PrePost[®] via SGI VizServer with NICE Software. CAE engineers, at the departmental level, can now allow multiple remote users create, collaborate, test, optimize, and verify new complex LS-DYNA simulations in a single system and without moving their data.

1.0 About SGI Systems

SGI systems used to perform the benchmarks outlined in this paper include the SGI[®] Rackable[®] standard depth cluster, SGI[®] ICE[™] X integrated blade cluster and the SGI[®] UV[™] 2000 shared memory system. They are the same servers used to solve some of the world's most difficult computing challenges. Each of these server platforms support LSTC LS-DYNA with its Shared Memory Parallel (SMP) and Distributed Memory Parallel (DMP) modes [1].

1.1 SGI[®] Rackable[®] Standard-Depth Cluster

SGI Rackable standard-depth, rackmount C2112-4RP4 servers support up to 512GB of memory per server in a dense architecture with up to 96 cores per 2U with support for up to 56 GB/s, FDR and QDR InfiniBand, twelve-core Intel[®] Xeon[®] processor E5-2600 v2 series and DDR3 memory running SUSE[®] Linux[®] Enterprise Server or Red Hat[®] Enterprise Linux Server for a reduced TCO (Figure 1).

SGI Rackable C2112-4RP4 configuration used in this paper:

- Intel[®] Xeon[®] 12-core 2.7 GHz E5-2697 v2
- Mellanox[®] Technologies ConnectX[®] Industry standard Infiniband FDR
- 5 GB RAM/core Memory Speed 1867MHz
- Altair[®] PBS Professional Batch Scheduler v11
- SLES or RHEL, SGI Performance Suite with Accelerate[™]
- Scratch file system was RAM (/dev/shm)



Figure 1: Overhead View of SGI Rackable Server with the Top Cover Removed

1.2 SGI[®] ICE[™] X System

SGI[®] ICE[™] X is one of the world's fastest commercial distributed memory supercomputer. This performance leadership is proven in the lab and at customer sites including the largest and fastest pure compute InfiniBand cluster in the world. The system can be configured with compute nodes comprising Intel[®] Xeon[®] processor E5-2600 v2 series exclusively or with compute nodes comprising both Intel[®] Xeon[®] processors and Intel[®] Xeon Phi[™] coprocessors or Nvidia[®] compute GPU's. Running on SUSS[®] Linux[®] Enterprise Server and Red Hat[®] Enterprise Linux, SGI ICE X can deliver over 172 teraflops per rack and scale from 36 to tens of thousands of nodes.

SGI ICE X is designed to minimize system overhead and communication bottlenecks, and offers, for example the highest performance and scalability, up to 4,096 cores processing in parallel for ANSYS Fluent computer fluid dynamics (CFD) or above 2000 cores for LS-DYNA topcrunch.org benchmarks with top-most positions six years running.

SGI ICE X can be architected in a variety of topologies with choice of switch and single or dual plane FDR Infiniband interconnect. The integrated bladed design offers rack-level redundant power and cooling via air, warm or cold water and is also available with storage and visualization options (Figure 2).

SGI ICE X configuration used in this paper:

- Intel[®] Xeon[®] 10-core 3.0 GHz E5-2690 v2
- Mellanox[®] Technologies ConnectX[®] Industry standard Infiniband FDR integrated interconnect Hypercube
- 3 GB of Memory/core Memory Speed 1867MHz
- Altair[®] PBS Professional Batch Scheduler v11
- SLES or RHEL, SGI Performance Suite with Accelerate[™]

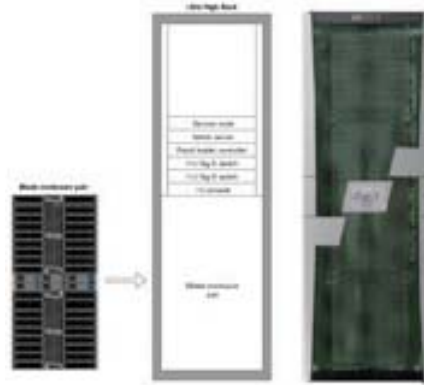


Figure 2: SGI ICE X Cluster with Blade Enclosure

1.3 SGI[®] UV[™] 2000

SGI UV 2000 server comprises up to 256 sockets (2,048 cores), with architectural support for 32,768 sockets (262,144 cores). Support for 64TB of global shared memory in a single system image enables efficiency of SGI UV for applications ranging from in-memory databases, to diverse sets of data and compute-intensive HPC applications all the while programming via the familiar Linux OS [2], without the need for rewriting software to include complex communication algorithms. TCO is lower due to one-system administration needs. Workflow and overall time to solution is accelerated by running Pre/Post-Processing, solvers and visualization on one system without having to move data (Figure 3).

Job memory is allocated independently from cores allocation for maximum multi-user, heterogeneous workload environment flexibility. Whereas on a cluster, problems have to be decomposed and require many nodes to be available, the SGI UV can run a large memory problem on any number of cores and application license availability with less concern of the job getting killed for lack of memory resources compared to a cluster.

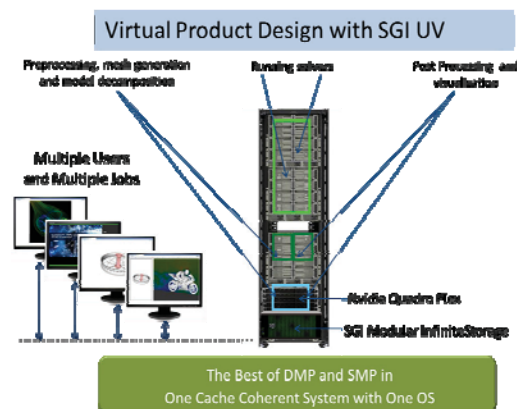


Figure 3: SGI UV CAE workflow running LSTC applications

SGI UV 2000 configuration used in this paper:

- 64 sockets (512 cores) per rack
- Intel[®] Xeon[®] 8 core 3.3 GHz E5-4627 v2
- SGI NUMALink[®] 6 Interconnect
- 4 GB of RAM/core Memory Speed 1867 MHz
- Altair[®] PBS Professional Batch Scheduler with CPUSET MOM v11
- SLES or RHEL, SGI Performance Suite with Accelerate[™]

1.4 SGI Performance tools

SGI[®] Performance Suite (Figure 4) takes Linux performance software to the next level. While hardware and processor technology continue to scale, managing software performance has become increasingly complex. SGI continues to extend technical computing performance for large scale servers and clusters. SGI Performance Suite incorporates the most powerful features and functionality from SGI[®] ProPack[™] 7, combined with several new tools and enhancements, and new, more flexible product packaging which allows you to purchase only the component or components that you need. For detailed information: <http://www.sgi.com/products/software/>

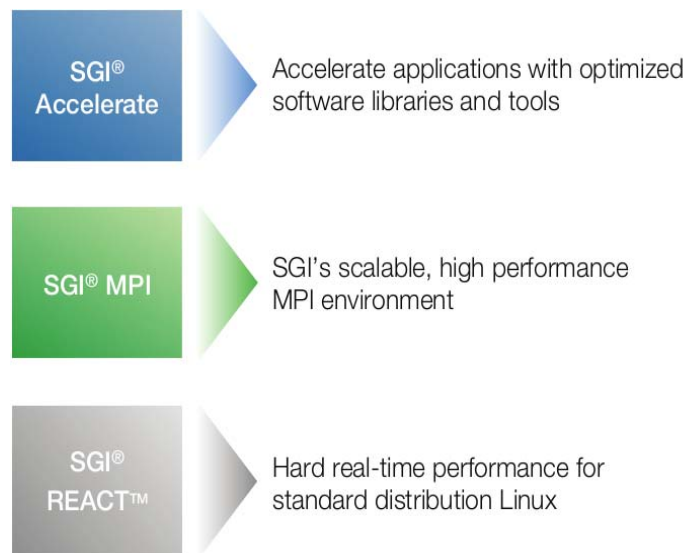


Figure 4: SGI Performance Suite Components

1.5 SGI System Management tools

SGI Management Center (Figure 5) provides a powerful yet flexible interface through which to initiate management actions and monitor essential system metrics for all SGI systems. It reduces the time and resources spent administering systems by improving software maintenance

procedures and automating repetitive tasks ultimately lowering total cost of ownership, increasing productivity, and providing a better return on the customer's technology investment. SGI Management Center is available in multiple editions which tailor features and capabilities to the needs of different administrators, and makes available optional features that further extend system management capabilities. For detailed information: <http://www.sgi.com/products/software/smc.html>

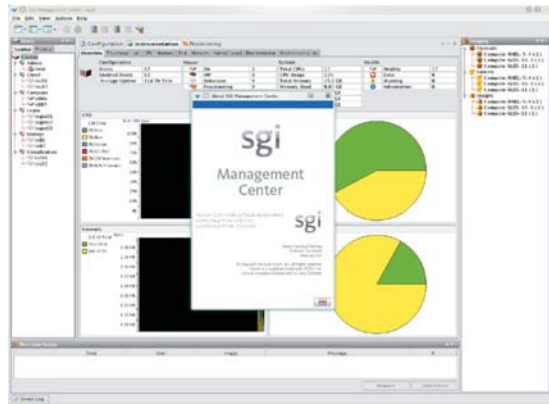


Figure 5: SGI Management Center Web Interface

1.6 Resource and Workload Scheduling

Resource and workload scheduling allows one to manage large, complex applications, dynamic and unpredictable workloads, and optimize limited computing resources. SGI offers several solutions that customers can choose from to best meet their needs.

