Benchmarking of three parallelized implementations of LS-Dyna on a HPC server cluster

Bruce W.T. Quinton and Anthony Kearsey
Dept. of Engineering and Applied Science
Memorial University of Newfoundland
St. John's, NL A1B 3X5 Canada

***Abstract*-This paper discusses the benchmarking of three parallelized implementations of the popular LS-Dyna® (Livermore Software Technology Corp.) finite element code on the 128-core x86-64 STePS2 high performance computing (HPC) server cluster. SMP, MPP and SMP-MPP hybrid implementations of LS-Dyna® are benchmarked over various numbers of nodes, CPUs and CPU cores.**

The SMP, MPP and MPP-SMP hybrid implementations of LS-Dyna were compiled for use with the HPMPI message passing interface.

Index Terms—LS-Dyna benchmark; SMP; MPP; Hybrid; parallel computing benchmark

I. Introduction

STePS^2 (Sustainable Technology for Polar Ships and Structures) is a collaborative multi-organization research project at Memorial University of Newfoundland (MUN) that will support development of a new generation of ships and structures for polar regions. Its goal is to develop sophisticated but practical tools for designers, builders and regulators of arctic ships and structures.

The project is a partnership between MUN, Husky Energy, the American Bureau of Shipping, Samsung Heavy Industries, BMT Fleet Technology and Rolls Royce Marine. In addition to the private sector participation, the National Research Council Institute for Ocean Technology is a research partner, and public funding has been awarded by the Atlantic Canada Opportunities Agency and the Research and Development Corporation Newfoundland and Labrador.

Project results will be primarily captured in new software. However the software will be backstopped by experimental and numerical research aimed at filling in gaps in understanding and methods that support the engineering software tools.

To aid end, it was desired to implement a High Performance Computing (HPC) parallel computing environment with which to run Livermore Software Technology Corporation's (LSTC) LS-Dyna®; a popular finite element code.

This paper outlines the hardware and software configuration; and subsequent benchmarking of three parallel implementations (SMP, MPP, and MPP-SMP Hybrid) of LS-Dyna on the STePS^2 HPC server cluster.

II. The STePS^2 HPC Cluster

The STePS^2 HPC cluster consists of a head node and sixteen compute nodes. The head node and compute nodes are interconnected via two networks: Ethernet and Fabric. All nodes operate using Intel_x86_64 Linux. Message passing for parallel computations is accomplished using HP-MPI v2.3.1. Jobs scheduling is by Torque (PBS). The cluster toolkit is provided by ROCKS 5.4. This configuration provides a multi-tasking head node driving a parallel computing environment that supports up to 128 parallel processes.

A. Hardware

The STePS^2 HPC cluster is manufactured by IBM. The head node is model x3650M2 and each of the compute nodes is model x3550M2. All nodes are rack mounted. All hardware components are "server" class. The 48-port Ethernet switch is model IBM BNT RackSwitch™ G8000R. The 36-port Fabric switch is model Voltaire® Grid Director™ 4036.

Head Node

The head node has dual quad-core CPUs; each with their own bank of 8 DIMM slots. The CPUs also support Intel® Turbo Boost Technology and hyperthreading capabilities; both of which are enabled. The DIMM slots in both memory banks are filled with 2GB EEC DDR3 chips operating at 800 MHz. Permanent storage is provided by two RAID arrays: a RAID 5 array of five 15000 RPM SAS 146GB drives hosts the main cluster partitions, while a RAID 5 array of three 15000 RPM SAS 300 GB drives provides backup and auxiliary storage. One of the partitions on the main RAID array is a Network Attached Storage (NAS) partition. This allows all attached compute nodes to access a common file store. The operating system (OS) is Red Hat Enterprise Linux 5.4 (kernel 2.6.18-164). The cluster toolkit is ROCKS 5.4. Table I outlines the key head node hardware.

Compute Nodes

Each of the compute nodes has the same CPU configuration as the head node, except that hyperthreading is not enabled. Previous experimentation has shown that enabling hyperthreading for fully subscribed CPUs running LS-Dyna has a negative effect on performance. The RAM configuration for each compute node is similar to that of the head node except that two of the DIMM slots are not filled in each of the memory banks; and the RAM frequency is higher at 1067 MHz. Permanent storage for each compute node is provided by a RAID 0 array of four 15000 RPM SAS 146GB drives. Data redundancy is sacrificed in favour of performance as no user data are stored on the compute nodes (i.e. only the OS and configuration are stored). In the event that any storage array is corrupted, the compute node will obtain and install a fresh, pre-configured OS installation from the head node via PXE upon its next first successful boot. The OS for the compute nodes is also similar to the head node. Table II outlines the main hardware features for the compute nodes.

