Performance Analysis and Comparison of MPI, OpenMP and Hybrid NPB -MZ

Héctor J. Machín Machín

Abstract—Chip multiprocessors (CMP) are widely used for high performance computing and are being configured in a hierarchical manner to compose a node in a parallel system. CMP clusters provide a natural programming paradigm for hybrid programs. Can current hybrid parallel programming paradigms such as hybrid MPI/OpenMP efficiently exploit the potential offered by such CMP clusters? In this research, with increasing the number of processors and problem sizes, we systematically analyze and compare the performance of MPI, OpenMP and hybrid NAS Parallel Benchmark Multi–Zone (NPB–MZ) on two supercomputers: DataStar p655 at San Diego Supercomputer Center (SDSC) and Hydra at Texas A&M Supercomputing Facilities to address the question. We also upload the performance data of NPB-MZ to Prophesy database and use Prophesy system to model the performance online.

Index Terms—Performance analysis, MPI (Message Passing Interface), OpenMP (Open Multi-Processing), NPB-MZ (NAS Parallel Benchmark Multi-Zone), MCM (Multi-Chip-Module), DCM (Dual-Chip-Module).

1 Introduction

Chip multiprocessors (CMP) are widely used for high performance computing and are being configured in a hierarchical manner to compose a node in a parallel system. CMP clusters provide a natural programming paradigm for hybrid programs. Through this paper we are going to use NAS Parallel Benchmarks with Multi-Zone executed on two supercomputer systems (DataStar p655 and Hydra) to find out if their potentials of the CMP clusters can be exploit by hybrid parallel programming paradigms such as hybrid MPI/OpenMP.

The remainder of this paper is organized as follows. Section 2 describes two supercomputing systems we used. Section 3 depicts NAS parallel benchmarks with multi-zone (NPB-MZ). Section 4 analyzes the performance of the NPB-MZ on two CMP clusters. Section 5 discusses processor partitioning. Section 6 presents performance modeling results, and concludes the paper in Section 7.

Fig. 1. DataStar p655

Table 1. DataStar p655 specifications

DataStar p655	
Hostname	dslogin.sdsc.edu
Manufacturer	IBM
CPU type	1.5, 1.7GHz Power4+ p655
Operating System	AIX 5.2L
Processors per Node	8
Nodes	272
Memory per Node	16, 32GB

2 Platforms

2.1 Datastar p655

The San Diego Supercomputer Center DataStar p655 [1] is primarily intended to run applications of very high levels of parallelism or concurrency, especially thosee with high parallel I/O requirements.

DataStar has 272 8-way P655+ compute nodes -- 176 nodes with 1.5-GHz Power4+ CPUs and 16 GB of memory, and 96 with 1.7 GHz Power4+ CPUs and 32 GB of memory. The nodes are connected via IBM's high-

• Héctor Machín is with the REU Summer 2008 at Texas A&M CS Department. Email: hjmachin@cs.tamu.edu.

speed Federation switch and have access to 130 TB of GPFS [1].

2.1.1 Power4 Chip

The functional unit of the POWER4 consists of two 64-bit implementations of the PowerPC AS Architecture. The