Altair Engineering PBS Professional[®] is SGI's preferred workload management tool for technical computing scaling across SGI's clusters and servers. PBS Professional is sold by SGI and supported by both Altair Engineering and SGI. Features:

- Policy-driven workload management which improves productivity, meets service levels, and minimizes hardware and software costs
- Integrated operation with SGI Management Center for features such as workload-driven, automated dynamic provisioning

1.7 SGI[®] VizServer[®] with NICE DCV

SGI[®] VizServer[®] with NICE DCV gives technical users remote 3D modeling tools through a web-based portal, allowing for GPU and resource sharing and secure data storage. (Figure 6)

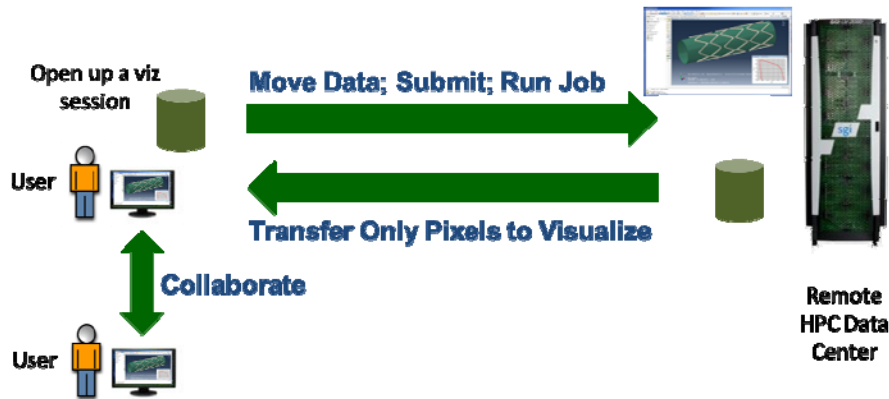


Figure 6: VizServer workflow

SGI[®] VizServer[®] with NICE DCV installed on a company's servers can provide LS-PrePost remote visualization capabilities through a software-as-a-service (SaaS) built in the company's private network. The LS-PrePost software is accessed through an easy-to-use web interface, resulting in simplicity for the end user. This solution provides intuitive help and guidance to ensure that less-experienced users can maximize productivity without being hindered by complex IT processes.

SGI[®] VizServer[®] with NICE DCV Components:

- **Engineer-friendly self-service portal:** The self-service portal enables engineers to access the LS-PrePost application and data in a web browser-based setting. It also provides security, monitoring, and management to ensure that users cannot leak company data and that IT managers can track usage. Engineers access the LS-PrePost application and data directly from their web browsers, with no need for a separate software installation on their local client.
- **Resource control and abstraction layer:** The resource control and abstraction layer lies underneath the portal, not visible to end users. It handles job scheduling, remote visualization, resource provisioning, interactive workloads, and distributed data management without detracting from the user experience. This layer translates the user request from the browser and facilitates the delivery of resources needed to complete the visualization or HPC tasks. This layer has a scalable architecture to work on a single cluster or server, as well as a multi-site WAN implementation.
- **Computational and storage resources:** The SGI[®] VizServer[®] with NICE DCV software takes advantage of the company's existing or newly purchased SGI industry-standard resources, such as servers, HPC schedulers, memory, graphical processing units (GPUs), and visualization servers, as well as the required storage to host application binaries, models and intermediate results. These are all accessed through the web-based portal via the resource control and abstraction layer and are provisioned according to the end user's needs by the middle layer.

The NICE DCV and EnginFrame software is built on common technology standards. The software adapts to network infrastructures so that an enterprise can create its own secure engineering cloud without major network upgrades. The software also secures data, removing the need to transfer it and stage it on the workstation, since both technical applications and data

stay in the private cloud or data center. These solutions feature the best characteristics of cloud computing—simple, self-service, dynamic, and scalable, while still being powerful enough to provide 3D visualization as well as HPC capabilities to end users, regardless of their location.

2.0 LS-DYNA

2.1 Versions used

LS-DYNA/MPP ls971 R3.2.1 or later. At R4.2.1, coordinate arrays were coded to double precision for the simulation of finer time-wise phenomena thus incurring a decrease in performance of 25% (neon) to 35% (car2car).

Compilers: Fortran: Intel Fortran Compiler 11.1 for EM64T-based applications

MPI: P-MPI, Intel MPI, Open MPI, SGI MPI

2.2 Parallel Processing capabilities of LS-DYNA

2.2.1 Underlying hardware and Software Notions

It is important to distinguish hardware components of a system and the actual computations being performed using them. On the hardware side, one can identify:

1. Cores, the Central Processing Units (CPU) capable of arithmetic operations.
2. Processors, the four, six, eight, ten or twelve core socket-mounted devices.
3. Nodes, the hosts associated with one network interface and address.

With current technology, nodes are implemented on boards in a chassis or blade rack-mounted enclosure. The board may comprise two sockets or more.

From the software side, one can identify:

1. Processes: execution streams having their own address space.
2. Threads: execution streams sharing address space with other threads.

Therefore, it is important to note that processes and threads created to compute a solution on a system will be deployed in different ways on the underlying nodes through the processors and cores' hardware hierarchy.

Note: Software processes or threads don't necessarily map one to one to hardware cores, and can also under or over-subscribe them.

2.1.2 Parallelism Background

Parallelism in scientific/technical computing exists in two paradigms implemented separately but sometimes combined in 'hybrid' codes: Shared Memory Parallelism (SMP) appeared in the 1980's with the strip mining of 'DO loops' and subroutine spawning via memory-sharing

threads. In this paradigm, parallel efficiency is affected by the relative importance of arithmetic operations versus data access referred to as ‘DO loop granularity.’ In the late 1990’s, Distributed Memory Parallelism (DMP) Processing was introduced and proved very suitable for performance gains because of its coarser grain parallelism design. It consolidated on the MPI Application Programming Interface. In the meantime, Shared Memory Parallelism saw adjunction of mathematical libraries already parallelized using efficient implementation through OpenMP™ (Open Multi-Processing) and Pthreads standard API’s.

Both SMP and DMP programs are run on the two commonly available hardware system levels:

- Shared Memory systems or single nodes with multiple cores sharing a single memory address space.
- Distributed Memory systems, otherwise known as clusters, comprised of nodes with separate local memory address spaces.

Note: While SMP programs cannot execute across clusters because they cannot handle communication between their separate nodes with their respective memory spaces, inversely, DMP programs can be used perfectly well on a Shared Memory system. Since DMP has coarser granularity than SMP, it is therefore preferable, on a Shared Memory system, to run DMP rather than SMP despite what the names may imply at first glance. SMP and DMP processing may be available combined together, in ‘hybrid mode’.

2.1.3 Distributed Memory Parallel implementations

Distributed Memory Parallel is implemented through the problem at hand with domain decomposition. Depending on the physics involved in their respective industry, the domains could be geometry, finite elements, matrix, frequency, load cases or right hand side of an implicit method. Parallel inefficiency from communication costs is affected by the boundaries created by the partitioning. Load balancing is also important so that all MPI processes perform the same number of computations during the solution and therefore finish at the same time. Deployment of the MPI processes across the computing resources can be adapted to each architecture with ‘rank’ or ‘round-robin’ allocation.

