# The Advantages of HP-MPI for MPP LS-DYNA

Yih-Yih Lin
*Hewlett-Packard Company*
*3000 Waterview Parkway*
*Richardson, TX 75080*
*Telephone: 1 (978) 841-7650*
*Email: yih-yih.lin@hp.com*

## Abstract

*The standard portable message-passing library MPI is the software tool that drives the parallelism in MPP LS-DYNA. MPI is required to operate in a complex environment: Currently, the major computer architectures include X86, X86_64, and Intel Itanium 2; the major operating systems include Linux, Window, and UNIX; diverse interconnects and switches, using different protocols, are offered by various vendors; and furthermore, in recent years most computer architectures have been evolved into multiple-core from single-core architecture. A well-implemented MPI should achieve the following goals in such a complex environment: (1) supporting all major computer architectures, operating systems, interconnects and switches; (2) being user friendly; (3) being optimized for the performance of the application. In this paper, an in-depth demonstration, using MPP LS-DYNA, on how HP-MPI achieved these goals is presented.*

## Introduction

Parallel applications, such as MPP LS-DYNA, run on a collection of computers that are connected with shared media or switches [1], which are commonly referred as interconnects and switches. Currently, the main computer architectures include X86, X86_64, and Intel Itanium 2, and the major operating systems include Linux and Microsoft Windows. Shared memory is a well-known shared media, and a variety of switches, using different communication protocols, are available from various vendors, such as Voltaire and Myrinet. The standard portable message-passing interface library (MPI) is the software tool that drives the parallelism in MPP LS-DYNA. There are several implementations of MPI, including MPICH, LAM/MPI, Intel MPI Library, Scali MPI Connect, and HP-MPI. In this paper, we will describe the advantages of HP-MPI over other MPI implementations in dealing with such a complex environment.

## HP-MPI Supports the Broadest Ranges of Computer Architectures, Operating Systems, and Interconnect Protocols

Unlike other MPI implementations, HP-MPI strives to support all major computer architectures, operating systems, and interconnect protocols with the same user interface. Table 1 lists computer architectures, operating systems, and interconnect protocols supported by HP-MPI. It is safe to claim that HP-MPI supports the broadest ranges of these three categories. For example, the generic MPICH, which works only with the protocol TCP/IP.

| | |
|---|---|
| Architectures | X86, X86_64 (including Intel Xeon and AMD Opteron), Intel Itanium 2 |
| Operating Systems | Linux, Microsoft Windows, HP-UX |
| Interconnect Protocols | Shared Memory, TCP/IP, VAPI, ITAPI, ELAN, MX, GM, UDAL, SRQ, RDMA |

Table 1. Architectures, operating systems, and interconnect protocols supported by HP-MPI.

## HP-MPI Is the Most User-Friendly Among All MPI Implementations

From its start, HP-MPI has been designed with the greatest concern about user-friendliness, and so it offers many advantages to MPP LS-DYNA users.

1. A single MPP LS-DYNA executable for all supported interconnect protocols: HP-MPI is a shared library on each platform and has a mechanism to select interconnect protocols automatically. As a result, HP-MPI allows LSTC to build a single MPP LS-DYNA executable that works on all interconnect protocols. The selection of protocol in MPP LS-DYNA/HP-MPI is simply an option in the mpirun command; for example, the mpirun option -*VAPI* will select the VAPI protocol.
2. Ready-to-use MPP LS-DYNA executable: Some MPI implementations, such as MPICH, have only one interconnect protocol available; and some other MPI implementations, such as MPI/LAM, have only a limited number of interconnect protocols available. Consequently, users of other MPI implementations than HP-MPI often have to build the MPP LS-DYNA executable themselves. In contrast, since HP-MPI allows a single MPP LS-DYNA to be built that works on a wide range of interconnect protocols, as described in point 1 above, the users have never had a need to rebuild the executable themselves.
3. A single user interface for all supported platforms and interconnect protocols: HP-MPI is built with a single source code for all the computer architectures and operating systems shown in Table 1. As a result, users of HP-MPI will use the same user interface to invoke MPP LS-DYNA regardless of the type of computer architectures or operating systems.
4. Sanity tests for interconnects and switches provided: As now, a simple ping-pong program to test data integrity and bandwidth is provided in HP-MPI. More sanity test programs are planned for future HP-MPI releases.
5. Cluster tools for users' convenience: Tools that allow users to use a cluster configuration more effectively are planned for future HP-MPI releases.
6. No requirement for end users to deal with HP-MPI license: HP-MPI has a built-in license for MPP LS-DYNA, and hence the users are not required to deal with HP-MPI license.

## Performance of HP-MPI Is Optimized

HP-MPI strives to optimize its performance when opportunity arises. The best example of this design philosophy is the recent implementation of CPU-binding (or affinity) feature in HP-MPI to get the most performance out of NUMA servers.

Moore's law, a statement made 40 years ago, expects the number of transistors on a chip doubles about every two years. In essence, the challenge of Moore's law is to increase the performance of processors without increasing the cost every so few years. However, such an expectation now appears unsustainable, by increasing transistor density alone, due to increased power

consumption and unmanageable complexity of the integrated circuit. The recent advent of multi-core architecture is an alternative answer to Moore's law. The idea behind the multi-core architecture is the replicating of identical cores, i.e., putting more than one core on a die; in essence, the multi-core architecture is to quit frequency race and to allow each core run at a more comfortable speed. Shown in Figure 1 is an example of a dual-core system, which has two processors, each of which has dual cores. (The system is called 2P/4C system because it has 2 processors and a total of 4 cores.) Reaping benefit from the multi-core architecture requires parallelism in applications, and MPP LS-DYNA, with its "massive parallelism," is one such an application.