TABLE I
HEAD NODE HARDWARE

<u>Head Node</u>		
Processors		
# of CPUs	2	
CPU type	Intel(R) Xeon(R) E5520	
Cores per CPU	4	
CPU Frequency	2.27	GHz
CPU Max Turbo Frequency	2.53	GHz
CPU Cache	8	MB
CPU Address Sizes	40 bits physical, 48 bits virtual	
QPI Speed	5.86	GT/s
Instruction Set	64-bit	
Hyperthreading	Yes and Enabled	
Memory		
Total Memory	32	GB
Memory per CPU	16	GB
Memory Slots	8 per CPU (all 8 filled)	
DIMM Size	2	GB
Type	DDR 3 ECC	
Memory Frequency	800	MHz
Storage		
Array 1		
RAID	RAID 5	
Number of disks	5	
Total Storage	584	GB
Storage per disk	146	GB
Disk Type	SAS	
Disk Speed	15000	RPM
RAID Controller	Hardware	
Array 2		
RAID	RAID 5	
Number of disks	3	
Total Storage	600	GB
Storage per disk	300	GB
Disk Type	SAS	
Disk Speed	15000	RPM
RAID Controller	Hardware	
Operating System		
Operating System	RHEL Server 5.4 (Tikanga)	
Linux Kernel	2.6.18-164	
Architecture	Intel x86_64	

TABLE II
COMPUTE NODE HARDWARE

Compute Nodes		
Processors		
Same as head node except	Hyperthreading not Enabled	
Memory		
Total Memory	24	GB
Memory per CPU	12	GB
Memory Slots	8 per CPU (6 of 8 filled)	
DIMM Size	2	GB
Type	DDR 3 ECC	
Memory Frequency	1067	MHz
Storage		
RAID	RAID 0	
Number of disks	4	
Total Storage	584	GB
Storage per disk	146	GB
Disk Type	SAS	
Disk Speed	15000	RPM
RAID Controller	Hardware	
Operating System		
Operating System	Same as head node	

Cluster Interconnects

As mentioned above, the STePS^2 cluster makes use of two forms of nodal interconnects: Ethernet and Fabric. The Fabric connection is a 40 GBit/s Infiniband® Fabric and the Ethernet connection is 1 GBit/s (GigE).

The Infiniband connection is used only for message passing via the MPI software (outlined below). Some of the notable communication protocols available are IPoIB, IBV and PSM.

The IP over Infiniband protocol (IPoIB) has not been implemented on the STePS^2 cluster because it is intended that normal IP communications take place on the GigE connection. Both the IBV (Infiniband Verb by Openfabrics) and the PSM (Performance Scaled Messaging by QLogic) communication protocols have been used in conjunction with LS-Dyna simulations in previous experiments, and it has been determined that the PSM protocol provides the best performance in this scenario. All tests outlined in this document have been performed using the PSM protocol.

The GigE connection is used for all non-MPI internodal communication; that is: file transfer, ssh, etc... It is possible to use the GigE connection for message passing via the MPI software, but there would be no practical benefit in this.

B. *Software*

The software on the cluster consists of the operating system (OS), the cluster environment, the job scheduler, the message passing interface (MPI), and the LS-Dyna finite element code.

Operating System and Cluster Environment

The OS on all nodes is the intel_x86_64 implementation of Redhat Enterprise Linux Server 5.4 with Linux kernel 2.6.18-164.

The cluster environment is configured through the ROCKS 5.4 cluster toolkit. The ROCKS toolkit simplifies cluster setup by installing a preconfigured parallel environment based on options determined by the end user. The basic ROCKS environment will provide a working parallel environment, however many additional features are available through the inclusion of ROLLs. ROLLs are special implementations of various software packages that are configured to interface directly with the ROCKS toolkit; either during initial cluster installation or post install. Some of the ROCKS ROLLs that were implemented on the STePS^2 cluster are the QLogic Infiniband Drivers and the Torque (PBS) job scheduler.