2.1.4 Parallelism Metrics

Amdahl’s Law, ‘Speedup yielded by increasing the number of parallel processes of a program is bounded by the inverse of its sequential fraction’ is also expressed by the following formula (where P is the program portion that can be made parallel, 1-P is its serial complement and N is the number of processes applied to the computation):

$$\text{Amdahl Speedup} = 1 / [(1-P) + P/N]$$

A derived metric is: Efficiency = Amdahl Speedup / N

A trend can already be deduced by the empirical fact that the parallelizable fraction of an application is depends more on CPU speed, and the serial part, comprising overhead tasks depends more on RAM speed or I/O bandwidth. Therefore, a higher CPU speed system will have a larger 1-P serial part and a smaller P parallel part causing the Amdahl Speedup to decrease. This can lead to a misleading assessment of different hardware configurations as shown by this example where, say System B has faster CPU speed than system A:

N	System a elapsed seconds	System B elapsed seconds
1	1000	810
10	100	90
Speedup	10	9

System A and System B could show parallel speedups of 10 and 9, respectively, even though System B has faster raw performance across the board. Normalizing speedups with the slowest system serial time remedies this problem:

Speedup	10	11.11
---------	----	-------

A computational process can exhibit:

- Strong scalability: Decreasing execution time on a particular dataset when increasing processes count.
- Weak scalability. Keeping execution time constant on ever larger datasets when increasing processes count.

It may be preferable, in the end, to use a throughput metric, especially if several jobs are running simultaneously on a system:

$$\text{Number of jobs/hour/system} = 3600/(\text{Job elapsed time})$$

The system could be a chassis, rack, blade, or any hardware provisioned as a whole unit.

2.3 Parallel execution Control

2.3.1 Submittal Procedure

Submittal procedure must ensure:

1. Placement of processes and threads across nodes and also sockets within nodes
2. Control of process memory allocation to stay within node capacity
3. Use of adequate scratch files across nodes or network

Batch schedulers/resource managers dispatch jobs from a front-end login node to be executed on one or more compute nodes so the following is a possible synoptic of a job submission script:

1. Change directory to the local scratch directory on the first compute node allocated by the batch scheduler.

2. Copy all input files over to this directory.
3. Create parallel local scratch directories on the other compute nodes allocated by the batch scheduler.
4. Launch application on the first compute node. The executable may itself carry out propagation and collection of various files between launch node and the others at start, and end of the main analysis execution. The launch script may also asynchronously sweep output files like d3plot* files to free up scratch directory.

2.3.2 Run Command with MPI tasks and OpenMP thread allocation across nodes and cores

For LS-DYNA, the deployment of processes, threads and associated memory is achieved with the following keywords in execution command [1]:

- -np: Total number of MPI processes used in a Distributed Memory Parallel job.
- ncpu=: number of SMP OpenMP threads
- memory, memory2: Size in words of allocated RAM for MPI processes. (A word is 4 or 8 bytes long for single or double precision executables, respectively.)

2.4 Tuning

2.4.1 Input/Output and Memory

To achieve the best runtime in a batch environment, disk access to input and output files should be placed on the high performance filesystem closest to the compute node. The high performance filesystem could be an in-memory filesystem (/dev/shm), a Direct (DAS) or Network (NAS) Attached Storage filesystem. In diskless computing environments, in-memory filesystem or Network Attached Storage are the only options. In cluster computing environments with a Network Attached Filesystem (NAS), isolating application MPI communications and NFS traffic will provide the best NFS I/O throughput for scratch files. The filesystem nomenclature is illustrated in Figure 7.

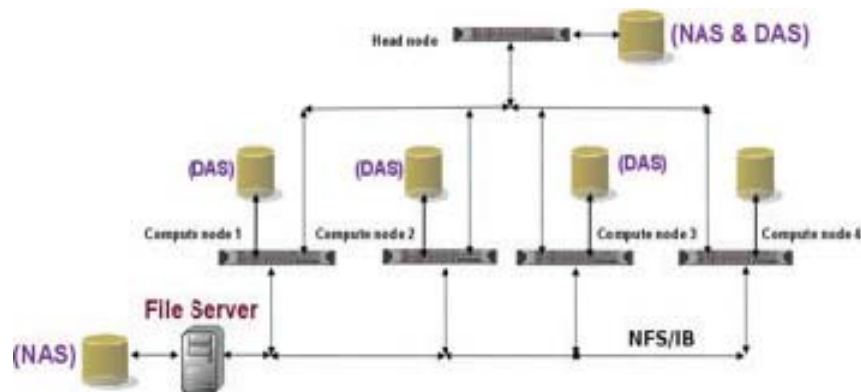


Figure 7: Example filesystems for Scratch Space

Having more memory per core will increase performance since it can be allocated for the analysis as well as the Linux kernel buffer cache to improve I/O efficiency. SGI's Flexible File I/O (FFIO) is a link-less library (which means it does not need to be linked to the application) bundled with SGI Accelerate. It implements user defined I/O buffer caches to avoid the operating system ones from thrashing when running multiple I/O intensive jobs or processes. This can be effective in Shared Memory Parallel systems or cluster computing environments using DAS or NAS storage subsystems. FFIO isolates user page caches so jobs or processes do not contend for Linux Kernel page cache. Hence, FFIO minimizes the number of system calls and I/O operations as echoed back by the `ie_close` sync and `async` values reflecting synchronous calls to disk—which should be as close to 0 as possible—to and from the storage subsystem and improves performance for large and I/O intensive jobs. (Ref [1], Chapter 7 Flexible File I/O).

2.4.2 Using only a subset of available cores on dense processors

Two ways of looking at computing systems are either through nodes which are their procurement cost sizing blocks or through cores which are their throughput sizing factors. When choosing node metrics, because processors have different prices, clock rates, core counts and memory bandwidth, optimizing for turnaround time or throughput will depend on running on all or a subset of cores available. Since licensing charges are assessed by the number of threads or processes being run as opposed to the actual number of physical cores present on the system, there is no licensing cost downside in not using all cores available so this may provide performance enhancement possibilities. The deployment of threads or processes across partially used nodes should be done carefully with consideration to the existence of shared resources among cores.

2.4.3 Hyper-threading

Hyper-threading (HT) is a feature of the Intel[®] Xeon[®] processor which can increase performance for multi-threaded or multi-process applications. It allows a user to run twice the number of OpenMP threads or MPI processes than available physical cores per node (over-subscription).

Beyond 2 nodes, with LS-DYNA, Hyper-threading gains are negated by added communication costs between the doubled numbers of MPI processes.

2.4.4 Intel[®] Turbo Boost

Intel[®] Turbo Boost is a feature of the Intel[®] Xeon[®] processor, for increasing performance by raising the core operating frequency within controlled limits constrained by thermal envelope. The mode of activation is a function of how many cores are active at a given moment when MPI processes, OpenMP or Pthreads are running. At best, Turbo Boost improves performance for low numbers of cores used, up to the ratio of the maximum frequency over baseline value. As more cores are used, Turbo Boost cannot increase the frequencies on all of them as it can on fewer

active ones. For example, for a base frequency of 3.0GHz, when 1-2 cores are active, core frequencies might be throttled up to 3.3GHz, but with 3-4 cores active, frequencies may be throttled up only to 3.2 GHz. For computational tasks, utilizing Turbo Boost often results in improved runtimes so it is best to leave it enabled, although the overall benefit may be mitigated by the presence of other performance bottlenecks outside of the arithmetic processing.

2.4.5 SGI Performance suite MPI, PerfBoost

The ability to bind an MPI rank to a processor core is key to control performance on the multiple node/socket/core environments available. From [3], ‘3.1.2 Computation cost-effects of CPU affinity and core placement [...]HP-MPI currently provides CPU-affinity and core-placement capabilities to bind an MPI rank to a core in the processor from which the MPI rank is issued. Children threads, including SMP threads, can also be bound to a core in the same processor, but not to a different processor; additionally, core placement for SMP threads is by system default and cannot be explicitly controlled by users.[...]’.

In contrast, SGI MPI, through its ‘omplace’ option enforces accurate placement of Hybrid MPI processes, OpenMP threads and Pthreads within each node. SGI MPI’s bundled PerfBoost facility linklessly translates P-MPI, IntelMPI, OpenMPI calls on the fly to SGI MPI calls.

2.4.6 SGI Accelerate LibFFiO

LS-DYNA/MPP/Explicit is not I/O intensive and placement can be handled by SGI MPI, therefore, libFFiO is not necessary. However, LS-DYNA/MPP/Implicit does involve I/O so libFFiO can compensate for bandwidth contention on NAS or slow drive systems.

