

Analysis of OpenMP and MPI Codes on Sun Fire Systems

Larry Meadows, Myungho Lee, Dominic Paulraj, Sanjay Goil
Compiler Performance Engineering Group

Brian Whitney
Strategic Application Engineering Group
Sun Microsystems, Inc.

Abstract

The paper presents the results of running the SPEC HPC benchmarks on several different Sun Fire systems with different interconnects.

1. Introduction

Sun's current product line includes workgroup, mid-size, and large servers, all based on the UltraSPARC III Cu processor architecture and the Fireplane interconnect. Additionally, these systems may be connected using several different network technologies.

The SPEC HPC benchmarks are the first application-level standard benchmarks to employ different parallel models: MPI, OpenMP, and mixed, or hybrid, MPI and OpenMP parallelism. These benchmarks provide a useful way to study the performance of different systems with different interconnects.

The rest of the paper is organized as follows. Section 2 gives an overview of the different Sun Fire systems and provides some details on memory and interconnect latency and bandwidth, as well as the software used. Section 3 describes the three SPEC HPC benchmarks, how they are parallelized, and their workload characteristics. Section 4 presents results on the various systems. Section 5 gives conclusions and opportunities for future work.

2. System Description

Three different Sun Fire systems were used to gather the results in this paper.

System 1	72-processor Sun Fire 15K with 900MHz Ultrasparc III Cu cpus and 144GB of physical memory
System 2	8, 8-processor Sun Fire V880 with 1050MHz Ultrasparc III Cu cpus and 16GB of physical memory, interconnected with Myrinet
System 3	3, 24-processor Sun Fire 6800 with 900MHz Ultrasparc III Cu cpus and 96GB of physical memory, interconnected with Sun Fire Link

Table 1 shows MPI latencies and bandwidths for each system. Table 2 shows the memory

latencies and bandwidths. Memory latencies were measured using `lmbench lat_mem_rd()`. Memory bandwidths were measured using the Stream COPY benchmark (the number of processors used for STREAM is limited by the number of processors in the machine). MPI latencies and bandwidths were measured using a Fortran program that times unidirectional MPI send and receive calls.

System	Interconnect	MPI Latency (usec)	MPI Bandwidth (MB/Sec)
System 1	Backplane	4.1	600
System 2	Myrinet	14	130
System 3	Sunfire Link	5.5	520

	System 1	System 2	System 3
Memory Latency (ns)			
Local Access	250	205	230
Remote Access	470	205	270
Memory BW (MB/Sec,N cpus)			
1	2013	1651	2356
4	3294	5589	4491
8	6525	8490	6934
16	12909		9239
24			9357
32	25889		

Sun One Studio 8 compilers were used to compile the codes. The Sun One Studio 8 performance analyzer was used in performance analysis. Sun HPC Clustertools 5.0 was used to run the MPI codes. The Solaris 9 operating environment was used on all systems.

3. SPEC HPC Benchmark Description and Analysis

This paper considers the results of running the SPEC HPC2002 medium data sets on the three benchmarks: CHEM2002, ENV2002, and SEIS2002. These benchmarks are referred to in this paper as Gamess, Wrf, and Seis, after the codes upon which they are based. The remainder of this section describes each benchmark, its parallelization strategies, how it is parallelized, and baseline performance characteristics.

3.1. Gamess

Gamess is a quantum chemistry code. It can be run in four different modes: serial, OpenMP, MPI, and hybrid MPI/OpenMP. The MPI parallelization uses a master-slave model. A single MPI process divides the work (a set of loop iterations) into pieces that are sent to each slave process. Each slave process then does the work.

This is an outline of the processing in the master process:

```
while (num_iter > 0) {
    MPI_RECV(recpt);    ! Recpt asks for work.
    chunk = num_iter / (num_proc*2); ! Compute the chunk size
    work[2] = work[1] + chunk; ! Compute the iteration range
    num_iter -= chunk+1; ! Reduce the number of iterations
    MPI_SEND(work, recpt);    ! Send the work to Recpt
    work[1] = work[2] + 1;
}
```

While there is work to be done, the master process waits for a slave process to request work; it then divides the remaining work by $2*N$ (where N is the number of slave processes) and sends that work to the slave process.

The slave processes repeatedly request work and perform calculations, then use a global reduction to assemble their results. The master process does not participate in the global reduction. This is an outline of this processing phase in the slave processes:

```
while (TRUE) {
    lp_next(ist, iend);
    if (ist == -1) break;
    for (i = ist; i <= iend; ++i)
        compute for iteration i;
}
mpi_barrier();
mpi_reduce();
mpi_broadcast();
```

On a sample run of 32 MPI processes, the MPI time was 37% of the total wall-clock time. About 18% of the total time was spent in MPI reductions, 14% in MPI barrier, and 5% in MPI broadcast and MPI receive. There is also a large variance in time spent in MPI barrier; this indicates a load imbalance, which is graphically visible when using the analyzer timeline.

