# LS-DYNA Communication Performance Studies

Ananthanarayanan Sugavanam and Guangye Li
*High Performance Computing, IBM*

## Abstract

*In recent years, MPP-DYNA, the message passing parallel version of LS-DYNA, has become more and more popular in car crash and metal stamping simulations due to its good scalability which may reduce the turn-around time significantly when more processors are used. However, so far, most users only use 16 or less processors for LS-DYNA simulation because of the limitation of the scalability on a larger number of processors. This paper analyzes the communication patterns, message sizes and costs of simulation of two models. It is concluded that the unbalanced work load among processes is the bottleneck for scalability. Our study shows that some special decomposition techniques including sliding interface decomposition and scaling on certain directions may produce more balanced work load and, therefore, improve scalability. It is our hope that this study provides some insight for the algorithm improvement which may lead to better MPP-DYNA scalability on a larger number of processors.*

## Introduction

The automotive industry has started to rely more and more on the message passing parallel (MPP) version of LS-DYNA for car crash and metal stamping simulations primarily due to its increasing reliability, and cost considerations. The parallel performance of the MPP version of LS-DYNA on a given hardware relies on a number of factors, such as single node compute performance, load balancing, communication characteristics of the interconnect (switch & network performance), the message passing details, the frequency of communication and the accuracy of the solution. As more and more crash simulations are relying on the MPP version, it is critical to understand some of these issues sufficiently. This paper will address the parallel performance issues of MPP-DYNA using some of the powerful tools developed at the IBM Research laboratories. This analysis will study the details of the work done by the different processes, message passing details and importance of load balancing by looking at different decomposition options, and will identify the performance bottlenecks in the application. This will be critically important for the LS-DYNA developers, and will help in improving methods to obtain better scalability as well as in using the MPP version of LS-DYNA more efficiently.

There have been other studies that addressed the communication performance issues in LS-DYNA. Lin [1], Roe, and Fong [2] have studied the performance of MPP-DYNA using some specific system tools to identify some of the communication performance issues. While this study uses similar system tools, the primary purpose is to identify, and associate the decomposition methods with performance bottlenecks to get better insight on improving scalability. With the type of imbalance seen in this study, even a perfectly balanced communication subsystem (with zero latency and infinite bandwidth) will have problem scaling beyond a dozen or so processes. Addressing the load balancing will allow LS-DYNA simulations to be done on hundreds of processors instead of dozens of processors.

In the following sections, domain decomposition methods, the description of tools used (both the system hardware and software tools) for the creation of LS-DYNA trace binary , description of two models, and three decomposition cases, timing results of the simulations and analysis of the results are given.

## Domain decomposition in finite-element methods

There is a lot of information on the domain decomposition methods and tools of finite element/finite volume or unstructured mesh for load balancing, and communication performance in the literature. Many tools developed at National Laboratories, and Universities such as, Recursive Coordinate Bisection (RCB), Recursive Spectral Bisection (RSB), and heuristic methods such as METIS [3], CHACO [4]. ) and JOSTLE [5] are widely used in disciplines such as computational fluid dynamics, and computational structural mechanics. However, in crash simulations, due to the presence of contacts, the domain characteristics dramatically change with time and pose some unusual challenges in load balancing. It appears that the Recursive Coordinate Bisections is the preferred method used for crash simulations due to the simplicity, effectiveness and physical characteristics.

## Description of the tools used

**System Description:**
All of the simulations for this exercise were done on an IBM p690 system. This is a shared memory system with 32 POWER4+ processors that run at a clock speed of 1.5 GHz. POWER4+ is an advanced microprocessor that has 2 processors in a chip with a shared 1.5 MByte L2 cache. The system has a large L3 cache (128 MByte) shared by 8 processors. The system supports small (4 KByte) and large (16 MByte) page support for the virtual memory. The details of the processor and the system may be found in [6].

**hpmcount and mpitrace tools:**
The HPM (Hardware Performance Monitor) Toolkit [7] was developed by Luiz DeRose of IBM Research for performance measurement of applications running on IBM systems (POWER3 & POWER 4), based on the ‘pmapi’ library and consists of

- A utility (**hpmcount**) which starts an application and provides at the end of execution wall clock time, hardware performance counter information, derived hardware metrics and resource utilization statistics.
- An instrumentation library (**libhpm**), which provides instrumented programs with a summary of output containing the above information for each instrumented region in a program (resource utilization statistics
- A graphical user interface (**hpmviz**)

The **hpmcount** utility measures depending on the environmental variables set, provides all of the details of the floating, and fixed point units and operations, L1, L2, L3-cache misses, Translation Lookaside Buffer (TLB) misses, and estimates the GFLOPS of the application.