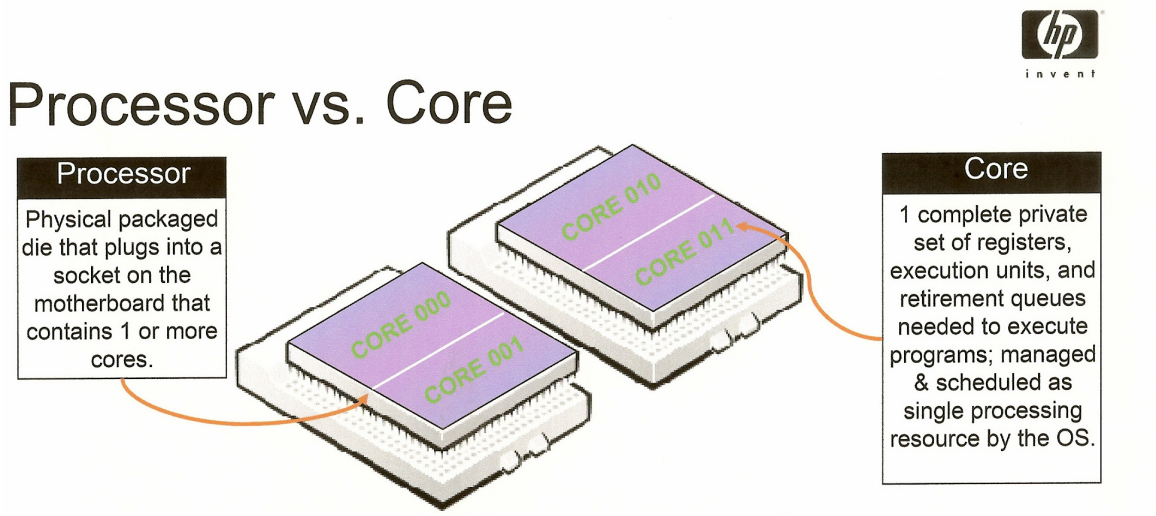


Figure 1. An example of 2-processor/4-core system, which means a processors populate 2 sockets with 2 cores per processor.

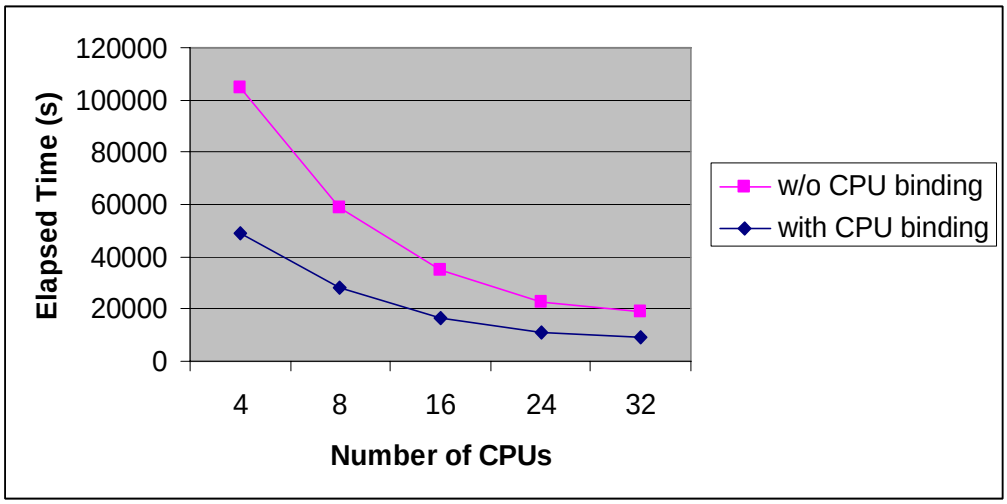


Figure 2. The elapsed times of the 3-vehicle-collision model on dual-core HP DL585 system with and without CPU binding.

One example for multi-core architecture is the multi-core AMD Opteron, which belongs to the class of NUMA (non-uniform memory access) systems. All NUMA systems use multiple memories that are physical distributed with the processors. The physically separate memories are addressed as a logically shared address space, and a memory reference can be made by any core to any memory location. For all NUMA machines, the memory access time depends on the location of a data word in memory. In other words, memory accessed by a core in a multi-core system can be either local (attaching to the core's processor) or remote, with different access times.

The CPU-binding feature in HP-MPI, invoked by the mpirun option *–cpu_bind*, restricts a core to always access that core's local memory, which reduces an application's overall memory access time and hence the application's elapsed time. Figure 2 shows the performance of MPP LS-DYNA/HP-MPI on dual core HP DL585 (based on AMD Opteron), with numbers of CPUs from 4 to 32, on the 3-vehicle-collision problem with and without CPU binding; the performance advantages of the former (with) over the latter (without) range from 7 to 12 percent.

## Conclusion

As demonstrated in this paper, there are numerous advantages of using MPP LS-DYNA with HP-MPI. Therefore, we encourage MPP LS-DYNA users to use HP-MPI on both Linux and Window platforms.

### References

1. David A. Patterson and John L. Hennessy [1996]. Computer Architecture: a Quantitative Approach, 2nd Edition, Morgan Kaufmann Publishers, Inc.