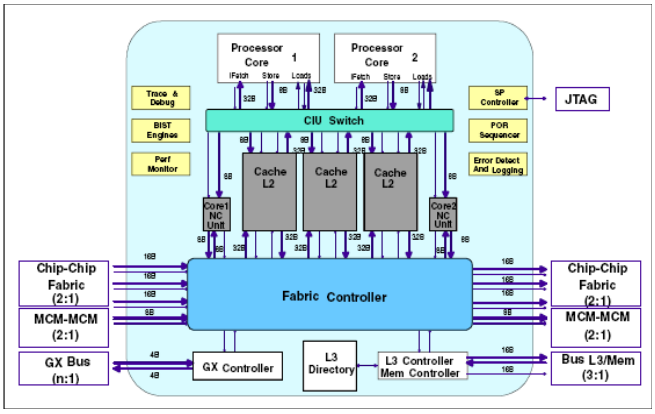


Fig. 2. Power4 Chip

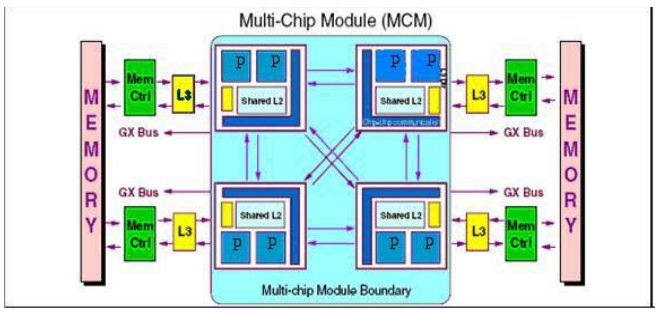


Fig. 3. Power4 MCM (Multi-Chip-Module), a p655 node

POWER4 has an L2 unified cache, divided into three equal parts. Each has its own independent L2 controller which can feed 32 bytes of data per cycle. The Core Interface Unit (CIU) connects each L2 controller to either the data cache or instruction cache in either of the two processors. The Non-Cacheable (NC) Unit is responsible for handling instruction serializing functions and performing any noncacheable operations in the storage topology. There is an L3 cache controller, but the actual memory is off-chip. The GX bus controller controls I/O device communications, and there are two 4-byte wide GX buses, one incoming and the other outgoing. The Fabric Controller is the master controller for the network of buses, controlling communications for both L1/L2 controllers, communications between POWER4 chips {4-way, 8-way, 16-way, 32-way} and POWER4 MCM's [1].

2.2 Hydra

The Texas A&M Hydra is a high-performance "IBM cluster 1600", based on IBM's Power5+ processor. The cluster consists of 40 p5-575 nodes, each having 16 Power5+ processors running at 1.9GHz and 32 GBytes of DDR2 DRAM. A p5-575 node, is a high-performance, Shared-Memory multi-processor (SMP), running the 64-bit version of AIX 5L (5.3) as a single system image [2].

Fig. 4. Hydra

Table 2. Hydra specifications

Hydra	
Hostname	hydra.tamu.edu
Manufacturer	IBM
CPU type	1.9GHz Power5+
Operating System	AIX 5.3L
Processors per Node	16
Nodes	40
Memory per Node	32GB

2.2.1 Power5 Chip

POWER5 is a microprocessor developed by IBM. It is an improved variant of the highly successful POWER4. The principal changes are support for Simultaneous multithreading (SMT) and an on-die memory controller. Each CPU supports 2 threads; since it is a multicore chip, with 2 physical CPUs, each chip supports 4 logical threads. The POWER5 can be packaged in a DCM (dual chip module), with one dual core chip per module, or an MCM with 4 dual core chips per module. POWER5+ (presented on 3Q 2005) packages in QCM, 2 dual core chips [2].

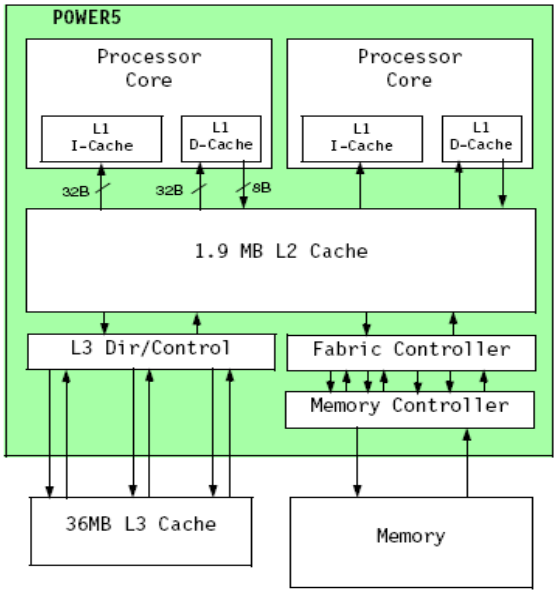


Fig. 5. Power5 Chip

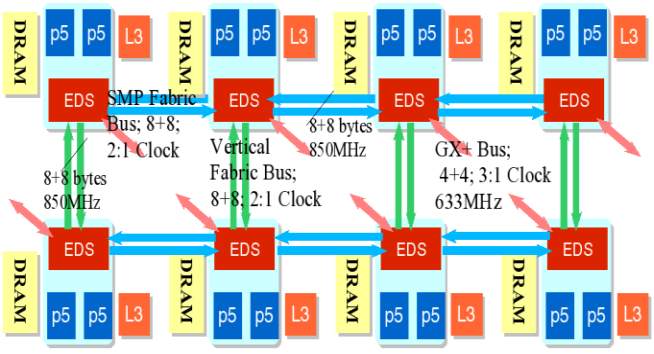


Fig. 6. A 16-way p5-575 node (8 DCMs with 2 Power5+ cores per DCM)

3 NAS Parallel Benchmarks – Multi Zone (NPB – MZ)

3.1 Introduction

For having a better understanding of these benchmarks we are going to provide a background in the following steps:

- Message Passing Interface (MPI)
- OpenMP
- Nas Parallel Benchmaks (NPB)

After this background we intend to go in more detail about NBP-MZ.

