

# Using Nodes in a Cluster Efficiently

Glenn R. Luecke, Ying Li, Martin Cuma

Iowa State University, University of Utah

grl@iastate.edu, yingli@iastate.edu, mcuma@chpc.utah.edu

November 4, 2005

## Abstract

The purpose of this paper is to evaluate how to use nodes in a cluster efficiently by studying the NAS Parallel Benchmarks (NPB) on Intel Xeon and AMD Opteron dual CPU Linux clusters. The performance results of NPB are presented both with one MPI process per node (1 ppn) and with two MPI processes per node (2 ppn). One would like to run all applications on a cluster with two processors per node using 2 ppn instead of 1 ppn in order to utilize the second processor on each node. However, the performance results from running the NPB and from the memory bandwidth benchmarks, show that better performance can sometimes be achieved using 1 ppn. Our performance results show that the Opteron/Myrinet cluster is able to achieve significantly better utilization of the second processor than the Xeon/Myrinet cluster.

## 1. Introduction

Message Passing Interface (MPI) [1] is a standard library specification for message-passing used when writing parallel programs for scientific applications. Many cluster computers are purchased with two processors on each node sharing a common memory since the cost of adding a second processor is typically only about 20% more. Nodes are interconnected with a communication network such as Ethernet, Myrinet, Quadrics, or Infiniband for MPI communication. Scientific applications can be run on such a cluster with one MPI process per node (1 ppn) or with two MPI processes per node (2 ppn). Clusters with 2 processors per node are normally set up with default execution of MPI applications to be 2 ppn in order to utilize the second processor. However, running applications with 1 ppn may provide better performance than running with 2 ppn.

In this paper, we first compare the performance of the NAS Parallel Benchmarks (NPB) [2] running with 1 ppn and 2 ppn on an Intel Xeon/Myrinet cluster and on an AMD Opteron/Myrinet cluster. We then attempt to explain these performance results by (1) investigating the memory bandwidth of the Xeon and Opteron processors, and (2) by investigating the impact of MPI and the communication network on the performance when using 1 and 2 ppn. Our conclusions are presented at the end.

While memory subsystem characteristics of Xeon and Opteron dual processor machines in single-node setup are well known [3,4], they are not aware of any published reports that investigate the performance issues encountered when using 1 ppn and 2 ppn on multi-node distributed clusters.

## **2. Hardware and Software Environment**

The Intel Xeon/Myrinet cluster used for this study was the “Tungsten” cluster at the National Center for Supercomputing Applications (NCSA) [5]. This cluster is a 2560 processor (1280 node) Intel Xeon 3.2 GHz with a 1 MB level 3 cache cluster running Red Hat Linux 9.0 kernel 2.4.20. Each node has 3 GB of memory. The Myrinet 2000 switch was used for communication between nodes using the M3F-PCIXD-2 Myrinet NICs on the 133MHz PCI-X riser cards. Myricom’s GM 2.0.21 libraries were used. Multiple 128 port Myrinet switches were used and cabled as Myricom recommends for maximum performance [9]. On this machine, only one job executes on each node at a time. The Load Sharing Facility (LSF) batch system was used to run jobs. The MPI used was ChaMPIon/Pro MPI 1.2.0-3. Benchmarks were compiled using version 8.0 of the Intel Fortran 95 compiler and version 8.1 of the Intel C compiler. The `-O3` compiler option was used for both compilers. Due to lack of allocation on Tungsten, the “hpc-class” cluster at Iowa State University (ISU) was used for the Performance Analysis section in this paper. This cluster’s hardware is nearly the same as on the Tungsten cluster. It is an 88 processor (44 node) Intel Xeon 2.8 GHz with a 512 KB level 2 cache cluster running Red Hat Linux 8.0 with each node having 2 GB of memory and using Myrinet 2000 for communication between nodes with PCIXD cards and running GM 2.0.6. On this machine, only one job executes on each node at a time. The OpenPBS batch system was used to run jobs. The MPI used was MPICH-GM 1.2.5.10. Benchmarks were compiled using version 7.1 of the Intel Fortran 95 and C compilers.

The AMD Opteron/Myrinet cluster used for this study was the “Delicatearch” cluster at the Center for High Performance Computing (CHPC), University of Utah [8]. This cluster is a 512 processor (256 node) AMD Opteron 1.4 GHz with a 1 MB level 2 cache cluster running SuSE Linux 9.0 with 2.6.11 kernel. Each node has 2 GB of memory. The Myrinet 2000 switch was used for communication between nodes using the M3F-PCIXD-2 Myrinet NICs on the 133MHz PCI-X riser cards. Myricom’s GM 2.0.21 libraries were used. Multiple 128 port Myrinet switches were used and cabled as Myricom recommends for maximum performance [9]. On this machine, only one job executes on each node at a time. The PBS batch system was used to run jobs. The MPI used was MPICH-GM 1.2.6.14b. Benchmarks were compiled using version 2.2 of the Pathscale Fortran and C compilers. For the NPB benchmarks the `-O3 -ipa -WOPT:aggstr=0 -CG:movnti=0` compiler options were used. The last two compiler options were used to turn off optimizations that cause problems with the NPB. All the other benchmarks were run using only the `-O3 -ipa`

options. In order to evaluate the performance improvement with the more recent Opteron processors, we also ran some of the NPB on a 32 node cluster at CHPC with 2.6 GHz Opteron nodes (with 1 MB level 2 caches). The software environment for this cluster is the same as for “Delicatearch”. Table 1 gives the performance ratios of the 2.6 GHz Opteron to the 1.4 GHz Opteron using 4 MPI processes for the class B NAS Parallel Benchmarks. The performance improvements for the other benchmarks that we could run on this small cluster were similar.

	CG	EP	MG	FT	LU	IS
1 ppn	1.5	1.8	1.9	1.7	1.7	1.5
2 ppn	1.9	1.8	2.0	1.5	1.9	1.5

Table 1. Performance ratios of the (2.6 GHz Opteron)/(1.4 GHz Opteron) for the class B NPB.

### 3. NPB Performance Results

This section describes and presents the performance results of running six of the eight NAS Parallel Benchmarks (NPB) on the Xeon cluster and Opteron cluster with 1 ppn and 2 ppn. The SP and BT benchmarks were not used since they require the number of MPI processes to be a square making it impossible to compare 1 ppn versus 2 ppn. For example, if 25 nodes were used then these benchmarks could be run with 1 ppn but not 2 ppn when running with a fixed number of nodes. The timing methodology provided by the NPB was used. It is important to perform multiple timings for each test to determine the variability of the timings. Our timing results varied less than 5%.

The NPB benchmarks CG, MG, FT, LU, and IS require that the number of processors must be a power of two, so 2, 4, 8, 16, 32, 64, 128, and 256 processes were used. There are no special requirements on the number of processors for the EP benchmark, so to be consistent with the other benchmarks the same number of processes were used for the EP benchmark. Each benchmark was run with different problem sizes: “class A” is a small problem, “class B” is a medium sized problem, and “class C” is a large sized problem. In some cases the class C problem was too large to run on the clusters used. This section lists the performance results for class B problem sizes to keep this section from becoming too long and since the class B performance scaling results are similar to the results for class A and class C. Performance results for class A and class C problem sizes are in Appendix 1.

We compare the performance of each of these six benchmarks in two different ways. First, we fix the number of nodes and then run the six benchmarks with 1 ppn and also with 2

ppn. For example, if there are 128 nodes, then we compare the performance of using these 128 nodes using 128 MPI processes (1 ppn) and then with 256 MPI processes (2 ppn). Most data centers give users exclusive use of the allocated nodes and users are charged for these nodes whether or not the application is run using 1 or 2 ppn. Hence users are faced with the following question: “For a fixed number of nodes, will 1 ppn or 2 ppn give the best performance?” Notice that the scalability of the application will affect these comparisons. For example, it may be that some applications will perform better using 128 MPI processes than when using 256 MPI processes.

The second way we compare performance is to fix the number of MPI processes instead of the number of nodes. For example, when fixing the number of MPI processes at 128 means that the 1 ppn performance numbers uses 128 nodes and the 2 ppn performance numbers use 64 nodes. This second way of comparing performance answers the question: “For a fixed number of MPI processes, will 1 ppn or 2 ppn give the best performance?”

### **Test 1: The CG Benchmark**

The Conjugate Gradient (CG) benchmark solves an unstructured sparse linear system by the conjugate gradient method. It uses the inverse power method to find an estimate of the largest eigenvalue of a symmetric positive definite sparse matrix with a random pattern of nonzeros [2]. The size of the square matrix is 14000 with 11 nonzeros per row/column for class A, 75000 with 13 nonzeros for class B and 150000 with 15 nonzeros for class C. Figures 1.B.a – 1.B.c show the performance of CG with class B problem. The performance scaling for classes A and C is similar and is shown in Appendix 1 in figures 1.A.a – 1.A.c and 1.C.a – 1.C.c.

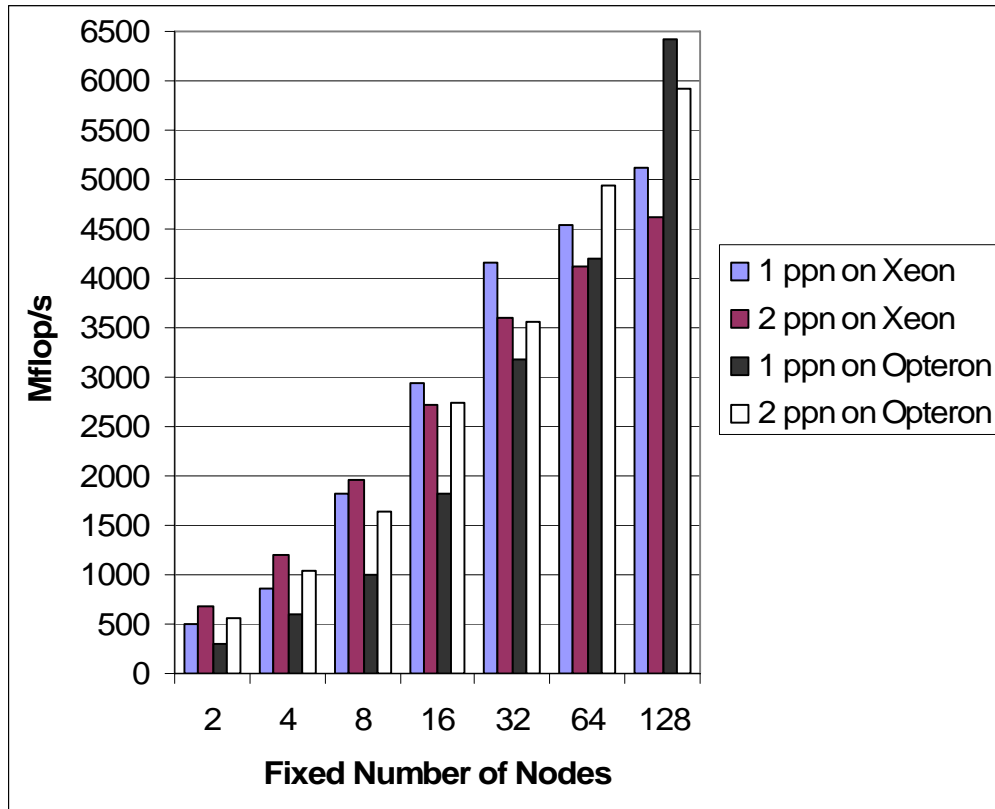


Figure 1.B.a: Class B CG benchmark for a fixed number of nodes.

From Figure 1.B.a, for the Xeon cluster, 2 ppn outperformed 1 ppn only for 2, 4, and 8 nodes for the class B problem size. For the Opteron cluster, 2 ppn outperformed 1 ppn for 2, 4, 8, 16, 32, and 64 nodes. Thus, using the second processor on nodes degraded performance for 16, 32, 64, and 128 nodes for the Xeon cluster but only degraded performance for 128 nodes for the Opteron cluster.

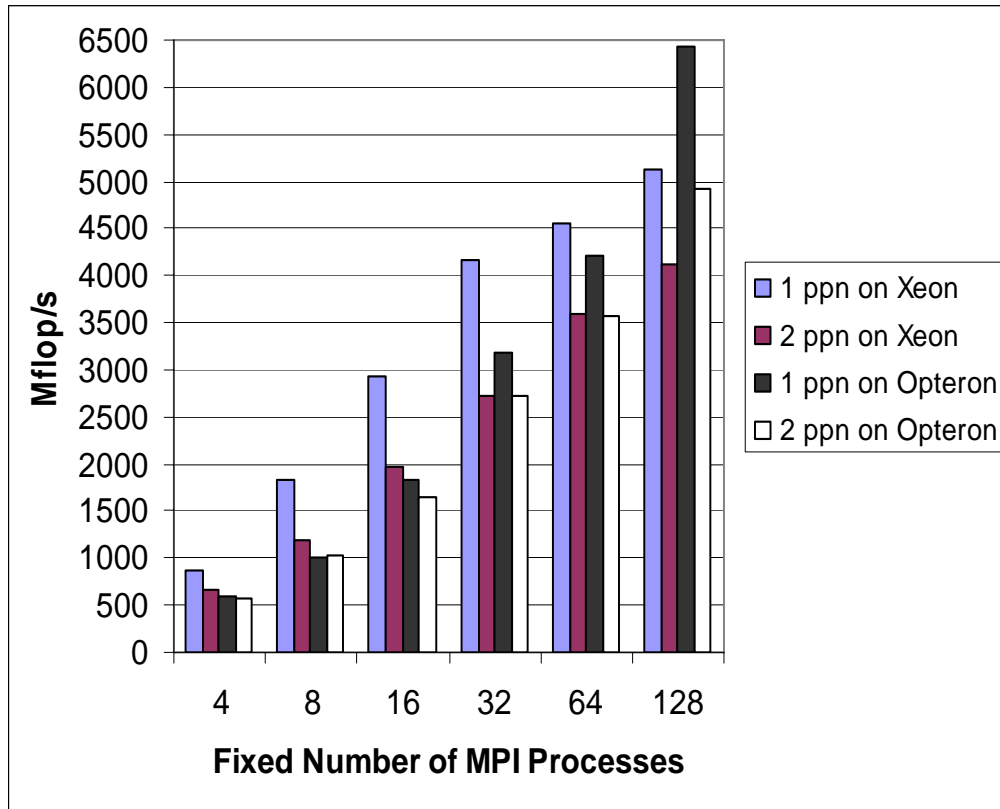


Figure 1.B.b. Class B CG benchmark for a fixed number of MPI processes.

Figure 1.B.b shows the performance comparisons for a fixed number of MPI processes running 1 ppn and 2 ppn on both clusters and figure 1.B.c shows the ratios  $(2 \text{ ppn Xeon}) / (1 \text{ ppn Xeon})$  and  $(2 \text{ ppn Opteron}) / (1 \text{ ppn Opteron})$ . Notice that the performance decrease when using 2 ppn on the Xeon cluster ranges from about 20% to 35% for this benchmark. Notice that the performance decrease when using 2 ppn on the Opteron cluster ranges from about 0% to 23%. Except for the Opteron cluster using 8 MPI processes, best performance is achieved when using 1 ppn when using a fixed number of MPI processes.

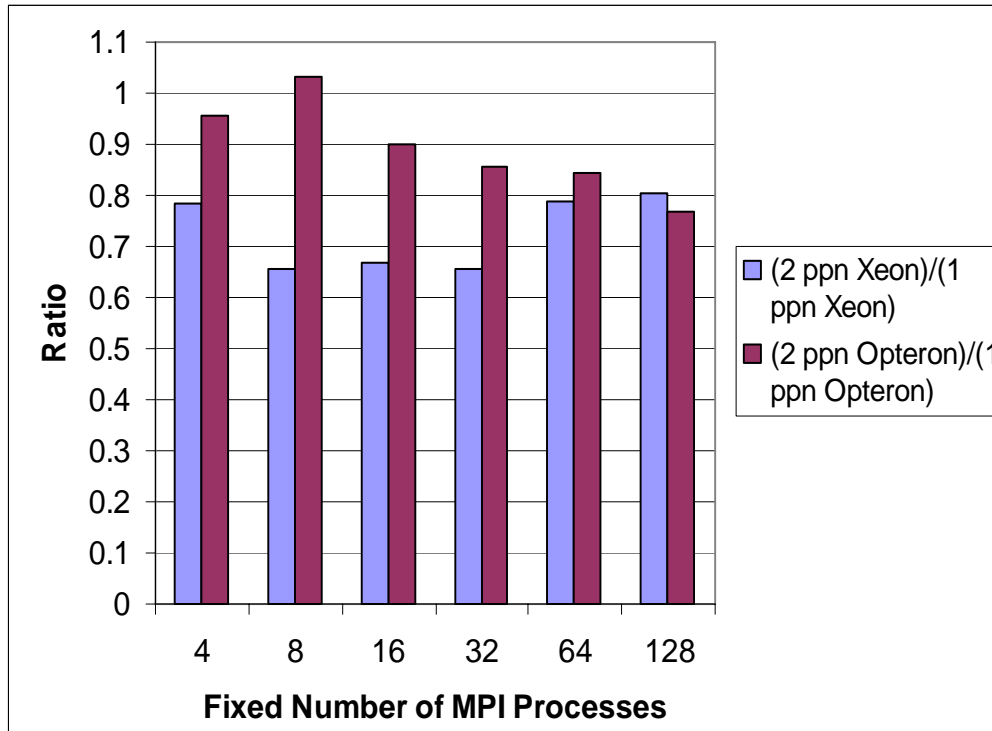


Figure 1.B.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

### Test 2: The EP Benchmark

The Embarrassingly Parallel (EP) Benchmark is typical of many Monte Carlo simulation applications.  $2^{28}$  pairs of random numbers are generated for class A,  $2^{30}$  pairs for class B and  $2^{32}$  pairs for class C. This benchmark requires almost no communication and puts minimal stress on the performance of the memory system [10, p. 10]. Figures 2.B.a – 2.B.c show the performance results of EP with class B problem. Similar results for class A and C are in Appendix 1. Since it is difficult to see performance differences for 2, 4, 8, and 16 nodes in Figure 2.B.a, Figure 2.B.aa has been added that shows the results for only these node counts.

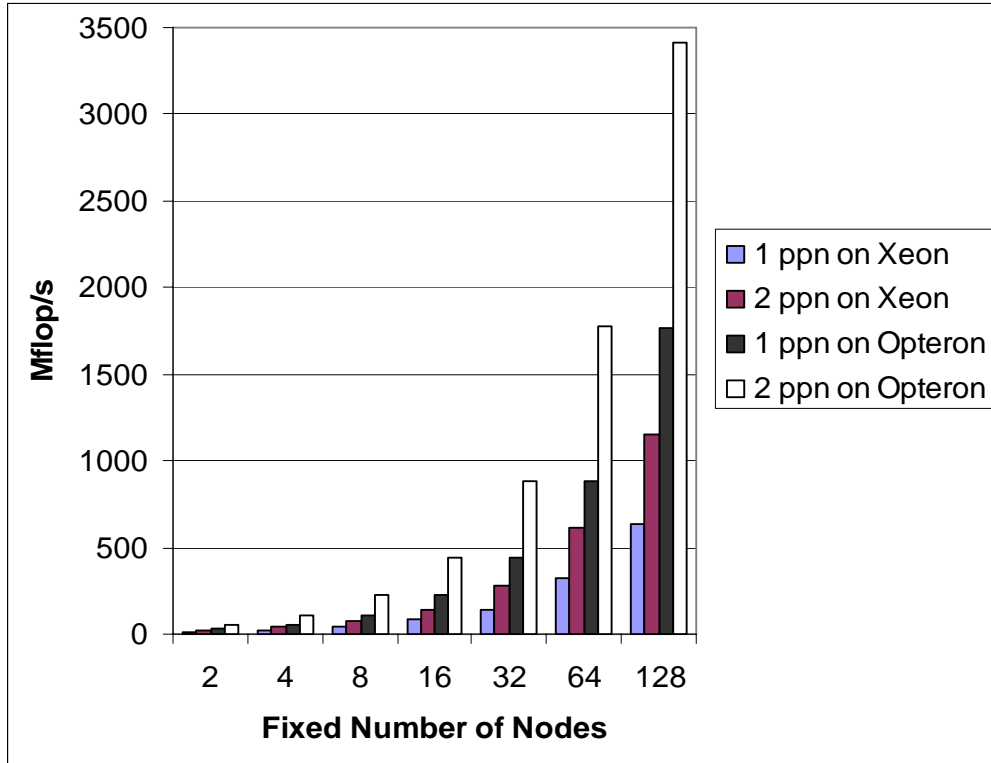


Figure 2.B.a. Class B EP benchmark for a fixed number of nodes.