3.0 Benchmarks description

The benchmarks used are the three TopCrunch (<http://www.topcrunch.org>) dataset--created by National Crash Analysis Center (NCAC) at George Washington University. The TopCrunch project was initiated to track aggregate performance trends of high performance computer systems and engineering software. Instead of using a synthetic benchmark, an actual engineering software application, LS-DYNA/Explicit, is used with real data. Since 2008, SGI has held top performing positions on the three datasets. The metric is: Minimum Elapsed Time and the rule is that all cores for each processor must be utilized.

LS-DYNA/Implicit [4], [5] has been covered in [7][8][9][10].

3.1 Neon Refined Revised

Vehicle based on 1996 Plymouth Neon crashing with an initial speed 31.5 miles/hour, (Figure 8). The model comprises 535k elements, 532,077 shell elements, 73 beam elements, 2,920 solid elements, 2 contact interfaces, 324 materials. The simulation time is 30 ms (29,977 cycles) and writes 68,493,312 Bytes d3plot and 50,933,760 Bytes d3plot [01-08] files at 8 time steps from start to end point (114MB).

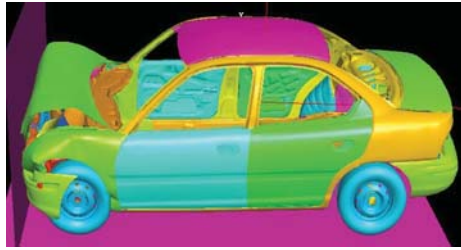


Figure 8: Neon Refined Revised

3.2 Three Vehicle collision

Van crashing into the rear of a compact car, which, in turn, crashes into a midsize car (Figure 9) with a total model size of 794,780 elements, 785,022 shell elements, 116 beam elements, 9,642 solid elements, 6 contact interfaces, 1,052 materials, and a simulation time of 150 ms (149,881 cycles), writing 65,853,440 Bytes d3plot and 33,341,440 Bytes d3plot[01-19] files at 20 time steps from start to end point (667MB). The 3cars model is difficult to scale well: most of the contact work is in two specific areas of the model, and so is hard to evenly spread that work out across a large number of processes. Particularly as the "active" part of the contact (which part is crushing the most) changes with time, so the computational load of each process will change with time.



Figure 9: Three Vehicle Collision

4.3 car2car

Angled 2 vehicle collision (Figure 10). The vehicle models are based on NCAC minivan model with 2.5 million elements. The simulation writes 201,854,976 Bytes d3plot and 101,996,544 Bytes d3plot[01-25] files at 26 time steps from start to end point (2624MB).

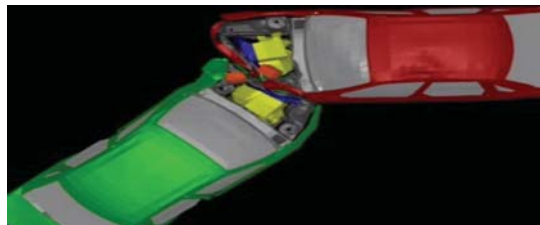


Figure 10: Car2car

5. Results

5.1 Introduction

Traditionally, benchmarking has been concerned with turnaround time as seen in following section.

5.2 Minimizing turnaround times

5.2.1 Looking up results on TopCrunch.org

Shortest elapsed times are frequently posted on topcrunch.org. To access them:

A) Browse TopCrunch.org, select 'Results', and use pull downs to select as shown on figure 11:

1. neon_refined_revised, (*not* the obsolete neon_refined),or
2. 3 Vehicle Collision or
3. car2car

B) Click on Search for each dataset as shown on figure 11.

C) Compare results across vendors, computer models, processors, interconnects and number of cores as shown in following sections.

Figure 11: TopCrunch.org menus pull downs

5.2.2 Neon_refined_revised

For same number of cores and processes (640), SGI ICE X presents 20% better performance over first competing entry. (Figure 12)

Vendor/Submitter Org.	Computer/Interconnect	Processor	#Nodes x #Processors per Node x #Cores Per Processor = Total #CPU	Time (Sec)	Benchmark Problem	Submission Date
SGI®/ HPC Applications Support	SGI® ICE-X/IB FDR	Intel® Xeon® E5-2690 v2 @3.00GHz Turbo Enabled	32 x 2 x 10 = 640	37	neon_refined_revised	02/26/2014
SGI/Applications Engineering	Altix ICE 8400EX?/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	96 x 2 x 6 = 1152	43	neon_refined_revised	09/25/2011
Dell/HPC Advisory Council	Dell PowerEdge R720xd/Mellanox Technologies Connect-IB FDR InfiniBand	Intel Xeon E5-2680 V2 @2.80GHz	32 x 2 x 10 = 640	46	neon_refined_revised	02/13/2014
SGI/Applications Engineering	Altix ICE 8400EX?/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	64 x 2 x 6 = 768	46	neon_refined_revised	09/25/2011
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	84 x 2 x 6 = 1008	47	neon_refined_revised	11/10/2010
SGI/Applications Engineering	Altix ICE 8400EX?/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	32 x 2 x 6 = 384	52	neon_refined_revised	09/25/2011
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	42 x 2 x 6 = 504	53	neon_refined_revised	11/10/2010
Dell/HPC Advisory Council	Dell PowerEdge R720xd/Mellanox Technologies Connect-IB FDR InfiniBand	Intel Xeon E5-2680 V2 @2.80GHz	16 x 2 x 10 = 320	55	neon_refined_revised	02/13/2014
SGI®/HPC Applications Support	SGI® Rackable CH-C2112 cluster/IB QDR	Intel® Xeon® E5-2670 @2.60GHz Turbo Enabled	16 x 2 x 8 = 256	60	neon_refined_revised	10/07/2012

Figure 12: [neon_refined_revised](#) TopCrunch.org results page

5.2.3 3cars

For lower number of cores and processes (560 vs 640), SGI ICE X presents 11% better performance over first competing entry. (Figure 13)

Vendor/Submitter Org.	Computer/Interconnect	Processor	#Nodes x #Processors per Node x #Cores Per Processor = Total #CPU	Time (Sec)	Benchmark Problem	Submission Date
SGI®/HPC Applications Support	SGI® ICE-X/IB FDR	Intel® Xeon® E5-2690 v2 @3.00GHz Turbo Enabled	28 x 2 x 10 = 560	348	3_Vehicle Collision	02/26/2014
SGI/Applications Engineering	Altix ICE 8400EX?/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	128 x 2 x 6 = 1536	375	3_Vehicle Collision	09/25/2011
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	85 x 2 x 6 = 1020	387	3_Vehicle Collision	11/10/2010
SGI/Applications Engineering	Altix ICE 8400EX?/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	64 x 2 x 6 = 768	388	3_Vehicle Collision	09/25/2011
SGI/Applications Engineering	Altix ICE 8400EX?/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	64 x 2 x 6 = 768	388	3_Vehicle Collision	09/25/2011
Dell/HPC Advisory Council	Dell PowerEdge R720xd/Mellanox Technologies Connect-IB FDR InfiniBand	Intel Xeon E5-2680 V2 @2.80GHz	32 x 2 x 10 = 640	390	3_Vehicle Collision	02/13/2014
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	64 x 2 x 6 = 768	397	3_Vehicle Collision	11/10/2010
SGI®/HPC Applications Support	SGI® Rackable CH-C2112 cluster/IB QDR	Intel® Xeon® E5-2670 @2.60GHz Turbo Enabled	32 x 2 x 8 = 512	431	3_Vehicle Collision	10/07/2012
SGI/Applications Engineering	Altix ICE8200EX/IP95 Blades with Mellanox ConnectX IB HCA DDR Fabric OFED v1.4	Intel® Xeon® Quad Core X5570 2.93GHz Turbo Boost E	128 x 2 x 4 = 1024	441	3_Vehicle Collision	03/31/2009
SGI/Applications Engineering	Altix ICE8200EX/IP95 Blades with Mellanox ConnectX IB HCA DDR Fabric OFED v1.4	Intel® Xeon® Quad Core X5570 2.93GHz Turbo Boost E	64 x 2 x 4 = 512	514	3_Vehicle Collision	03/31/2009
SGI/Applications Engineering	Altix ICE 8400EX?/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	32 x 2 x 6 = 384	519	3_Vehicle Collision	09/25/2011
Dell/HPC Advisory Council	Dell PowerEdge R720xd/Mellanox Technologies Connect-IB FDR InfiniBand	Intel Xeon E5-2680 V2 @2.80GHz	16 x 2 x 10 = 320	522	3_Vehicle Collision	02/13/2014
BULL/BULL	bullx blade system (B510)/IB QDR	Intel® Xeon® E5-2680 @2.70GHz Turbo Enabled	32 x 2 x 8 = 512	530	3_Vehicle Collision	07/04/2012
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	32 x 2 x 6 = 384	531	3_Vehicle Collision	11/10/2010
SGI/Applications Engineering	Rackable? C2112-4TY14/Mellanox® Technologies ConnectX-2® IB QDR MT26428	Intel® Xeon® Hexa Core X5675 3.07GHz	32 x 2 x 6 = 384	549	3_Vehicle Collision	09/25/2011
SGI/Applications Engineering	Altix ICE8200EX/Mellanox ConnectX IB HCA DDR Fabric OFED v1.3	Intel Xeon E5472 3.00GHz	128 x 2 x 4 = 1024	568	3_Vehicle Collision	04/08/2008
Intel/Intel/SSG	Sandy Bridge-EP system/IB FDR	Intel® Xeon® E5-2680 @2.70GHz	16 x 2 x 8 = 256	628	3_Vehicle Collision	07/09/2012