3.1.1 Message Passing Interface (MPI)

In the MPI programming model, a computation comprises one or more processes that communicate by calling library routines to send and receive messages to other processes. In most MPI implementations, a fixed set of processes is created at program initialization, and one process is created per processor. However, these processes may execute different programs. Hence, the MPI programming model is sometimes referred to as multiple program multiple data (MPMD) to distinguish it from the SPMD model in which every processor executes the same program [3].

3.1.2 OpenMP

OpenMP is an implementation of multithreading, a method of parallelization whereby the master "thread" (a series of instructions executed consecutively) "forks" a specified number of slave "threads" and a task is divided among them. The threads then run concurrently, with the runtime environment allocating threads to different processors.

By default, each thread executes the parallelized section of code independently. "Work-sharing constructs" can be used to divide a task among the threads so that each thread executes its allocated part of the code. Both Task parallelism and Data parallelism can be achieved using OpenMP in this way [4].

3.1.3 NAS Parallel Benchmarks (NPB)

NAS Parallel Benchmars has been developed for the performance evaluation of highly parallel supercomputers. These benchmarks consist of five parallel kernels and three simulated application benchmarks. Together they mimic the computation and data movement characteristics of large scale computational fluid dynamics (CFD) application.

The principal distinguishing feature of these benchmarks is their "pencil and paper" specification - all details of these benchmarks are specified only algorithmically. In this way many of the difficulties associated with conventional benchmarking approaches on highly parallel systems are avoided [5].

NPB are well-known problems for testing the capabilities of parallel computers and parallelization tools. They exhibit mostly fine-grain exploitable parallelism and are almost all iterative, requiring multiple data exchanges between processes within each iteration. However, many important scientific problems feature several levels of parallelism, and this property is not reflected in NPB. To remedy this deficiency the NPB Multi-Zone (NPB-MZ) versions were created [6,7].

3.2 NPB - MZ

To mimic NPB applications, the NPB Multi-Zone (NPB-MZ) versions were created, which contain three families of multi-zone benchmarks, derived from the NPB. These multi-zone benchmarks stress the need to exploit both levels of parallelism for efficiency and to balance the computational load [6].

Problem sizes and verification values are given for benchmark classes S, W, A, B, C, and D.

Table 3. Aggregate problem size and the number of zones for each problem class. *Gx*, *Gy*, and *Gz* are aggregate spatial dimensions.

Class	Aggregate Size	Zones (*x-zones* × *y-zones*)		
	$(G_x \times G_y \times G_z)$	LU-MZ	SP-MZ	BT-MZ
S	$24 \times 24 \times 6$	4×4	2×2	2×2
W	$64 \times 64 \times 8$	4×4	4×4	4×4
A	$128 \times 128 \times 16$	4×4	4×4	4×4
B	$304 \times 208 \times 17$	4×4	8×8	8×8
C	$480 \times 320 \times 28$	4×4	16×16	16×16
D	$1632 \times 1216 \times 34$	4×4	32×32	32×32

The application benchmarks Lower-Upper Symmetric Gauss-Seidel (LU), Scalar Penta-diagonal(SP), and Block Tri-diagonal (BT) solve discretized versions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions. Each operates on a structured discretization mesh that is a logical cube. In realistic applications, however, a single such mesh is often not sufficient to describe a complex domain, and multiple meshes or zones are used to cover it [6].

3.2.1 Serial Implementation

The serial implementation of NPB – MZ starts with the original single-zone problem of LU, SP, and BT being subdivided into multiple zones. Then solutions for each zone are then initialized. The benchmarking loop starts with a time step loop which contains a procedure to exchange boundary values of different zones. Then discrete partial differential equation solvers LU, SP, and BT are used for obtaining solution updates within each zone in the new LU-MZ, SP-MZ, and BT-MZ, respectively. The solving stage includes procedures for performing forcing term (right-hand-side) calculations and the Lower-Upper (for LU-MZ) or Alternative Directional Implicit (for SPMZ and BT-MZ) algorithm. Finally solution is then verified for all zones for a given problem class.

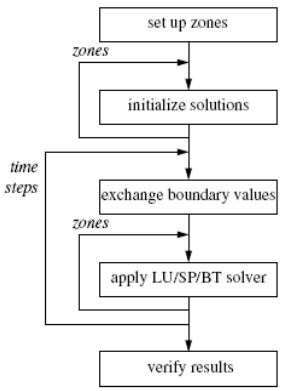


Fig 7. Schematic flow graph of the multizone benchmarks in sequential execution. Loops (back arrows) are annotated with their induction variable.

3.2.2 Hybrid Implementation (MPI + OpenMP)

As clusters of symmetric multiprocessor machines have become popular, more and more applications take advantage of the hardware architecture by using the hybrid programming model which uses MPI for communication between symmetric multi-processor nodes and OpenMP for parallelization within one node.