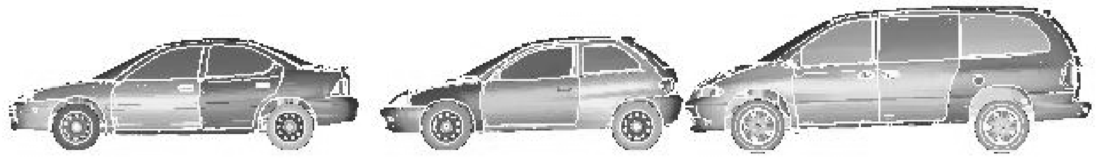
Walkup [8] at IBM Research developed a wrapper library called **libmpitrace** that is widely used at many of the DOE and DOD laboratories on big POWER3 & POWER4 clusters, and has a mechanism for profiling the MPI communication calls for applications written in C, C++, and

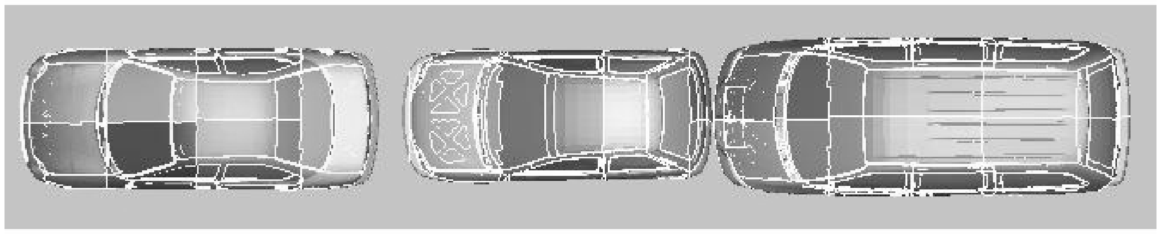
FORTRAN. This is a very low overhead utility library that has to be linked with an existing unstripped binary of an application, and upon execution of the application generates trace files for each of the processes. Each of these trace files contain information on how much elapsed time is spent on each of the message passing, and collective communication calls, size of the messages, and number of calls made. A version of this **libmpitrace** library includes the **libhpm** library mentioned earlier and, the combined library called the **libmpihpm** provides in addition to the details of MPI communication profiles, the hardware performance counter details such as the specifics of the floating and fixed point performance.

**Trace Binary Creation**
The LS-DYNA binary used for this study is **mpp970_s_3535a_ibmpwr3_51**, which is a 64-bit single precision binary. For obtaining the tracing details, a new binary called **lsdyna_trace** was built by linking with the **libmpihpm library** with **mpp970_s_3535a_ibmpwr3_51**. To estimate the overhead of the profiling binary, runs were made with both the binaries for 10 time steps, and it was found that there was no difference in elapsed time between these two binaries.

**Simulations with the Trace Binary**
The models studied here are two models – a publicly available three car collision model, and another smaller model. With the three car collision model, 3 cases were studied – the original decomposition, the merged contact model and a special decomposition that includes scaling, and the decomposition of sliding interfaces. Studies on the second model were restricted to cases with the sliding interface decomposition and scaling.

## Three car crash model





Decomposition of the 3-car model on 16 CPUs

This model has a total of 794,780 elements (9,642 solid elements, 116 beam elements and 785,022 shell elements). The original model has 9 sliding interfaces, most of which are small (less than 5000 slave nodes). LSTC modified this model by merging these 9 sliding interfaces to

2 sliding interfaces, which is called the “merged model” in the following context. The merged model can be obtained from the web site http://www.topcrunch.org. The simulation time in our experiments is 50 milliseconds which correspond to 50000 time steps (full simulation for this case would be 150 milliseconds). The experiments were done from 8 to 32 processors with the three cases described earlier. All these simulations were done on the same 32-CPU p690 running at 1.5 GHz system described earlier, using small (4KByte) pages, and with the default shared memory system option “MP_SHARED_MEMORY=yes”.

