Hybrid (MPP+OpenMP) version of LS-DYNA

LS-DYNA Forum 2011

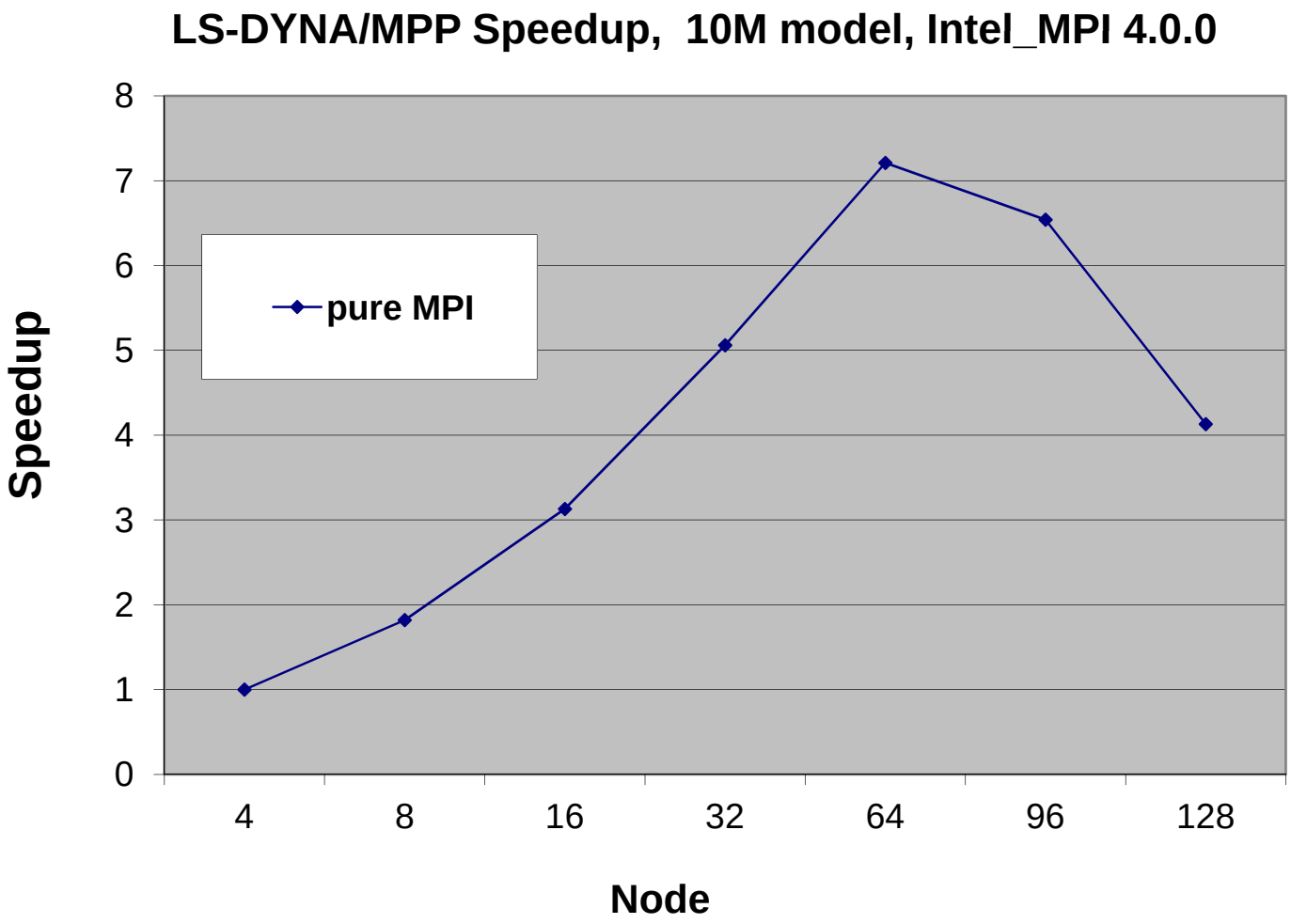
Jason Wang

Oct. 12, 2011

Outline

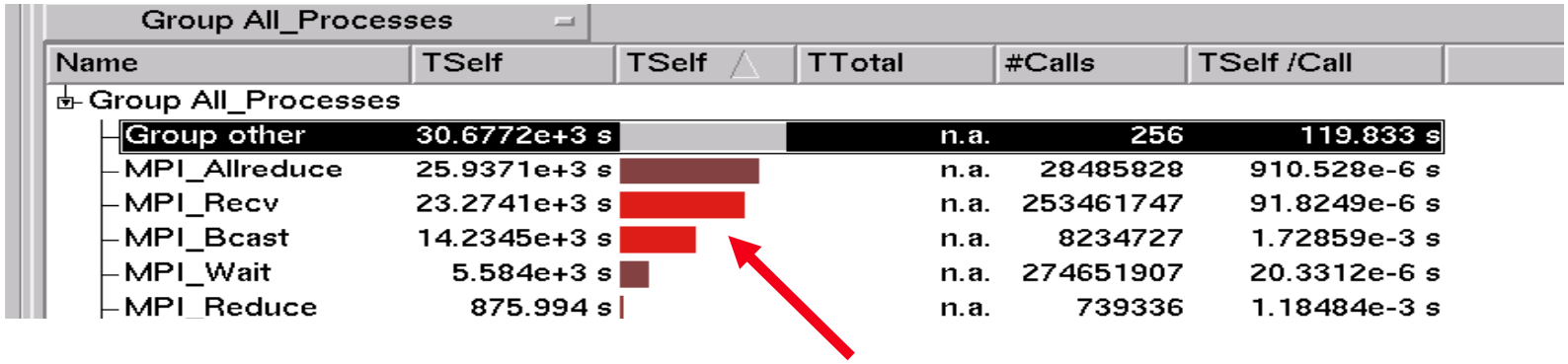
1) *Why MPP HYBRID*
2) *What is HYBRID*
3) *Benefits*
4) *How to use HYBRID*

Why HYBRID LS-DYNA



Why HYBRID LS-DYNA

32x8 cost profile

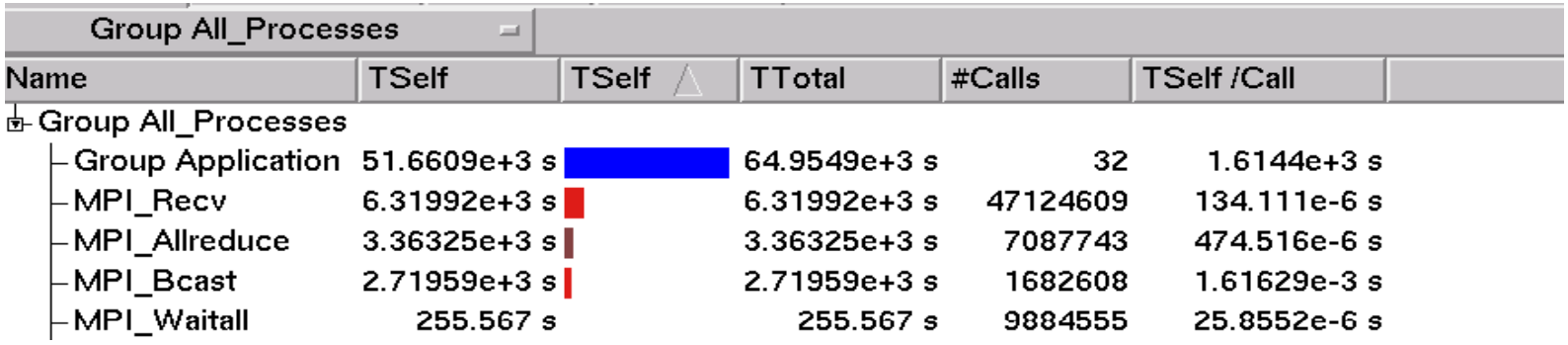


Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
Group other	30.6772e+3 s		n.a.	256	119.833 s
MPI_Allreduce	25.9371e+3 s		n.a.	28485828	910.528e-6 s
MPI_Recv	23.2741e+3 s		n.a.	253461747	91.8249e-6 s
MPI_Bcast	14.2345e+3 s		n.a.	8234727	1.72859e-3 s
MPI_Wait	5.584e+3 s		n.a.	274651907	20.3312e-6 s
MPI_Reduce	875.994 s		n.a.	739336	1.18484e-3 s

