Evaluation of MPIP with IRS Benchmark: Final Report
CSC 591C – Cluster Computing
May 2, 2003

Anwar Ali **Annika Edwards** **Nhon Nguyen**

Introduction

In this project we seek to understand fully how MPIP is used, and how the output can help us to determine where communication time should be reduced. For example, if a large amount of time is being spent on an MPI_BARRIR call in several processes, this may indicate a load imbalance.

We will use the IRS Benchmark Code (IRS) which executes on both SMP and multi-node systems, to measure and compare the performance of a large application on the cluster. This will help us understand more about large scale applications on SMP machines and parallel architectures.

Installation of MPIP and IRS Benchmark

MPIP
First download and extract the file: "MPIP_2002_07_08.tar". Then run "confgure". Change "Makefile" to use "mpicc" compiler. Finally build MPIP with "make".

MeshTV/Silo (Need Silo only for IRS application)
First download and extract the file: "meshtv4_3_1.linux.tar.gz", and follow, "INSTALL_NOTES" for Linux. Next Modify "configure" as follows:

```
Change: $CPP to use "gcc"
Change all paths "/usr/local/*" to "/usr/*"
```

Then modify "confugure.in" by changing all paths `"/usr/local/*"` to `"/usr/*."` Run "configure" and "make."

The IRS Benchmark Code
Download and extract the file: "irs1.3.tar". Go to "scripts" directory, modify all "`irs_*`" scripts; change the path

```
"#!/usr/local/bin/perl" to "#!/usr/bin/perl".
```

Go to the "build" directory. Modify "Makefile" for the following:

Silo Library Path:

```
 SILO_LIBS = -lsilo
 SILO_LIBPATH = -L/home/nvnguyen/meshtv020506/lib
 SILO_INCPATH = -I/home/nvnguyen/meshtv020506/include
```

MPI Path:

```
MPI_LIBPATH = -L/opt/mpich-1.2.4..8/lib
MPI_INCPATH = -I/opt/mpich-1.2.4..8/include
```

OpenMP path

```
OPENMP_LIBPATH = -L/opt/intel/compiler70/ia32/lib
OPENMP_INCPATH = -I/opt/intel/compiler70/ia32/include
```

Now modifiy make file to compile with “mpicc". And build with:

Build: **make** <build-option>

opt:	builds the optimized code
debug:	builds the debug code
lint:	runs lint on the code
gcc:	runs gcc as a syntax checker on the code

Benchmark Testing

Inputs

IRS provides a set of decks, labled zrad.XXXX, that input for optimized for XXXX number of processes. We used the three input files zrad.0008.seq, zrad.0008, and zrad.0064. zrad.0008.seq is optimized to run on pure OpenMP, while zrad.0008 and zrad.0064 can be used with MPI alone or with an MPI/OpenMP hybrid.

Methodology

We ran the following tests:

Pure OpenMP Threads

We on the code compiled for pure OpenMP as follows:

```
irs -k omp_seq   zrad.0008.seq
irs -k omp_<nds> zrad.0008.seq -threads
```

The firt test runs the code without threads. The second run will run the same file with threads. The number of threads used was set using,

```
export OMP_NUM_THREADS=<nthrds>
```

where <nthrds> was set to 2 and 4.

Pure MPI

We ran the code compiled for pure MPI on 2, 4, and 8 processors the input file zrad.0008. We also used zrad.0064 when using the MPIP profiling tool. This was accomplished with the following command:

```
mpirun -np <nds>  irs -k mpi.<nodes> zrad.<nprocs>
```

where <nprocs> was set to 0008 and 0064

MPI and OpenMP Threads

We ran the code compiled for MPI+OpenMP on 2, 4, 8 processors, with the number of threads varying from 2 to 8.

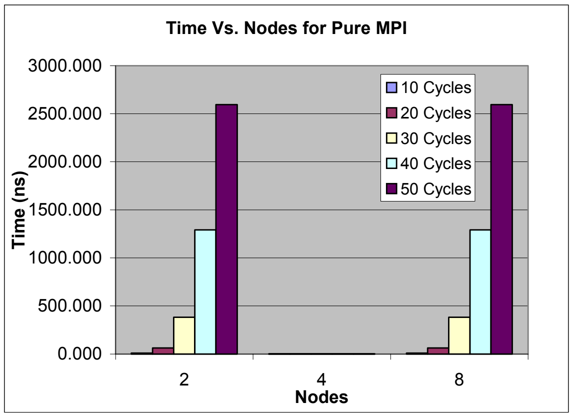
```
export OMP_NUM_THREADS=<nthrds>
mpirun -np <nds> irs -k mpi/omp.<nds><thrds> zrad.<nprcs> -threads
```

Problems

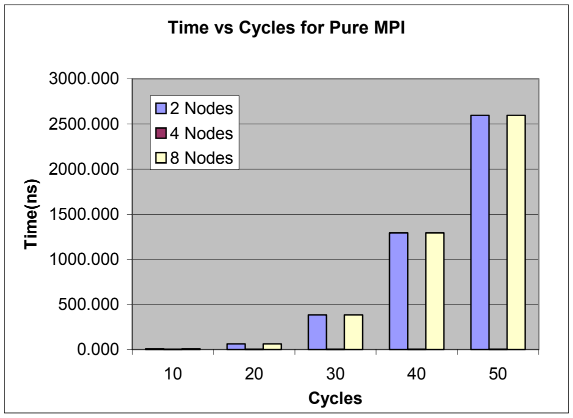
We were unable to run the IRS code to completion. When we attempted to let the program continue beyond a certain number of cycles the program would hang until killed. The cycle number where this occurred varied depending on the number of processes being used. We chose fifty cycles as a minimum cycle where all programs still made progress.