**Fig. 1 Elapsed Time**

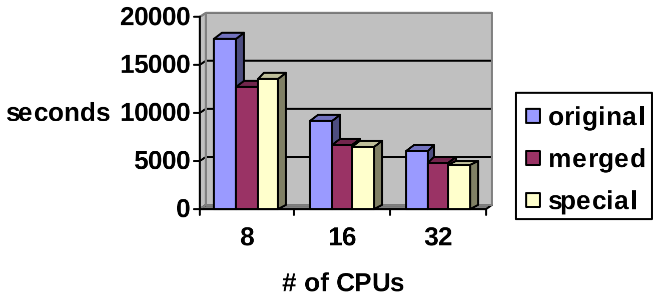


Fig. 1 shows the comparisons of these 3 cases in terms of elapsed time for the simulation. In these charts, ‘merged’ refers to the merged contact case, and ‘special’ refers to the special decomposition (with the decomposition of sliding interface number 6 and default scaling) case. The original case took significantly more elapsed time, and had poor scalability. The elapsed time for the case with merged contacts was significantly better than the original case. The most benefits were seen for the 8-CPU simulation. The special decomposition case has overall improved performance and better scalability particularly for the larger number of processors, compared to the merged contact case. For some of the other models, there is a significant performance benefit with the special decomposition. The overall hardware counted GFLOPS for the 32-CPU case improved by about 20% from original to the special decomposition case.

To measure the load balance, we compared the total numbers of floating point divides in each processes case (the divide instructions take typically many more cycles (15 to 18) than other floating point operations, and hence this disparity will be a good measure of load imbalance). Table 1 shows the difference in the floating point divide (FDIV) operations between different processes illustrating the load imbalance for the original model. For the 8 CPU case, there is 27% variation between the least and the most FDIV operations between the processes. This percentage variation stays roughly the same as we increase the number of processors.

For the merged contact case, and the special decomposition case, there is much less variation in the number of FDIV operations (about 15%), which improves the scaling performance noticeably as seen in Table2, that presents the results for the merged contact case. In addition, the number of FDIV operations has decreased considerably (about 20%). This reduction in variation of the total divide operations clearly alleviates some of the load balancing problems, and significant performance gains in overall elapsed time are obtained in all of the 8, 16 and 32-way simulations.

| # of CPUs | Fewest FDIV instructions in billions | Most FDIV instructions in billions | % variation |
|---|---|---|---|
| 8 | 73 | 100 | 27 |
| 16 | 36 | 52 | 31 |
| 32 | 18 | 26 | 31 |

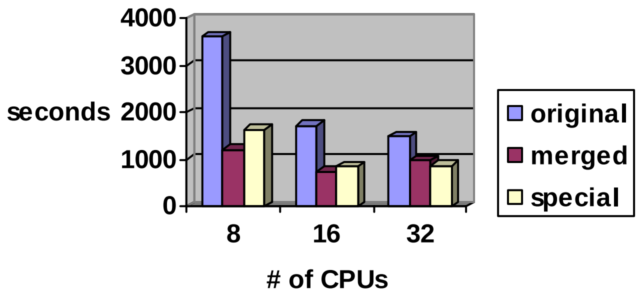
Table 1 Variation of floating point divide operations (original)

| # of CPUs | Fewest FDIV instructions in billions | Most FDIV instructions in billions | % variation |
|---|---|---|---|
| 8 | 67 | 78 | 14 |
| 16 | 34 | 39 | 13 |
| 32 | 17 | 20 | 15 |

Table 2 Variation of floating point divide operations (merged contact)

Total communication time is the time spent on all the message passing calls for each of the processes in the simulation. There is a separate file for each process, and generally there will be some variation between times spent on each of the calls between processes due to load balancing. Figure 2 shows the total communication time for the process with the least communication time (there is a big variance in communication time between processes due to load imbalance, and typically the least communication time corresponds to the process that does the most work or that has most floating point divide operations) for the 3 cases studied.

**Fig.2 Total Communication Time**



For the merged contact case, and the special decomposition case, there is a significant reduction in the total communication time. Also, the fraction of the total time spent on communication is reduced by 50% for the 8 and 16 processors with the ‘merged’ and ‘special’ cases. For the 32-CPU case, there is a significant reduction in the total communication time (about 30% reduction).