MPI cost becomes dominant

4x8 cost profile

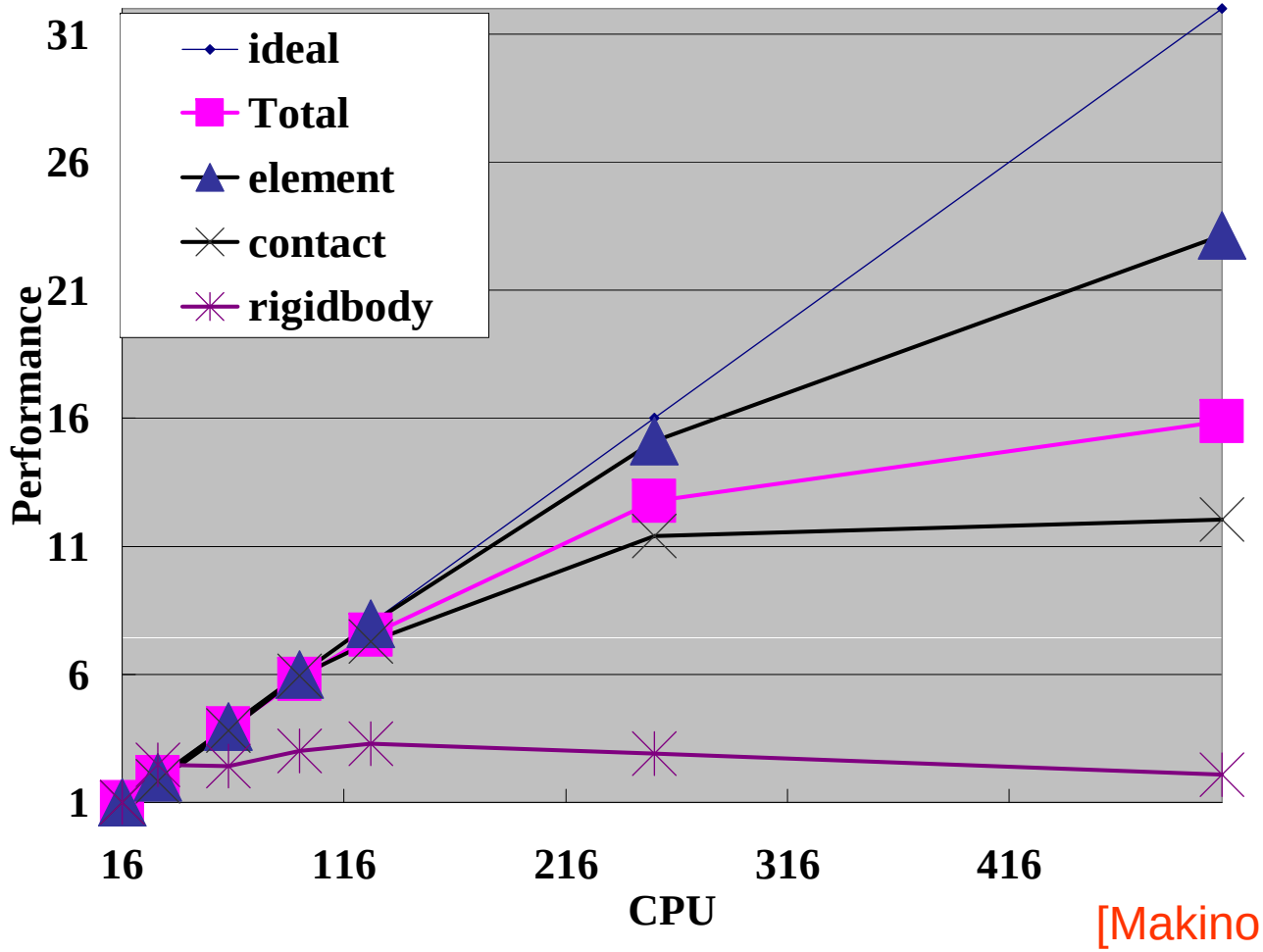


Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
Group Application	51.6609e+3 s		64.9549e+3 s	32	1.6144e+3 s
MPI_Recv	6.31992e+3 s		6.31992e+3 s	47124609	134.111e-6 s
MPI_Allreduce	3.36325e+3 s		3.36325e+3 s	7087743	474.516e-6 s
MPI_Bcast	2.71959e+3 s		2.71959e+3 s	1682608	1.61629e-3 s
MPI_Waitall	255.567 s		255.567 s	9884555	25.8552e-6 s

Why HYBRID LS-DYNA

- The scalability depends on the numbers of CPU's. There is not an ideal scaling for a large numbers of CPU's.