Results

Pure MPI



Graph 1



Graph 2

	Time (ns)		
Cycle	2 Nodes	4 Nodes	8 Nodes
10	9.232	1.200	9.232
20	62.247	1.814	62.247
30	383.571	2.610	383.571
40	1292.231	2.711	1292.231
50	2595.515	2.736	2595.515

Table 1.

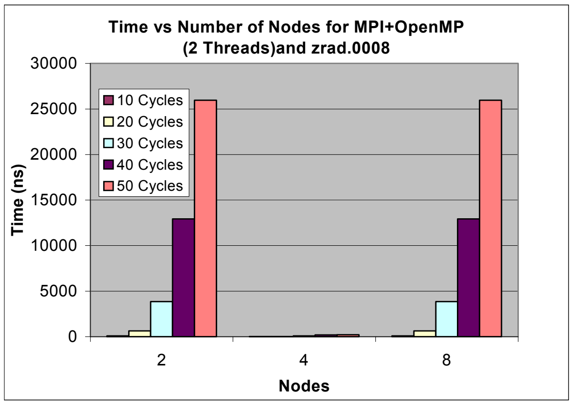
Graph 1 and 2 show how the time the program reaches a cycle number varies as the nodes increase. The performance of IRS is optimal when there are four nodes. It seems implausible but the cause may be that when the number of nodes is below four the program can not compute fast enough to compensate for the communication and synchronization. Parent thread will have to send larger messages to get the data distributed evenly on the two processors. If there are greater then four processors the synchronization may overwhelm the computation. I say this is implausible because this experiment was run using the zrad.0008 input file. This should be optimized to run on eight processors. The experiment was run several times to make sure the behavior was repeatable. We observed the same result for every run we made, so the problem is not easily attributed to other student traffic on the server. There seems to be no explanation to this.

Pure OpenMP

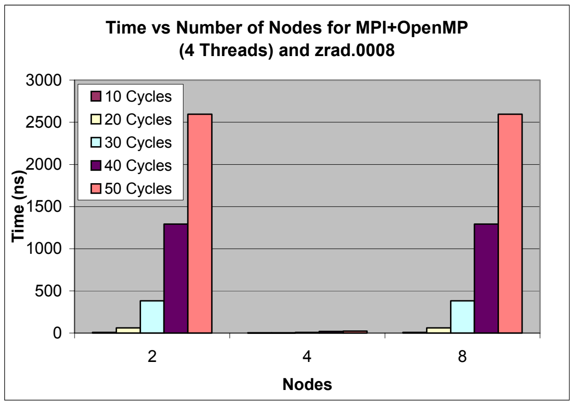
We varied the number of nodes from 1, 2, 4, 8. The data that we collected was the same for all 4 runs of the code. We rechecked parameters passed to program and ran the tests several times at different times of day. The results were still identical. The graphs of this strange fact are no included since they no more clearly show the equality. Increasing the number of threads does not seem to have any effect on the performance of this application.

We expected to see the time to complete the cycles decrease from one node to two nodes since the threads are distributed across the two processors on the node. The explanation for this did is unclear to us. It is possible that the threads are not placed on two separate processors.

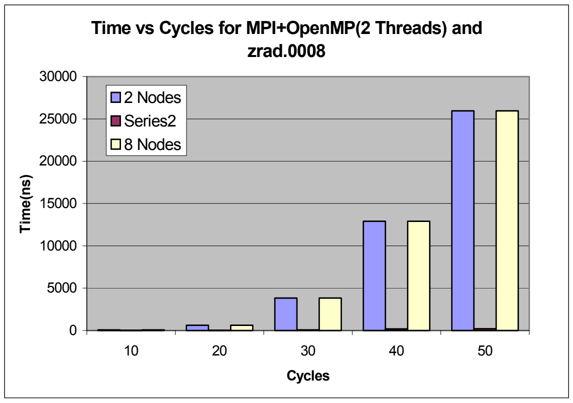
MPI + OpenMP



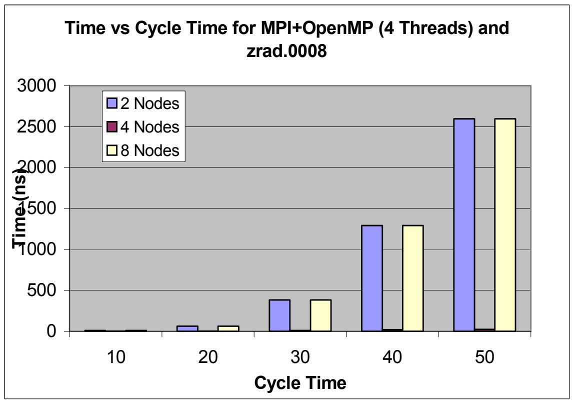
Graph 3.