The total communication time can be broken into essentially two parts – the time spent on ghost cell updates (exchanging information between neighboring domains) and collective communication time, such as a global summation, synchronization, broadcast and similar operations. In the context of the MPI library, typically the ghost cell updates are done with blocking or non-blocking sends and receives such as MPI_SEND, MPI_RECEIVE, MPI_ISEND, MPI_IRECEIVE calls. The global operations are done with calls such as MPI_BARRIER (synchronization), MPI_ALLREDUCE (a reduction operation such as summation done on the all the processors), MPI_BCAST (communicating a single value or a vector to all the processes), MPI_ALLGATHER (collecting and distributing data globally) and similar others. With a high performance network, the global communications are done very efficiently, and typically more time on these global communication calls represent load imbalance in the computations.

For the original decomposition case, Figure 4 shows the break up of communication time between the ghost cell update (near neighbor communication), and the global communication time.

**Fig. 3 Distribution of Communication Time (Original)**

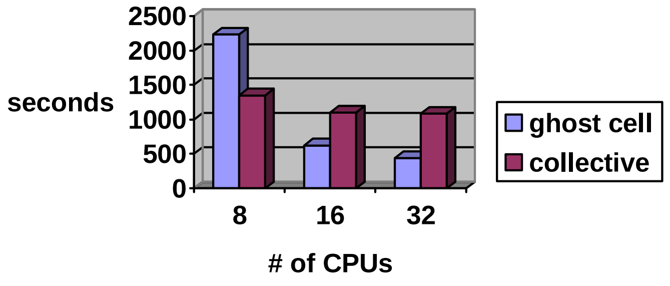
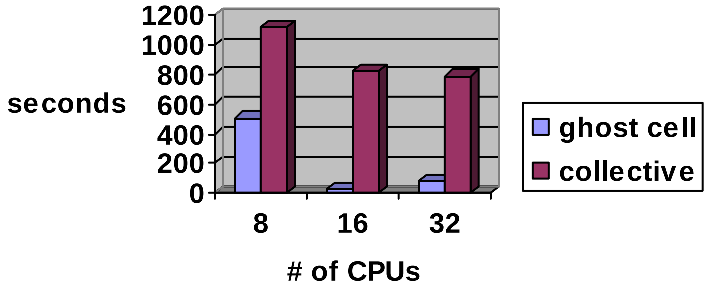


**Fig4. Distribution of Communication Time (Merged Contact)**



With the original decomposition case, for the 8 CPU case, the ghost cell update is quite dominant, suggesting that imbalance is driven by computations in a domain to finish and waiting for the posted messages to arrive. However, for the 16, and 32-CPU cases, the ghost cell updates are roughly only a fraction (about 60% of the collective communication time for the 16-CPU case, and about 40% for the 32-CPU case). Figure 4 shows the same distribution for the merged contact case.

Clearly, the ghost cell update is done considerably faster for this case compared to the original case. Even though the collective communication times are smaller than the original case (by at least 20% or so), there is still some load imbalance that shows up relatively prominently as the processors increase. The 16 CPU simulations take considerably less time for ghost cell updates for this case.

Next we will take a look at the message size distribution in the ghost cell update process for the original case. The dominant message passing MPI call is the non-blocking sends (MPI_Isend). Table 3 shows the details of the MPI_Isend calls. There are over 1 million MPI_Isend calls, and most of these are short messages, meaning less than 3500 Byte. (As before, these correspond to the process with the least total communication time). Clearly, the short messages dominate in the message passing phase, implying that latency of the communication subsystem is also important. Also, it appears that the percentage of smaller messages increases with the numbers of processes. Typically, IP based communication have much higher latencies, compared to proprietary switches, and shared memory systems. Also, the message volume increases for the 16 processor case – this behavior is nonlinear.

| **#of CPUs** | **# of MPI_Isends in millions** | **Average message size in Byte** | **% of messages < 3500 Byte** |
|---|---|---|---|
| **8** | **1.31** | **2536** | **75** |
| **16** | **1.96** | **3705** | **76** |
| **32** | **2.15** | **1490** | **88** |

Table 3 Distribution of Message Sizes

Next we will take a look at the collective communication call details, since they appear to dominate in the total communication time. The three collective communication calls MPI_Bcast, MPI_Allreduce & MPI_Alltoall contribute to collective communication timings. Out of these, MPI_Allreduce which does the global reduction takes a much bigger chunk of the collective communication time (about 80% of the time). Table 4 gives the average message size, and time taken for the MPI_Allreduce calls for the special decomposition case. From the well known PALLAS communication benchmark, it is known that on an IBM p690 system, the time taken for 500,000 MPI_Allreduce calls with about 1000 Byte messages is only 40 seconds. So the extra time indicates load imbalance, and this implies that load imbalance is still an issue with the improved decomposition.