Figure 13: 3cars TopCrunch.org results page

5.2.4 car2car

For similar number of cores and processes (2000), SGI ICE X with Mellanox® industry standard Infiniband presents comparable performance with proprietary interconnects (Figure 14).

Vendor/Submitter Org.	Computer/Interconnect	Processor	#Nodes x #Processors per Node x #Cores Per Processor = Total #CPU	Time (Sec)	Benchmark Problem	Submission Date
CRAY Inc./Cray Inc.	CRAY XC30/Aries Interconnect	Intel Xeon E5-2690 v2 3.0 GHZ	150 x 2 x 10 = 3000	931	car2car	03/11/2014
CRAY Inc./Cray Inc.	CRAY XC30/Aries Interconnect	Intel Xeon E5-2690 v2 3.0 GHZ	100 x 2 x 10 = 2000	1112	car2car	03/13/2014
SGI®/HPC Applications Support	SGI® ICE-X/IB FDR	Intel® Xeon® E5-2690 v2 @3.00GHz Turbo Enabled	100 x 2 x 10 = 2000	1207	car2car	02/26/2014
CRAY Inc./Cray Inc.	CRAY XC30/Aries Interconnect	Intel Xeon E5-2690 v2 3.0 GHZ	75 x 2 x 10 = 1500	1315	car2car	03/14/2014
CRAY Inc./Cray Inc.	CRAY XC30/Aries Interconnect	Intel Xeon E5-2690 v2 3.0 GHZ	50 x 2 x 10 = 1000	1680	car2car	03/14/2014
SGI/Applications Engineering	Altix ICE 8400EX7/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	128 x 2 x 6 = 1536	1769	car2car	09/25/2011
SGI®/HPC Applications Support	SGI® Rackable CH-C2112 cluster/IB QDR	Intel® Xeon® E5-2670 @2.60GHz Turbo Enabled	64 x 2 x 8 = 1024	1887	car2car	10/07/2012
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	128 x 2 x 6 = 1536	1936	car2car	11/10/2010
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	85 x 2 x 6 = 1020	2039	car2car	11/10/2010
BULL/BULL	bullx blade system (B510)/IB QDR	Intel® Xeon® E5-2680 @2.70GHz Turbo Enabled	64 x 2 x 8 = 1024	2072	car2car	08/08/2012
SGI/Applications Engineering	Altix ICE8200EX/IP95 Blades with Mellanox ConnectX IB HCA DDR Fabric OFED v1.4	Intel® Xeon® Quad Core X5570 2.93GHz Turbo Boost E	128 x 2 x 4 = 1024	2316	car2car	03/31/2009
SGI/Applications Engineering	Altix ICE 8400EX7/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	64 x 2 x 6 = 768	2361	car2car	09/25/2011
SGI/Applications Engineering	Altix ICE 8400/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Six Core X5680 3.33GHz	64 x 2 x 6 = 768	2464	car2car	11/10/2010
BULL/BULL	bullx blade system (B510)/IB QDR	Intel® Xeon® E5-2680 @2.70GHz Turbo Enabled	32 x 2 x 8 = 512	2823	car2car	08/08/2012
SGI/Applications Engineering	Altix ICE8200EX/IP95 Blades with Mellanox ConnectX IB HCA DDR Fabric OFED v1.4	Intel® Xeon® Quad Core X5570 2.93GHz Turbo ON	64 x 2 x 4 = 512	3130	car2car	03/31/2009
SGI/Applications Engineering	Altix ICE 8400EX7/Mellanox® Technologies ConnectX-2® IB QDR	Intel® Xeon® Hexa Core X5690 3.47GHz	32 x 2 x 6 = 384	3646	car2car	09/25/2011
SGI/Applications Engineering	Rackable? C2112-4TY14/Mellanox® Technologies ConnectX-2® IB QDR MT26428	Intel® Xeon® Hexa Core X5675 3.07GHz	32 x 2 x 6 = 384	3814	car2car	09/25/2011
SGI/Applications Engineering	Altix XE1300/Mellanox® Technologies MT26428 ConnectX® IB QDR	Intel® Xeon® Six Core X5670 2.93GHz	32 x 2 x 6 = 384	4005	car2car	08/09/2010
Intel/Intel/SSG	Intel SR1600UR system/QDR IB	Intel® Xeon® Six Core X5670 2.93GHz	64 x 2 x 6 = 768	4009	car2car	03/28/2010
SGI/Applications Engineering	Altix XE1300/Mellanox® Technologies MT26428 ConnectX® IB QDR	Intel® Xeon® Six Core X5670 2.93GHz	30 x 2 x 6 = 360	4113	car2car	08/09/2010
IBM/IBM/Microsoft	System x® iDataPlex7 dx360 M2/ConnectX Infiniband	Intel® Xeon® Quad Core X5550	64 x 2 x 4 = 512	4196	car2car	05/28/2010
BULL/BULL	bullx blade system (B510)/IB QDR	Intel® Xeon® E5-2680 @2.70GHz Turbo Enabled	16 x 2 x 8 = 256	4294	car2car	08/08/2012

Figure 14: car2car TopCrunch.org results pages

5.2.5 Car2Car tuning

Figure 15 shows how going from Double to Single Precision when possible can affect performance. LS-DYNA's version chosen can also affect results as mentioned in 2.1 when going back to R3.2.1. Then adjusting the otherwise automatic decomposition can improve results [1]. Lastly, Turbo Boost as described in 2.4.4 and dual rail [6], further improve performance where last entry cumulates Single Precision, R3.2.1, custom decomposition, Turbo Boost mode and dual rail.

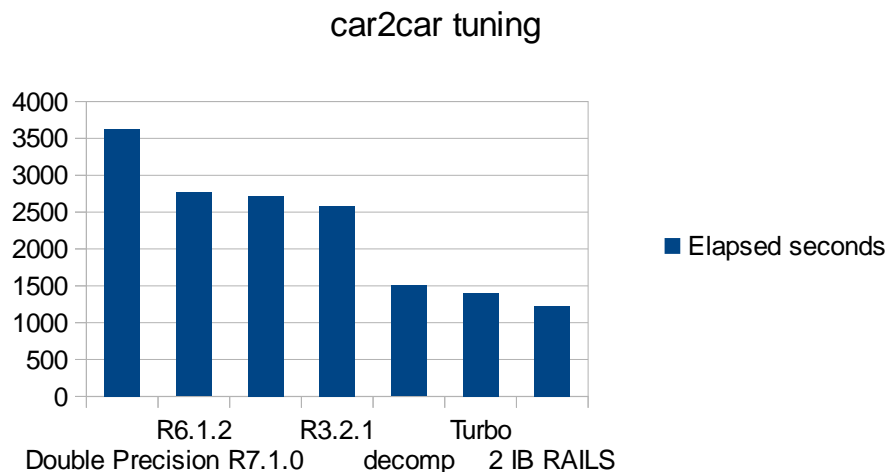


Figure 15: Car2car tuning

5.2.6 Car2Car resource usage

Linux collectl/colplot was first run on the first compute node to verify CPU usage and correct placement on physical cores instead of virtual ones since HyperThreading is not desired. Figure 16 shows full CPU utilization for cores 0,1 and 18,20 and none for subsequent virtual cores 20,21.