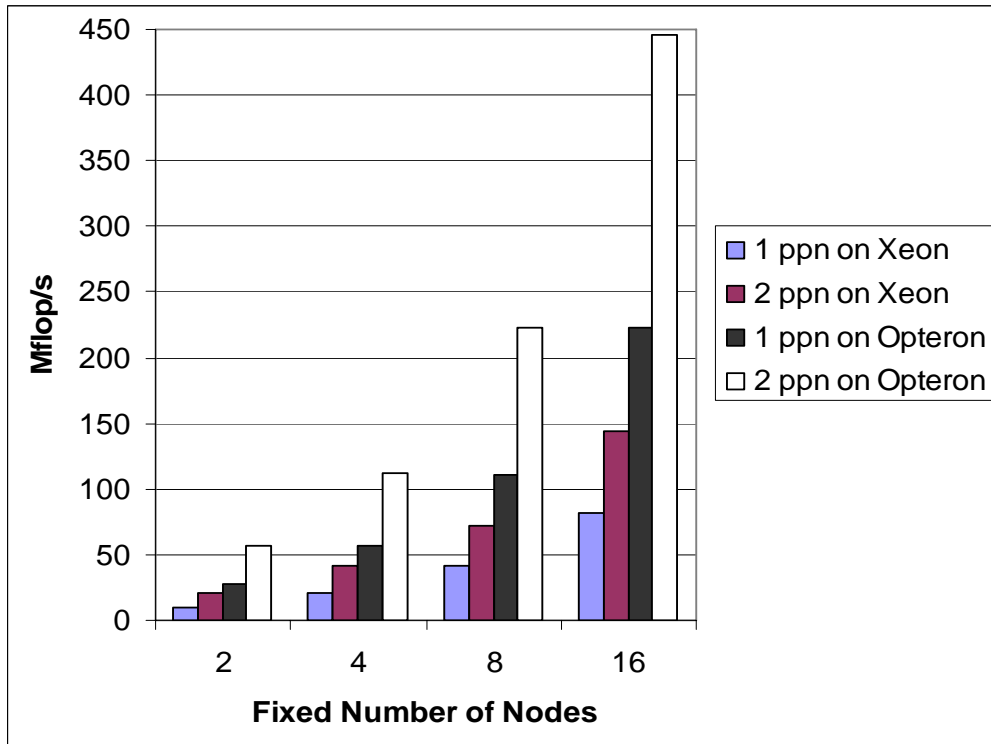


Figure 2.B.aa. Class B EP benchmark for a fixed number of nodes.

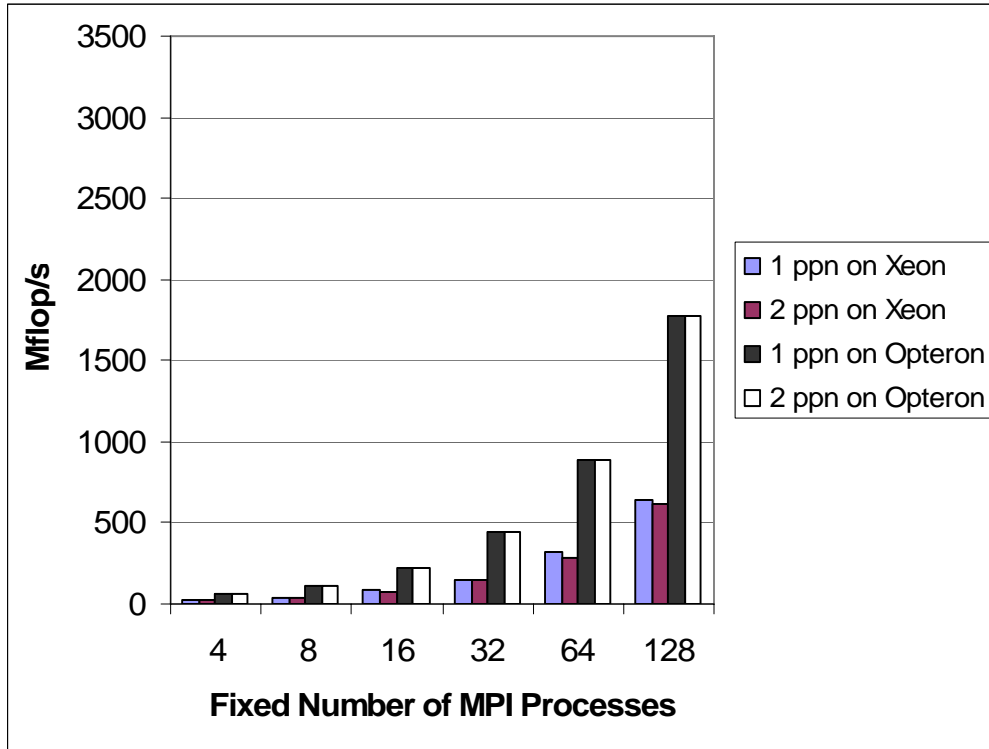


Figure 2.B.b. Class B EP benchmark for a fixed number of MPI processes.

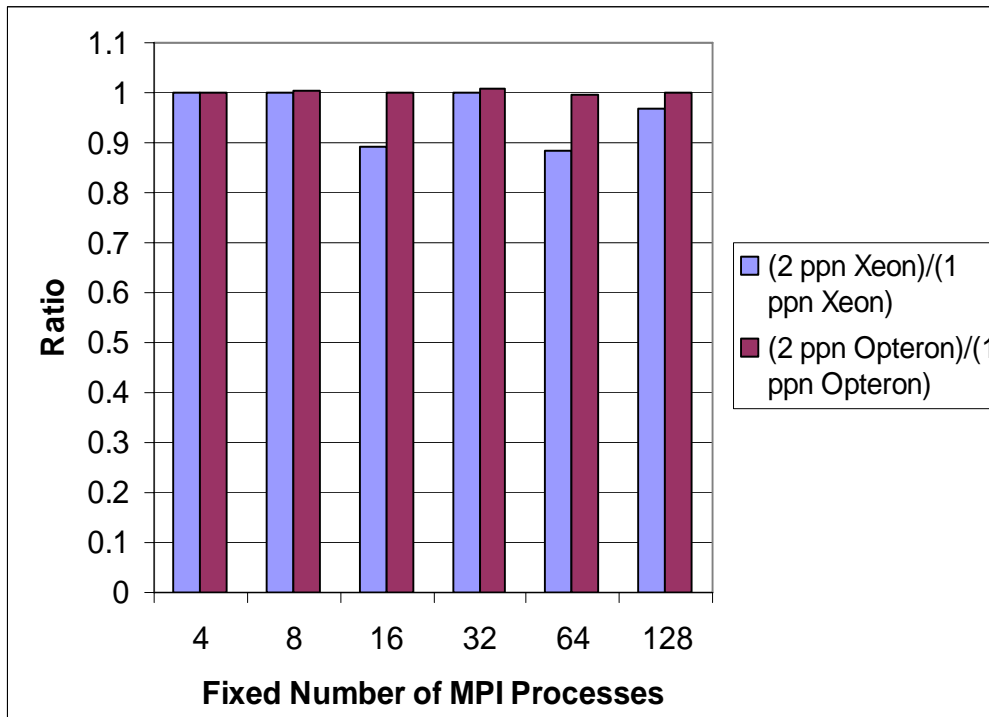


Figure 2.B.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

For the EP benchmark, the 2 ppn performance is nearly the same as the 1 ppn performance for both clusters. This is not surprising since this benchmark does not require much memory bandwidth within nodes and has little MPI communication. Notice that the 1.4 GHz Opteron cluster always more than doubly outperformed the 3.2 GHz Xeon cluster.

### Test 3: The MG Benchmark

MG (MultiGrid) is a simple 3D multigrid benchmark. It demonstrates the capabilities of a very simple multigrid solver in computing a three dimensional potential field. Results from [10, p. 29] indicate that the communication network performance is the primary key to good performance for this benchmark. The problem size is  $256^3$  with 4 iterations,  $256^3$  with 20 iterations and  $512^3$  with 20 iterations for class A, B and C respectively. Figures 3.B.a – 3.B.c show the performance results of MG with class B problem, similar results for class A are in Appendix 1. (Note: class C problem exceeds the user memory limit on the Xeon and Opteron clusters used for this study.)

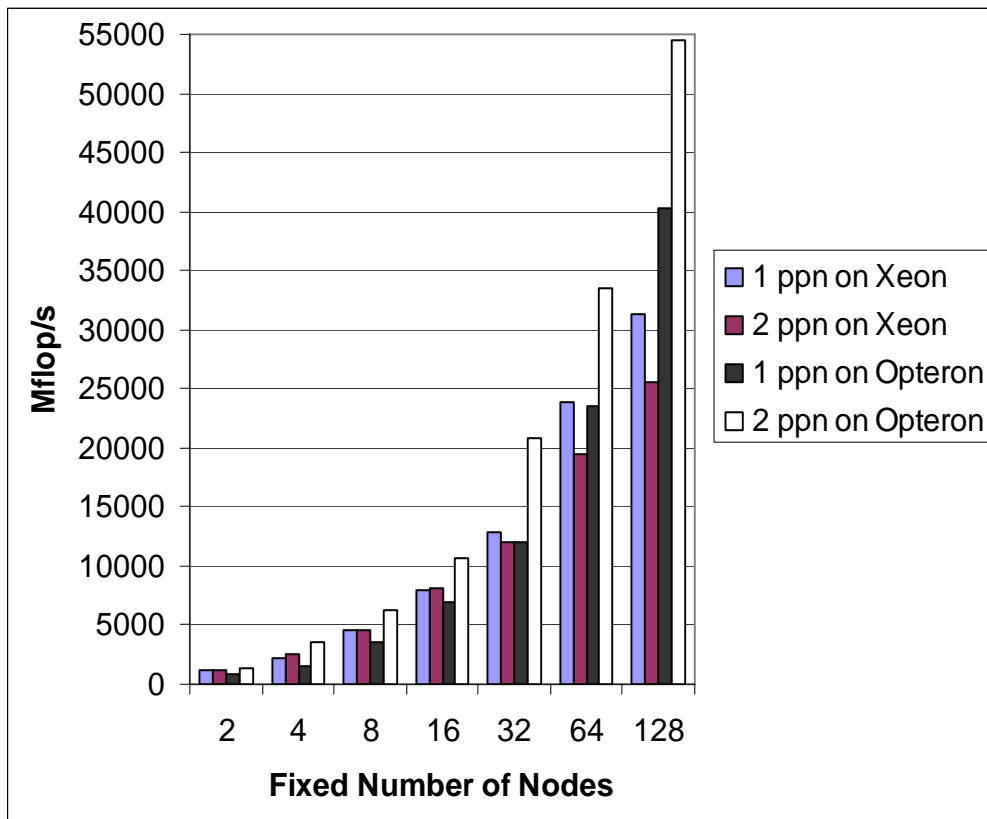


Figure 3.B.a. Class B MG benchmark for a fixed number of nodes.

Since it is difficult to see performance differences for 2, 4, 8, and 16 nodes, Figure 3.B.aa has been added that shows the results for only these node counts.

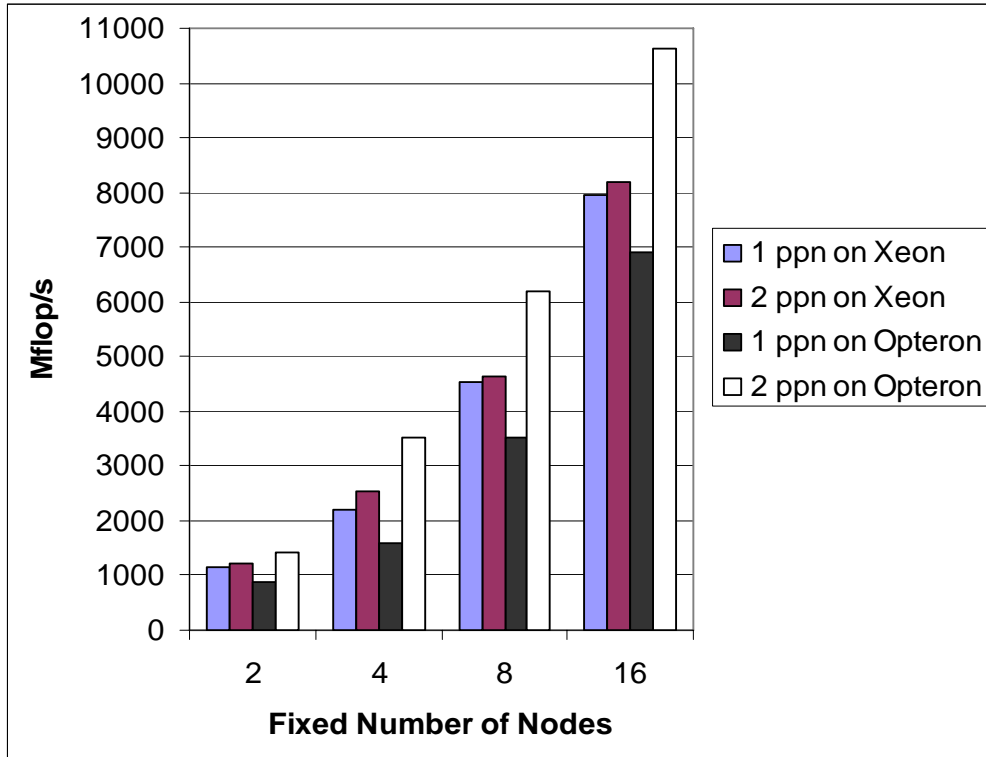


Figure 3.B.aa. Class B MG benchmark for a fixed number of nodes.

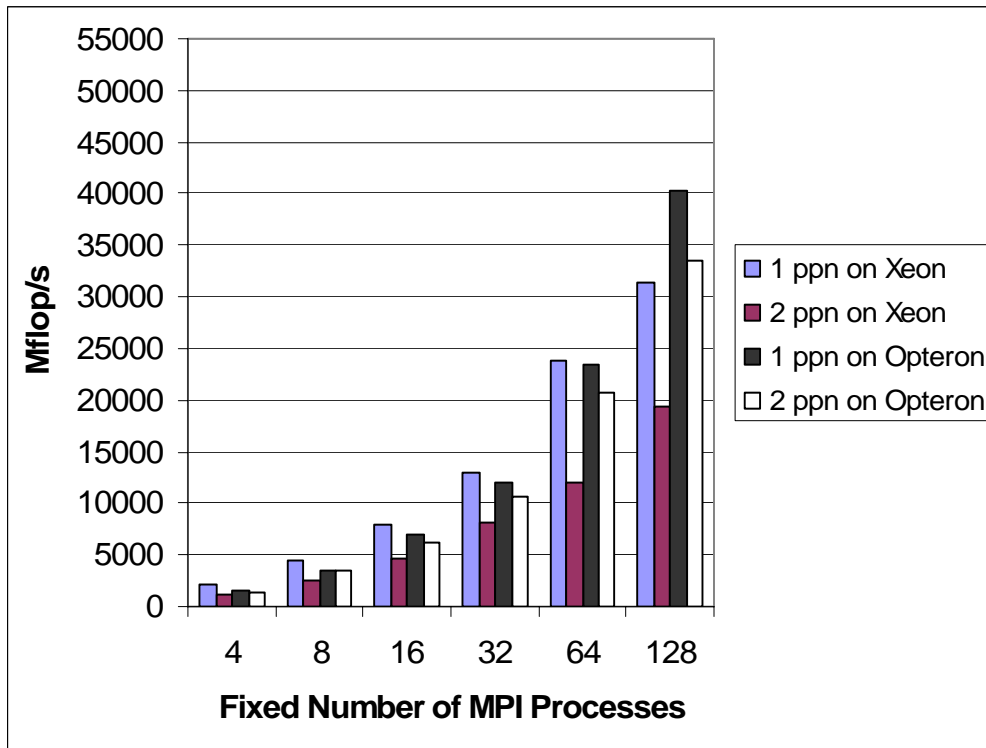


Figure 3.B.b. Class B MG benchmark for a fixed number of MPI processes.

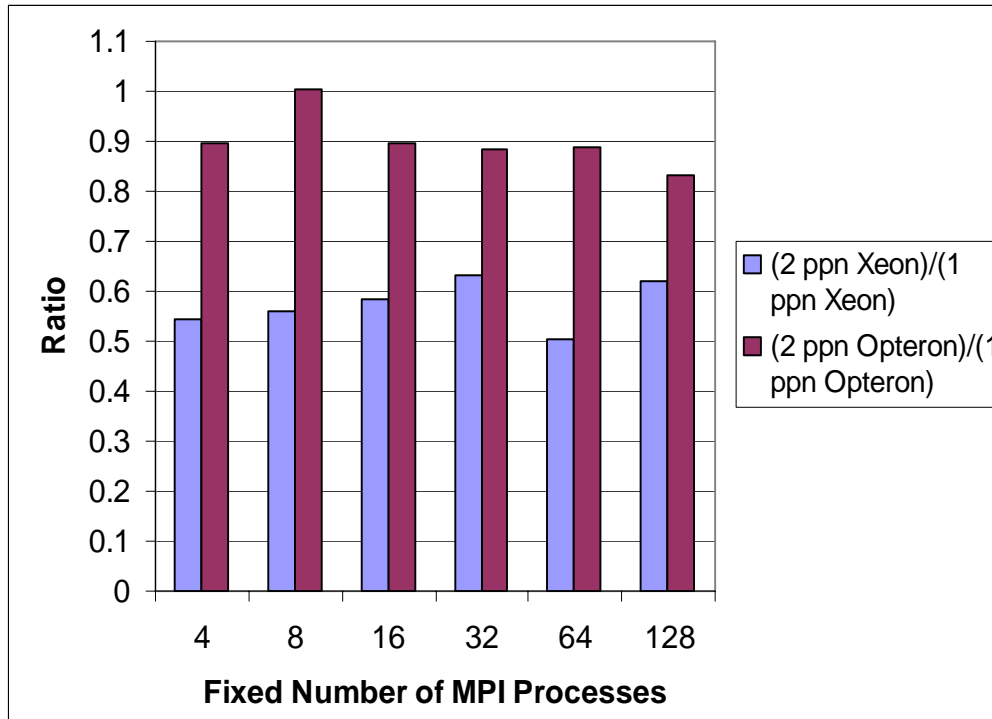


Figure 3.B.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

As seen from figure 3.B.a., for the Xeon cluster, 2 ppn slightly outperformed 1 ppn for 2, 4, 8 and 16 nodes for a fixed number of nodes. For the Opteron cluster, 2 ppn significantly outperformed 1 ppn in all cases for a fixed number of nodes. Figure 3.B.c. shows a 37% to 50% decrease in performance when using 2 ppn instead of 1 ppn for the Xeon cluster! Figure 3.B.c. also shows a 0% to 17% decrease in performance when using 2 ppn instead of 1 ppn for the Opteron cluster.

#### Test 4: The FT Benchmark

FT (Fourier Transform) is a 3-D fast-Fourier transform partial differential equation benchmark. It implements the time integration of a three-dimensional partial differential equation using the Fast Fourier Transform (FFT). The performance of the FT benchmark is affected by the memory bandwidth, cache size, interconnect performance and processor performance, in roughly that order [10, p.31]. The problem size is  $256^2 \times 128$ ,  $256^2 \times 512$ , and  $512^3$  for class A, B and C respectively. Figures 4.B.a. – 4.B.c. show the performance results of FT with class B problem, similar results for class A are in Appendix 1. (The class C problem exceeds the user memory limit on the Xeon and Opteron clusters used in this study.).

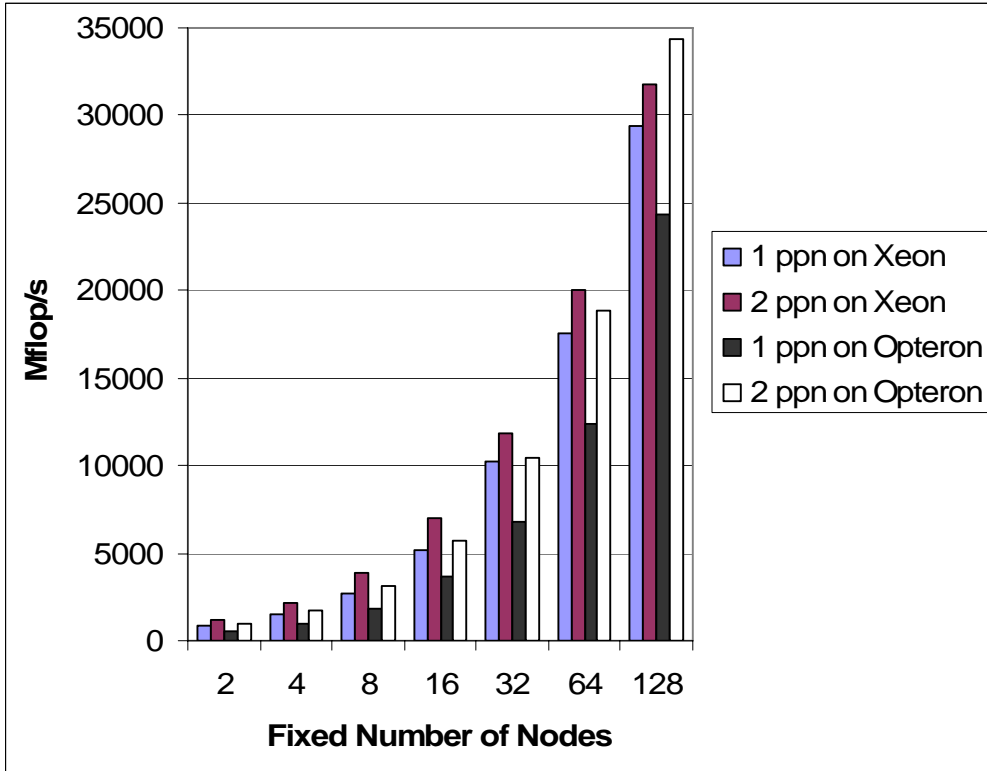


Figure 4.B.a. Class B FT benchmark for a fixed number of nodes.