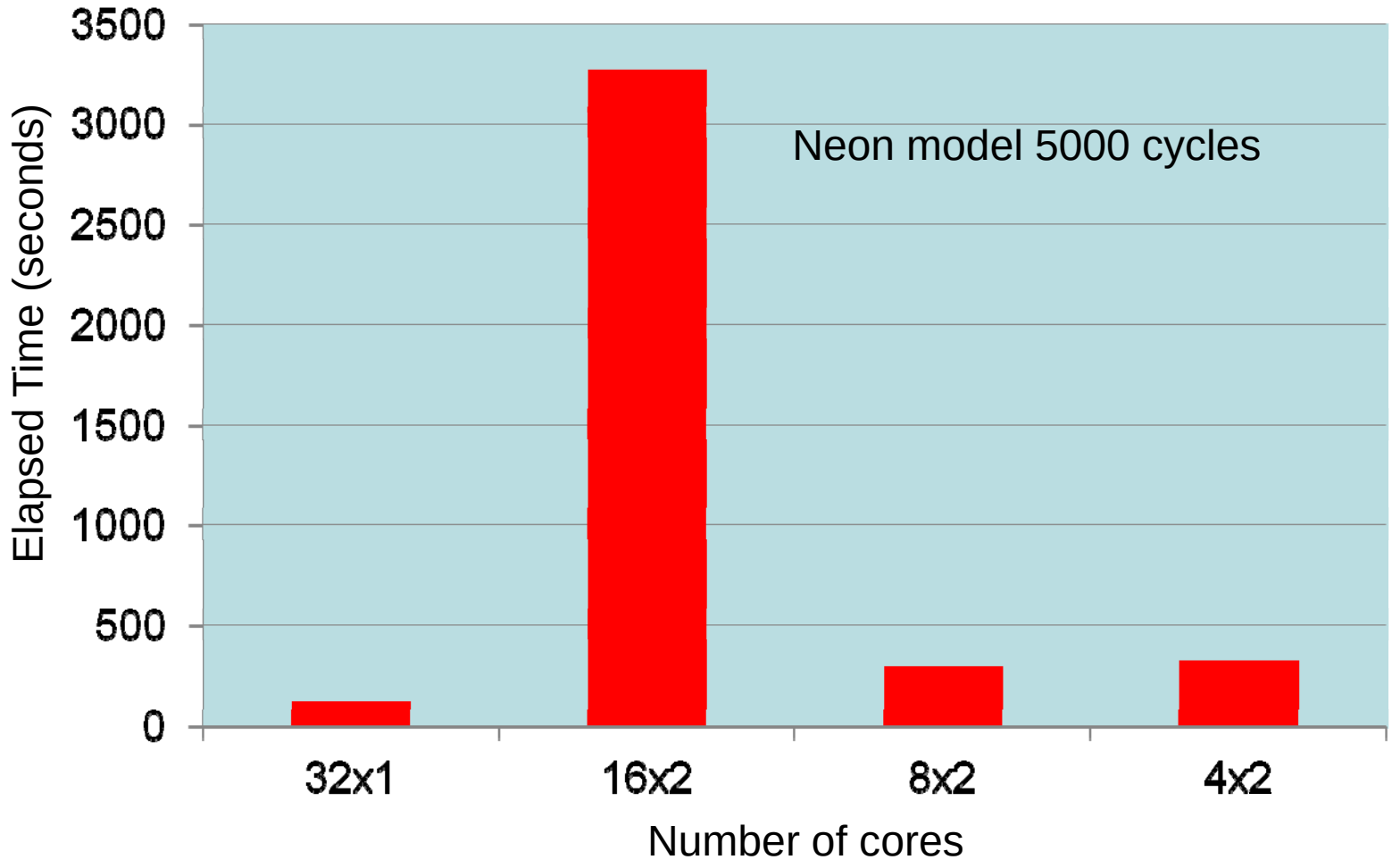


[Makino, 2008]

Why HYBRID LS-DYNA

Network bandwidth problem:

New chip with more cores (8/socket, 16/socket, etc)



Why HYBRID LS-DYNA

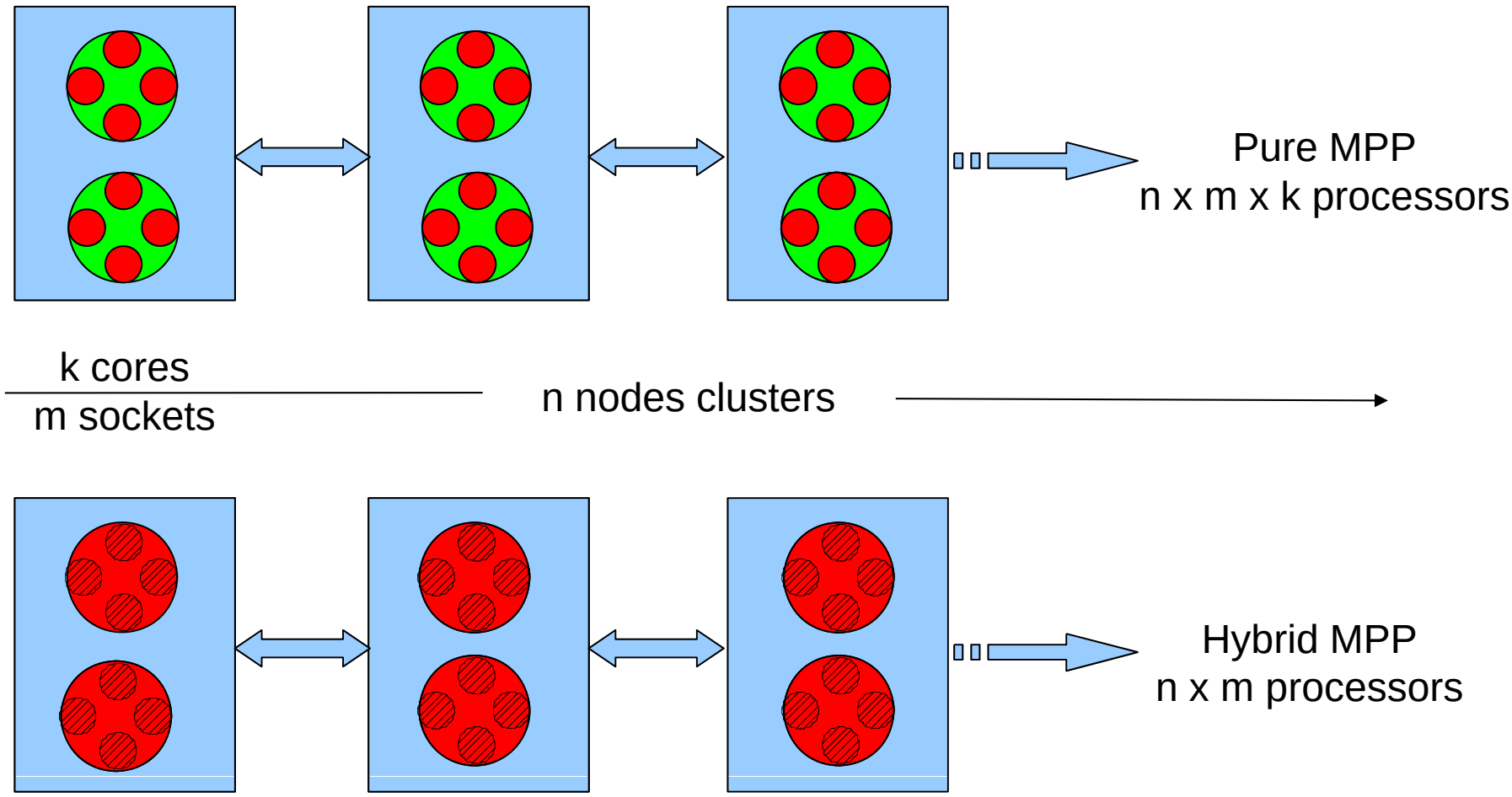
1. Large number of cores per LS-DYNA/MPP job
   - Scalability is not always good while increasing number of cores. Most of computer center find their own “sweet spot” and keep core/job.
   - Lots of contact definitions in the model which reduces the scalability
   - Contact does not scale linearly while increasing cores

2. Network bandwidth problem
   - Interconnect cannot catch up with core count increase

3. Numerical Noise
   - Numerical variations from changing number of cores due to decomposition

4. Implicit LS-DYNA/MPP
   - Requires lots of memory
   - Needs very good disk I/O capabilities for out of core solution

What is HYBRID



HYBRID LS-DYNA/MPP

1. *Large Number of Cores*
2. *Limited Network Resource*
3. *Scalability and Numerical Consistency*
4. *Implicit Solver on Explicit Hardware*