Job Scheduler

As mentioned above, the Torque (PBS) job scheduler was implemented as a ROCKS ROLL. This installed a preconfigured job scheduling environment that worked "out-of-the-box". All benchmarking tests were submitted through the job scheduler. Note: Normally a job scheduler makes as efficient use of cluster resources as possible, but this is not ideal for benchmarking tests. For example, because simulation data (i.e. output files) for all benchmarking tests are being written to the common NAS drive, running multiple benchmarking tests simultaneously would necessarily require multiple sets of output data being written concurrently. This may negatively impact simulation times due to competition over hard disk resources. In order to eliminate competition for any cluster resources between benchmarking tests, all tests were run sequentially. This was accomplished by having each job (i.e. each benchmarking test) reserve the use of all 128 compute cores; whether that benchmarking tests actually made use of all 128 or not. Once one job finished, another started.

Message Passing Interface

The STePS^2 cluster uses the HP-MPI v2.3.1 message passing interface (MPI) for interfacing parallel computations. This is one of the MPI implementations recommended by LSTC for use with LS-Dyna. HP-MPI v2.3.1 implements the PSM communication protocol for Infiniband communication. It also allows the user to bind processes to CPU cores relatively easily.

LS-Dyna finite element code

The LS-Dyna finite element code is highly popular in the automobile and aerospace industries, among others. It has three major implementations: Symmetric Multiprocessing (SMP), Massively Parallel Processing (MPP), and hybrid MPP-SMP.

The SMP implementation makes use of multiple processors operated by a single OS that have access to a single shared memory. SMP processing for LS-Dyna has the problem that multiple processors sharing the same memory bus, coupled with too many synchronization checks results in a computational bottleneck that practically limits its scalability to 8 processors [1].

The MPP implementation makes use of multiple processors that may or may not be operated by a single OS, do not have access to shared memory (i.e. have distributed memory), and are connected by some type of link. MPP processing for LS-Dyna has the problem that results will not always be consistent across various numbers of MPP processes. This is due to the way the problem is decomposed into parallel processes, and may be aggravated by large numbers of processes (typically larger than 128) [2].

The hybrid MPP-SMP implementation addresses the scalability issues of the MPP implementation by employing a number of SMP processes for every MPP process [3]. For example, 32 MPP processes running on 32 quad-core CPUs where each MPP process has 4 SMP processes, actually employs 128 cores. This keeps the number of MPP processes down; and thereby alleviates the numerical noise associated with large numbers of MPP processes.

III. Benchmarking

Each of the parallelized implementations of LS-Dyna was benchmarked on the STePS^2 HPC by having it solve a "problem" utilizing various numbers of compute cores (from 4 to 128). The same problem was given in each case. For the purpose of this paper, the only important result is the amount of time the implementation took to solve the problem versus the number of cores provided. This result will be called the "run time" from this point on.

A. *The "Problem"*

LSTC has published four benchmarking problems that are publically available for download from www.topcrunch.org. The problem chosen for these benchmarking tests was the "neon_refined_revised" problem [4] shown in Fig. 1. This model simulates a head on collision of a Dodge Neon with a rigid wall. The model is quite detailed and contains over 500,000 elements. This model is a "real world" problem, not a synthetic benchmark.

B. *Test Matrix*

Four categories of benchmarking tests were executed: SMP, MPP without CPU binding, MPP with CPU binding, and hybrid MPP-SMP (with CPU binding).

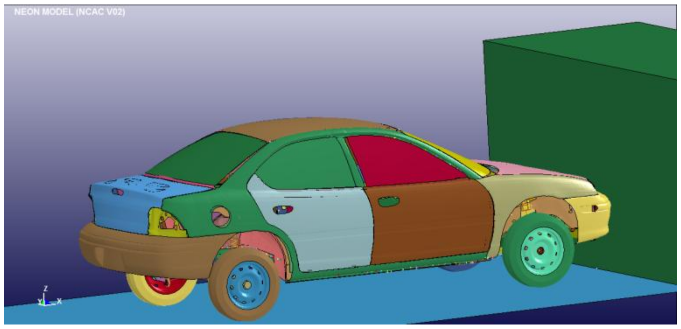

Figure 1. “Neon_revised_refined” benchmarking simulation.

Each category was executed using both single and double precision versions of their respective codes. All tests were executed three times and the resulting run times were averaged before being reported here. Table III outlines benchmarking test matrix.

C. Results

Run time results for single and double precision benchmark tests for each of the four categories of tests are given in Fig. 2.