The OpenMP parallelization is mostly done at the loop level. There are two major OpenMP parallel loops; in a 20 processor run they account for 66% of the execution time. They are both written using OpenMP dynamic scheduling, for example:

```
c$omp do schedule(dynamic, 20)
```

Hybrid parallelism is accomplished by combining the MPI and OpenMP models, so it is possible to analyze this code by combining the analyses for MPI and OpenMP.

3.2.Wrf

Wrf is a mesoscale weather model. Like Gamess it can be run in four different modes. The MPI parallelization is done using a two dimensional data decomposition. Boundary data is exchanged in a stencil pattern using a generalized stencil package from Argonne National Labs. Each MPI process then operates on its local data. The stencil package uses asynchronous messages to

exchange the boundary elements followed by MPI wait calls to wait for the message passing to complete.

In a 20 processor run, approximately 15% of the total wall-clock time was spent in MPI. 12.5% of the total time was spent in exchanging data. The remainder of the MPI time was spent in broadcasting initialization data at the beginning of the simulation.

The OpenMP parallelism concentrates on the solve_em routine. The following information is based on a 20-cpu OpenMP run. 99.5% of the user cpu cycles are spent in this routine; 91% of the user cpu cycles are spent in parallel loops in this routine. However, 14.8% of total CPU cycles are wasted by threads that complete their iterations and must wait in a barrier until the remaining threads complete; this load imbalance limits scaling.

Like Gamess, hybrid parallelization simply combines the MPI and OpenMP models.

3.3.Seis

Seis is a seismic benchmark. Unlike Gamess and Wrf it cannot be run in hybrid mode; the available modes are serial, MPI, and OpenMP. The same executable is run four times, with four different data sets, resulting in widely different profiles for the different runs.

The first and fourth runs account for the bulk of the time in both MPI (93%) and OpenMP (97%) so this analysis ignores the second and third runs. While those runs might become important for large processor counts, the current implementation of Seis does not work properly for more than 32 cpus.

The MPI version of the first run is more than four times slower than the OpenMP version on 16 cpus. Inspection of the timeline shows that virtually all of the time is spent in the read() system call. It turns out that every MPI process reads the same file at the same time. This appears to be causing a bottleneck in the operating system, but time did not permit further investigation. Therefore, the remainder of the seis analysis is focused on the fourth run.

For the code executed in the fourth run, the parallelism for both MPI and OpenMP is at the same level. The OpenMP is written in a SPMD fashion with private variables used for most computation. Data is moved between threads using an explicitly allocated shared buffer. For MPI, data is moved between processes using message passing. Thus, the performance of the fourth run of Seis provides a way to compare SPMD shared-memory programming with MPI programming on a single SMP.

For the MPI version of the fourth run, about 6% of the time is spent in MPI on 16 processors. Almost all of this time is spent in point-to-point messages.

4. Performance Measurements and Results

All performance measurements are based on wall-clock time. So that systems of different clock-rates can be compared, all times are first normalized to a 900MHz clock. Then, times are divided into the time for the serial version of the benchmark as measured on a 900MHz V880. Thus, reported numbers are speedups versus the serial version.

4.1. Gamess

Table 3 shows the performance of Gamess on various systems in the three modes OpenMP, MPI, and hybrid.

Table 3: Gamess MPI and OpenMP Performance

	64	32	16	8
System 1/OpenMP	13.0	12.8	11.3	5.4
System 2/OpenMP				5.9
System 3/OpenMP			9.8	5.8
System 1/MPI	23.6	18.3	11.3	6.1
System 2/MPI	23.6	18.3	11.1	
System 3/MPI	20.8			

OpenMP can run only on a single node, so not all processor combinations were possible on all systems. Time did not allow running all MPI combinations.

MPI scales much better than OpenMP. This is due to the level of parallelism. The OpenMP parallelism is at the loop level, so the amount of work per processor is limited by the number of loop iterations.

The lower memory latency of System 2 compensates for the higher latency of the Myrinet interconnect, so Systems 1 and 2 perform virtually the same. System 3 is worse in both MPI and OpenMP, perhaps because of the limited memory bandwidth compared to System 1.

Table 4 shows hybrid performance for various combinations of M processes x N threads.

Table 4: Gamess Hybrid Performance

	64		32			16			8
	1+16x4	8x8	1+8x4	8x4	4x8	1+4x4	8x2	4x4	4x2
System 1	35.9	31.0	21.4	18.3	15.2	11.8	10.5		4.9
System 2		33.3		19.1	16.5		10.3	9.1	4.9
System 3	32.7		20.8						

The columns labelled MxN are the result of running M processes with N threads each using the Sun Clustertools mprun command. The columns labelled 1+Nx4 are the result of running N processes with 4 threads each plus one extra process as the master process (see Section 3.1). Also, in the 1+Nx4 case, the 4 threads in each worker process were bound to a 4-processor Uniboard.