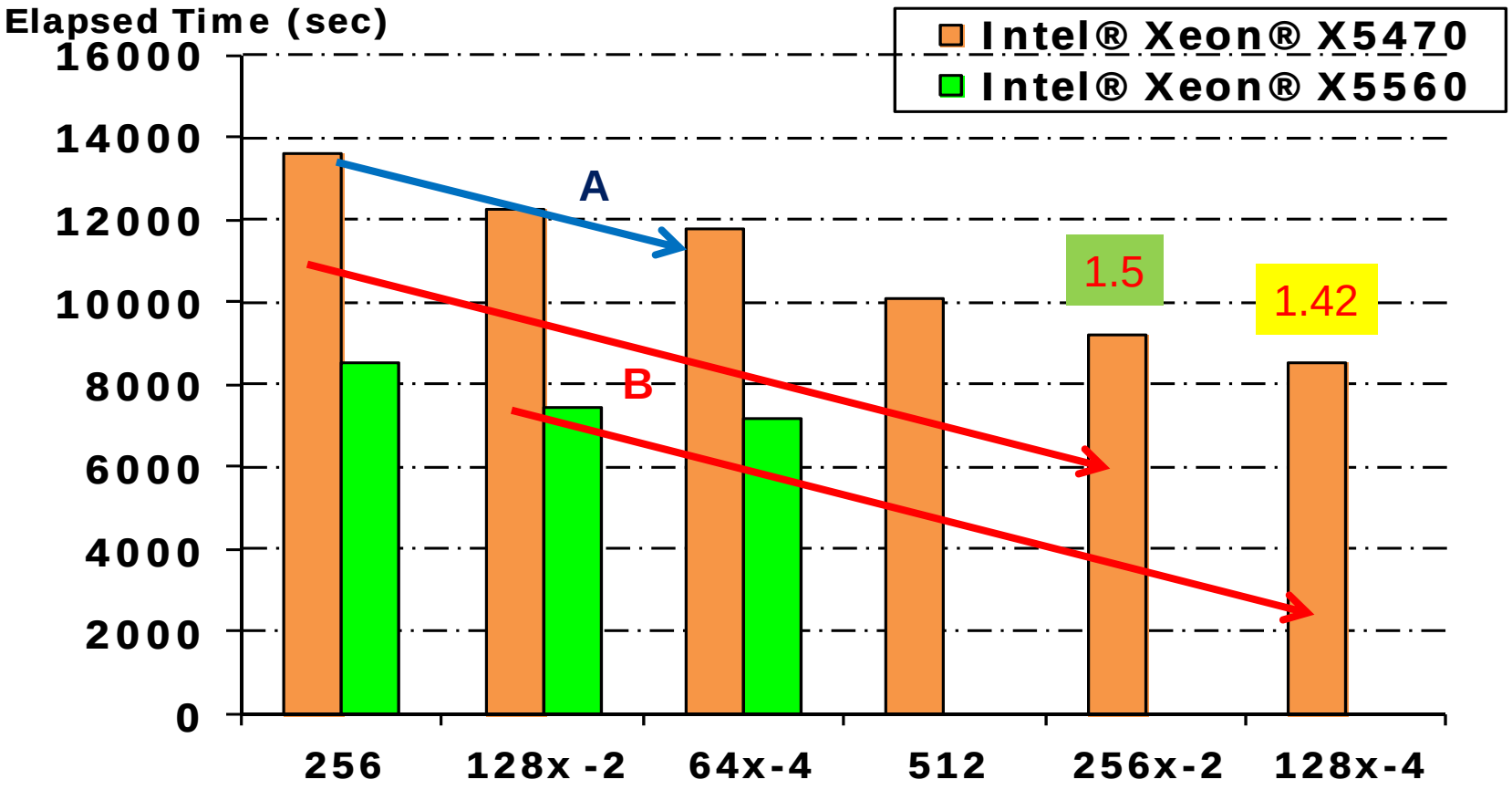
Large Number of Cores

- Current Multi-core chip is the platform

  ➢Coarse grain: Domain decomposition is very efficient for MPI algorithm
  ➢Fine grain: OpenMP is very friendly and efficient for loop parallel

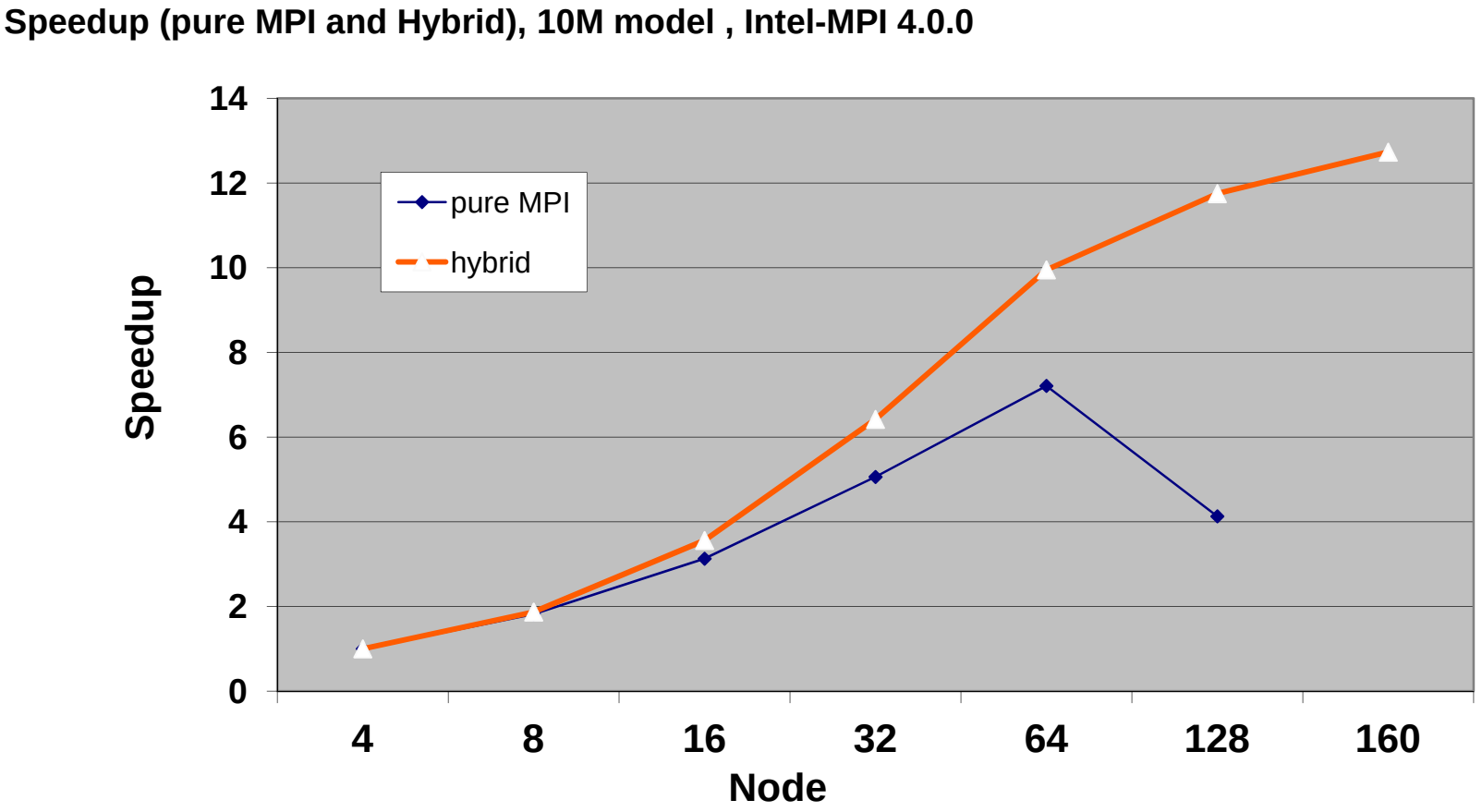
- Using same number of cores, LS-DYNA/MPP hybrid greatly reduce the network traffic