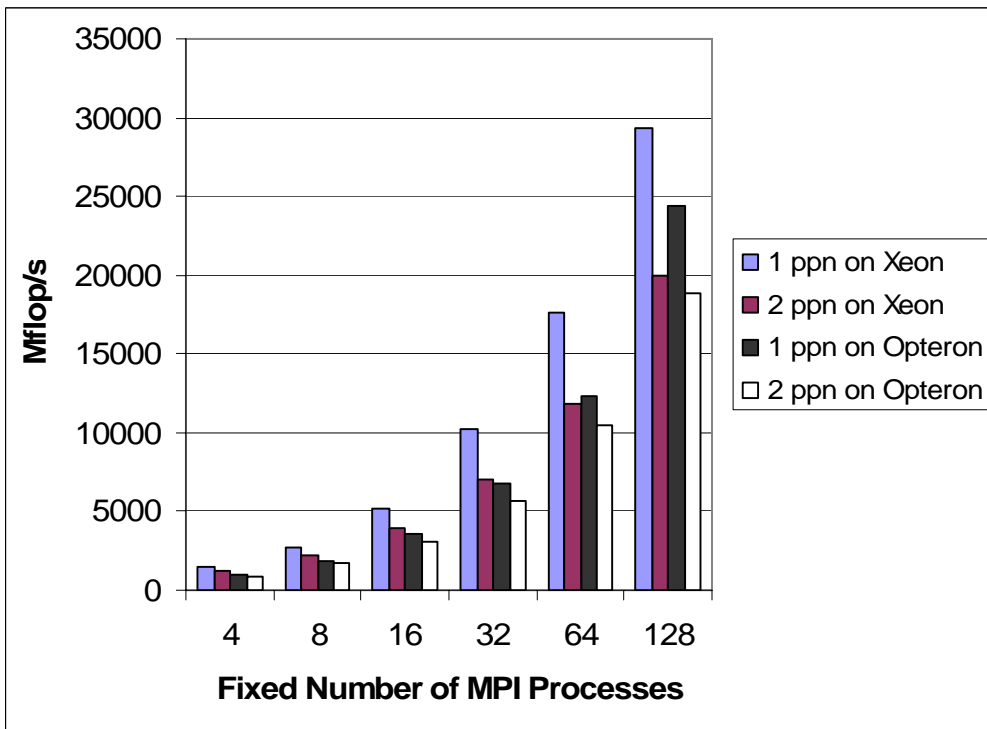


Figure 4.B.b. Class B FT benchmark for a fixed number of MPI processes.

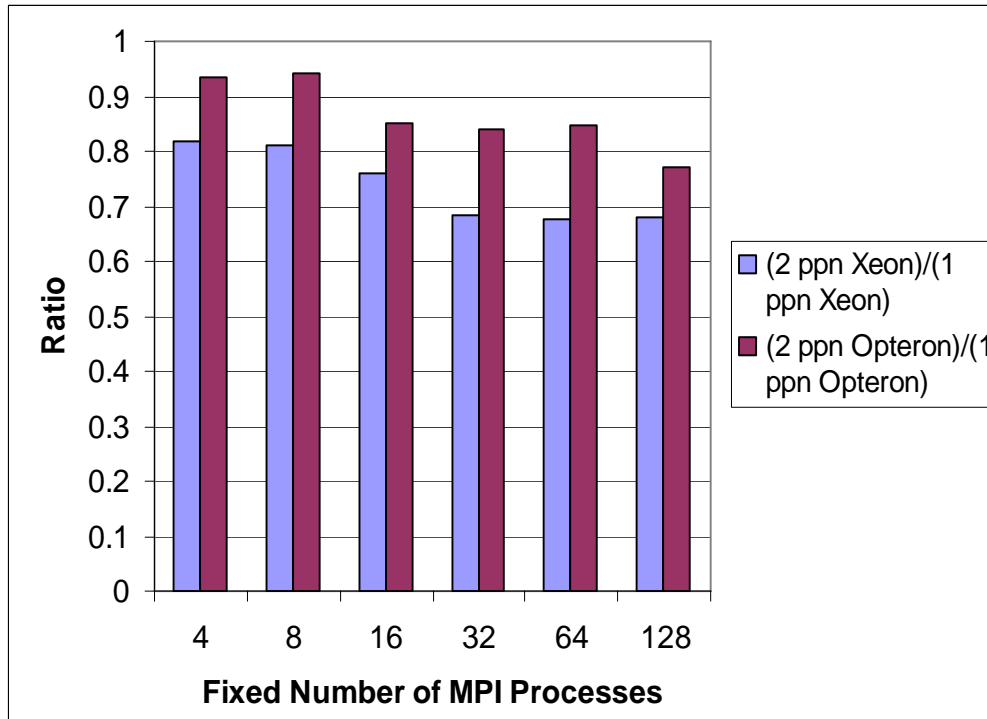


Figure 4.B.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

For the FT benchmark with a fixed number of nodes, 2 ppn always performs better than 1 ppn for both clusters. Notice that performance increases for 2 ppn over 1 ppn are greater for the Opteron cluster than for the Xeon cluster. Figure 4.B.c. shows a 20% to 33% decrease in performance when using 2 ppn instead of 1 ppn for the Xeon cluster. Figure 4.B.c. also shows a 5% to 23% decrease in performance when using 2 ppn instead of 1 ppn for the Opteron cluster.

### Test 5: The LU Benchmark

Benchmark LU (LU factorization) makes a regular-sparse, block (5 x 5) lower and upper triangular factorization of a matrix. The problem size is  $64^3$ ,  $102^3$ , and  $162^3$  for class A, B and C respectively. Figures 5.B.a. – 5.B.c. show the performance results of LU with class B problem, similar results for classes A and C are in Appendix 1. Note that since it is difficult to see performance differences for 2, 4, and 8 nodes in Figure 5.B.a, Figure 5.B.aa has been added that shows the results for only these node counts.

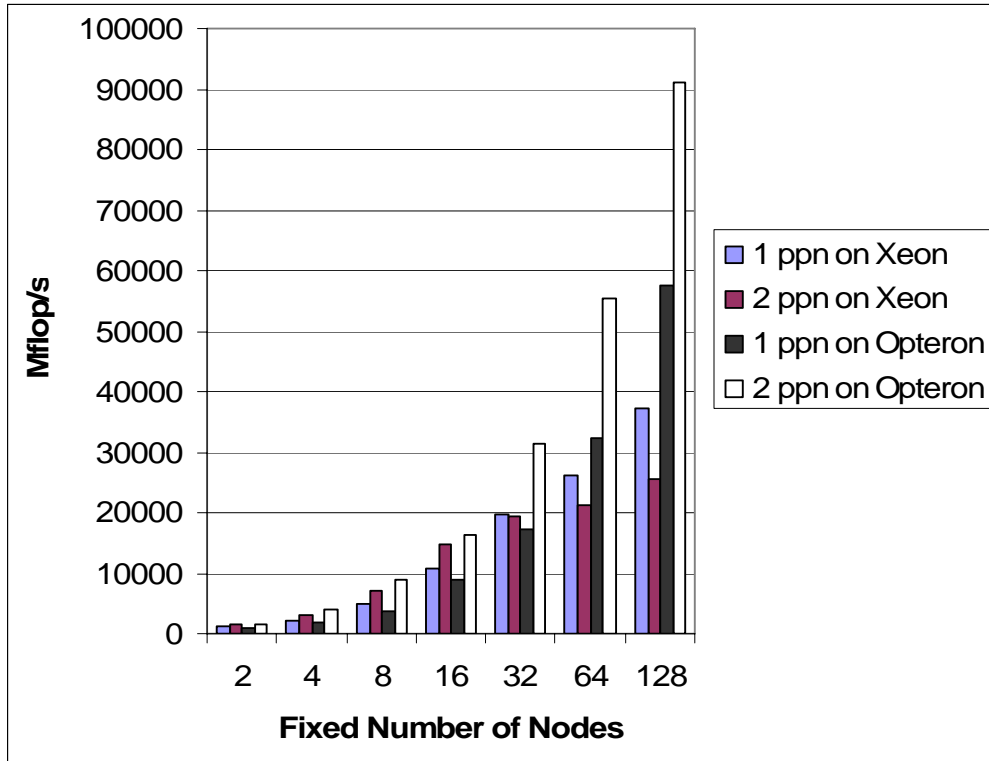


Figure 5.B.a. Class B LU benchmark for a fixed number of nodes.

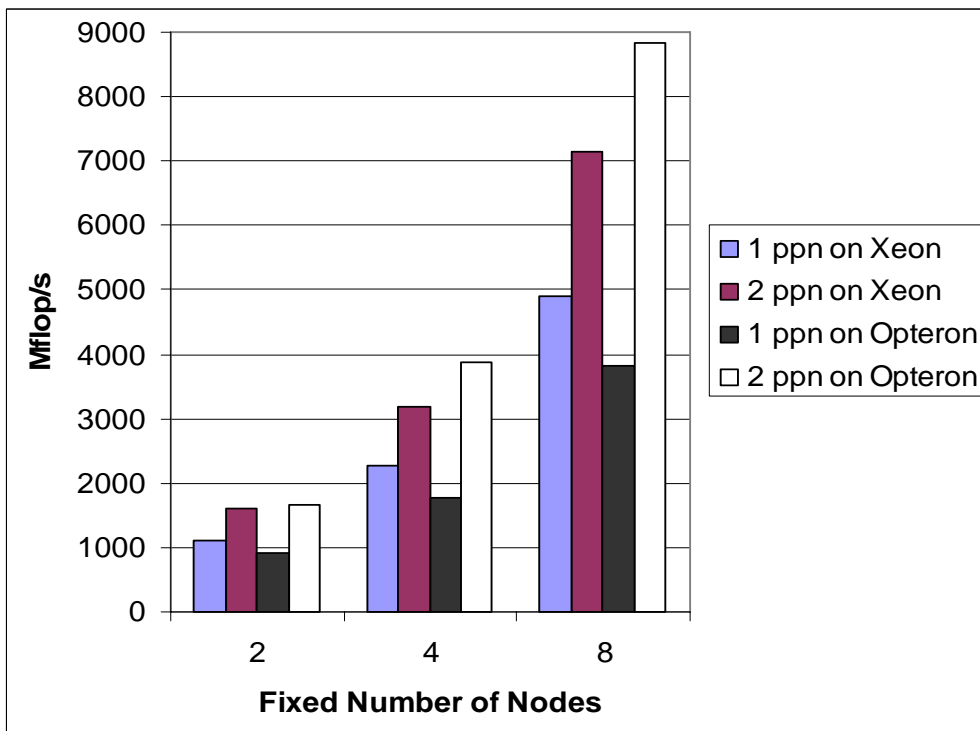


Figure 5.B.aa. Class B LU benchmark for a fixed number of nodes.

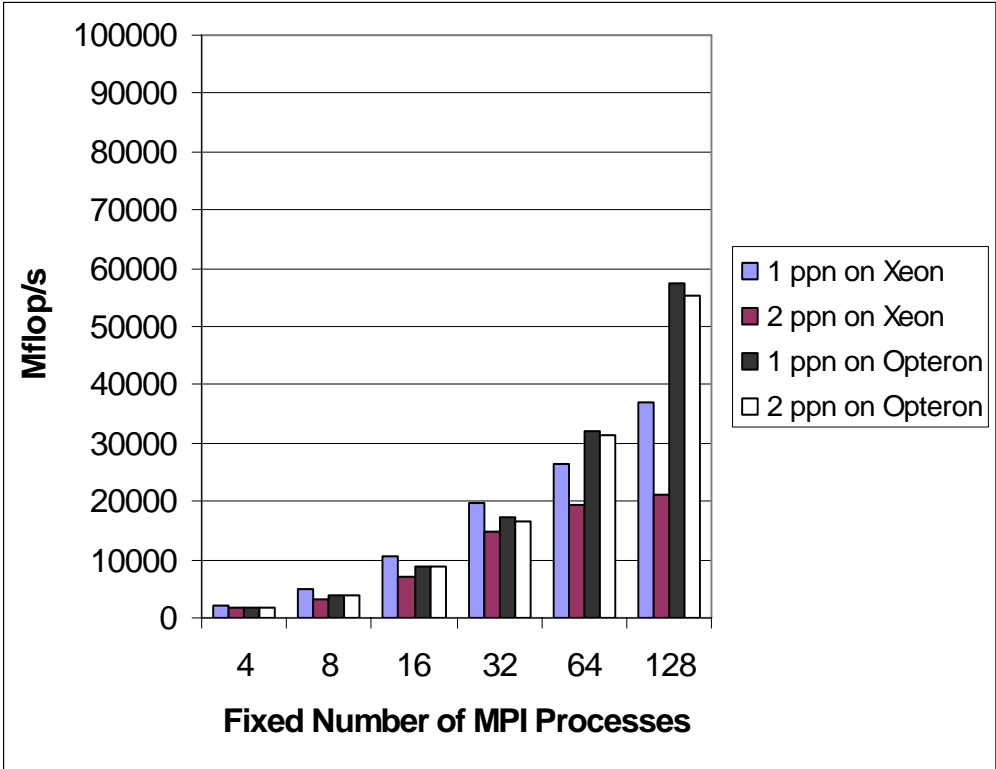


Figure 5.B.b. Class B LU benchmark for a fixed number of MPI processes.

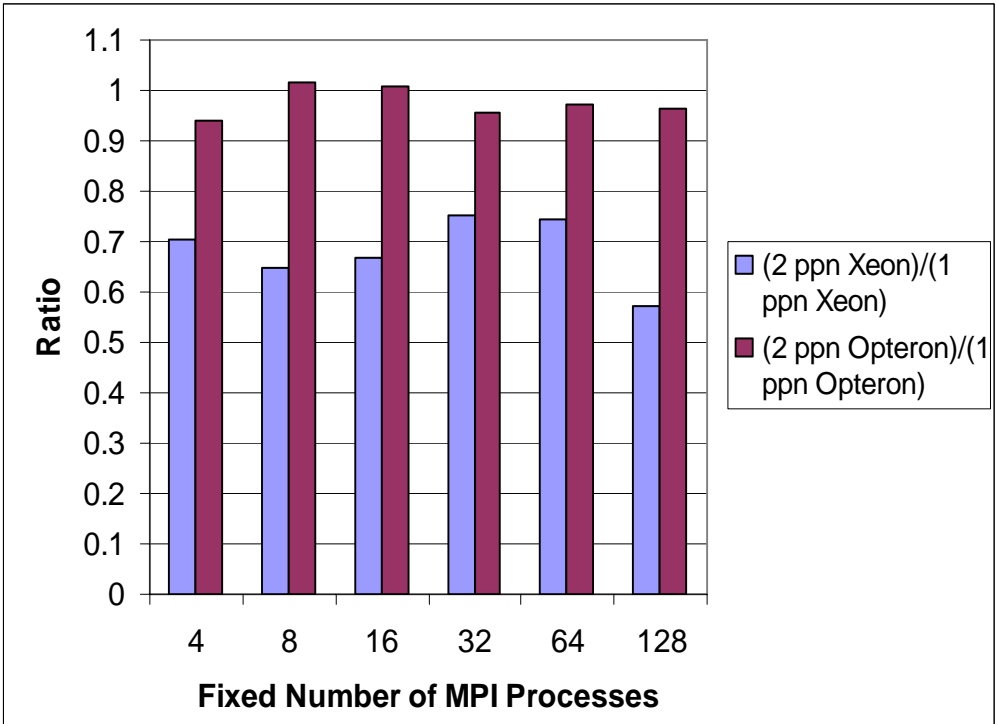


Figure 5.B.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

Figure 5.B.a. again shows that the Opteron cluster is able to efficiently utilize the second processor significantly better than the Xeon cluster. Figure 5.B.c. shows a 25% to 43% decrease in performance when using 2 ppn instead of 1 ppn for the Xeon cluster while there is only a 0% to 6% decrease in performance when using 2 ppn instead of 1 ppn for the Opteron cluster.

### Test 6: The IS Benchmark

Benchmark IS (Integer Sort) is a large integer sort. It performs a sorting operation that is important in "particle method" codes. It tests both integer computation speed and communication performance. This benchmark exercises a combination of interconnect, processor, and memory performance [10. p. 32]. The problem size is  $2^{23}$ ,  $2^{25}$ , and  $2^{27}$  for class A, B and C respectively. Figure 6.B.a. – 6.B.c. show the performance results of IS with class B problem, similar results for class A are in Appendix 1. (Note: class C problem exceeds the user memory limit on the Xeon cluster.)

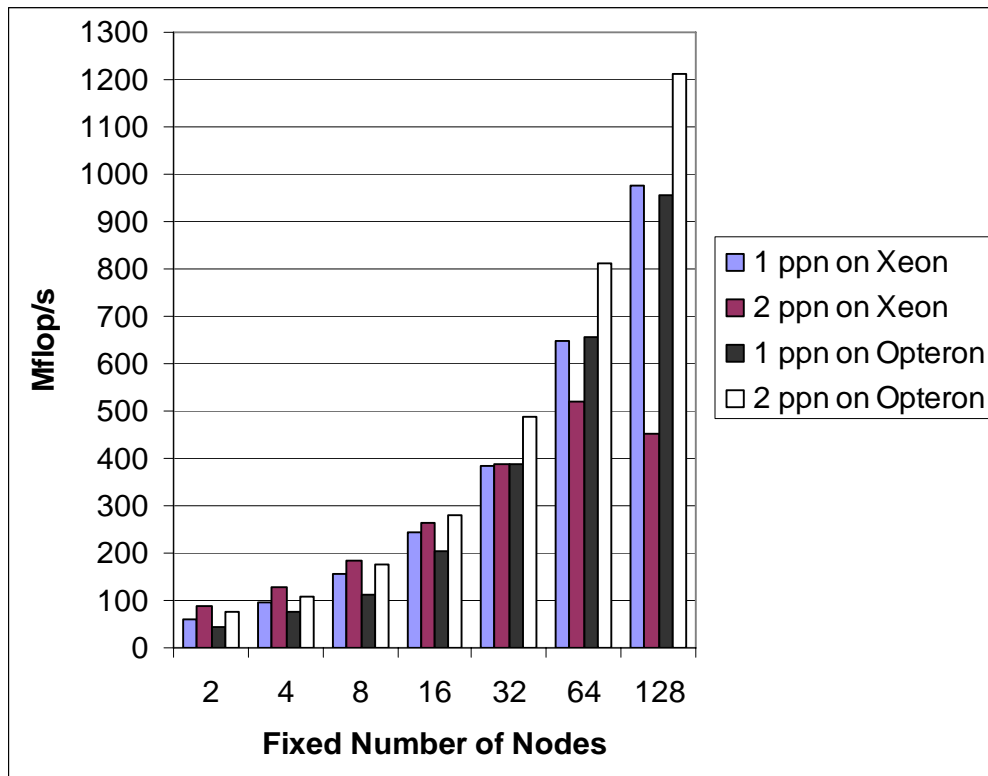


Figure 6.B.a. Class B IS benchmark for a fixed number of nodes

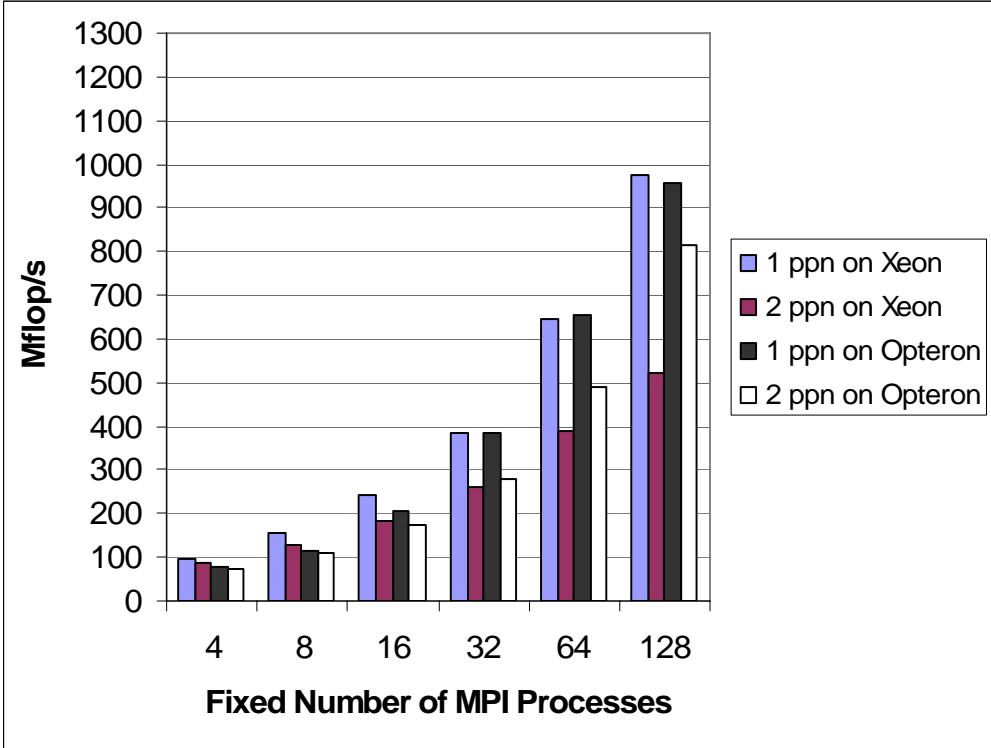


Figure 6.B.b. Class B IS benchmark for a fixed number of MPI processes.