The worst runtime for both precisions is posted by hybrid SMP-MPP when using 4 cores. This run time (single: 3136 s, double: 5028 s) represents 1 MPP process running 4 SMP processes and should notionally be comparable to the SMP implementation's run time for 4 cores (i.e. single: 2334 s, double: 4085 s). One factor that may have influenced this difference is the trade-off between CPU binding and Intel Turbo Boost Technology. The 4 SMP processes in the hybrid case were bound to the 4 cores of a single quad-core CPU (i.e. one CPU was fully subscribed); while the 4 processes of the SMP implementation were split between 2 quad-core CPUs (i.e. 2 processes on one of the node's quad-core CPUs, and 2 processes on the other). Intel Turbo Boost technology allows CPUs that are not fully subscribed to operate at a higher frequency than their rated frequency (i.e. up to 2.57 GHz versus 2.27 GHz for these CPUs). The under-subscribed SMP case may have taken advantage of the turbo boost. The CPU bound MPP implementation had the best time for the 4 core case, followed by the non bound MPP case.

TABLE III
COMPUTE NODE HARDWARE

Cat	Single Precision			Double Precision		
	Type	**Total cores**	**MPP Proc**	**Type**	**Total cores**	**MPP Proc**
I	smp	4	N/A	smp	4	N/A
	smp	8	N/A	smp	8	N/A
II	mpp	4	4	mpp	4	4
	mpp	8	8	mpp	8	8
	mpp	16	16	mpp	16	16
	mpp	32	32	mpp	32	32
	mpp	64	64	mpp	64	64
	mpp	128	128	mpp	128	128
III	mpp-nobind	4	4	mpp-nobind	4	4
	mpp-nobind	8	8	mpp-nobind	8	8
	mpp-nobind	16	16	mpp-nobind	16	16
	mpp-nobind	32	32	mpp-nobind	32	32
	mpp-nobind	64	64	mpp-nobind	64	64
	mpp-nobind	128	128	mpp-nobind	128	128
IV	hybrid	4	1	hybrid	4	1
	hybrid	8	2	hybrid	8	2
	hybrid	16	4	hybrid	16	4
	hybrid	32	8	hybrid	32	8
	hybrid	64	16	hybrid	64	16
	hybrid	128	32	hybrid	128	32

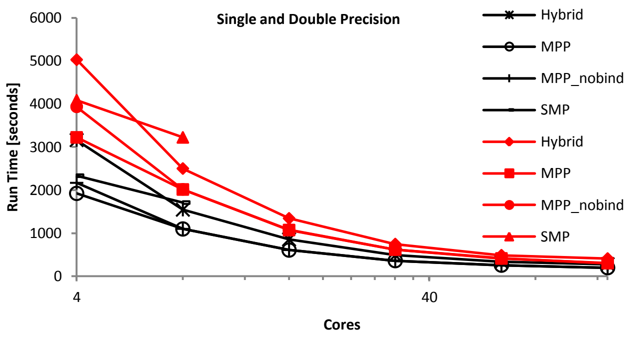

Figure 2. Run time results for single (black with hollow data points) and double (red with solid data points) precision benchmark tests.

The results over 8 cores (i.e. 2 fully subscribed CPUs in a single node) show that the hybrid implementation is now better for both single and double precision than the SMP implementation. There is almost no difference between the CPU bound and unbound MPP implementation times; which are again the fastest overall. This illustrates the speedup attained by implementing MPP processes over just SMP processes.

The run times progressively decrease as the LS-Dyna implementations utilize more and more cores (each time doubling from 4 to 128). The MPP implementation (with either bound or unbound processes) gave the fastest benchmark times up to and including 128 cores; with negligible difference in performance between the bound and unbound MPP processes from 8 cores on.

Fig. 3 shows the ratio of run times (i.e. t_{hybrid}/t_{MPP}) versus number of cores. It is apparent from Fig. 3 that while the hybrid implementation was slower than the MPP implementations (i.e. the ratio is always greater than 1.0) the gap narrows up to 64 cores, but then starts to widen again. Further investigation is required to determine the cause of this widening at 128 cores.

Fig. 4 shows the "speedup" and "relative speedup" associated with utilizing increasing numbers of cores for all four categories of the single precision case. The "speedup" is the ratio of run time for *n* cores over the run time for 4 cores (for each implementation). The "relative speedup" is the ratio of runtime for *n* cores versus the run time for *n/2* cores.