Large Number of Cores



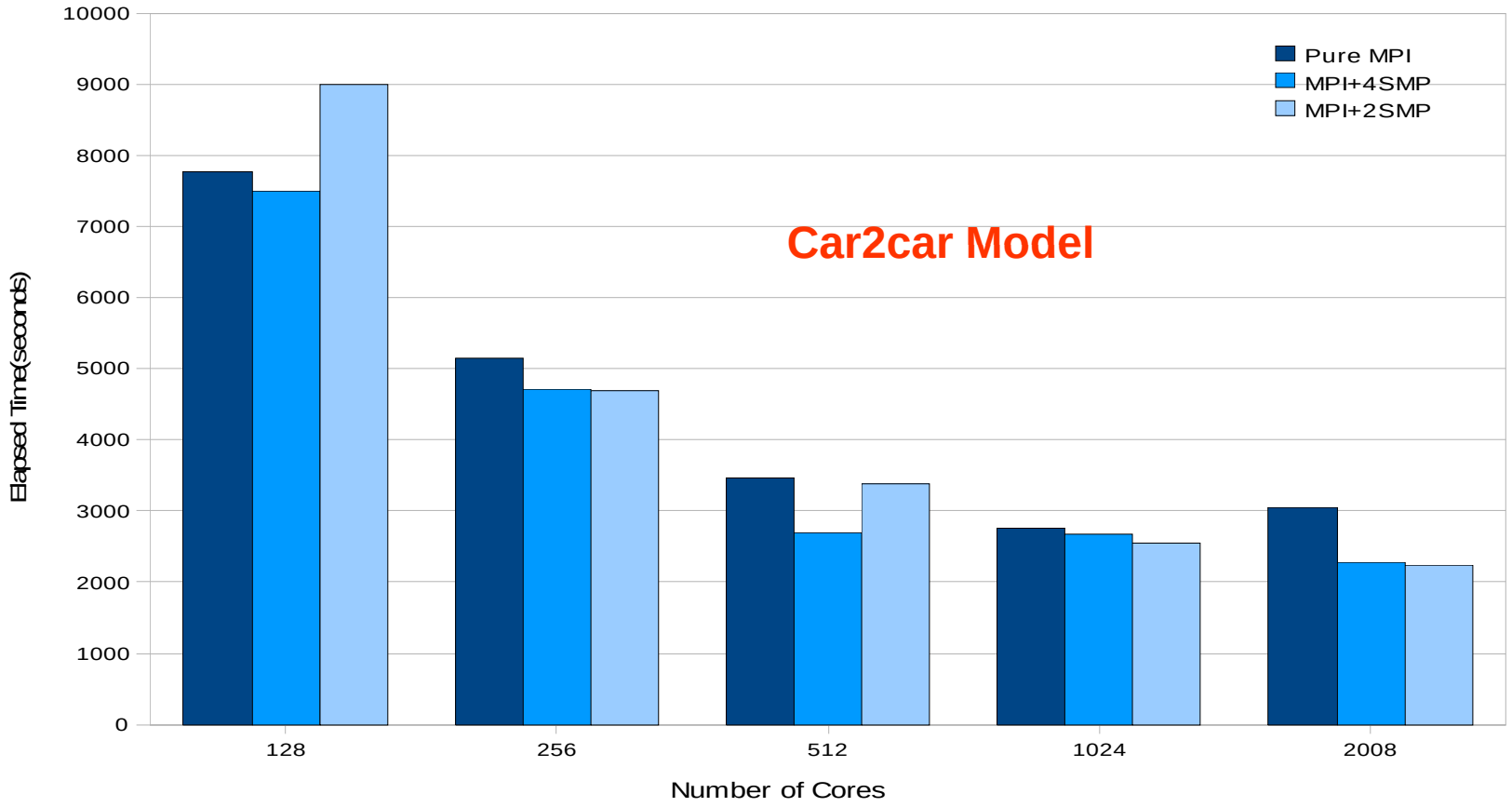
A. Constant cores: Gain from reducing the MPI traffic
B. Constant MPP: Gain from OpenMP

Large Number of Cores



Large Number of Cores

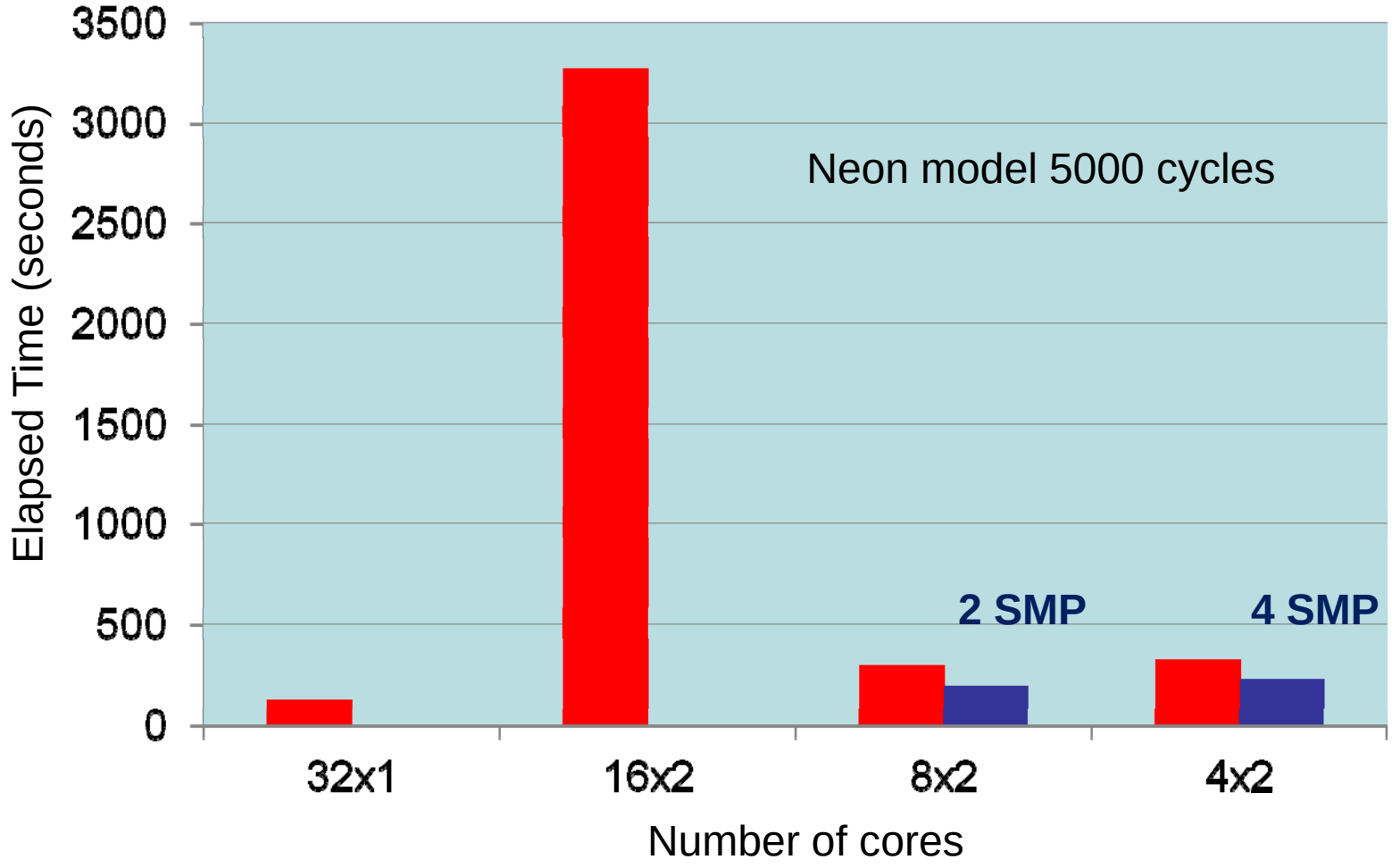
Performance Comparison on Windows Server 2008