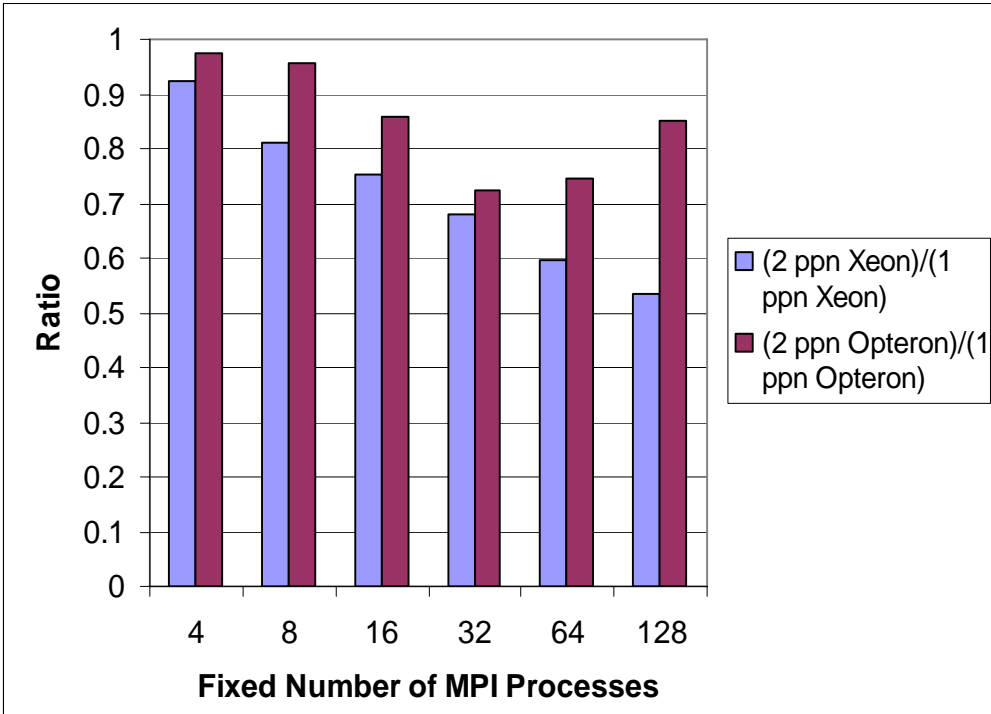


Figure 6.B.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

Figure 6.B.a shows that the Opteron cluster is able to efficiently utilize 2 ppn significantly better than the Xeon cluster. Figure 6.B.c. shows an 8% to 47% decrease in performance when using 2 ppn instead of 1 ppn for the Xeon cluster. Figure 6.B.c. also shows a 2% to 28% decrease in performance when using 2 ppn instead of 1 ppn for the Opteron cluster.

## **4. Performance Analysis**

The purpose of this section is to investigate the various performance factors that determine whether 1 ppn or 2 ppn will yield the best performance. Before beginning this study, we thought that the memory bandwidth of each node was the sole factor in determining whether 1 ppn or 2 ppn would give the best performance. However, there are other factors! In Section 4.1 we discuss the impact of cache effects and scalability on 1 ppn and 2 ppn performance. In section 4.2 we investigate the impact of the memory bandwidth of the Xeon and Opteron nodes on 1 ppn and 2 ppn performance. In section 4.3, we investigate the impact of the MPI implementation and the performance of the communication network on 1 ppn and 2 ppn performance.

### **4.1 Impact of Cache Effects and Scalability**

In Section 3, one of the ways the NPB were compared was using a fixed number of nodes and measuring the performance when using 1 ppn and 2 ppn. For example, when 128 nodes are used, the performance when using 128 nodes with 128 MPI processes was compared with using 128 nodes and 256 MPI processes. When comparing these performance numbers for a fixed number of nodes on either the Xeon or Opteron clusters, one must be aware of a possible cache effects. When increasing the number of MPI processes often the problem being computed for each MPI process will become small enough to fit into the processor's cache. When this happens, performance usually increases. However, notice that cache effects will not affect the performance comparisons of 1 and 2 ppn when using a fixed number of MPI processes since both the Xeon and Opteron have caches local to each processor and the processors do not share a cache. The largest cache on the 1.4 GHz Opteron and 3.2 Xeon is 1 MB.

The scalability of the MPI program should also be taken into consideration when comparing performance numbers for a fixed number of nodes. It is well known that for a fixed problem size the performance of an MPI program usually increases as the number of MPI processes increases but only up to some point. When this point is reached the performance will often decrease. Looking at the performance data for the NPB for a fixed number of nodes, it appears that scalability problems often limit performance when using 2 ppn with a large number of nodes. However, when comparing performances with a fixed

number of MPI processes, clearly scalability issues do not affect the performance comparisons.

## 4.2 Impact of Memory Bandwidth within Nodes

The memory system within nodes should be able to support the concurrent streaming of data between processors and the shared memory. The purpose of this section is to evaluate the Xeon and Opteron's ability to support concurrent streaming of data between processors and their shared memory. This is done by investigating a variety of patterns commonly found in scientific applications and includes investigating the performance of the STREAM benchmarks [7]. Arrays used for these tests were taken to be much larger than the 1 MB level 2 cache size to ensure accurate memory bandwidth measurements. Arrays were declared to be of type real\*8 with 300000\*17 elements. (17 was used because strides range from 1 to 17.) The timing methodology is explained in Appendix 2. Notice that caches are flushed prior to measuring times for each trial. Tests were performed using different strides for arrays to measure memory bandwidth under varying degrees of memory traffic. Stride 1 array accesses provided the least memory traffic and stride 17 the heaviest. When running timing programs several times, average timings varied less than 0.05 milliseconds. Because of lack of allocation on the NCSA 3.2 GHz Xeon cluster, all Xeon tests in this section were run using the 2.8 GHz Xeon cluster at ISU.

### Test 1: Uniform Memory Reads and Writes

The following is the MPI code executed by each MPI process that was used to evaluate uniform concurrent memory reads. When 1 ppn is used, this means that only one processor executes this code. When 2 ppn is used, then both processors on the node execute this code. If the node has good memory bandwidth, then the 2 ppn time will be the same as the 1 ppn time.

```
do i = 1, n*stride, stride
  s = s + A(i)
enddo
```

The following is the MPI code executed by each MPI process that was used to evaluate uniform concurrent memory writes.

```
do i = 1, n*stride, stride
  A(i) = 1.d0
enddo
```

N was taken to be 300000 and stride was taken to be 1, 2, ..., 17. Notice that the same

number of elements is read/written for each stride value. The timing results for memory reads are shown in the Figure 7.a and timing results for memory writes are shown in Figure 7.b. For the Xeon cluster, the time of memory reads/writes ratio for (2 ppn)/(1 ppn) is between 1.8 and 2.0. This shows that the Xeon's ability to handle concurrent uniform reads and writes is poor even in the stride 1 case where memory to processor traffic is light. This ratio for the Opteron processor varies between 1.0 and 1.01, nearly perfect!

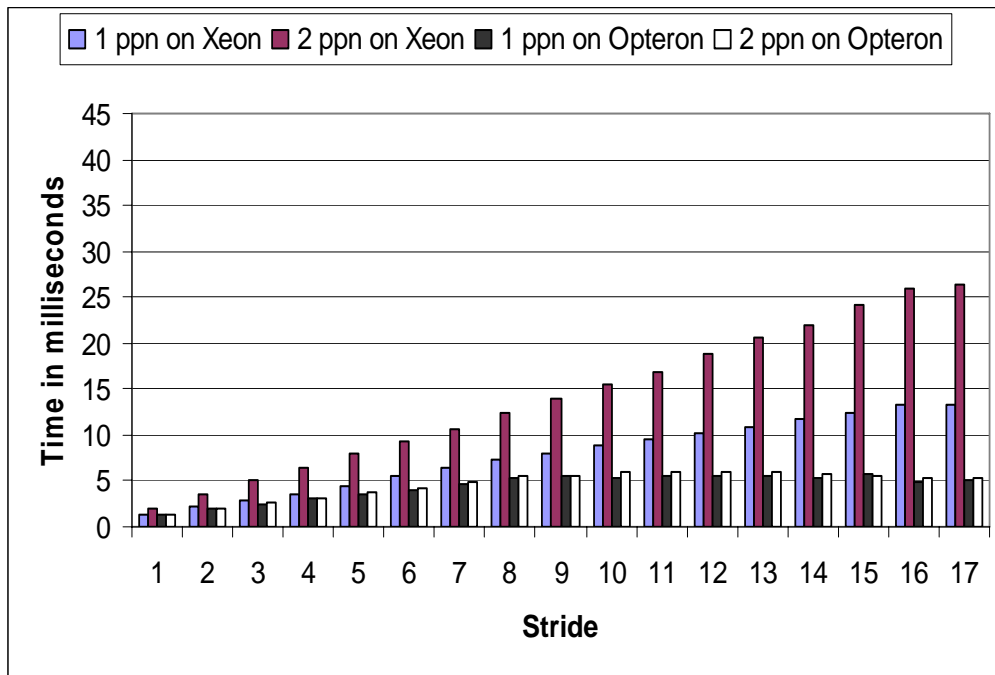


Figure 7.a. Read performance comparisons for different strides.

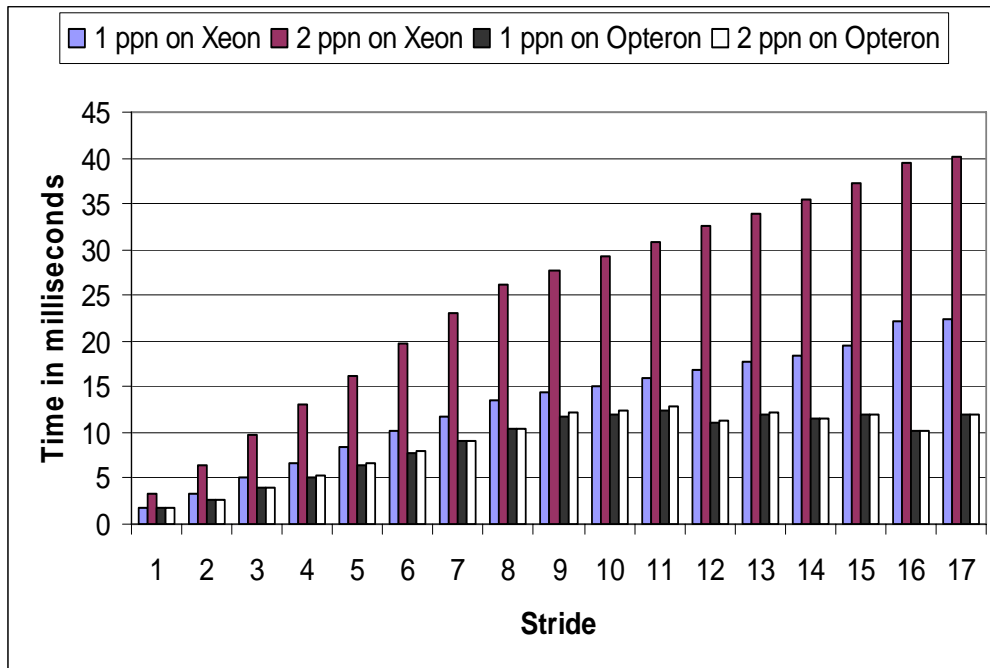


Figure 7.b. Write performance comparisons for different strides.

## Test 2: Random Memory Reads and Writes

We next measure the ability of the Xeon and Opteron processors to execute random memory reads and writes. This is done by first defining an index array using the Fortran random number generator, `random_number`, that generates `real*8`, uniform random numbers between 0 and 1.

```

integer,parameter :: n = 300000
real*8 :: A(n), B(n)
integer :: index(n)
...
call random_number(B)
B = float(n)*B ! 0 < B(i) < n
index(1:n) = B(1:n) ! truncation to integer but some values may be zero
do i = 1, n
  if ( index(i) == 0 ) index(i) = i
enddo

```

The following is the MPI code executed by each MPI process that was used to evaluate random concurrent memory reads.

```

do i = 1, n

```

```

s = s + A(index(i))
enddo

```

The following is the MPI code executed by each MPI process that was used to evaluate random concurrent memory writes.

```

do i = 1, n
  A(index(i)) = 1.d0
enddo

```

N was taken to be 300000, the same value used for uniform reads/writes. Since we are using random accesses, using different strides is not needed. For random reads, the average timings are 15.7 milliseconds and 26.8 milliseconds for 1 ppn and 2 ppn respectively on the Xeon cluster giving a 2 ppn to 1 ppn ratio of 1.7. For the Opteron, the times are 6.36 milliseconds and 6.02 milliseconds for 1 ppn and 2 ppn, respectively, giving a 2 ppn to 1 ppn ratio of 1.06!

For random writes, the average timings are 23.6 milliseconds and 41.2 milliseconds for 1 ppn and 2 ppn respectively on the Xeon cluster giving a 2 ppn to 1 ppn ratio of 1.74. For the Opteron, the times are 9.82 milliseconds and 10.1 milliseconds for 1 ppn and 2 ppn, respectively, giving a 2 ppn to 1 ppn ratio of 1.03!

### Test 3: Concurrent Memory Reads and Writes

To measure the effects of two-way traffic between memory and processors, we use the following operations from the STREAM benchmarks [7]. “FLOPS” means “floating point operations”.

Name	Operation	FLOPS per iteration:
COPY	$a(i) = b(i)$	0
SCALE	$a(i) = q*b(i)$	1
SUM	$a(i) = b(i) + c(i)$	1
TRIAD	$a(i) = b(i) + q*c(i)$	2

Table 2: STREAM Benchmarks

The following is the MPI code executed by each MPI process that was used to measure the performance of these operations using the same value for n, 300000, as above.

```

do i = 1, n*stride, stride
  operation
enddo

```

where “operation” is the copy, scale, sum, or triad operation listed in Table 2. The performance results of these four operations are shown in Figures 8.a – 8.d.

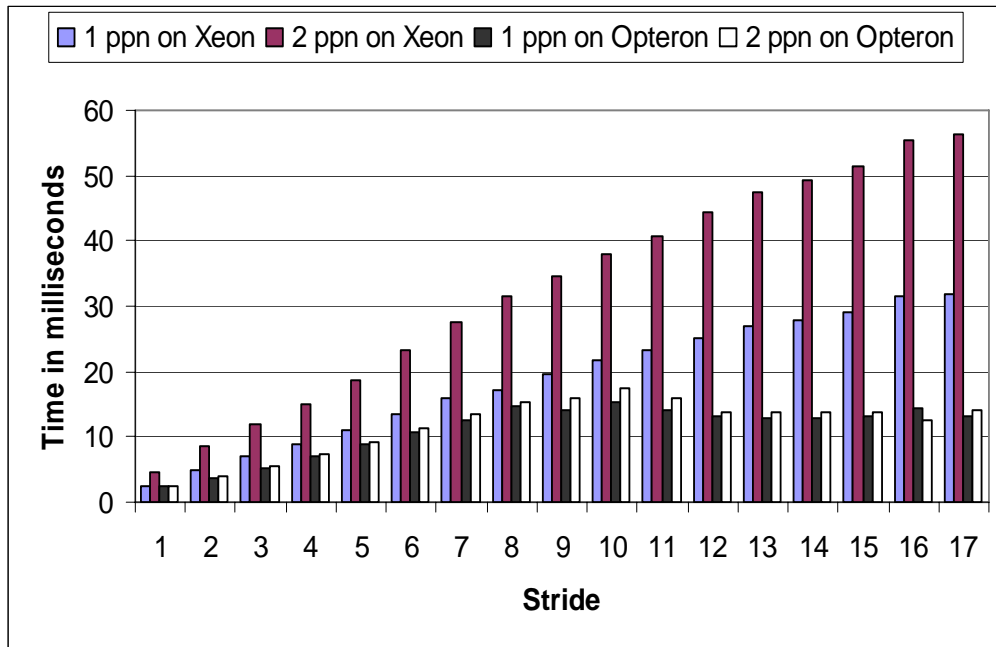


Figure 8.a. Performance of the COPY operation for different strides.

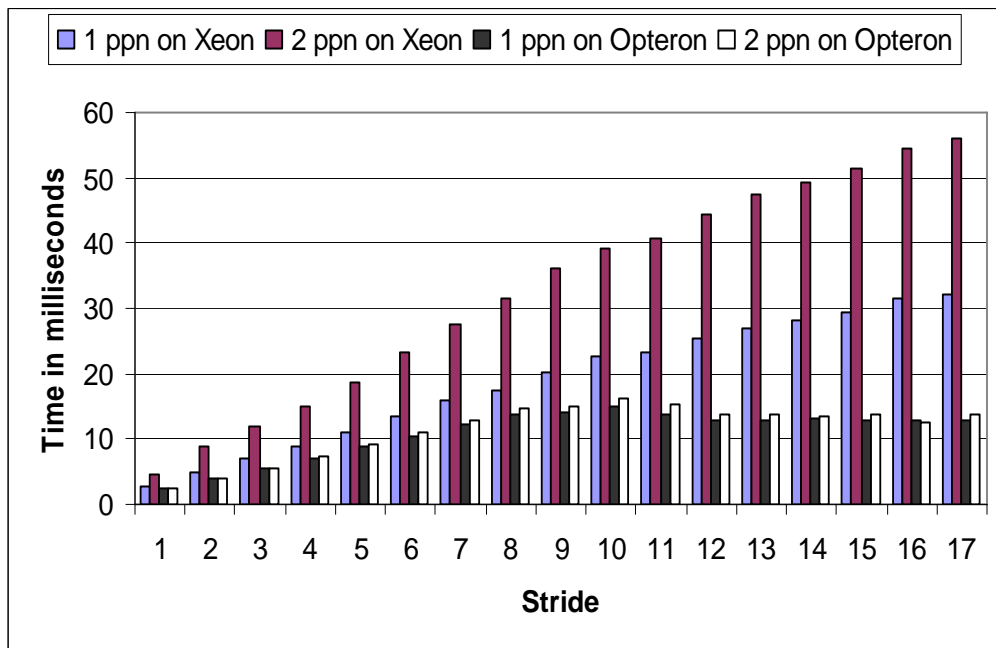


Figure 8.b. Performance of the SCALE operation for different strides.

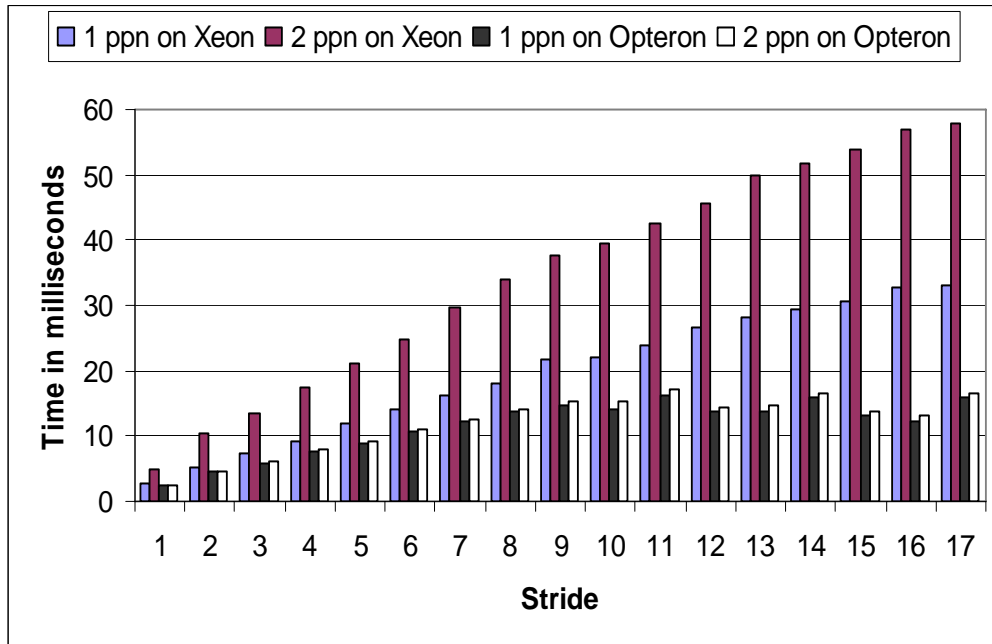


Figure 8.c. Performance of the SUM operation for different strides.

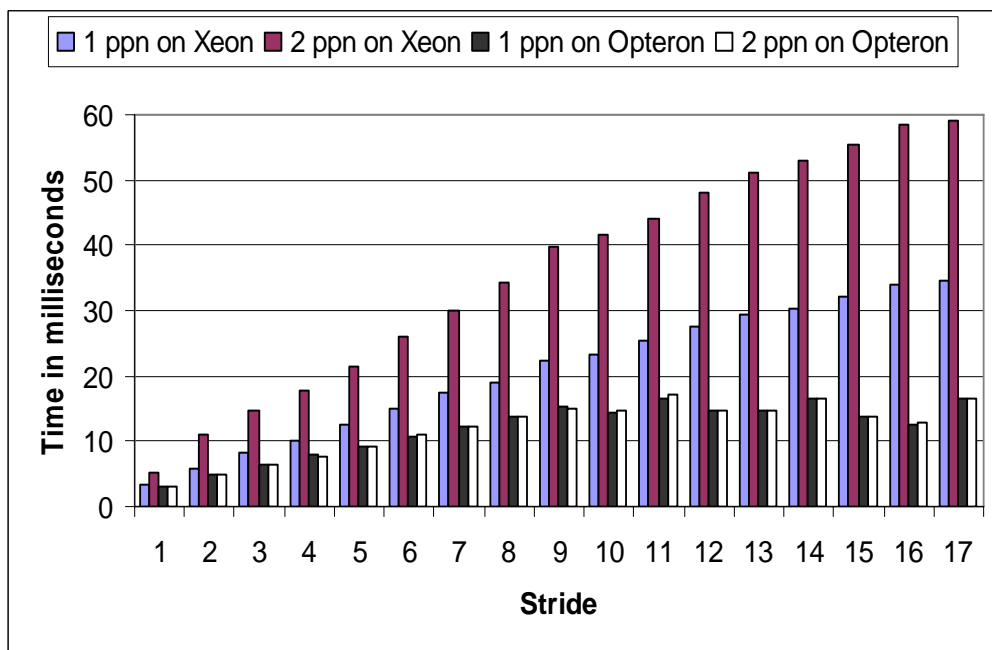


Figure 8.d. Performance of the TRIAD operation for different strides.

For these tests the Xeon time ratios of 2 ppn to 1 ppn again range from 1.7 to 2.0. The Opteron time ratios of 2 ppn to 1 ppn range from 1.00 to 1.04, again nearly perfect!

The purpose of this section was to evaluate the Xeon and Opteron's ability to support concurrent streaming of data between processors and their shared memory. The above tests show that bandwidth of the memory system of the Xeon processor is not able to support concurrent streaming of data between processors and their shared memory. However, the bandwidth of the memory system of the Opteron processor fully supports the concurrent streaming of data between processors and their shared memory.