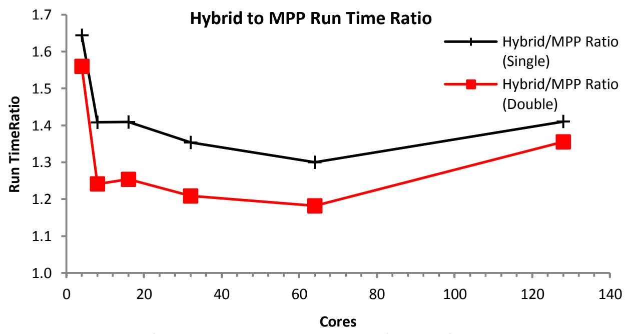

Figure 3. Ratio of runtimes versus number of cores for hybrid and MPP LS-Dyna implementations.

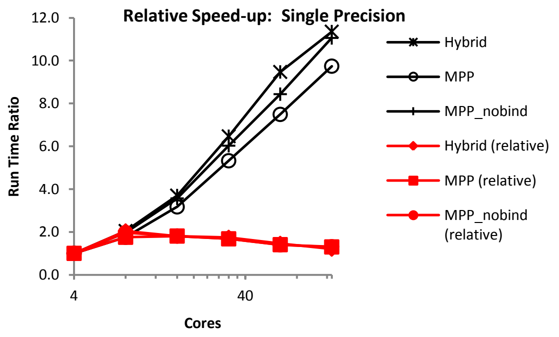

Figure 4. "Speedup" and "Relative Speedup" versus number of cores for single precision LS-Dyna parallelized implementations.

We can see that the hybrid implementation has the best "speedup", followed closely by the unbound MPP implementation. Note that the "relative speedup" is not significantly different for any of the implementations, and it is decreasing; thus demonstrating diminishing returns with increased resources. The SMP implementation results have not been included in this graph. The "speedup" and "relative speedup" for the SMP implementation are both 1.4. The results for double precision are similar and are omitted for brevity.

Fig. 5 shows the ratio of run times for double over single precision versions of all LS-Dyna implementations. This illustrates the relative cost of running double precision versus single precision for the same "problem". The graph generally shows that as the number of parallel processes increases (past 8), the relative cost of running double precision versus single precision decreases; especially for the MPP implementation, which shows a decrease approximately 20% over the range from 8 to 128 cores.

IV. Summary

Three parallelized implementations of LS-Dyna were benchmarked on the STePS^2 HPC cluster using a real-world test “problem”. The results show that the MPP implementation of LS-Dyna offers the best performance as a function of run time only. It outperforms both the SMP and hybrid MPP-SMP

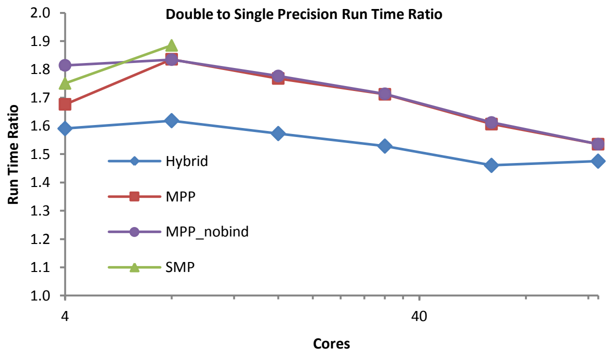

Figure 5. Ratio of run times for double over single precision for the parallelized implantations of LS-Dyna.

implementations across the range of cores tested. One notable point regarding the hybrid MPP-SMP implementation is that the relative cost of its use to perform double precision versus single precision calculations is much better than the other parallelized implementations.

Acknowledgment

The first author would like to thank the following for their financial support:

The Sustainable Technology for Polar Ships & Structures (STePS^2) project at Memorial University of Newfoundland.

MITACS Accelerate Internship Program.

BMT Fleet Technolgoy Ltd.

References

[1] B. Wainscott and J. Wang, “Message Passing and Advanced Computer Architectures,” *6th International LS-DYNA Users Conference,* Detroit, 2000.

[2] J. Wang, Livermore Software Technology Corp. Private Communication, October, 2011.

[3] N. Meng, J. Wang, and S. Pathy, “New Features in LS-DYNA® HYBRID Version,” *11th International LS-DYNA Users Conference,* Detroit, 2010.

[4] J. Hallquist, “neon_refined_revised,” *http://www.topcrunch.org*, Accessed: October 3, 2011