Limited Network Resource

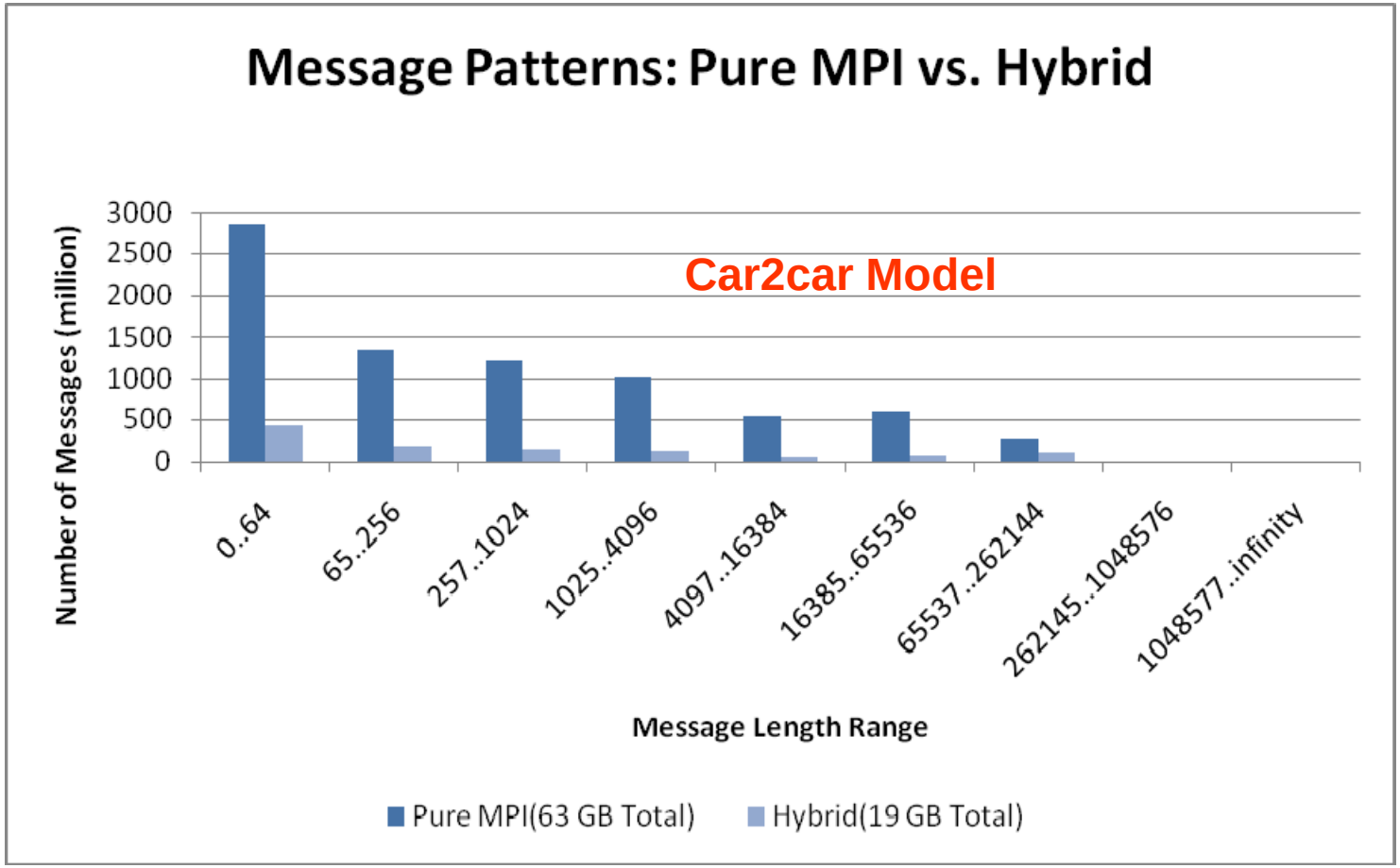
Network bandwidth problem:

New chip with more cores (8/socket, 16/socket, etc)



Limited Network Resource

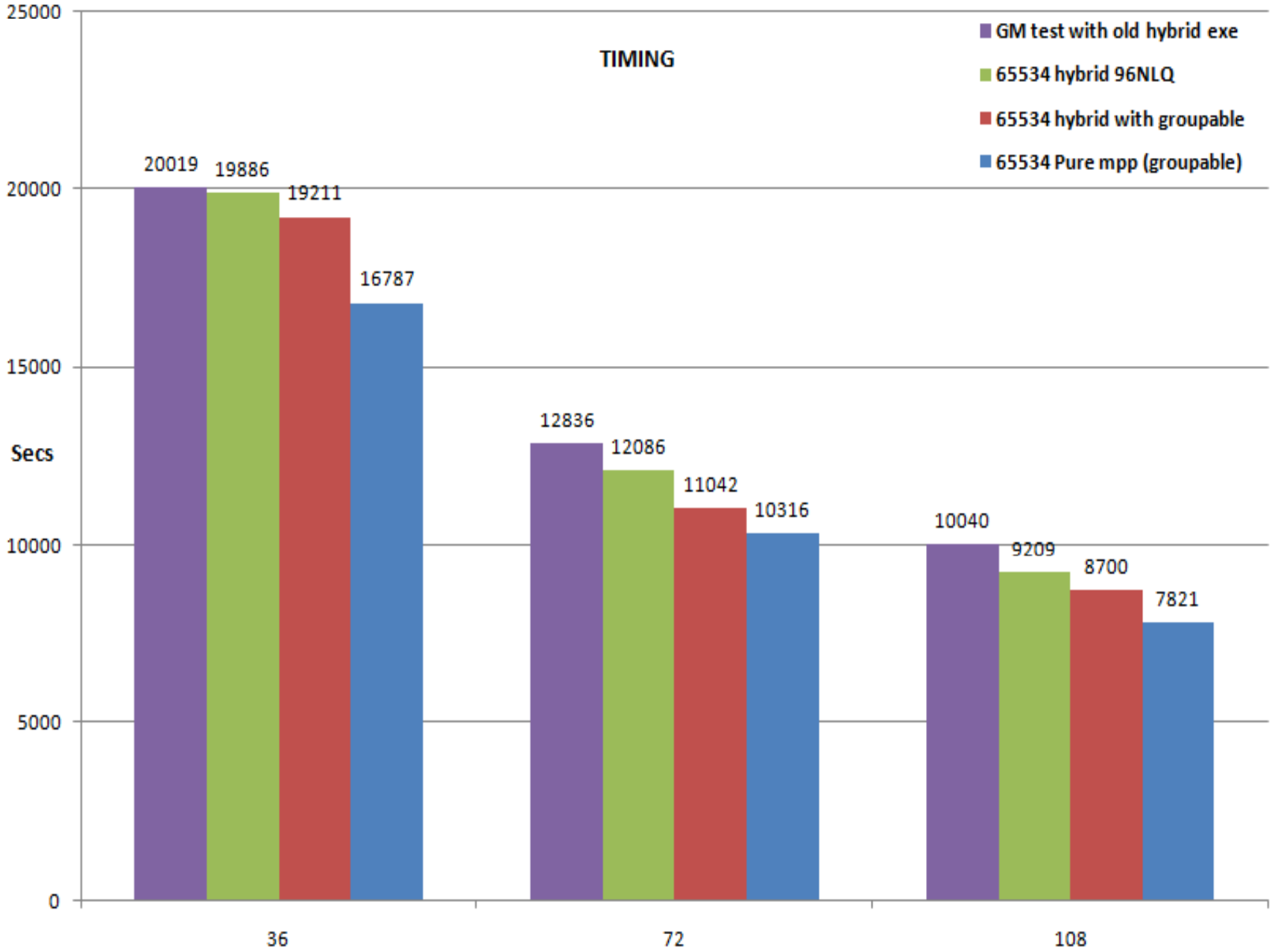
Message Across Network



- Hybrid greatly reduce the amount of data through network and provide better scaling to large number of processors