The MPI+OpenMP implementation of the multi-zone benchmarks starts by defining the number of MPI processes at compilation time in order to avoid dynamic memory allocation. Then each process is first assigned with a group of zones and a given number of OpenMP threads. There is no dynamic load adjustment at runtime. As in the sequential version, solutions for the zones assigned to each process are then initialized, followed by the time step loop. There is no communication during the LU, SP, or BT solving stage. Finnally the last stage (verification) performs a reduction of solutions and residues from all zones for a given problem class [7].

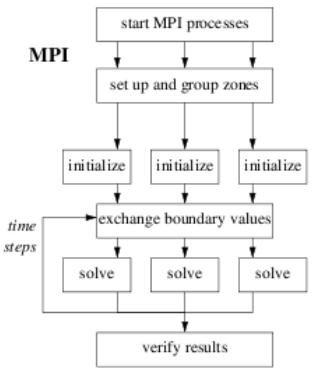


Fig 8. Coarse grained parallelization with zone groups for the multi-zone benchmarks using MPI and OpenMP.

4 Scalability Analysis

Through the quest of comparing the performance of MPI, OpenMP and the Hybrid NPB-MZ over CMP clusters, we developed a scalability analysis wich consists:

- Compiling all benchmarks from 1 to 1024 processors (DataStar p655) and from 1 to 128 (Hydra).
- Running jobs for MPI, OpenMP and Hybrid NPB-MZ increasing each run the number of processors with a fixed problem size.
- Running jobs for MPI, OpenMP and Hybrid NPB-MZ increasing problem size fixed number of processors.
- Collecting the execution time from the output files and make a detailed comparison for each programming paradigm.

4.1 Description

Each benchmark was compiled for problem classes: A, B, C, D and E. Then they where runned for MPI, using the Load Level (LL) submit file, and the collected execution time results until now are shown in table 4. The results that are not shown in this table are still waiting in queues in both DataStarp655 and Hydra.

Table 4. MPI Run Results Class A

	Problem Class: A					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
1	215.63	203.53	185.1	180.25	101.14	99.43
2	104.28	98.28	93.34	88.79	51.59	46.22
4	53.29	49.05	46.89	45.23	32.35	23.08
8	51.6	41.54	24.25	22.23	17.69	12.22
16	51.44	41.43	11.53	9.81	8.94	6.45
32	X	X	X	X	X	X
64	X	X	X	X	X	X
128	X	X	X	X	X	X
256	X	X	X	X	X	X
512	X	X	X	X	X	X
1024	X	X	X	X	X	X

Table 5. MPI Run Results Class B

	Problem Class: B					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
1	961.35	942.31	739.36	728.75	574.59	477.11
2	432.88	398.2	329.31	325.46	297.59	239.12
4	247.21	169.45	174.79	169.24	159.63	112.5
8	128.14	100.97	102.75	95.3	97.9	66.13
16	64.49	52.91	42.91	41.51	48.25	30.7
32	51.62	48.61	26.04	21.62	X	X
64	51.29	48.57	13.21	10.86	X	X
128	X	X	X	X	X	X
256	X	X	X	X	X	X
512	X	X	X	X	X	X
1024	X	X	X	X	X	X

Table 6. MPI Run Results Class C

	Problem Class: C					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
1	3551.16	3276.65	3613.48	3348.87	2852.55	2052.44
2	1749.48	1690.92	1834.34	1684.01	1506.59	1026.08
4	1005.26	808.08	892.98	728.67	734.47	513.72
8	515.5	421.94	458.54	349.45	450.79	283.51
16	259.5	203.75	188.88	175.71	250.14	148.72
32	114.65	103.5	105.25	100.73	X	X
64	59.91	54.79	52.06	44.29	X	X
128	58.9	51.01	26.41	25.73	X	X
256	54.04	X	14.65	X	X	X
512	X	X	X	X	X	X
1024	X	X	X	X	X	X

Table 7. MPI Run Results Class D

	Problem Class: D					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
1	X	X	X	X	X	X
2	X	X	X	X	X	X
4	X	X	X	X	X	X
8	9120.09	8413.51	9392.26	8278.24	12549.21	10782.45
16	4674.1	4217.35	4570.03	4368.73	6563.82	5987.29
32	2627.06	2399.75	2262.49	1871.73	X	X
64	1337.17	1070.64	1139.11	939.62	X	X
128	688.42	544.73	573.58	533.51	X	X
256	392.34	X	396.77	X	X	X
512	334.68	X	190.23	X	X	X
1024	334.55	X	74.36	X	X	X