The best performance is achieved in hybrid mode for Gamess. This is probably because fewer MPI processes lead to less load imbalance, and the OpenMP parallelism scales well for small numbers of threads. Reserving an additional thread for the master process and binding to Uniboards adds additional performance. Again, the limited memory bandwidth of System 3 appears to be a bottleneck; however, the Sun Fire Link allows the three-node system to perform at close to the level of the large shared-memory 15K.

4.2.Wrf

Table 5 shows the performance of Wrf with MPI and OpenMP. Wrf does not scale well with OpenMP, again because of the loop-level parallelism. All systems scale reasonably well up to 32 nodes. The slowdown on System 3 for 64 processes is probably due to memory bandwidth and system activity (mprun used 24, 24, and 16 cpus in this case).

Table 5: Wrf MPI and OpenMP Performance

	64	32	16	8
System 1/OpenMP	11.4	9.9	8.0	6.1
System 2/OpenMP				6.2
System 3/OpenMP			10.9	7.1
System 1/MPI	33.7	23.1	11.5	6.4
System 2/MPI	35.9	22.6	12.1	
System 3/MPI	24.2	22.8	12.1	7.1

Table 6 shows the performance of Wrf in hybrid mode.

Table 6: Wrf Hybrid Performance

	64		32			16			8	
	16x4*	8x8	8x4*	8x4	4x8	4x4*	8x2	4x4	2x4*	4x2
System 1	30.5	19.6	20.6	18.2	14.4	11.6	11.4	10.4	7.4	6.7
System 2		24.7		18.8	16.5		12.1	11.1		6.4
System 3	27.3	25.0	20.5	19.4	12.7	11.4	12.1	11.1		

The columns labelled Nx4* use binding to Uniboards. Hybrid parallelism does seem to improve 64-node performance for System 3, but this may be due to process binding. The MPI decomposition is well balanced, so increasing the work done by a given MPI process does not help performance, and the additional overhead of OpenMP actually decreases performance in most cases. Binding to Uniboards does help.

4.3.Seis

Recall that only the fourth dataset of the Seis benchmark was run. Since Seis is not a hybrid code, different configurations of MPI processes were used on Systems 2 and 3 to explore the effects of the interconnect. The OpenMP data is presented in Table 7, and the MPI data in Table 8.

Table 7: Seis4 OpenMP Performance

	64	32	16	8
System 1	54.2	28.5	14.5	7.5
System 2				7.5
System 3			14.4	7.5

Table 8: Seis4 MPI Performance, System 1

	64	32	16		8	
System 1	27.9	25.1		14.3		7.5
System 2		4x8	2x8	4x4	2x4	4x2
		20.0	13.7	13.6	7.3	7.4
System 3		2x16	2x8	1x16	2x4	1x8
		25.7	13.8	14.2	7.3	7.6

The notation NxM in Table 8 means N nodes with M mpi processes on each node. OpenMP scales better than MPI, probably because OpenMP has less overhead (data copies rather than message passing). The configuration of MPI processes doesn't matter much, indicating that the interconnects are very efficient.

5. Conclusions and Future Work

Both Myrinet and Sun Fire Link are very efficient interconnects. The V880 cluster performs very well on MPI and hybrid codes; however, if large SMP systems are required the SF15K is a good candidate. The 6800 cluster with Sun Fire Link is a good mixture of relatively fat nodes and high bandwidth, low latency interconnect, but the limited memory bandwidth may reduce performance of bandwidth intensive problems.

More study is required to find out exactly why Gamess performs better in hybrid mode but Wrf does not. The problem with Seis dataset 1 needs to be investigated so the full Seis performance can be presented. The bandwidth requirements of the codes need to be investigated so that performance on the 6800 can be better explained. The behavior of lower-performance interconnects like Gigabit Ethernet should be explored.

References

1. Sun ONE Studio 8 Compiler Collection(tm)
<http://developers.sun.com/prodtech/cc/reference/docs/index.html>
2. Nawal Copt, Marty Itzkowitz, and Larry Meadows, "OpenMP Support in Sun's SPARC Compilers and Tools", SunTech 2001.
3. Sun Fire 6800 Server, <http://www.sun.com/servers/midrange/sunfire6800/index.html>
4. Sun Fire 15K server, <http://www.sun.com/servers/highend/sunfire15k/index.xml>
5. Sun Fire V880 Server, <http://www.sun.com/servers/entry/880/index.html>
6. Solaris 9 Operating System, <http://www.sun.com/software/solaris>
7. HPC Clustertools 5, <http://www.sun.com/servers/hpc/software/specifications.html>
8. Matthijs Muller, et.al., "SPEC HPG Benchmarks for Large Systems"
9. Barbara Perz, Ron Larson, "Technical Cluster Grid Performance", SUPerG, Spring, 2003.