### 4.3 Impact of MPI and the MPI Communication Network

The performance of an MPI application depends on the performance and architecture of the MPI communication network being used. The Myrinet communication network used in both clusters supports high bandwidth, low latency, and concurrent message passing between distinct nodes. Both clusters have only one Myrinet connection per node, so we contacted Myricom to find out if concurrent messages could be sent over this single Myrinet connection. A developer from Myricom replied:

"Concurrent sends are progressing in parallel, but they have to share the link bandwidth (more precisely, messages are split into 1k to 4k fragments, and fragments from concurrent messages are sent over a Myrinet-NIC link in a round-robin fashion)."

Myrinet cards and switches support the concurrent sending and receiving of messages up to the rated bi-directional bandwidth, but there is some overhead for concurrently processing messages. When sending small messages in an application code, it is unlikely that concurrent sends (or receives) will degrade performance since messages will be processed and sent quickly; however, when large messages are sent (or received), the single Myrinet data path will degrade performance.

The impact of the performance of the communication network was measured by measuring the MPI performance. However, the MPI performance depends on the efficiency of the implementation of the MPI routine and on the performance of the communication network. Let's examine the impact of 1 ppn versus 2 ppn on point-to-point and collective routines.

**Point-to-point routines.** The MPI on both clusters uses the most efficient point-to-point communication depending on whether the communication is within a node or between nodes. Typically, MPI sends and receives are faster within a node than between nodes for Myrinet 2000 clusters. When using 1 ppn, all point-to-point communication will occur between nodes. When using 2 ppn, some communication may be within nodes and some may be between nodes. The following test was run to measure the degree of concurrency

achieved when simultaneous sends are issued from node 0 to nodes 1 and 2. We first send an array A from rank 0 to rank 2 and measure the time.

```
call mpi_barrier(comm, ierror)
t = mpi_wtime()
if (rank == 0) then
  call mpi_send(A, n, mpi_double_precision, .... with destination = 2)
endif
if (rank == 2) then
  call mpi_recv(A, n, ..... with source=0)
endif
time = mpi_wtime() - t
```

We then compare these times with the time to execute two concurrent sends from node 0 to nodes 1 and 2.

```
call mpi_barrier(comm, ierror)
t = mpi_wtime()
if (rank == 0) then
  call mpi_send(A, n, mpi_double_precision, .... with destination = 2 ....)
endif
if (rank == 1) then
  call mpi_send(A, n, mpi_double_precision, .... with destination = 4)
endif
if (rank == 2) then
  call mpi_recv(A, n, ..... with source=0 ....)
endif
if (rank == 4) then
  call mpi_recv(A, n, ..... with source=1 ....)
endif
time = mpi_wtime() - t
```

Figure 9 shows the ratios of the time for concurrently sending two messages of the same size from one node to two other nodes divided by the time to send a message of the same size from one node to another node. Notice that concurrency is achieved up to the bandwidth limit of the Myrinet connection when using a single network card.

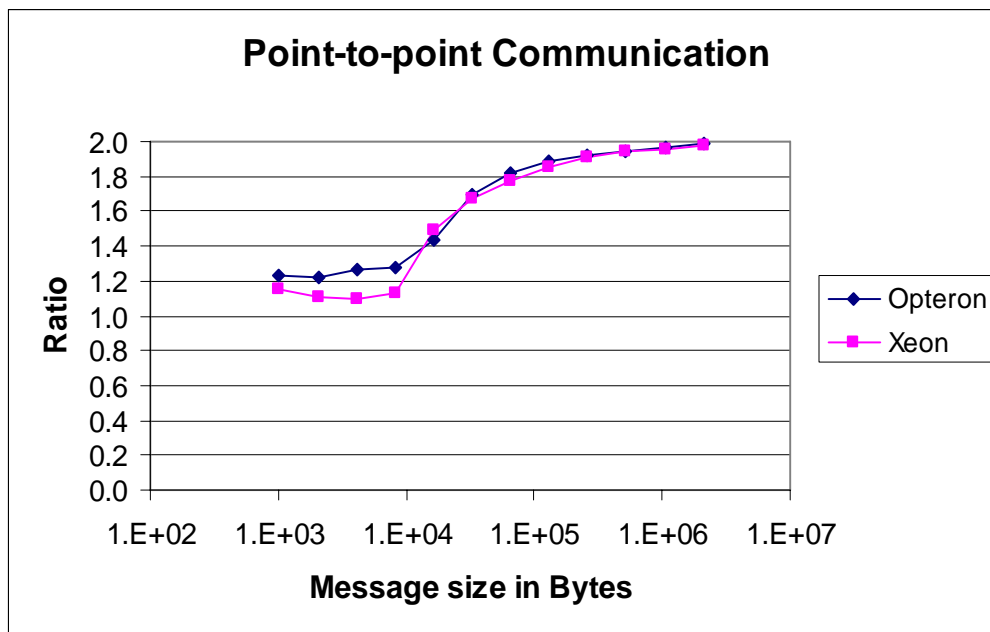


Figure 9. (2 ppn time)/(1 ppn time) ratios for point-to-point communication.

**Collective Routines.** The collective routines used for both clusters do not take advantage of the fact that communication within nodes is much faster than between nodes when using 2 ppn. For example, when running 2 ppn and performing a broadcast from node 0, a good implementation for the Myrinet communication network would be to first broadcast the message to all other nodes and then complete the broadcast within each node.

Figures 10 and 11 show the ratios of 2 ppn to 1 ppn times for `mpi_bcast` and `mpi_alltoall` using the Opteron cluster with 64 nodes and 64 MPI processes (1 ppn) and 32 nodes using 64 MPI processes (2 ppn) for a variety of message sizes. The purpose of presenting these results is to not extensively investigate the performance of collective routines using 1 ppn and 2 ppn for different messages sizes and for different numbers of MPI processes, but to illustrate that sometimes 1 ppn performance is better than 2 ppn and sometimes it is the other way around. We ran these two tests with 64 MPI processes. (We also ran these tests using 16 and 32 MPI processes and the behavior was similar.) We chose `mpi_bcast` since it does not require much communication and we chose `mpi_alltoall` since it requires extensive communication.

Figures 10 and 11 show the variability of the relative performance of 1 and 2 ppn for these two collective routines. Notice that the 2 ppn to 1 ppn ratio for `mpi_alltoall` increases with message size. This is likely due to the fact that each node has a single Myrinet port and that, when running 2 ppn, each node is sending and receiving twice the number of large messages to and from all other nodes. (Those collective routines implemented using point-to-point routines would be node-aware in the sense that the point-to-point routines were

node aware.)

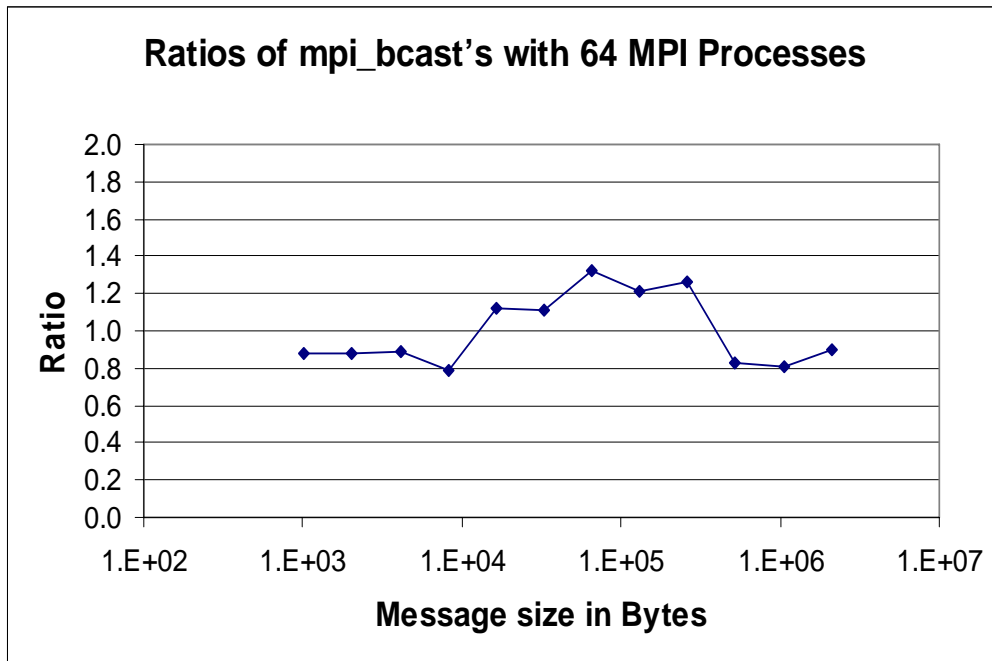


Figure 10. (2 ppn)/(1 ppn) ratios for mpi\_bcast on the Opteron cluster using 64 MPI processes.

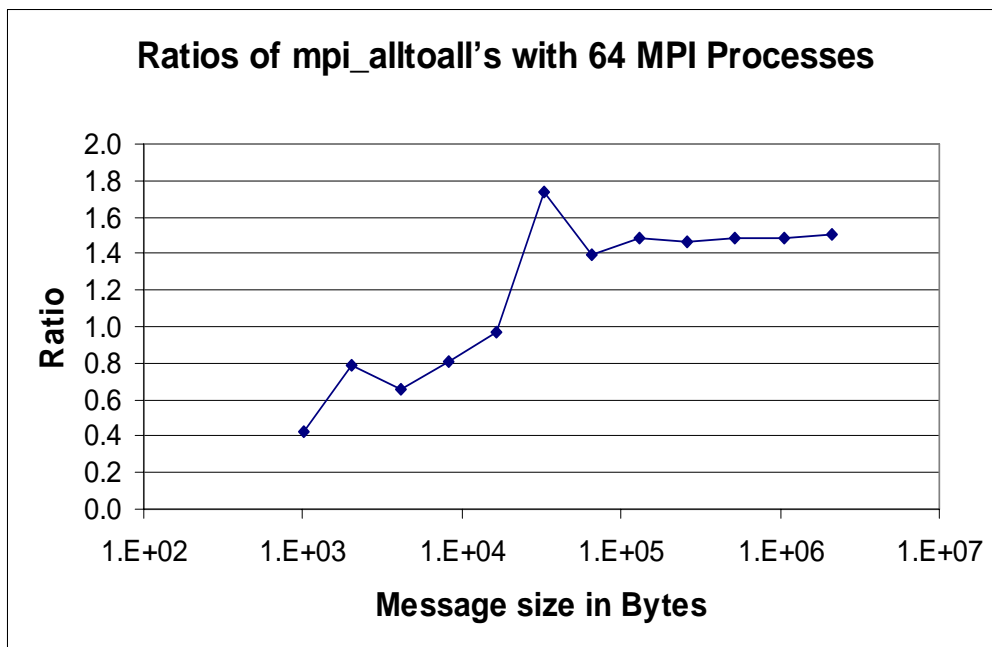


Figure 11. (2 ppn)/(1 ppn) ratios for mpi\_alltoall on the Opteron cluster using 64 MPI processes.

In summary, the MPI implementation as well as the performance and architecture of the MPI communication network affect the relative performance of running applications with 1 and 2 ppn. The MPI implementation used is “node aware” for point-to-point MPI routines but not for collective routines.

## 5. Conclusions

In this paper, we first compared the performance of the NAS Parallel Benchmarks (NPB) [2] running with 1 ppn and 2 ppn on an Intel Xeon/Myrinet cluster and on an AMD Opteron/Myrinet cluster. Performance for the NPB were compared in two different ways: (1) fixing the number of nodes and measuring 1 ppn and 2 ppn performance, and (2) by fixing the number of MPI processes and comparing 1 ppn and 2 ppn. We then attempted to explain these performance results by (1) investigating the impact of cache effects and scalability, (2) investigating the memory bandwidth of the Xeon and Opteron processors, and (3) by investigating the impact of MPI and the communication network on the performance of 1 and 2 ppn. The ChaMPIon/Pro MPI 1.2.0-3 was used on the Xeon/Myrinet cluster and the MPI used on the Opteron/Myrinet cluster was MPICH-GM 1.2.6..14b. When comparing these two clusters, it would have been better to have used exactly the same MPI implementation, but this was not possible. The impact of using two different MPI implementations is not known, but we suspect the impact is small.

The results presented in section 4.2 show that the dual processor Opteron fully supports concurrent streaming of data between memory and the two processors. The results presented in section 4.2 also show that the memory bandwidth of the dual processor Xeon is so low that one processor is idle or nearly idle for memory intensive applications. Since the memory bandwidth of the Opteron processor fully supports the concurrent streaming of data between memory and the two processors, one might conclude that running 2 ppn on the Opteron cluster will give the same performance as running 1 ppn using twice the number of nodes. However, this is not true since the MPI implementation, the selection and frequency of use of MPI routines, the topology of the communication network being used for MPI communication, and the sizes of the messages being sent all affect the relative performance of 1 ppn and 2 ppn.

Section 3 presents the 1 ppn and 2 ppn performance results for the NPB. An examination of these results shows that 1 ppn and 2 ppn performance can vary greatly from one application to another. When running applications with 1 ppn, the second processor in the node is not being used, so one would like the performance of 2 ppn to be close to the 1 ppn performance for a fixed number of MPI processes. When fixing the number of MPI processes on the Opteron cluster, 1 ppn outperforms 2 ppn for all class A, class B and class C problems, but most of the time the 2 ppn performance is close to the performance of the 1 ppn performance. However, when fixing the number of MPI processes on the Xeon

cluster, 1 ppn outperforms 2 ppn for all class A, class B and class C problems and most of the time the 2 ppn performance is much worse than the 1 ppn performance.

In conclusion, one would like to run all applications on a cluster with two processors per node using 2 ppn instead of 1 ppn in order to utilize the second processor on each node. However, the performance results from running the NPB show that better performance can sometimes be achieved using 1 ppn. The performance results in this paper also show that the Opteron/Myrinet cluster is able to achieve significantly better utilization of the second processor than the Xeon/Myrinet cluster.

## 6. Acknowledgments

We would like to thank NCSA for giving us access to their Xeon cluster. We also would like to thank University of Utah for access to their Opteron/Myrinet cluster. We also thank Myricom's technical personnel for reviewing the discussion in this paper concerning their Myrinet switch and cards.

## 7. References

1. The Message Passing Interface (MPI) Standard, <http://www-unix.mcs.anl.gov/mpi/>
2. The NAS Parallel Benchmarks, <http://www.nas.nasa.gov/Software/NPB/>
3. Chona S. Guiang, Kent F. Milfeld, Avijit Purkayastha, and John R. Boisseau, *Memory Performance of Dual-Processor Nodes: Comparison of Intel, Xeon, and AMD Opteron Memory Subsystem Architectures*, Clusterworld 2003 conference, [http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF03/Chona\\_S.pdf](http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF03/Chona_S.pdf)
4. Douglas M. Pase, James Stephens, *Performance of Two-Way Opteron and Xeon Processor-Based Servers for Scientific and Technical Applications*, Linux Clusters: The HPC Revolution 2005 conference, [http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF05/14-Pase\\_D.pdf](http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF05/14-Pase_D.pdf)
5. The IA-32 Linux Cluster at the National Center for Supercomputing Application (NCSA), University of Illinois Urbana-Champaign, <http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IA32LinuxCluster/>
6. Glenn R. Luecke, Marina Kraeva, Jing Yuan, Silvia Spanoyannis, *Performance and Scalability of MPI on PC Clusters*, Performance Evaluation & Modeling of Computer Systems, November, 2002, and CONCURRENCY AND COMPUTATION: PRACTICE

AND EXPERIENCE, 2004: volume 16, pages 79-107. Also see,  
<http://www.public.iastate.edu/~grl/publications.html>

7. STREAM: Sustainable Memory Bandwidth in High Performance Computers,  
<http://www.cs.virginia.edu/stream/>

8. Center for High Performance Computing at the University of Utah,  
<http://www.chpc.utah.edu/>

9. Myricom, <http://www.myri.com/>

10. Pase, Douglas M.; *Performance of Voltaire InfiniBand in IBM 64-Bit Commodity HPC Clusters*, IBM Corporation 2005, IBM Systems and Technology Group, Department MXSA, Research Triangle Park, NC 27709, May 2005,  
[ftp://ftp.software.ibm.com/eserver/benchmarks/wp\\_IB\\_Performance\\_053105.pdf](ftp://ftp.software.ibm.com/eserver/benchmarks/wp_IB_Performance_053105.pdf)

## Appendix 1: Class A & C NPB Performance Results

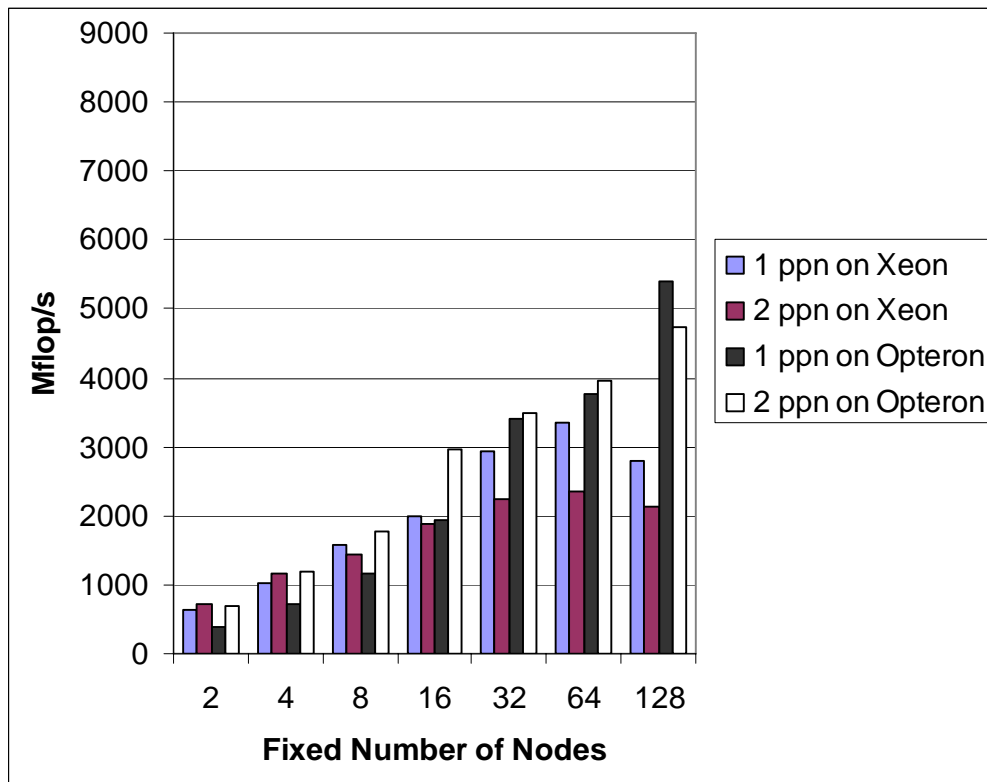


Figure 1.A.a. Class A CG benchmark for a fixed number of nodes.

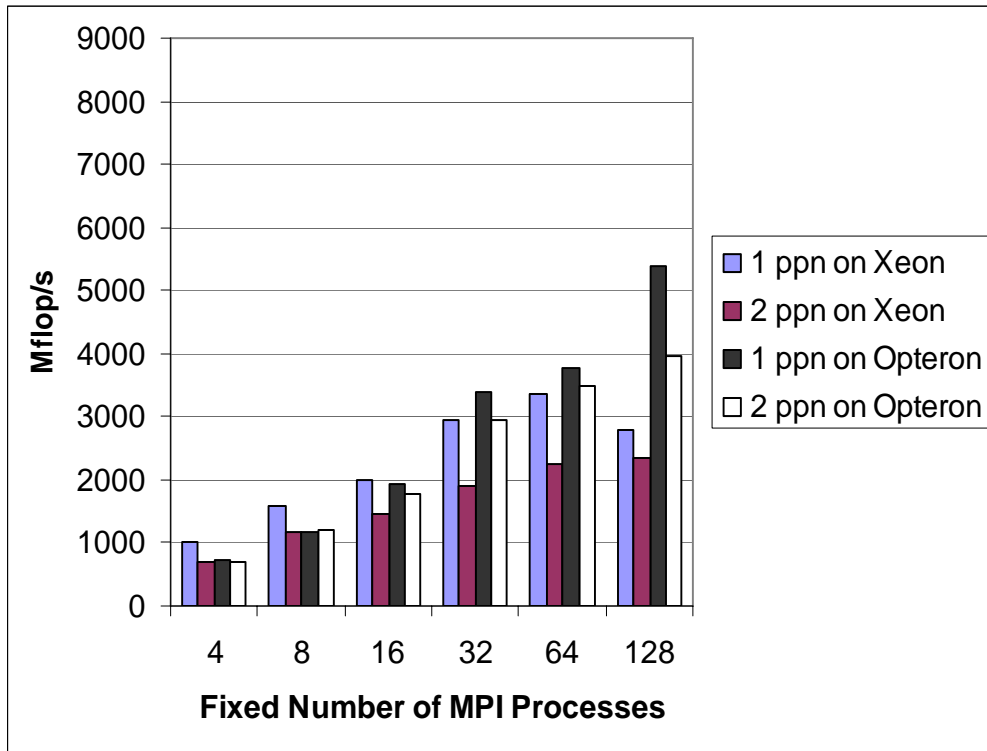


Figure 1.A.b. Class A CG benchmark for a fixed number of MPI processes.