Table 8. MPI Run Results Class E

	Problem Class: E			
	BT-MZ		SP-MZ	
# procs	DataStar	Hydra	DataStar	Hydra
1	X	X	X	X
2	X	X	X	X
4	X	X	X	X
8	X	X	X	X
16	X	X	X	X
32	X	X	X	X
64	X	X	X	X
128	13586.45	12455.78	12874.35	11862.16
256	7215.32	X	6384.99	X
512	3717.34	X	3535.82	X
1024	1966.09	X	1786.78	X

Theses tables 4, 5, 6, 7 and 8 show all the results collected until now. The Xs in a specific table space means that the program cannot run for the specific problem class and number of processors. For example the benchmark LU-MZ can only run for up to 16 processors for any problem class. Problem classes D, and E restricts where the number of processors should begin, for D is 8 and for E is 128. But like we mentioned LU-MZ can only run up to 16 processors it is needless to mention this benchmark in the table for problem class E since it would not have any collectable results. Also it is important to remind the reader that the limit of processor number in Hydra is 128 processors.

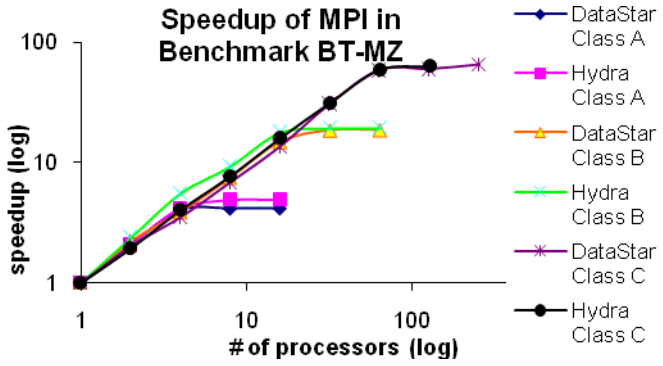


Fig 8. Graph for speedup for benchmark BT-MZ, for problem classes A-C, from 1 up to 256 processors for MPI code.

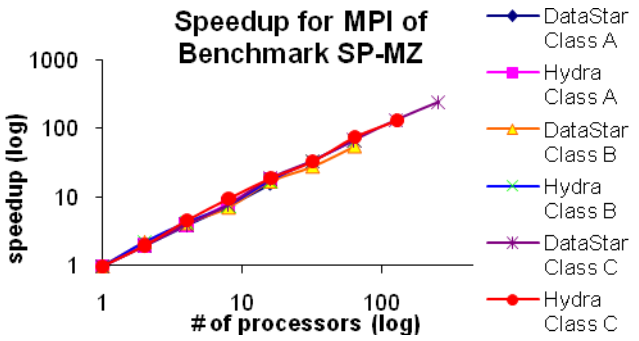


Fig 9. Graph for speedup for benchmark SP-MZ, for problem classes A-C, from 1 up to 256 processors for MPI code.

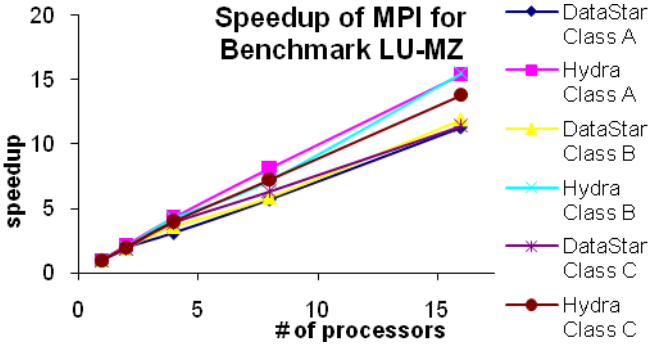


Fig 10. Graph for speedup for benchmark LU-MZ, for problem classes A-C, from 1 up to 16 processors for MPI code.

The figures 8, 9 and 10 show the speed-up for MPI code for problem classes A, B and C. It is notably show that the speed-up of Hydra exceeds the speed-up of DataStar. This implicitly shows that the execution time goes down quicklier in Hydra than in DataStar p655. The reason for this is the difference in CPU type, Hydra is a Power5 where DataStar p655 is a Power4. Figures 8 and 9 axis where put on logarithm base to show all points with equal distance. The figure 8 also shows at what point the speedup does not increase more.