| **# of CPUs** | **# of calls** | **Ave size of message** | **Time in seconds** |
|---|---|---|---|
| **8** | **567818** | **700** | **1100** |
| **16** | **567982** | **788** | **266** |
| **32** | **417350** | **1280** | **607** |

Table 4 MPI_Allreduce message sizes and elapsed time

## Smaller Model

For the three car collision model, we only used a sliding interface decomposition for the special decomposition case because scaling factors do not play a significant role in the overall performance improvement for this model. To show the effect of both sliding interface decomposition and scaling factors, a smaller model (about 200,000 elements) is chosen for this experiment. The full simulation corresponds to 65,000 cycles, and the full simulation was done on this model for the 2 cases: the default decomposition case, and the special decomposition case. The special decomposition included the decomposition on one large sliding interface with an entry 'sx 15' for the scaling shift in the axial direction.

Fig 5 Elapsed time comparisons

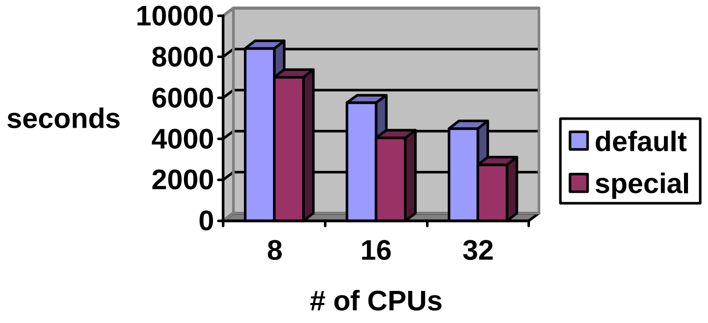


Figure 5 shows the elapsed time for both of these 2 cases for 8, 16 and 32 CPUs. The computational performance in the 'special' case significantly improves with the increased number of processors. The 32 CPU elapsed time decreased by about 40% with the 'special' decomposition. The 'special' case also scales significantly better (over 36%). A comparison of the spread of number of floating point divide operations between the processes show that the 'special' case is a lot tighter, suggesting a much better load balancing.

Fig 6. Total Communication Time

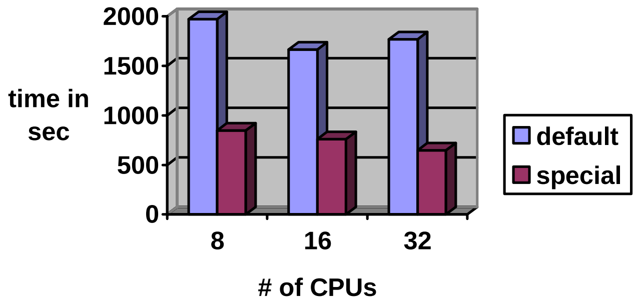


Figure 6 shows the total communication time of the process with the minimum total communication time for both these cases. There is a dramatic reduction in the total communication time with the 'special' case. The collective communication MPI calls appear to dominate the total communication as before, and the collective communication time increase with the increased number of processes. Clearly, for this model, there is a significant advantage with the 'special' decomposition.

## Conclusions

We studied the communication patterns and costs of the message passing parallel version of LS-DYNA (MPP-DYNA) using the IBM performance analysis tools. Our analysis suggests that the work load imbalance is the bottleneck of the turn-around (elapsed) time of car crash simulation using MPP-DYNA. Merging small contacts and special decomposition techniques may reduce the turn-around time significantly for some models.

## Acknowledgements

The authors would like to thank Dr. Jeff Zais for his comments and corrections.

### References

1. Lin, Yih-Yih. "A Quantitative Approach for Determining the Communication and Computation Costs in MPP LS-DYNA Simulations, FEA Worldwide News, January 2004.
2. Roh, Youn-Seo, and Fong, Henry, "Recent Developments of LS-DYNA Performance Optimization", K-1-33, 4th European LS-DYNA Users Conference
3. http://www-users.cs.umn.edu/~karypis/metis/
4. http://www.cs.sandia.gov/CRF/chac.html
5. http://www.gre.ac.uk/jostle
6. The POWER4 processor Introduction and Tuning Guide, http://www.ibm.com/redbooks SG24-7041-00, 2001
7. http://hpcf.nersc.gov/software/ibm/hpmcount/
8. http://www.hpcx.ac.uk/support/documentation/ IBMdocuments/mpitrace
9. http://www.topcrunch.org