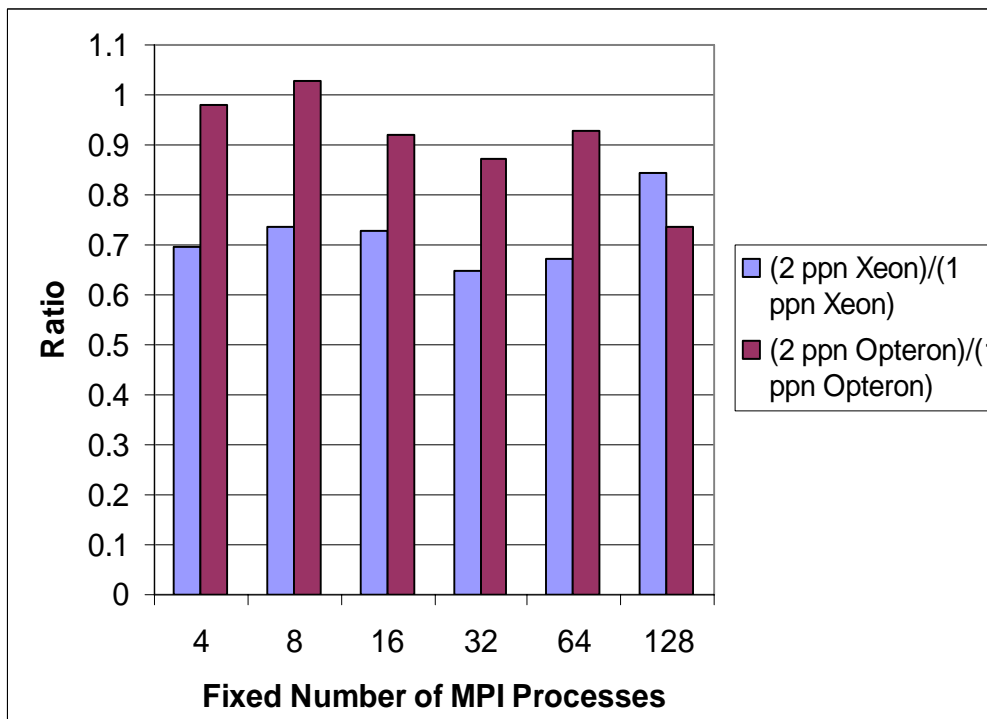


Figure 1.A.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

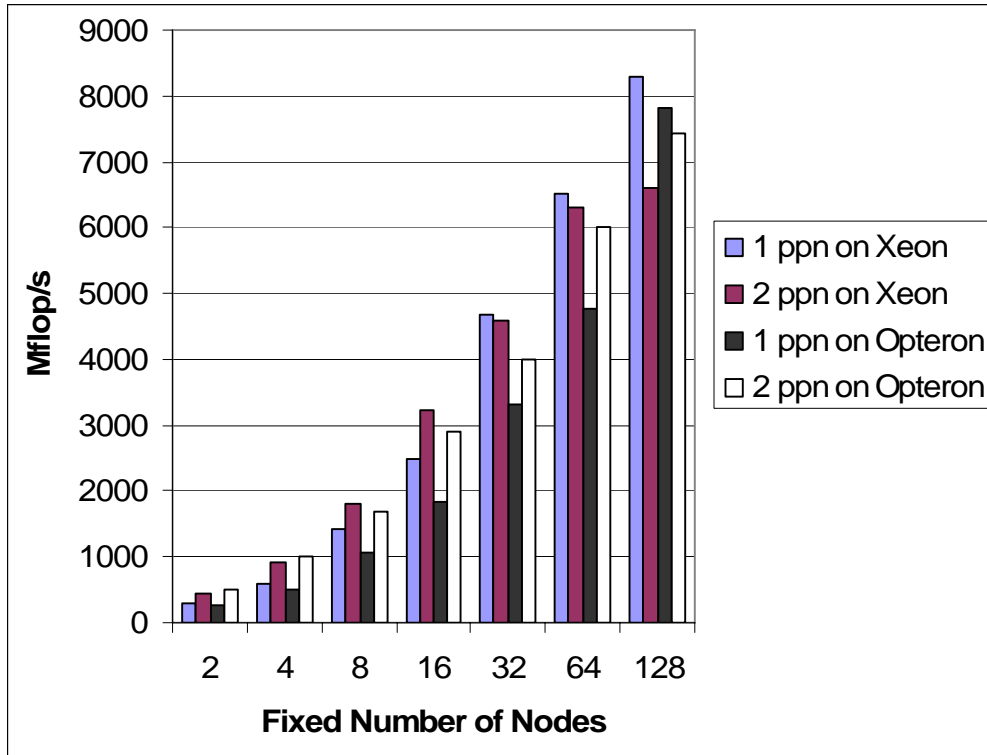


Figure 1.C.a. Class C CG benchmark for a fixed number of nodes.

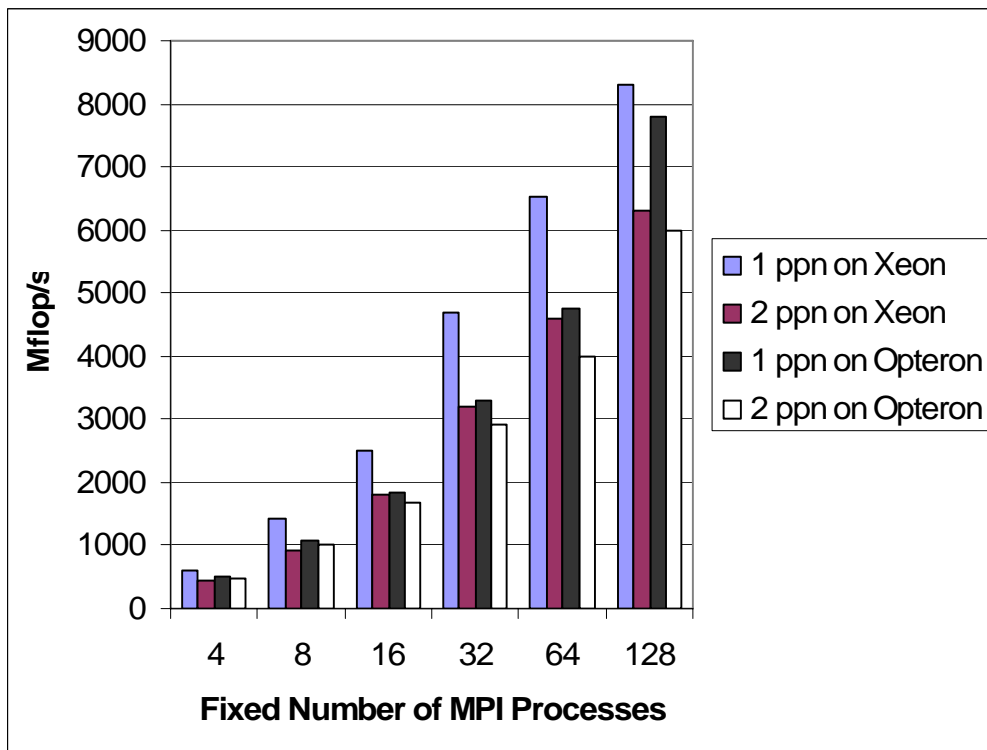


Figure 1.C.b. Class C CG benchmark for a fixed number of MPI processes.

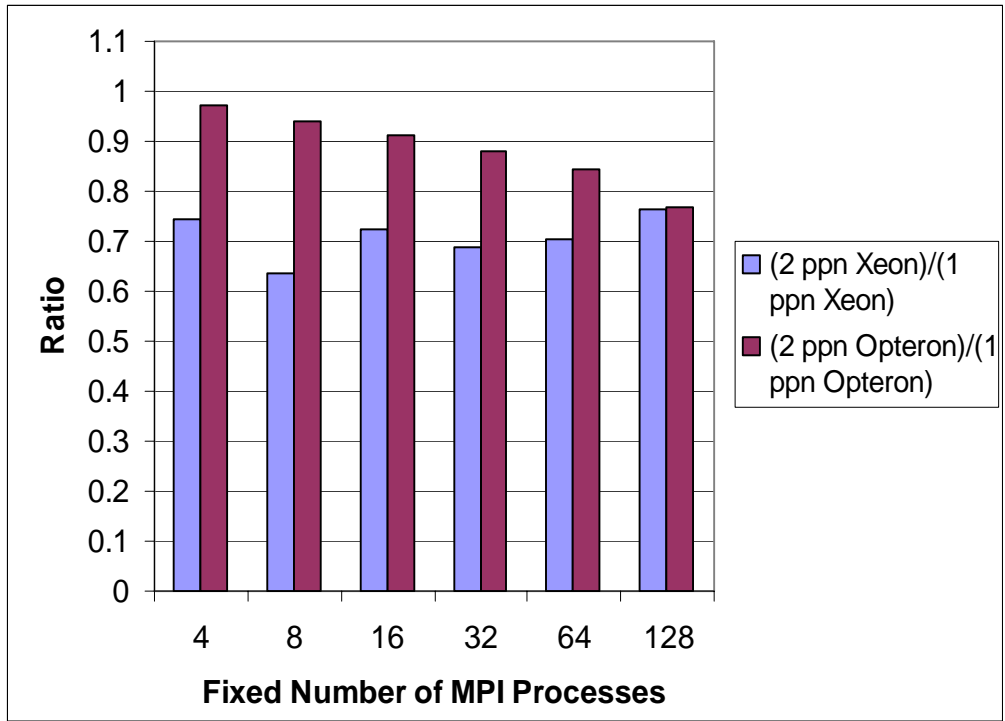


Figure 1.C.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

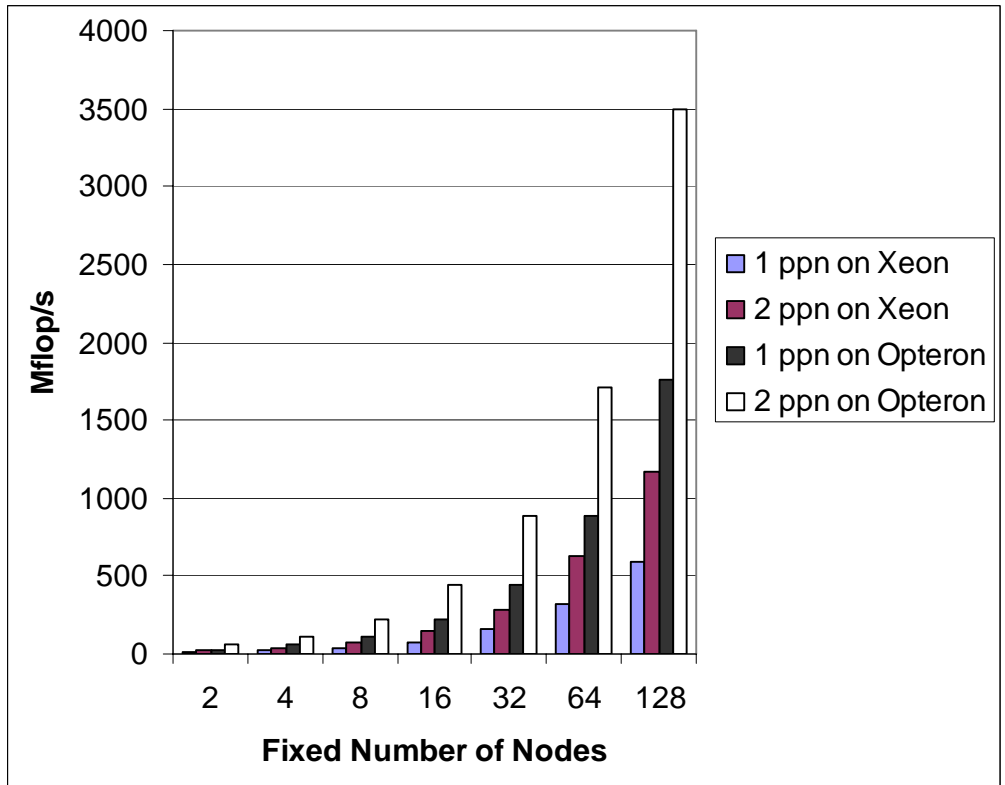


Figure 2.A.a. Class A EP benchmark for a fixed number of nodes.

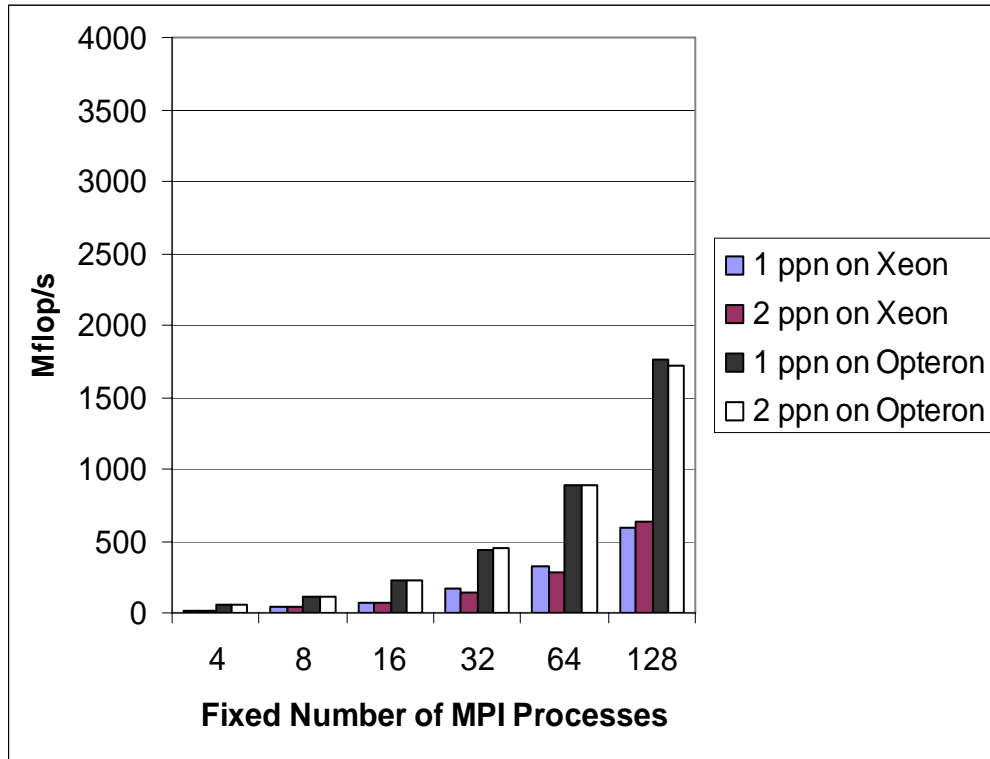


Figure 2.A.b. Class A EP benchmark for a fixed number of MPI processes.

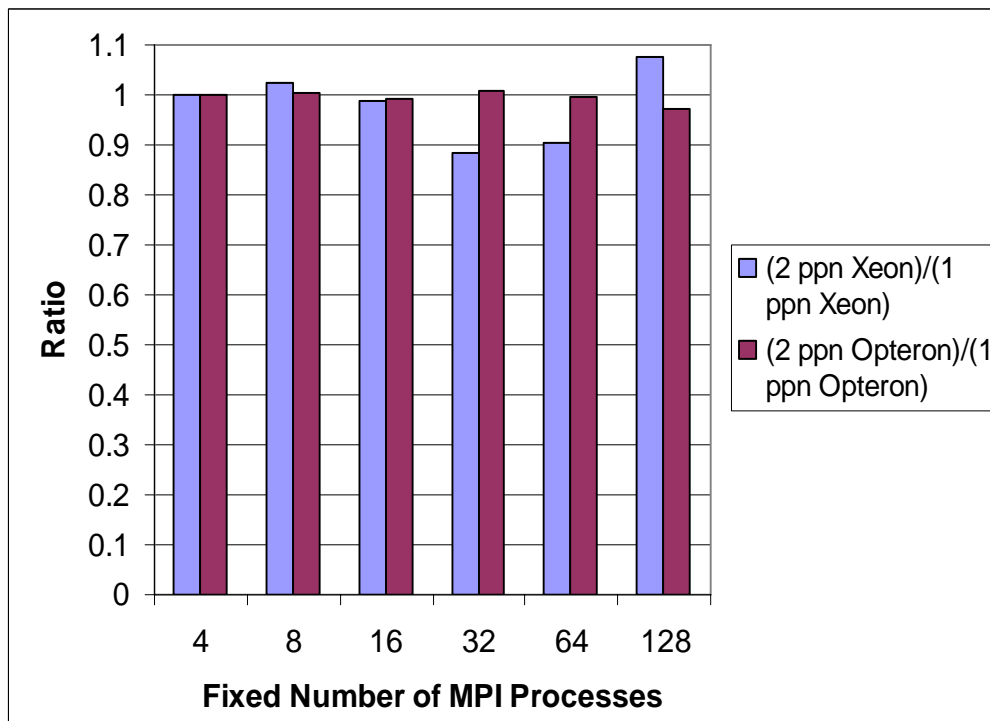


Figure 2.A.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

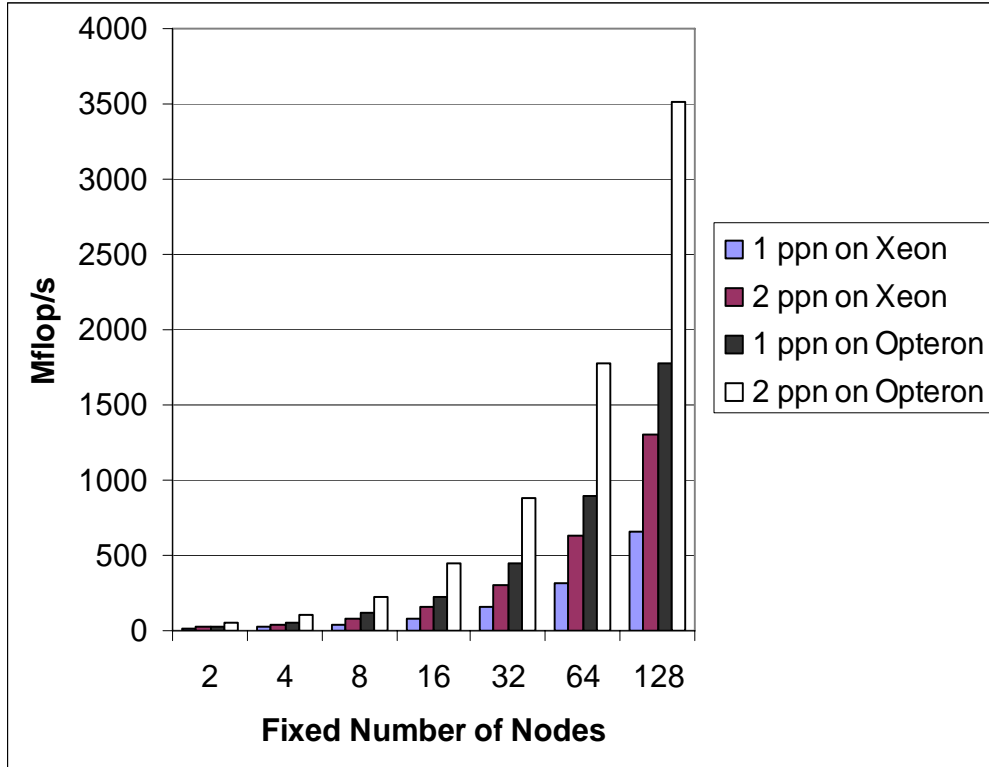


Figure 2.C.a. Class C EP benchmark for a fixed number of nodes.

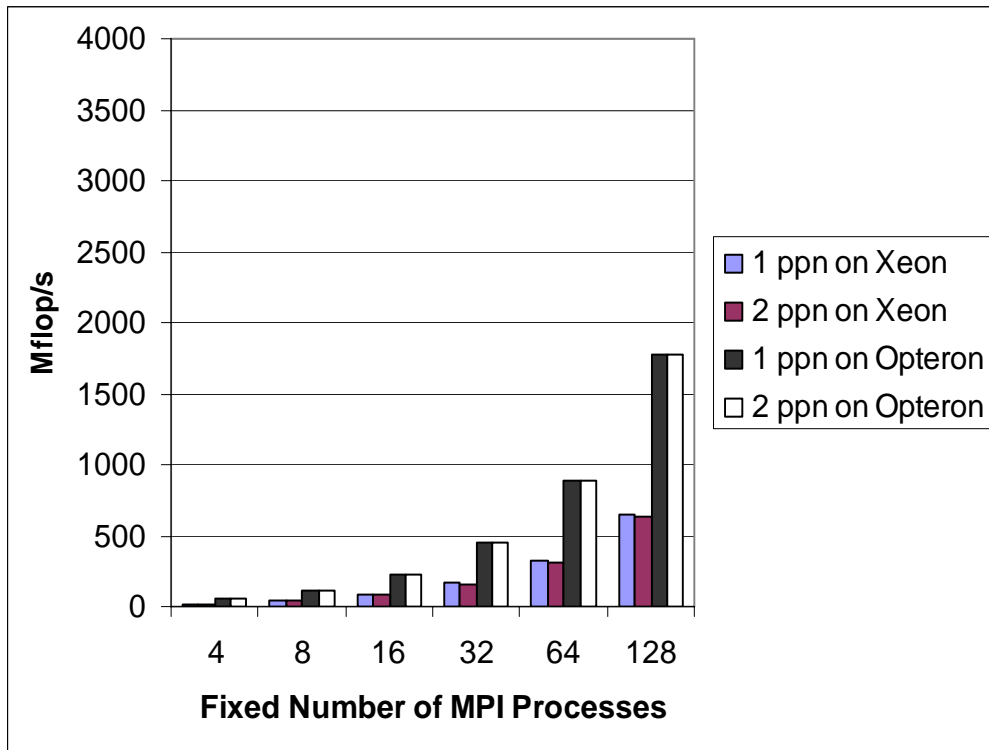


Figure 2.C.b. Class C EP benchmark for a fixed number of MPI processes.