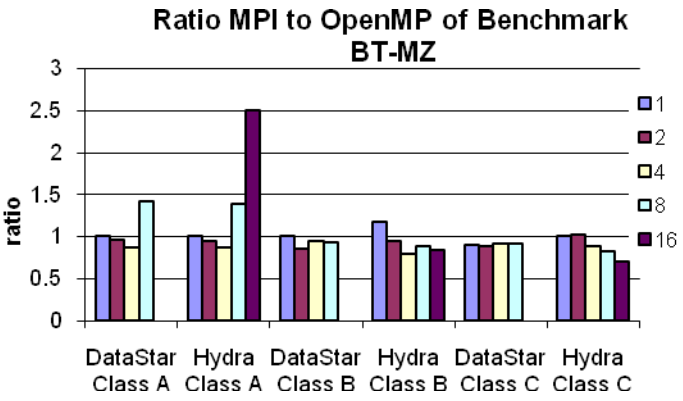


Fig 11. Graph for ratio MPI to OpenMP, benchmark BT-MZ.

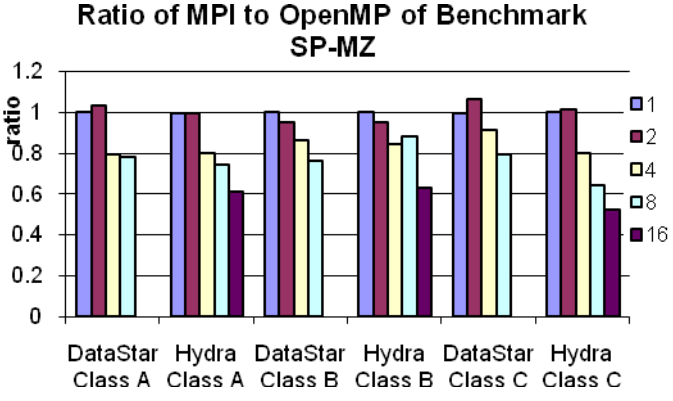


Fig 12. Graph for ratio MPI to OpenMP, benchmark SP-MZ.

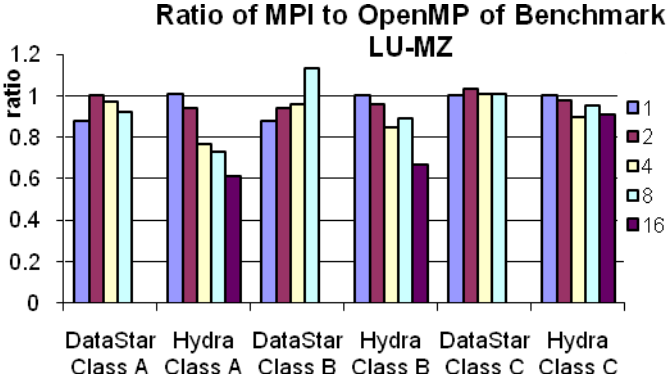


Fig 13. Graph for ratio MPI to OpenMP, benchmark LU-MZ.

Table 9. OpenMP Run Results Class A

	Problem Class: A					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
1	215.03	203.16	185.85	181.75	114.37	98.12
2	108.25	103.07	90.48	89.9	51.71	49.08
4	61.29	56.54	59.46	56.2	33.48	29.86
8	36.45	29.78	31.04	29.88	19.17	16.72
16	X	16.61	X	16.07	X	10.51

Table 10. OpenMP Run Results Class B

	Problem Class: B					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
1	953.76	795.92	738.9	730.76	649.45	476.3
2	504.26	423.16	345.16	342.25	316.22	248.74
4	261.83	215.29	204.28	202.62	166.65	131.96
8	138.09	114.17	135.66	107.79	86.42	74.08
16	X	62.73	X	65.97	X	45.81

Table 11. OpenMP Run Results Class C

	Problem Class: C					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
1	3944.79	3273.82	3641.35	3346.65	2858.75	2053.49
2	1995.21	1659.13	1735.81	1665.93	1469.79	1044.52
4	1087.56	916.66	979.24	913.71	728.82	573.43
8	568.6	516.87	579.45	547.07	448.31	299
16	X	289.15	X	339.21	X	162.72

The Tables 9, 10 and 11 show the execution time results for OpenMP runs. To begin the comparison between the programming paradigms, we made a direct comparison between the execution time of MPI and OpenMP. The ratio from MPI to OpenMP is the execution time of MPI over the execution time of OpenMP. The figures 11, 12 and 13 show us that for most of the cases MPI is faster than OpenMP, that's why the ratio in most cases is below 1. Before making more experiments we can suggest another question, is Message Passing programming model more efficient than Shared Memory?