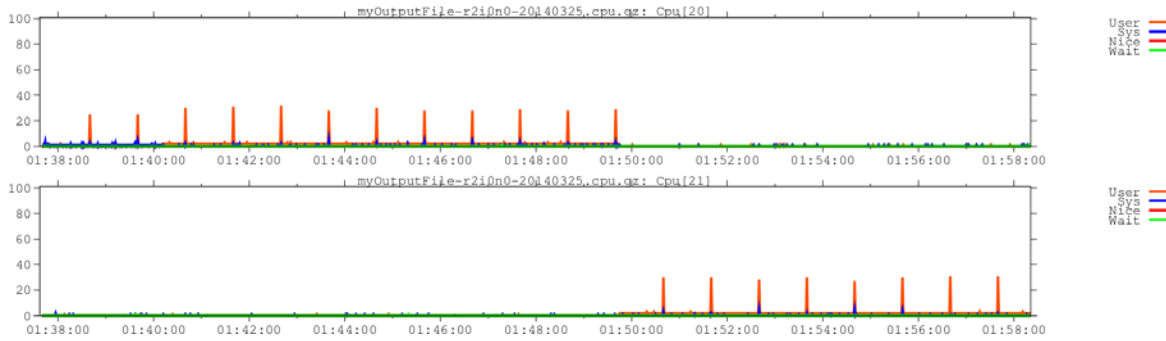


Figure 16: Core usage from 0 to 39

5.2.7

5.2.7 Car2Car MPI profiling

SGI MPInside [11] was run to get basic profiling and construct the area plot stack across all 2000 MPI processes attributed to computation time and MPI calls. Figure 17 shows elapsed seconds on the Y axis for the complete range of ranks 0 to 1999. the light purple, teal and dark blue bands indicate that across all ranks, a little less than half of the running time was compute, with the majority of the communication time spent in Bcast and Recv calls.

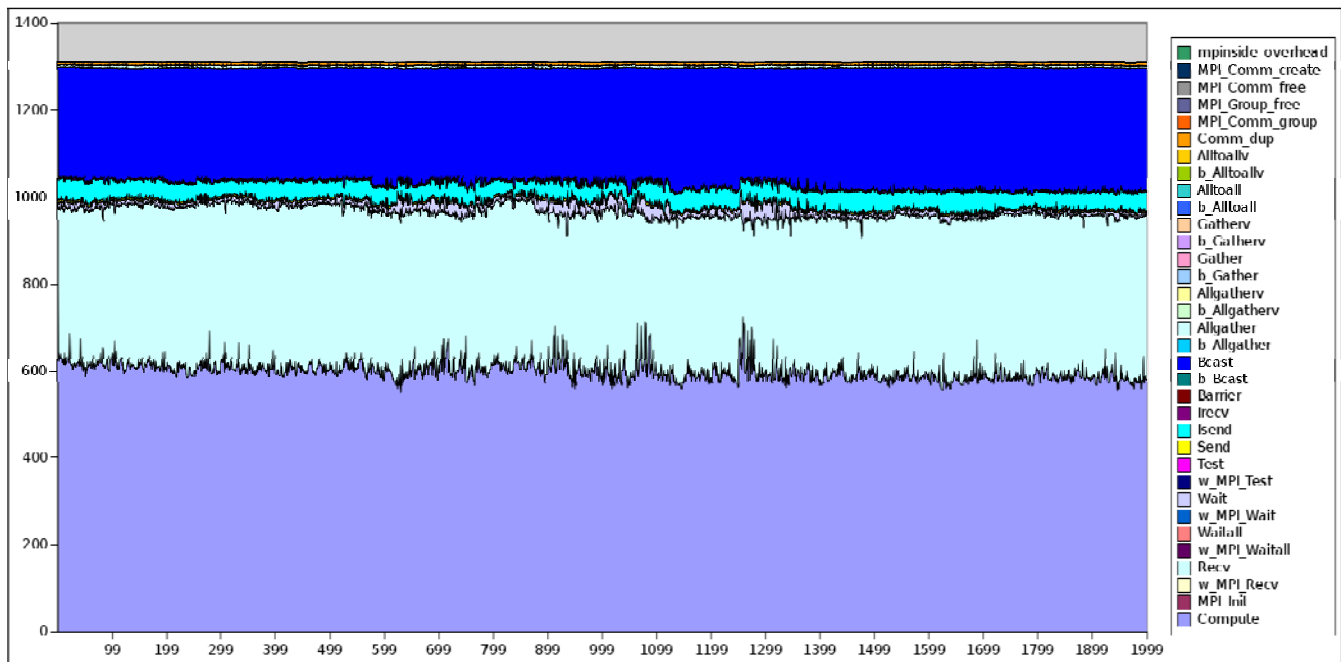


Figure 17: Cumulative area plot stack from SGI MPInside

The histogram of the request sizes distribution for rank first to last from which previous plot is available. Rank 1999 is shown on Figure 18 showing expensive Recv and Bcast in 128-256 and 0-32 bytes message length bands, respectively.

>>> Rank 1999 Sizes distribution <<<<

Sizes	Recv	Send	Isend	Irecv	Barrier	Bcast
65536	0	0	0	0	0	5260
32768	2455	0	0	0	0	2
16384	7145	0	0	0	0	114
8192	1801	0	78	0	0	11
4096	599	480	52	480	0	14
2048	247232	481402	105	241421	0	6
1024	1202253	721867	240035	721866	0	14
512	719944	960892	1679946	960891	0	25
256	9560	959955	7203	959951	0	3
128	967133	959935	242591	479972	0	21
64	2391	239990	491788	239989	0	1
32	2401	1	23956	0	0	12
0	3629915	4	3610941	0	3272	292692

>>> Rank 1999 Size distribution times<<<<

Sizes	Recv	Send	Isend	Irecv	Barrier	Bcast
65536	0	0	0	0	0	3.00101
32768	0.01090	0	0	0	0	0.00018
16384	0.03185	0	0	0	0	0.00626
8192	0.00384	0	0.00013	0	0	0.22016
4096	0.00112	0.00058	5E-05	0.0002	0	0.00012
2048	38.9114	0.48212	8.1E-05	0.12406	0	3.2E-05
1024	3.25882	0.61298	0.17572	0.94710	0	0.00224
512	0.52911	0.53767	1.83301	0.37697	0	0.00012
256	0.17519	0.50377	0.00457	0.59256	0	0.00043
128	307.803	1.04413	0.30725	0.14172	0	0.00561
64	0.04047	0.11488	0.52372	0.07279	0	48.5074
32	0.00169	0	0.03405	0	0	0.37118
0	22.2937	0.00038	40.7483	0	0.23487	302.809

Figure 18: Histogram of the request sizes and times distribution

The distribution of Bcast requests and times of messages in band 0-32 Bytes of 292692 requests for 303 seconds was used to target SGI MPI optimizations heuristics to improve time. As a result, the same Bcast requests saw their times reduced to, 284 for an overall elapsed time decrease from 1400 to 1367 seconds.

"Compute" time as measured by MPInside is the time that a given rank spent that wasn't attributable to a profiled MPI call. It could be time that the rank is busy doing something besides communication, possibly leading to increased total run time, or that rank could be blocked on an MPI call waiting for some other rank to catch up. Profiling with `MPINSIDE_EVAL_SLT=y` and `MPINSIDE_EVAL_COLLECTIVE_WAIT=y` will help find out whether wait time is a significant factor, i.e. whether collectives like Bcast spend a significant amount of time waiting for slower ranks to arrive. Adding a feature to MPInside or integrating with another profiling tool like perf oroprofile in a future release will allow to isolate idle time within the compute times intervals.

This will help gain more specific insights into how much of the observed differences in send late time and collective barrier time are due to computational load imbalance.

Considering Recv and Bcast times, if a lot of send late time in Recv (large time in `w_MPI_Recv` column of MPInside output) or a lot of waiting for synchronization at the start of Bcast (large value in `b_Bcast` column) are seen, then optimization efforts could focus on areas of the code where differences in the compute time might reflect a load imbalance. Also, if there is other meaningful work a rank could do while waiting, there might be some performance to be gained by overlapping communication and computation with `MPI_Ibcast` (new in MPT 2.10) or `MPI_Irecv` and delaying blocking until work can't proceed without more data.

MPInside reports such as the one above are also available with the communication modeled instead of being measured. In particular MPInside is able to tell what will be the communication with a perfect interconnect. Knowing this asymptotic value is very useful. It can tell if it worth trying optimizing, trying other library, enhancing the hardware for a particular application.