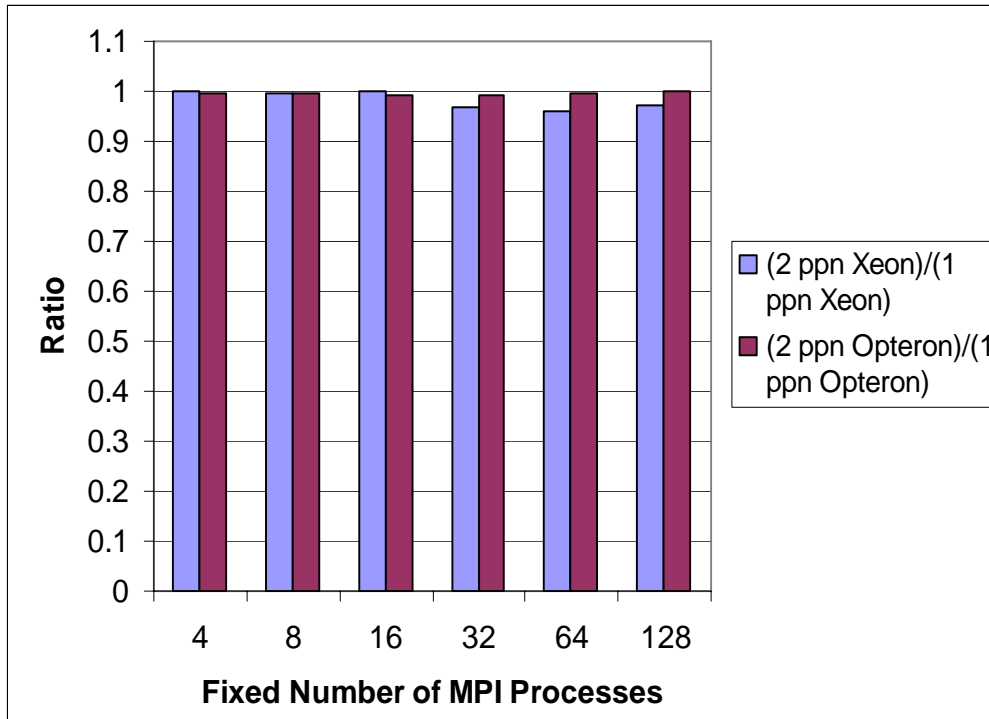


Figure 2.C.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

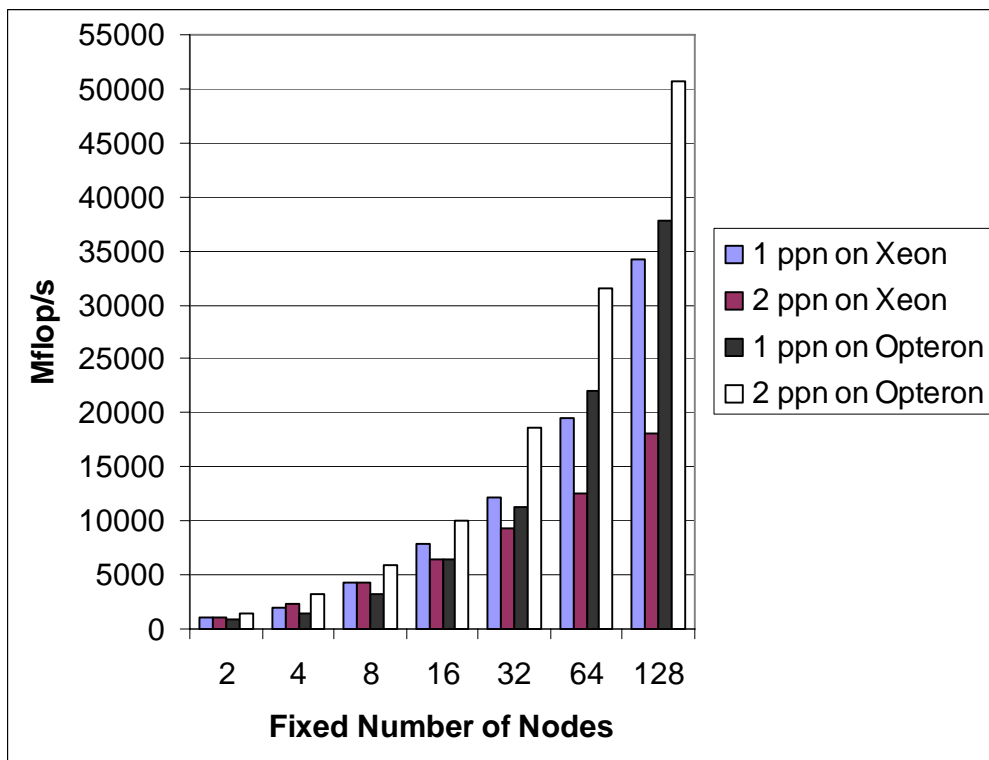


Figure 3.A.a. Class A MG benchmark for a fixed number of nodes.

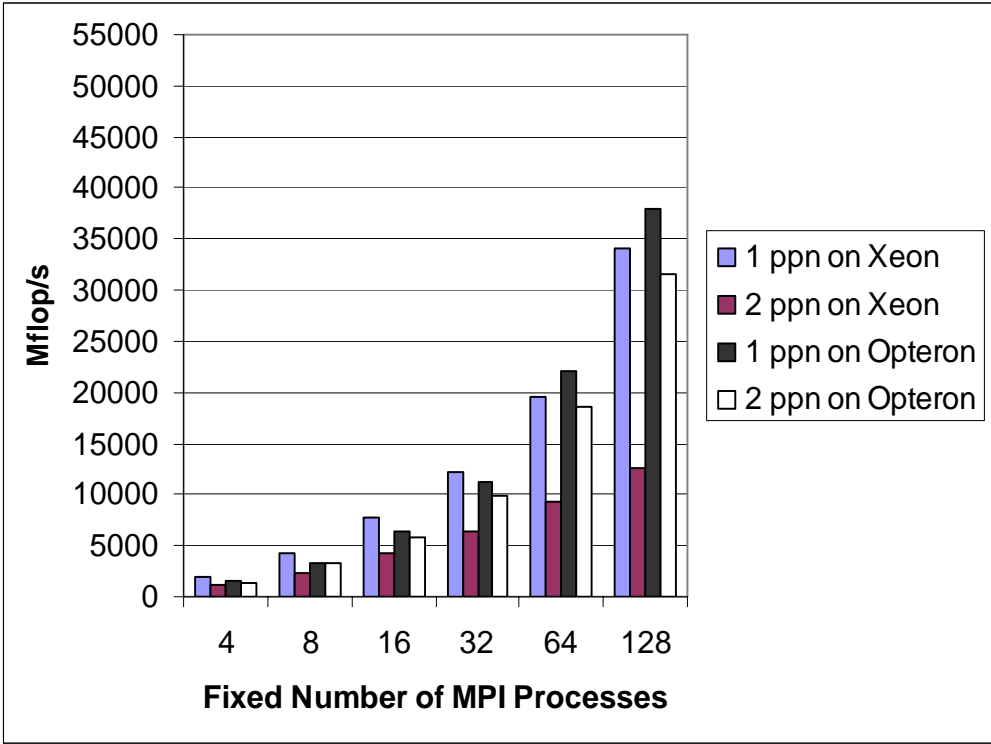


Figure 3.A.b. Class A MG benchmark for a fixed number of MPI processes.

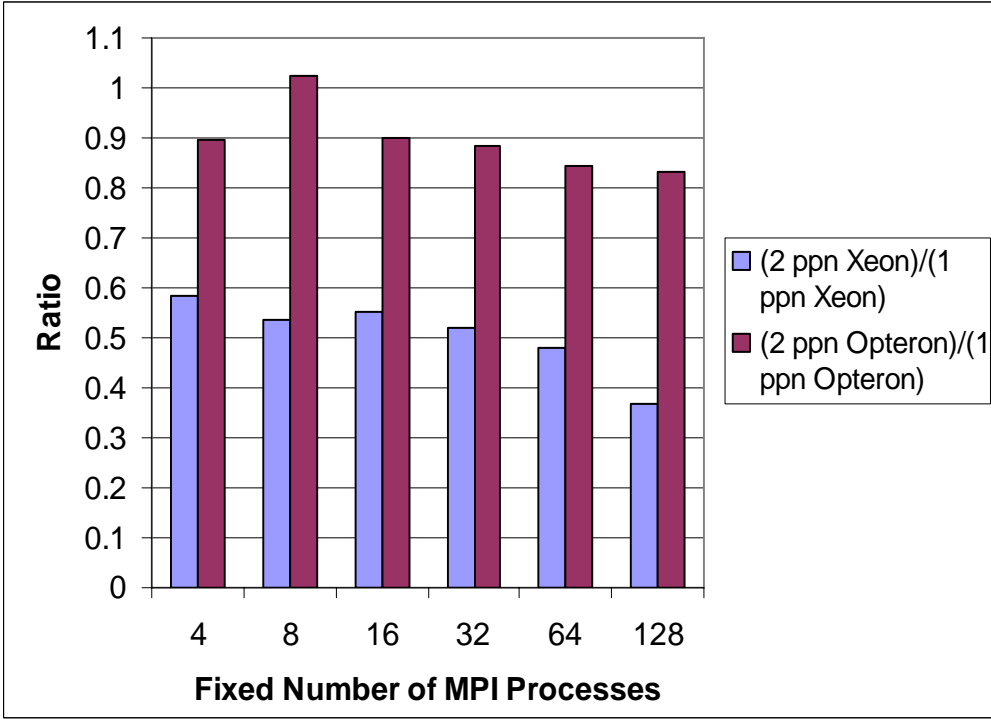


Figure 3.A.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

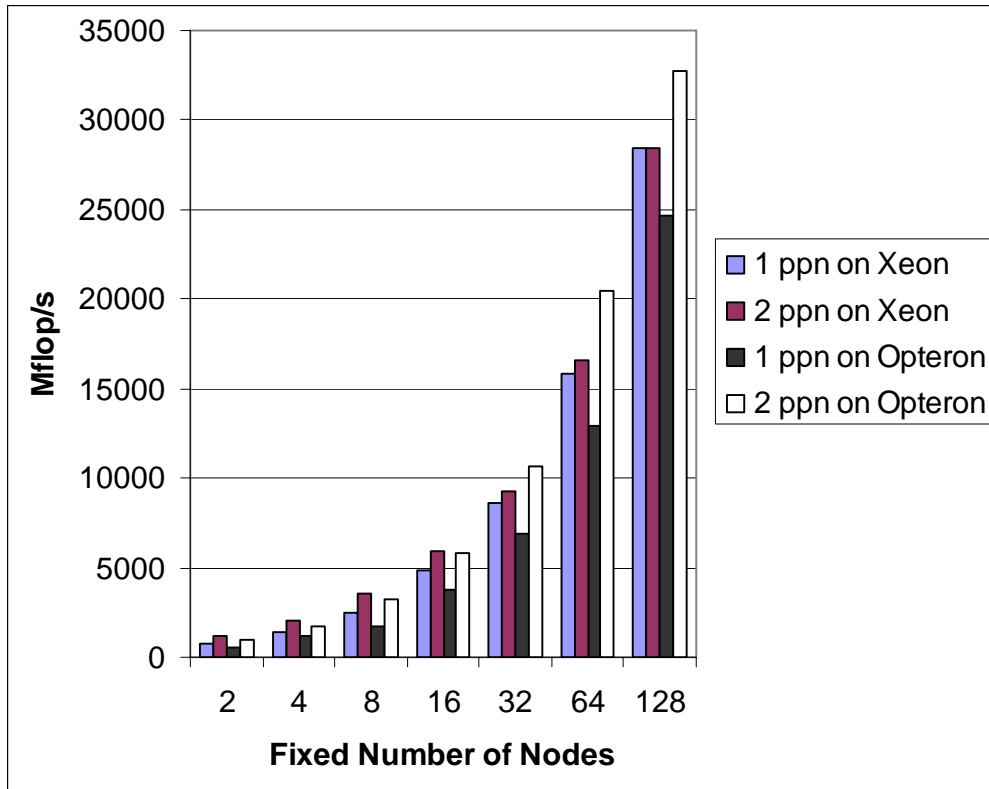


Figure 4.A.a. Class A FT benchmark for a fixed number of nodes.

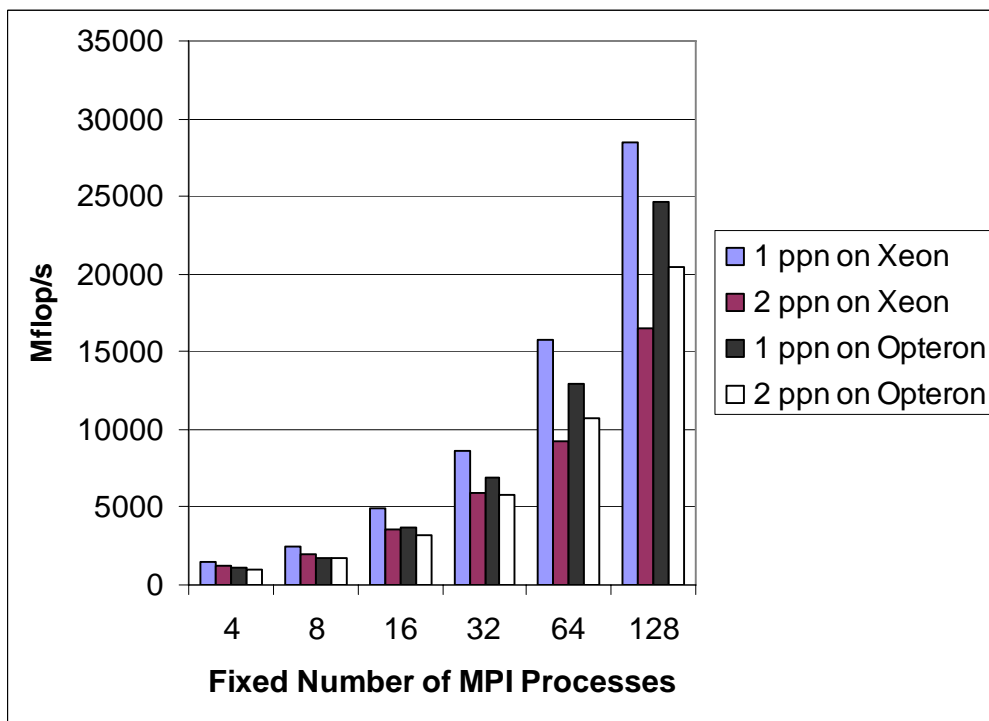


Figure 4.A.b. Class A FT benchmark for a fixed number of MPI processes.

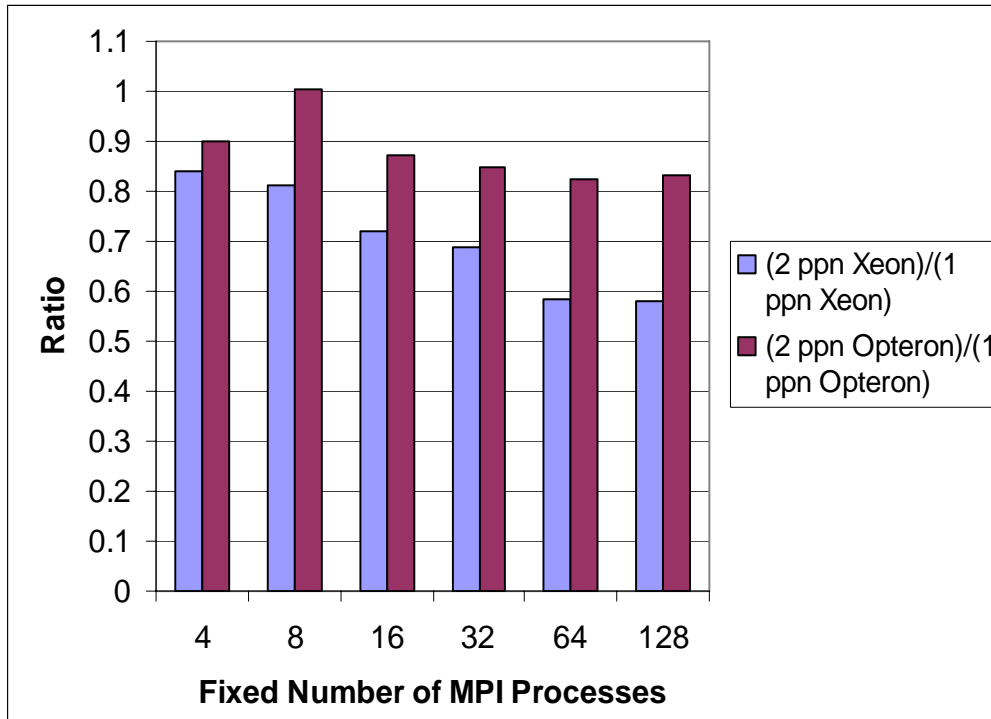


Figure 4.A.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

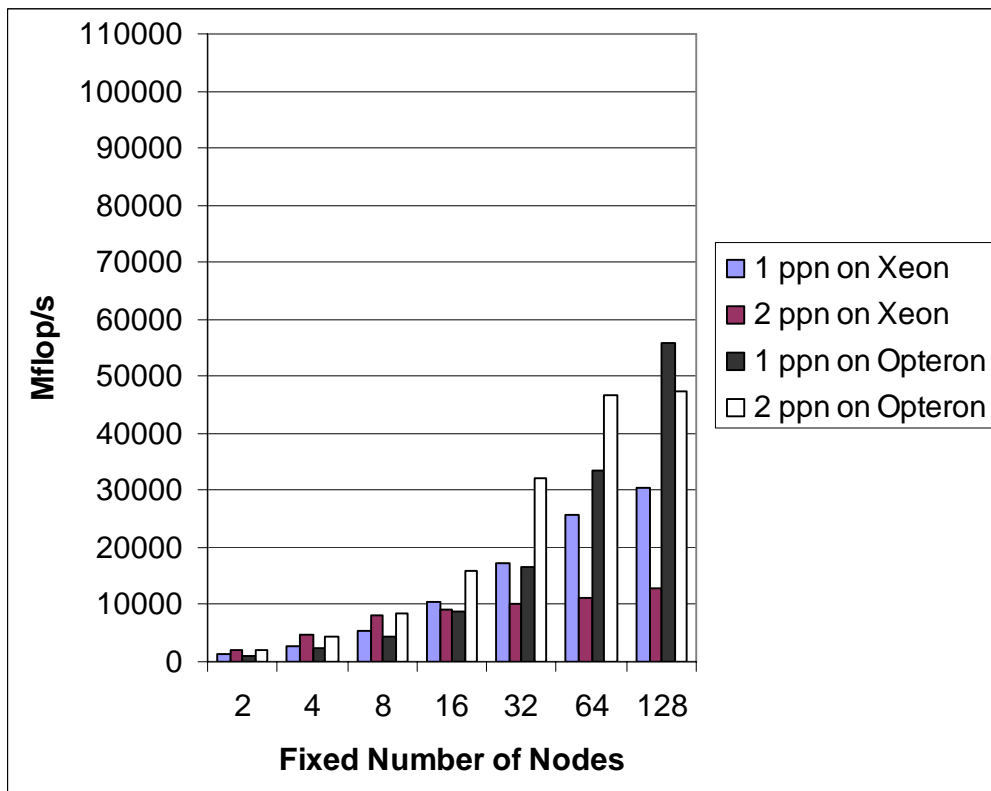


Figure 5.A.a. Class A LU benchmark for a fixed number of nodes.

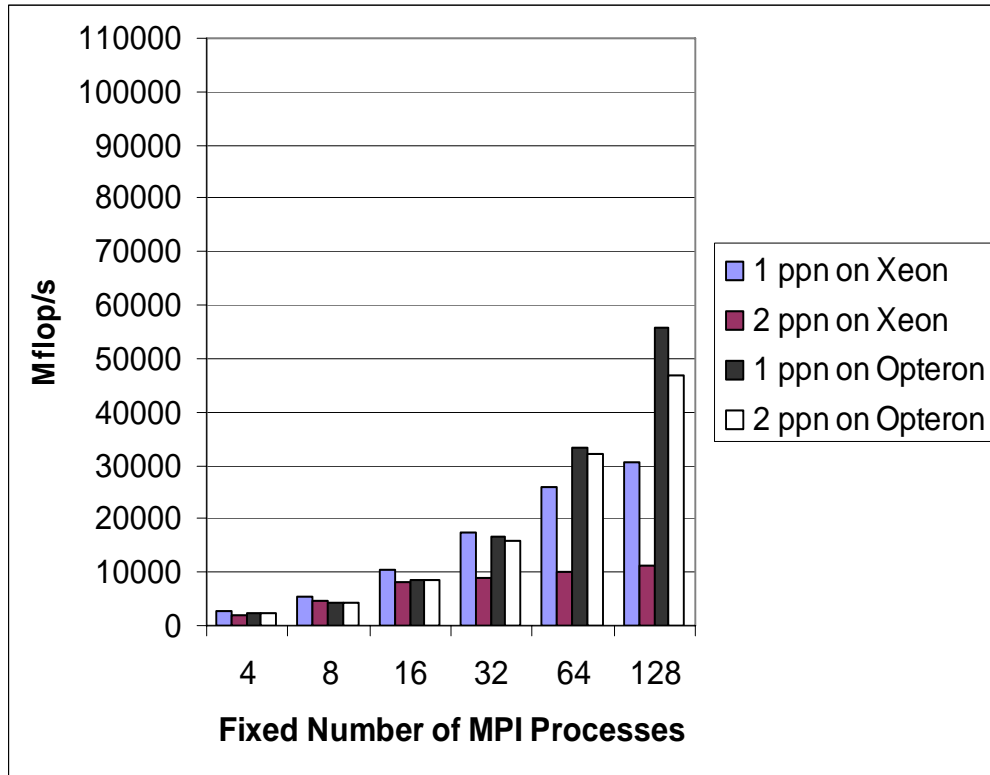


Figure 5.A.b. Class A LU benchmark for a fixed number of MPI processes.