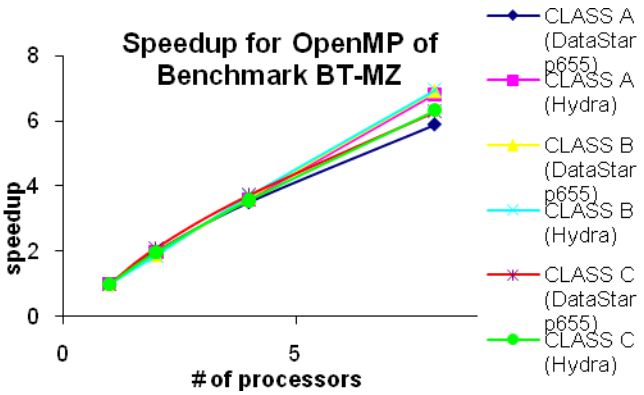


Fig 14. Graph for speed-up for benchmark BT-MZ, for problem classes A, B and C, from 1 up to 8 processors for OpenMP code.

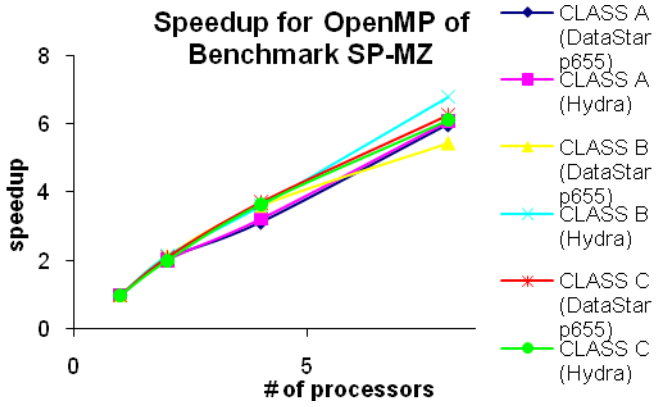


Fig 15. Graph for speed-up for benchmark SP-MZ, for problem classes A, B and C, from 1 up to 8 processors for OpenMP code.

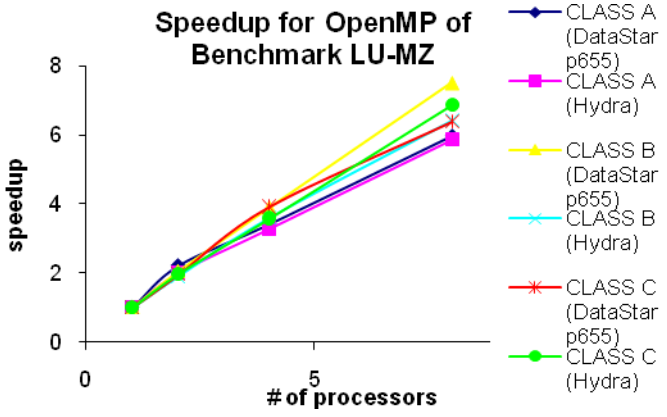


Fig 16. Graph for speed-up for benchmark LU-MZ, for all problem classes A, B and C from 1 up to 8 processors for OpenMP code.

OpenMP for problem classes A, B and C. This states the as the speed-up graphs for MPI that Hydra is faster than DataStar p655. OpenMP is code is runned for this problem classes because it can only run on one node so the maximum number of processors is 8 for DataStar p655 and 16 for Hydra.

Tables 12, 13, 14, 15 and 16 show the execution results for Hybrid (MPI/OpenMP) runs. To follow up the comparison of the programming paradigms we now compare the Hybrid with MPI. In figures 17 and 18 shows the ratio from MPIto Hybrid. This is the execution time of MPI over the execution time of Hybrid. The figures show explicitly that Hybrid in most cases is faster than MPI. It is very important to remark some results that states that Hybrid is two and three times faster than MPI. There are no speed-up graphs for Hybrid code because it does not have a sequential basis execution time to develop it.

Table 12. Hybrid Run Results Class A

	Problem Class: A					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
16	17.65	X	14.01	X	9.86	X
32	X	X	X	X	X	X
64	X	X	X	X	X	X
128	X	X	X	X	X	X
256	X	X	X	X	X	X
512	X	X	X	X	X	X
1024	X	X	X	X	X	X

Table 13. Hybrid Run Results Class B

	Problem Class: B					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
16	17.63	X	48.37	X	42.98	X
32	30.56	31.19	28.16	30.39	X	X
64	18.01	16.38	12.55	15.31	X	X
128	X	X	X	X	X	X
256	X	X	X	X	X	X
512	X	X	X	X	X	X
1024	X	X	X	X	X	X