Times on a perfect interconnect (`MPINSIDE_MODEL=PERFECT+1.0`) for Recv and Bcast will probably be much lower but still nonzero. The time that remains is attributable to waiting on one or more ranks on the other end of the Recv or Bcast to catch up - similar to SLT or collective waiting time, but might be shorter if zero transfer time in between compute intervals leads to the ranks being in closer synchronization. SGI mpiplace might be able to speed up execution by mapping ranks to a different sequence of nodes based on rank to rank matrix signature of communications obtained by MPInside to minimize inter node and inter switch transfer costs.

6. Maximizing throughput

6.1 Introduction

In a production environment, throughput is more important than turnaround time of individual jobs except for the exception case of high priority task which needs to be accomplished in the shortest delay. An HPC computation center might comprise some number of shared memory systems and cluster systems. Each of these systems have a total number of cores. The throughput optimization consists in deploying the jobs to be performed on varying numbers of

cores across these systems while not oversubscribing. Jobs to be performed fall into several cases in terms of their respective serial elapsed time. Often, an analyst, within a given production cycle will handle one such case.

6.2 Nomenclature for a throughput environment

Following are variables and parameters used to describe the throughput environment:

- `system`: either a Shared Memory System or cluster enclosing a total of `NbCoresSystem`. `system=1, NbSystems`
- `NbSystems`: Total number of above system's
- `NbCoresSystem`: Number of cores comprised by a given system
- `case`: a given class of dataset showing a similar `Elapsed1Case` serial elapsed time
- `NbCases`: Total number of above case 's
- `job`: a particular job, which will belong to a particular case, `job=1, NbJobsCase`
- `NbJobsCase`: Total number of above job belonging to a given case.
- `Elapsed1Case`: serial elapsed time of a particular case.
- `CoresJob`: number of cores chosen for a particular job
- `ElapsedCoresCase`: Elapsed time of a particular case when executed over some number of cores.
- `ElapsedCoresJobCase`: Elapsed time for a given job deployed over `coresJobCase`.

Derived variables:

- $\text{RateCoresCase} = 1 / \text{ElapsedCoresCase}$
- $\text{RateCoresJobCase} = 1 / \text{ElapsedCoresJobCase}$

An approximate linear relationship can be postulated between jobs running on a certain number of cores versus 1 core (serially):

- $\text{ElapsedCoresCase} = \text{Elapsed1Case} / \text{coresCase}$
- $\text{ElapsedCoresJobCase} = \text{Elapsed1Case} / \text{coresJobCase}$

Thus,

$$\text{RateCoresCase} = \text{coresCase} / \text{Elapsed1Case}$$

$$\text{RateCoresJobCase} = \text{coresJobCase} / \text{Elapsed1Case}$$

A metric to maximize would be:

$$\begin{aligned} \bullet \quad \text{Throughput} &= \sum_{\text{case}=1, \text{NbCases}} \sum_{\text{job}=1, \text{NbJobsCase}} \text{RateCoresJobCase} \\ &= \sum_{\text{case}=1, \text{NbCases}} \sum_{\text{job}=1, \text{NbJobsCase}} \text{coresJobCase} / \text{Elapsed1Case} \end{aligned}$$

Subject to the constraints:

- For $\text{job}=1, \text{NbJobsCase}$ For $\text{case}=1, \text{NbCases}$; $[\text{coresJobCase} \geq 1]$ which reflects that any job will be deployed on at least 1 core.
- $\sum_{\text{case}=1, \text{NbCases}} \sum_{\text{job}=1, \text{NbJobsCase}} \text{coresJobCase} \leq \text{NbCoresSystem}$

which reflects that total cores of jobs summed across cases must fit within the number of cores available in a system

6.3 Optimization method

Maximizing throughput from the preceding section is a Linear Programming (LP) problem [10] which is defined as finding the maximum or minimum value of a linear expression

$$ax + by + cz + \dots$$

(called the objective function), subject to a number of linear constraints of the form

$$Ax + By + Cz + \dots \leq N$$

or

$$Ax + By + Cz + \dots \geq N.$$

The largest or smallest value of the objective function is called the optimal value, and a collection of values of x, y, z, \dots that gives the optimal value constitutes an optimal solution. The variables x, y, z, \dots are called the decision variables.

The Simplex method [12] to solve Linear Programming problems has been implemented as an online solver here: <http://www.zweigmedia.com/RealWorld/simplex.html> as shown in Figure 19.

```

Maximize p = (1/2)x + 3y + z + 4w subject to
x + y + z + w <= 40
2x + y - z - w >= 10
w - y >= 10

```

Solution:

Optimal Solution: p = 115; x = 10, y = 10, z = 0, w = 20

Solve Example Erase Everything

Figure 19: Online Simplex Method Implementation

6.4 Examples

6.4.1 One case, multiple jobs

In this example, there is only 1 system with 40 cores on which one wants to allocate 4 jobs:

NbSystems=1

NbCores1=40

NbCases=1

NbJobs1=4

ElapsedCoresCase: Elapsed1=1000 seconds

Throughput = cores1/1000+cores2/1000+cores3/1000+cores4/1000

Corresponding template and results are shown in figure 20 where the extra 3 constraints impose an equality between the 4 jobs.

```

Maximize p = cores1+cores2+cores3+cores4 subject to
cores1>=1
cores2>=1
cores3>=1
cores4>=1
cores1-cores2=0
cores2-cores3=0
cores3-cores4=0
cores1+cores2+cores3+cores4<=40

```

Solution:

Optimal Solution: p = 40; cores1 = 10, cores2 = 10, cores3 = 10, cores4 = 10

Solve Example Erase Everything Rounding: 6 significant digits

Figure 20: One case, multiple jobs throughput example

6.4.2 Three cases, one job/case

In this example, there is only 1 system with 40 cores on which one wants to allocate 1 job for 3 different cases:

NbSystems=1

NbCores1=40

NbCases=3

NbJobs1=1 NbJobs2=1 NbJobs3=1

ElapsedCoresCase: Elapsed11=1000 Elapsed12=5000
Elapsed13=12000

Throughput=cores1/Elapsed11+cores2/Elapsed12+cores3/Elapsed13

Corresponding template and results are shown in figure 21 where the extra 2 constraints impose an equality between the 3 jobs.

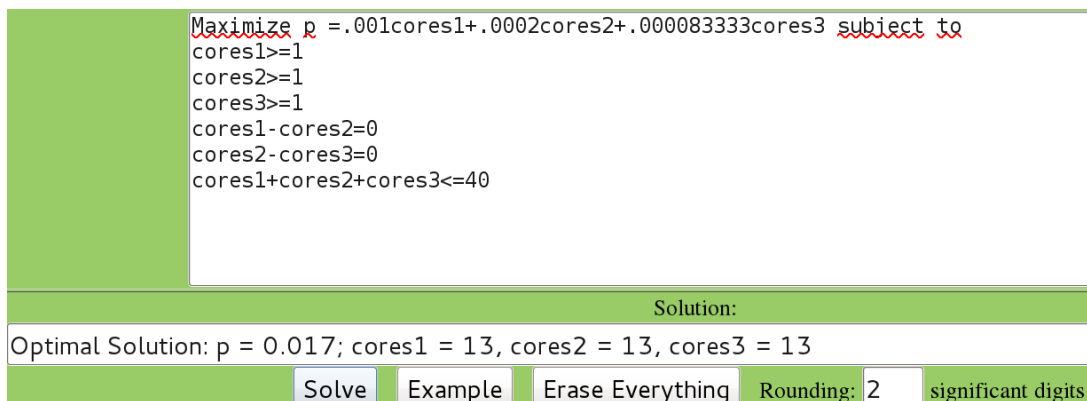


Figure 21: Three cases, one job per case throughput example

6.4.3 Three cases, several jobs/case

In this example, there is only 1 system with 80 cores on which one wants to allocate 4, 3 and 2 jobs for 3 different cases, respectively:

NbSystems=1

NbCores1=80

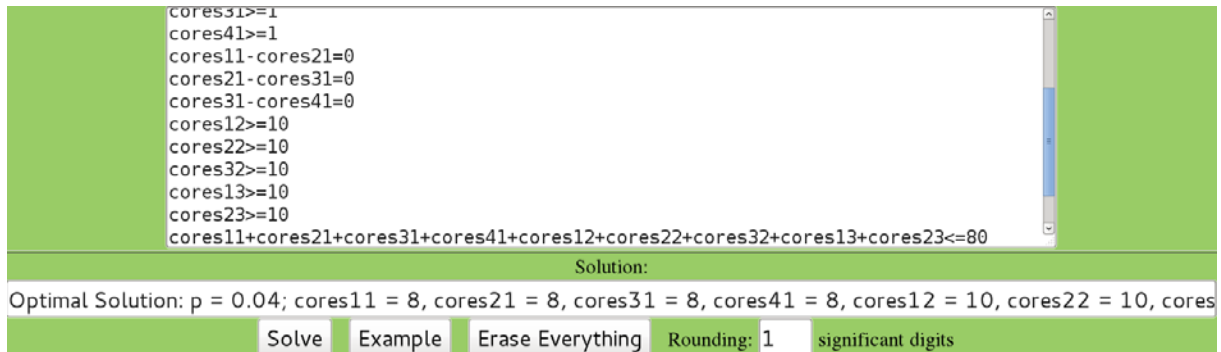
NbCases=3

NbJobs1=4 NbJobs2=3 NbJobs3=2

ElapsedCoresCase: Elapsed11=1000 Elapsed12=5000 Elapsed13=12000

Throughput=cores11/Elapsed11+cores21/Elapsed11+cores31/Elapsed11
+cores41/Elapsed11+cores12/Elapsed12+cores22/Elapsed12+cores32/E
lapsed12+cores13/Elapsed13+cores23/Elapsed13

Corresponding template and results are shown in figure 22 where the extra constraints impose minimum rates for each case.



```

cores31>=1
cores41>=1
cores11-cores21=0
cores21-cores31=0
cores31-cores41=0
cores12>=10
cores22>=10
cores32>=10
cores13>=10
cores23>=10
cores11+cores21+cores31+cores41+cores12+cores22+cores32+cores13+cores23<=80

```

Solution:

Optimal Solution: p = 0.04; cores11 = 8, cores21 = 8, cores31 = 8, cores41 = 8, cores12 = 10, cores22 = 10, cores32 = 10, cores13 = 10, cores23 = 10

Solve Example Erase Everything Rounding: 1 significant digits

Figure 22: Three cases, several jobs per case throughput example

6.4.4 Three systems, three cases, several jobs/case

In this example, first system is a UV 2000 (**1.3**) with 512 cores, second system is a Rackable cluster (**1.1**) with 64 nodes of 24 cores each and third system is an ICE X cluster (**1.2**) with 288 nodes of 20 cores each. One wants to allocate 4, 3 and 2 jobs for 3 different cases, respectively on these three systems:

NbSystems=3

NbCores1=512 NbCores2=64*24 NbCores3=288*20

NbCases=3

NbJobs1=4 NbJobs2=3 NbJobs3=2

ElapsedCoresCase: Elapsed11=1000 Elapsed12=5000 Elapsed13=12000

Throughput=cores11/Elapsed11+cores21/Elapsed11+cores31/Elapsed11
+cores41/Elapsed11+cores12/Elapsed12+cores22/Elapsed12+cores32/Elapsed12+cores13/Elapsed13+cores23/Elapsed13

Corresponding template and results are shown in figure 23 where the extra constraints impose minimum rates for each case.

The screenshot shows a linear programming problem with the following constraints:

```

Type your linear programming problem below. (Press "Example" to see how to set it up.)
cores11>=1
cores11-cores21=0
cores21-cores31=0
cores31-cores41=0
cores12-cores22=0
cores22-cores32=0
cores13-cores23=0
cores11+cores21+cores31+cores41<=512
cores12+cores22+cores32<=1536
cores13+cores23<=5760

```

The solution is:

```

Solution:
Optimal Solution: p = 1.2992; cores11 = 128, cores21 = 128, cores31 = 128, cores41 = 128, cores12 = 512, cores22 = 512, cores32 = 512, cores13 = 2880, cores23 = 2880

```

Buttons: Solve, Example, Erase Everything, Rounding: 6 significant digits

Figure 23: Three systems, three cases, several jobs/case

Optimal Solution: $p = 1.2992$; $\text{cores11} = 128$, $\text{cores21} = 128$, $\text{cores31} = 128$, $\text{cores41} = 128$, $\text{cores12} = 512$, $\text{cores22} = 512$, $\text{cores32} = 512$, $\text{cores13} = 2880$, $\text{cores23} = 2880$

6. Conclusions

This study showed how interconnect, processor architecture, core frequency, Turbo Boost, and hyper-threading effects on turnaround and throughput performance can be gauged for LS-DYNA runs. All these effects are seen to be dependent on the datasets and solution methods used. Procurement of the right mix of resources should therefore be tailored to the mix envisaged. Upgrading a single system attribute like CPU frequency, interconnect, or number of cores, diminishes returns if the others are kept unchanged. Elapsed time performance can be translated into derived metrics such as turnaround times or throughput and the cost to achieve them can be broken up into acquisition, licensing, energy, facilities and maintenance components.

7. Attributions

LS-DYNA, is a registered trademark of Livermore Software Technology Corp. SGI, Rackable, NUMalink, SGI Ice X, SGI UV, ProPack are registered trademarks or trademarks of Silicon Graphics International Corp. or its subsidiaries in the United States or other countries. Xeon is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries. Linux is a registered trademark of Linus Torvalds in several countries. SUSE is a trademark of SUSE LINUX Products GmbH, a Novell business. All other trademarks mentioned herein are the property of their respective owners.

References

- [1] LS-DYNA[®], KEYWORD USER'S MANUAL, VOLUME I, Appendix O, August 2012, Version 971 R6.1.0
- [2] SGI. Linux Application Tuning Guide. Silicon Graphics International, California, 2009.
- [3] Yih-Yih Lin and Jason Wang. "Performance of the Hybrid LS-DYNA on Crash Simulation with the Multicore Architecture". In 7th European LS-DYNA Conference, 2009.

- [4] Dr. C. Cleve Ashcraft, Roger G. Grimes, and Dr. Robert F. Lucas. “A Study of LS-DYNA Implicit Performance in MPP”. In Proceedings of 7th European LS-DYNA Conference, Austria, 2009.
- [5] Dr. C. Cleve Ashcraft, Roger G. Grimes, and Dr. Robert F. Lucas. “A Study of LS-DYNA Implicit Performance in MPP (Update)”. 2009.
- [6] Olivier Schreiber, Michael Raymond, Srinivas Kodiyalam, [LS-DYNA® Performance Improvements with Multi-Rail MPI on SGI® Altix® ICE cluster](#), 10th International LS-DYNA® Users Conference, June 2008
- [7] Olivier Schreiber, Scott Shaw, Brian Thatch, and Bill Tang. “LS-DYNA Implicit Hybrid Technology on Advanced SGI Architectures”. <http://www.sgi.com/pdfs/4231.pdf>, July 2010.
- [8] Olivier Schreiber, Tony DeVarco, Scott Shaw and Suri Bala, ‘Matching LS-DYNA Explicit, Implicit, Hybrid technologies with SGI architectures’ In 12th International LS-DYNA Conference, May 2012.
- [9] Leveraging LS-DYNA Explicit, Implicit, Hybrid technologies with SGI hardware, Cyclone Cloud Bursting and d3VIEW, Olivier Schreiber*, Tony DeVarco*, Scott Shaw* and Suri Bala† *SGI, †LSTC, 9th European Users Conference, 3-4th June 2013-Manchester, UK
- [10] Leveraging LS-DYNA Explicit, Implicit, Hybrid Technologies with SGI hardware and d3VIEW Web Portal software, Olivier Schreiber*, Tony DeVarco*, Scott Shaw* and Suri Bala† *SGI, †LSTC <http://www.sgi.com/pdfs/4426.pdf>, August 2013
- [11] Daniel Thomas, [Jean-Pierre Panziera](#), [John Baron](#): MPIInside: a performance analysis and diagnostic tool for MPI applications. [WOSP/SIPEW 2010](#): 79-86, ACM, (2010) <http://www.sgi.com/products/software/sps.html>
- [12] Finite Mathematics, Finite Mathematics & applied Calculus, Sixth Edition, *Waner, Costenoble*, An Internet Resource for Students and Instructors, Illinois State U, Eastern Michigan U, Michigan State U, Oakland CC, Arizona State U, SUNY Oswego, U of Texas EP, [FiniteMath.org](#), [AppliedCalc.org](#), [FiniteAndCalc.org](#)