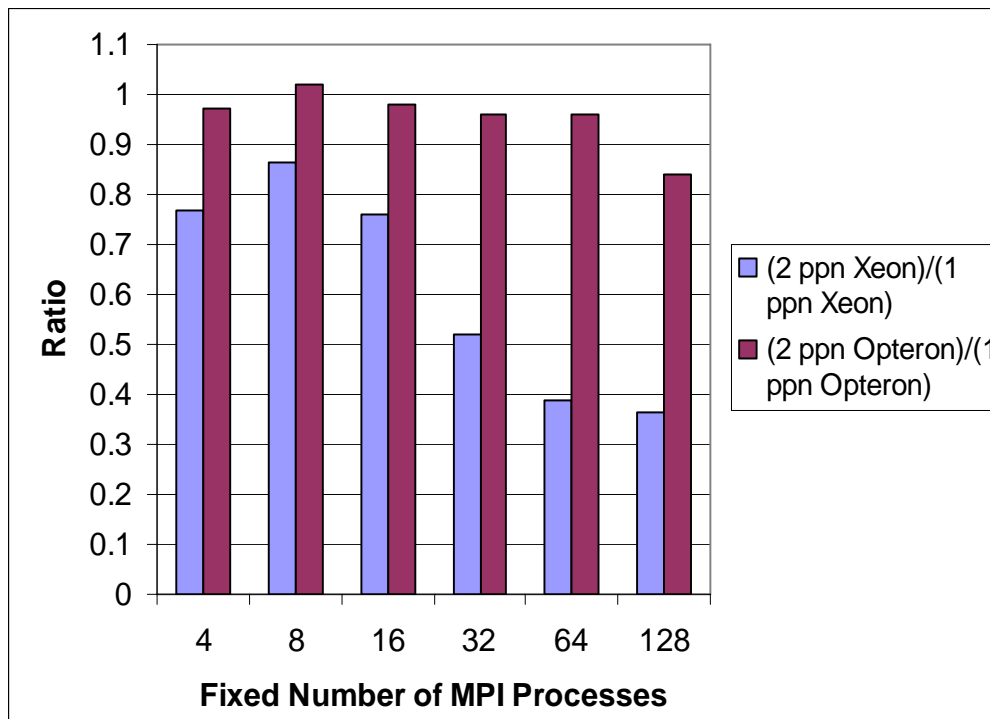


Figure 5.A.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

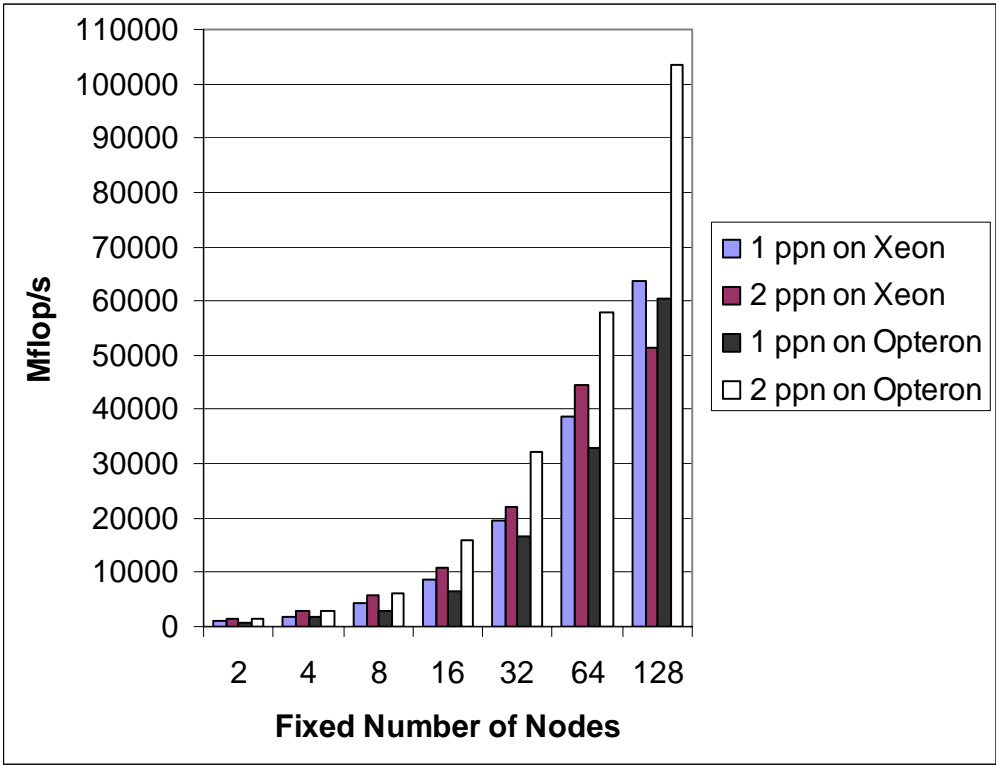


Figure 5.C.a. Class C LU benchmark for a fixed number of nodes.

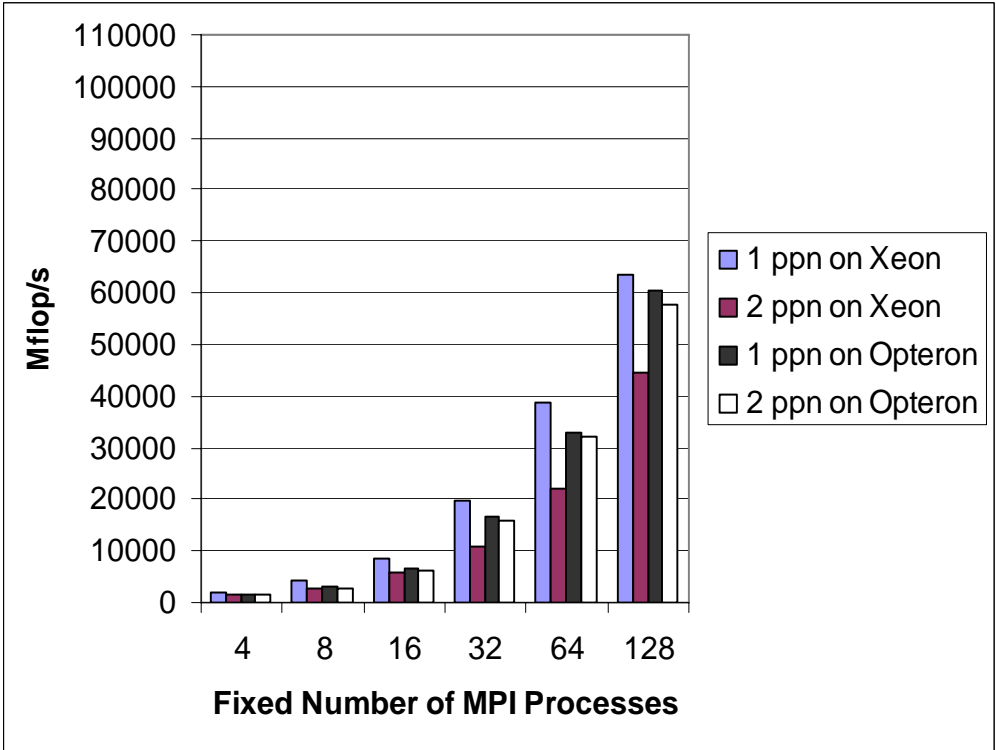


Figure 5.C.b. Class C LU benchmark for a fixed number of MPI processes.

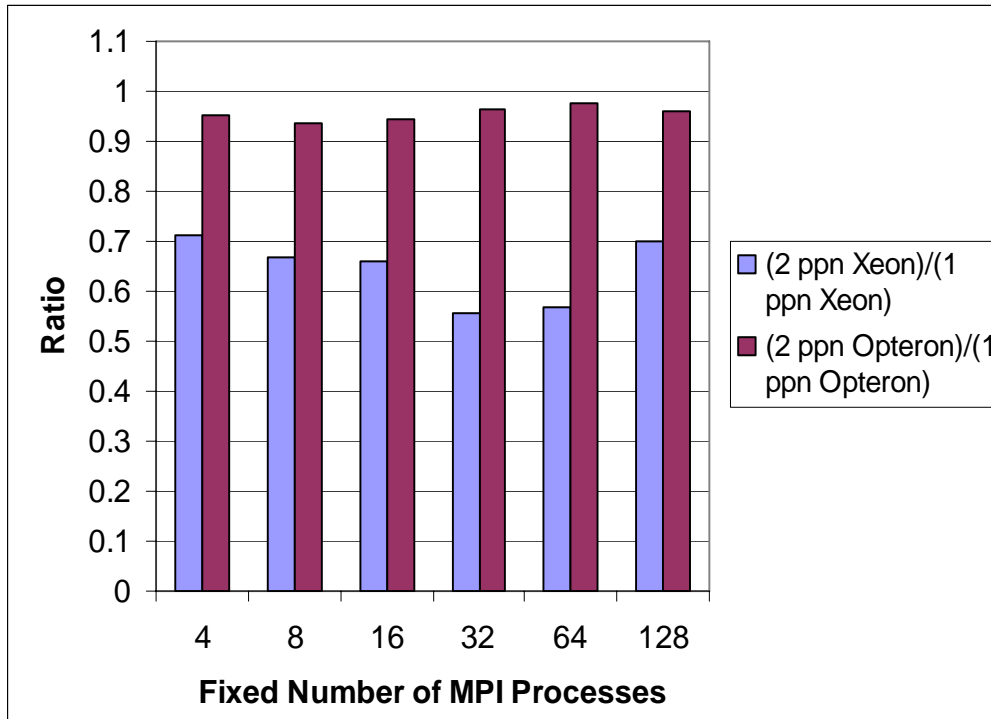


Figure 5.C.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

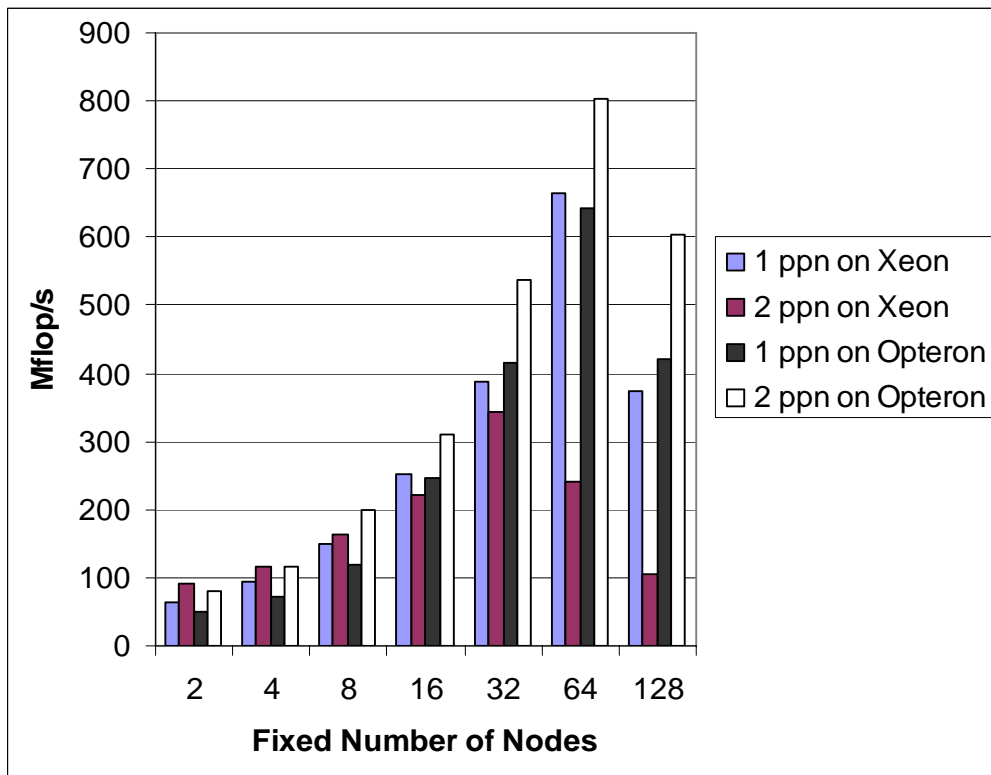


Figure 6.A.a. Class A IS benchmark for a fixed number of nodes.

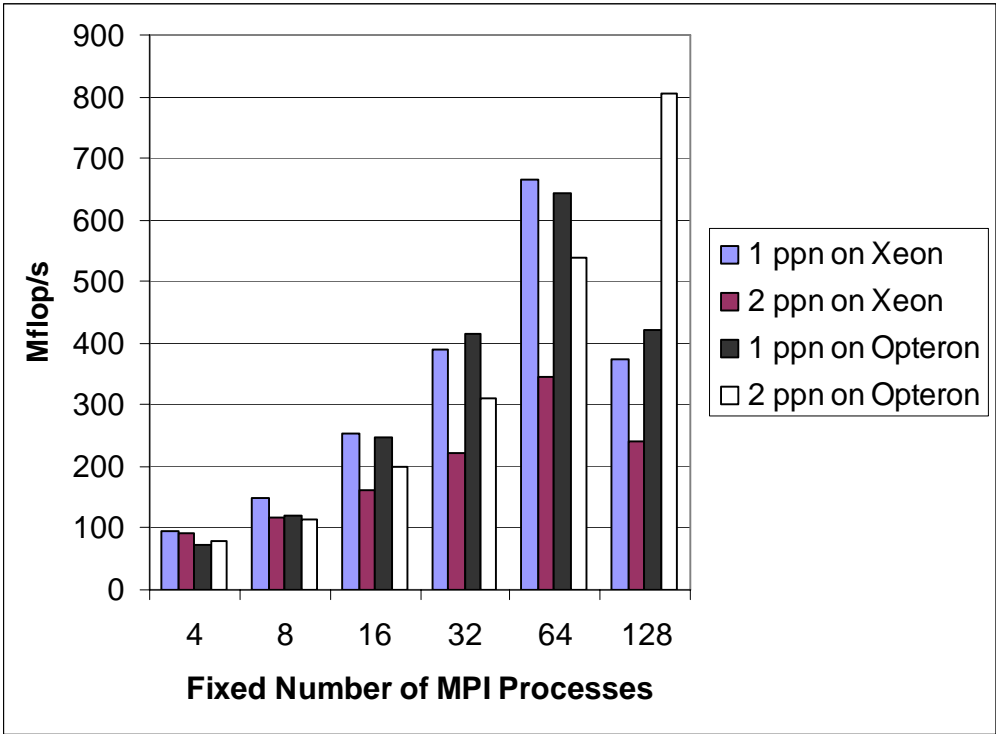


Figure 6.A.b. Class A IS benchmark for a fixed number of MPI processes.

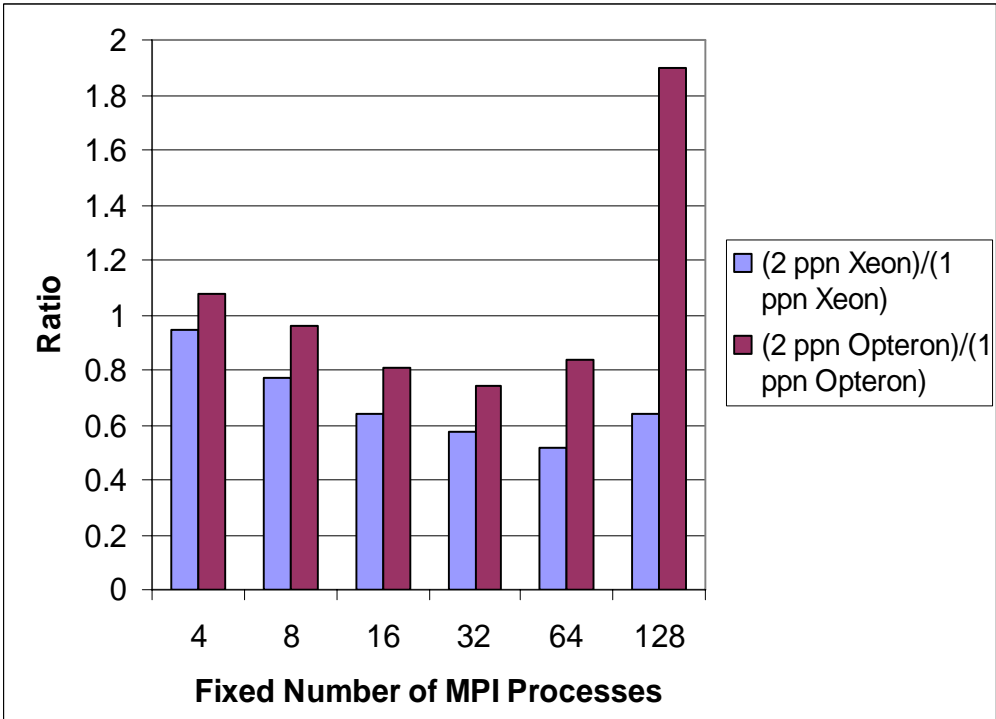


Figure 6.A.c. (2 ppn)/(1 ppn) ratios for the Xeon and Opteron clusters.

## Appendix 2:

### Timing Methodology for Memory Bandwidth Tests

Timing is performed by first flushing the caches on all processors by changing the values in the `real*8` array `flush(ncache)` where `ncache` is an integer parameter large enough so that the flush array is at least as large as the size of the largest (level 3) cache. There are many different ways to flush a cache. One way is to call the Fortran intrinsic: `call random_number(flush)`. This gives random values between 0 and 1 to the flush array. As this is being done, all other data is removed from all (data) caches since both the Xeon and Opteron processors use a least recently used cache replacement scheme. The random number generator uses a lot of time to execute, so we flushed the caches by simply executing the loop: `flush(1:ncache) = flush(1:ncache) + 0.1d0`. Any `real*8` number can be used instead of 0.1d0 that will not cause an overflow to occur. The size of the largest cache on the Opteron and 3.2 GHz Xeon is 1 MB and for the 2.8 GHz Xeon 512 KB.

`Ntrial` is the number of timing tests performed in each single test, and it was set to 50 for all tests. Tests are run with strides equal 1, 2, 3, ..., `max_stride=17`. Stride 1 array accesses provided the least memory traffic and stride 17 the heaviest.

```
integer,parameter :: ncache= 1024*1024/8! number of 8 byte words in the largest cache
integer,parameter :: ntrial=50, max_stride=17
real*8 :: flush(ncache), array_time(ntrial), max_time(ntrial), A(300000*17)
...
do stride = 1, max_stride
  do ktrial = 1, ntrial
    flush(1:nflush) = flush(1:nflush) + 0.1d0 ! flush the cache
    call mpi_barrier(comm,ierror)
    t = mpi_wtime()
    do i = 1, n*stride, stride
      s = s + A(i) ! memory read
    enddo
    array_time(ktrial) = mpi_wtime() - t
    call mpi_barrier(comm, ierror)
    s = s + flush(mod(ktrial,nflush)) ! prevent compiler from splitting out the cache flushing
  enddo
  call mpi_reduce(array_time,max_time,ntrial,dp,mpi_max,0,comm,ierror)
  ...
  print*, 'flush(1)+array_time(1)+s=', flush(1)+array_time(1)+s ! prevent dead code elimination
enddo ! for the stride loop
```

In the above timing code, the first call to `mpi_barrier` guarantees that all processors reach this point before calling `mpi_wtime`. The second call to `mpi_barrier` is to make sure that no processor starts the next iteration (flushing the cache) until all processors have completed executing the code to be timed. To prevent the compiler from considering all or part of the code to be timed as dead code and eliminating it, the value of `flush(1)+array_time(1)+s` was printed. The call to `mpi_reduce` calculates the maximum of `array_time(ktrial)` for each fixed `ktrial` and places this maximum in `max_time(ktrial)` on processor 0 for all values of `ktrial`.

Figure 12 shows 100 times measured for the COPY operation. Notice that the first time trial is much longer than the rest. This is typical of all the times performed in section 3.1. Because the first time trial was usually much longer than subsequent time trials, the first time trial was thrown away and an average was taken of the remaining time trials.

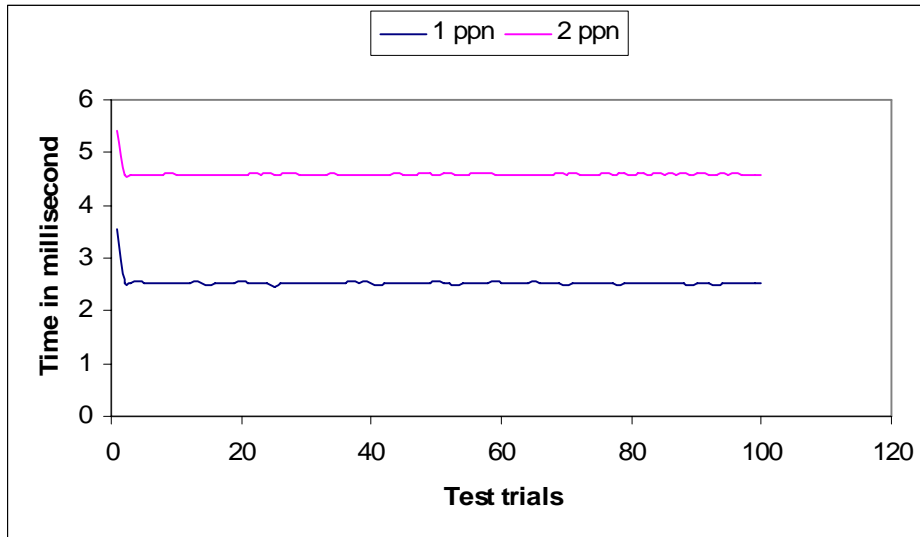


Figure 12. 100 time trials for COPY with stride=1 and a 300000\*8 Bytes message.