Graph 4.



Graph 5.



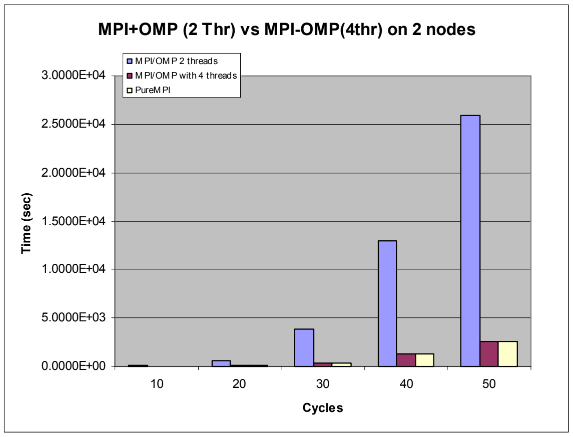
Graph 6.

These graphs show the same result as for the pure MPI case. There is really no difference in the performance seen for 2 or 8 processors. The performance is best for four nodes. The fact that the performance stays low for four nodded with 1, 2 and 4 threads on a processor, means this is the optimal case.

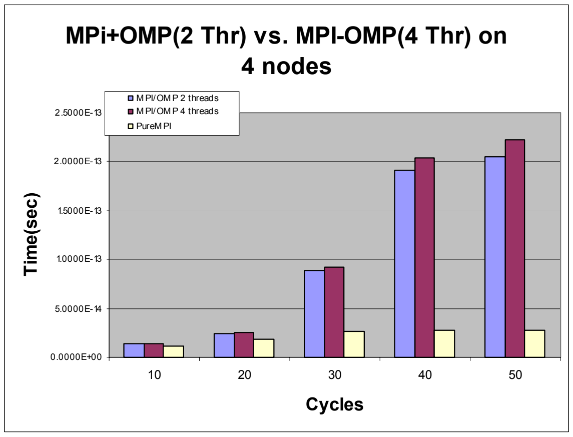
These graphs shows that the performance of IRS improves as you increase the number of nodes when keeping the threads count the same.

When the number of threads is increased while keeping the number of nodes constant at 4 the performance degrades from pure MPI. But when the number of nodes is held constant at 2 or 8 the performance degrades for 2 threads, and equal to the optimal for 4 threads.

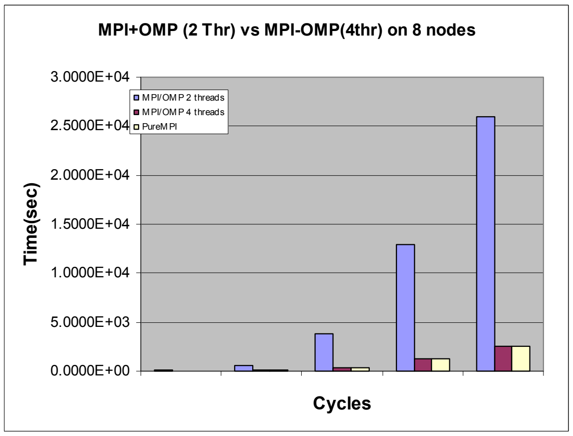
A single thread works the best for this application, on 8 nodes. This is the expected result since the zrad.0008 is optimized for this case.



Graph 7



Graph 8



Graph 9

MPIP TESTING

The same tests mentioned in the IRS Benchmark testing section were completed for MPIP to determine where the major bottlenecks in the program were.   The MPIP portion of this report is included in the file mpip.pdf

References

[1] **mpiP: Lightweight, Scalable MPI Profiling**
http://www.llnl.gov/CASC/mpip/

[2] **The IRS Benchmark Code**
http://www.llnl.gov/asci/purple/benchmarks/limited/irs/

[3] **Parallel Implicit Solvers for Radiation Transport Systems**
http://research.nianet.org/~dimitri/ASCI/

[4] **The download and reference page for MeshTV/Silo**
http://www.llnl.gov/meshtv/

[4] **Statistical Scalability Analysis of Communication Operations in Distributed** Applications, Jeffrey S. Vetter, Michael O. McCracken,Proc.

[5] **ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPOPP, 2001)**
http://llnl.gov/CASC/people/vetter/people/pubs/ppopp01_scal_analysis.pdf

[6] **An Empiracal Performance Evaluation of Scalable Scientific Applications, Jeffrey S. Vetter,Andy Yoo,Supercomputing Conf. Tech Paper(2002).**
http://sc-2002.org/paperpdfs/pap.pap222.pdf

Project Web Page

http://www4.ncsu.edu/~aredward/csc591c/index.html