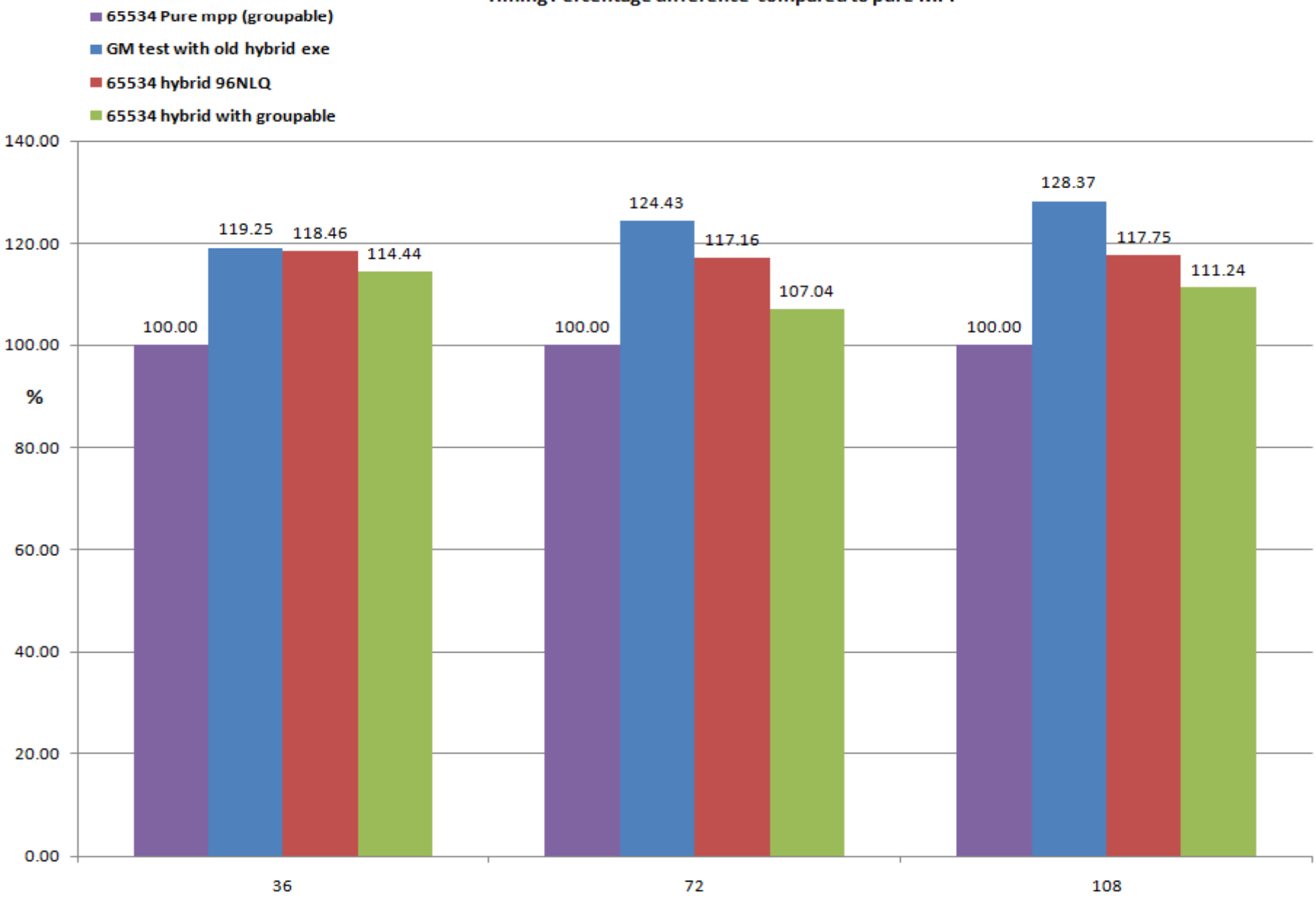
Scalability and Numerical Consistency

NCAP Model

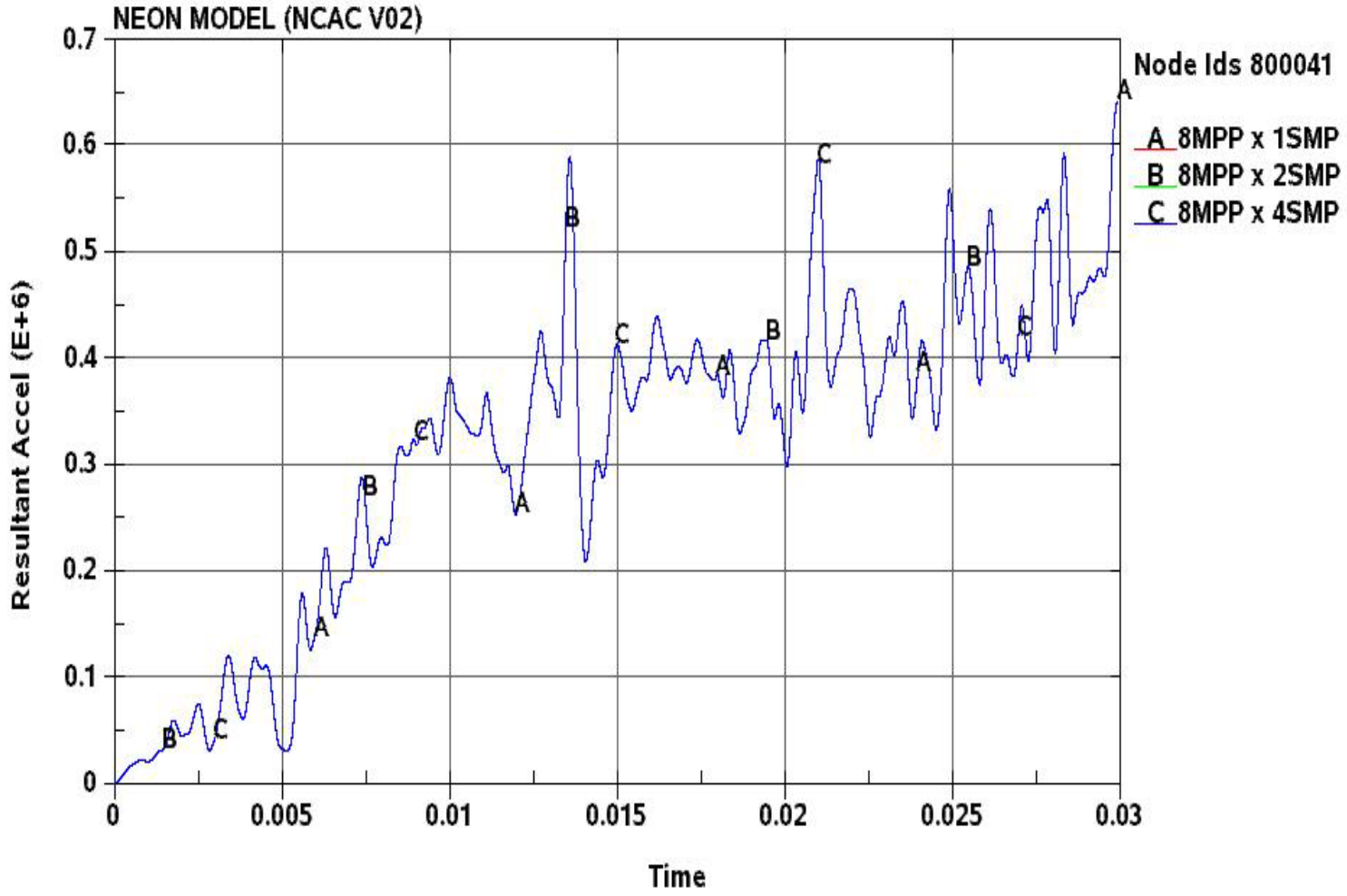


Scalability and Numerical Consistency

NCAP Model



Scalability and Numerical Consistency



- Consistent results is be obtained with fix decomposition and changing number of SMP threads ncpu=-#

Implicit Solver on Explicit Hardware

Typical hardware configuration for explicit analysis

- High end X86-64 dual sockets node (dual cores to six cores)
- 2-4GiB/core, 4-24GiB/socket
- 1-2 disk/node

If you can use in core solver on all available cores, you are fine!

If you need “memory=1000m” for your implicit job

In core solver:

Pure MPP:

1000m*8B/DYNA_WORD = 8GiB/MPP(1 core)

3 MPP/socket + 3 idle core/socket

Hybird MPP:

~1000m*8B/DYNA_WORD = ~8GiB/MPP(2 cores)

3x-2 MPP/socket

Implicit Solver on Explicit Hardware

Out of core solver:

Pure MPP:

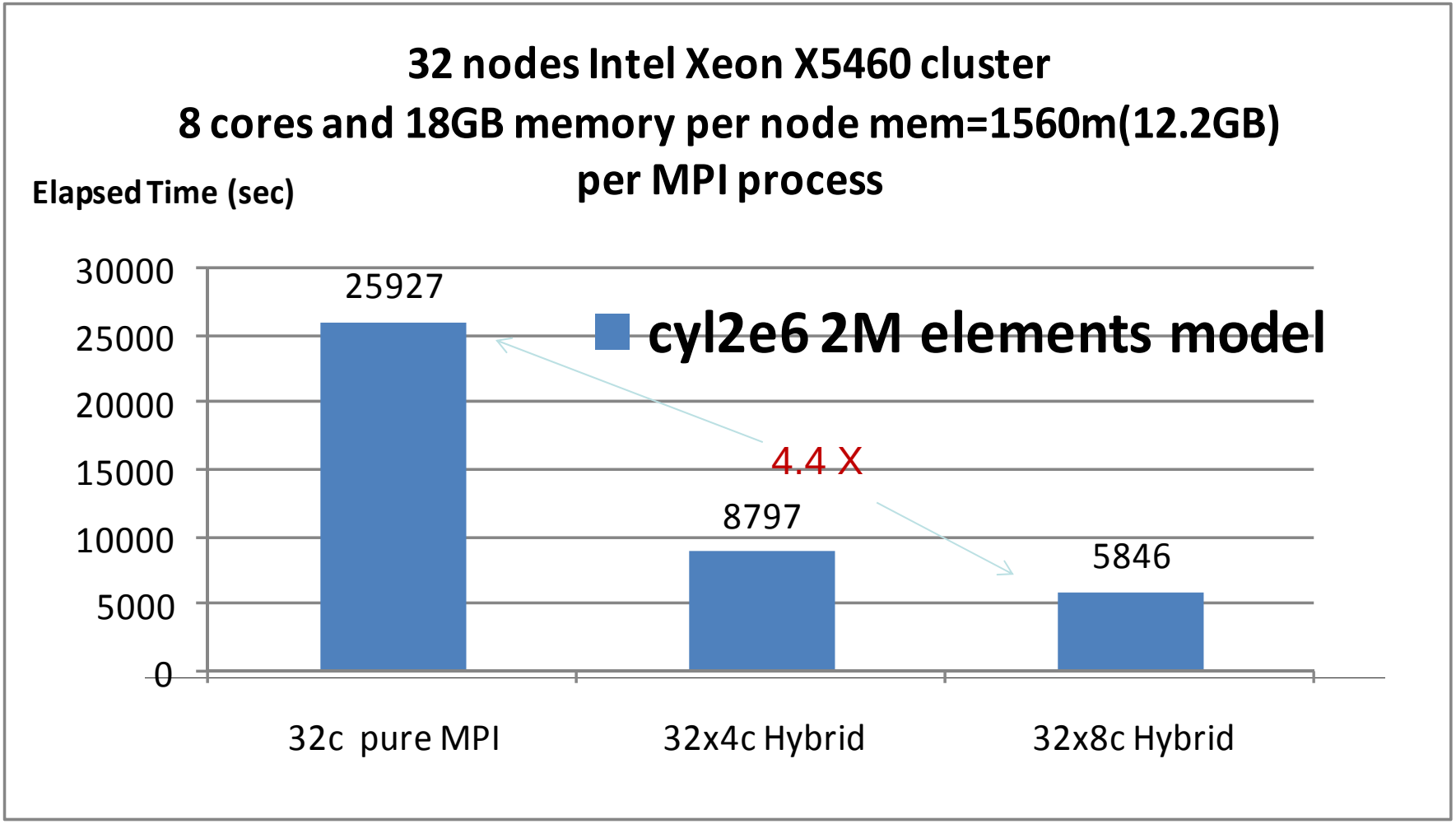
500m*8B/DYNA_WORD = 4GiB/MPP(1 core)
6 MPP/socket
12 MPP are writing to the disk through 1 I/O subsystem

Hybird MPP:

~1500m*8B/DYNA_WORD = ~12GiB/MPP(3 cores)
2x-3 MPP/socket
2MPP/socket
4 MPP are writing to the disk

Less MPI message traffic and I/O traffic

Implicit Solver on Explicit Hardware



Implicit Solver on Explicit Hardware

Performance on Linux AMD64 systems

No. of cores (node x socket x core)	WCT of Factor Matrix (seconds)	WCT for job to complete (seconds)
16 x 4 x 1	2055	14417
16 x 4 x 2	985	13290
16 x 4 x 4	582	29135
16 x 4 x 4omp (Hybrid)	960	9887

How to run Hybrid LS-DYNA/MPP

There is a consistent option (ncpu=-N) in LS-DYNA SMP version. Many customers used to run their jobs with the option in SMP era, even though there is about 10-15% performance penalty with the option.

LSTC added the option into LS-DYNA Hybrid version. So customers can use the option for getting consistent numerical result. However, there is a condition here. The condition is you need to fix the number of MPI processes at first.

For example, you select 12 MPI processes, then you can run your job in this way.

*mpiexec **–ppn M** –np 12 mpp971hyb i=input memory=xxxm memory2=xxm **ncpu=-N** p=pfile*

12 cores:	12 MPI processes x 1 OMP thread	(1 nodes x 12 cores)	M=12, N=1
24 cores:	12 MPI processes x 2 OMP threads	(2 nodes x 12 cores)	M= 6, N=2
36 cores:	12 MPI processes x 3 OMP threads	(3 nodes x 12 cores)	M= 4, N=3
48 cores:	12 MPI processes x 4 OMP threads	(4 nodes x 12 cores)	M= 3, N=4
……			
72 cores:	12 MPI processes x 6 OMP threads	(6 nodes x 12 cores)	M= 2, N=6

Then you can get consistent results with 12c, 24c,36c,48c, 60c, and 72c.

How to run Hybrid LS-DYNA/MPP

- Turn hyper threading off
- OMP_NUM_THREADS to SMP upper limit
- General variables for MPI

  Platform (HP) MPI

  -cpu_bind_mt=MASK_CPU:*string*

  -e MPI_THREAD_AFFINITY=packed

  Intel MPI

  -env I_MPI_PI_DOMAIN=*string*

  -env I_MPI_PIN_ORDER=compact

  -env KMP_AFFINITY=compact

How to run Hybrid LS-DYNA/MPP

- How to find out the *string*
- Find the core ordering

  cat /proc/cpuinfo | grep –i “physical id”

  Example: Dual 6 cores 0 0 0 0 0 0 1 1 1 1 1 1
- Pin application to cores sharing local resource

  Example: 3 SMP/MPP on each node

	CPU #1	CPU #0	HEX #
1st MPP	0 0 0 0 0 0	0 0 0 1 1 1	7
2nd MPP	0 0 0 0 0 0	1 1 1 0 0 0	38
3rd MPP	0 0 0 1 1 1	0 0 0 0 0 0	1C0
4th MPP	1 1 1 0 0 0	0 0 0 0 0 0	E00

String = *7,38,1C0,E00*

Conclusions

- *Better performance with same number of cores*
- *Consistent results while changing core count*
- *Flexibility using the computing resources*
- *Explicit/Implicit sharing the same hardware*