Table 14. Hybrid Run Results Class C

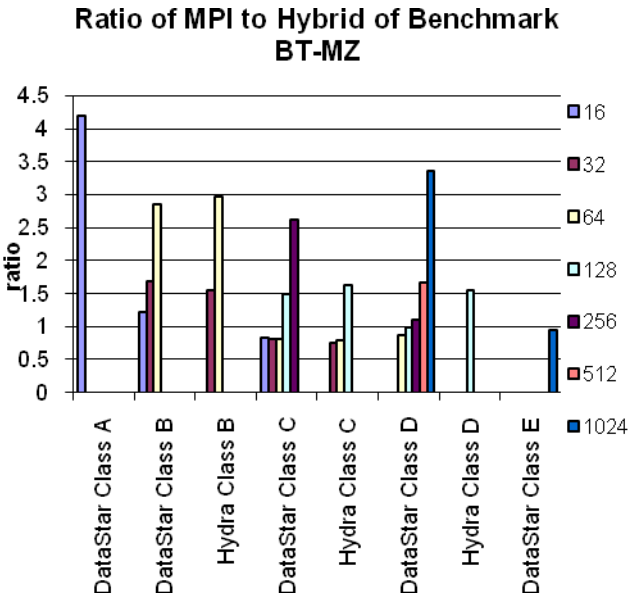
	Problem Class: C					
	BT-MZ		SP-MZ		LU-MZ	
# procs	DataStar	Hydra	DataStar	Hydra	DataStar	Hydra
16	312.07	X	259.12	X	228.77	X
32	139.75	135.86	170.46	137.4	X	X
64	72.95	69.72	80.13	74.9	X	X
128	39.29	31.54	41.48	30.84	X	X
256	20.63	X	16.58	X	X	X
512	X	X	X	X	X	X
1024	X	X	X	X	X	X

Table 15. Hybrid Run Results Class D

	Problem Class: D			
	BT-MZ		SP-MZ	
# procs	DataStar	Hydra	DataStar	Hydra
16	X	X	X	X
32	X	X	X	X
64	1524.36	X	1295.8	X
128	704.32	684.59	616.65	598.75
256	352.01	X	303.4	X
512	201.37	X	177.08	X
1024	99.7	X	77.09	X

Table 16. Hybrid Run Results Class E

	Problem Class: E			
	BT-MZ		SP-MZ	
# procs	DataStar	Hydra	DataStar	Hydra
16	X	X	X	X
32	X	X	X	X
64	X	X	X	X
128	X	X	X	X
256	X	X	X	X
512	X	X	X	X
1024	2038.09	X	1942.9	X



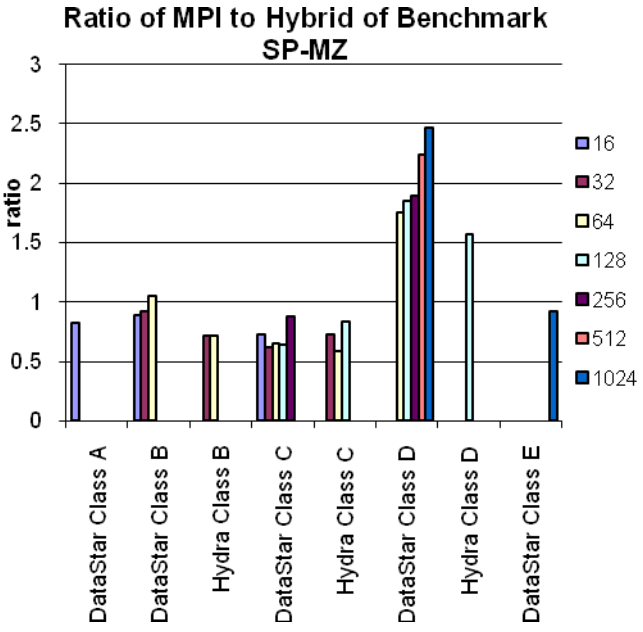


Fig 18. Graph for ratio MPI to Hybrid, benchmark SP-MZ.

5 PROCESSOR PARTITIONING

Processor partitioning [8] is developed by executing different runs of MPI code with a fixed number of processors, but varying the number of nodes and task per node. This way we can find what is the most efficient configuration for running a parallel program.

Table 17. DataStar p655, BT-MZ, Class B results with 8 processors

8 processors BT-MZ Class B		
# nodes	# task_per_node	execution time
8	1	124.99
4	2	125.05
2	4	127.52
1	8	129.37

Table 18. DataStar p655, BT-MZ, Class B results with 32 processors

32 processors BT-MZ Class B		
# nodes	# task_per_node	execution time
32	1	56.03
16	2	56.36
8	4	56.91
4	8	57.93

Table 19. DataStar p655, BT-MZ, Class B results with 64 processors

64 processors BT-MZ Class B		
# nodes	# task_per_node	execution time
64	1	56.64
32	2	56.91
16	4	57.34
8	8	58.09

Table 20. DataStar p655, BT-MZ, Class C results with 8 processors

8 processors BT-MZ Class C		
# nodes	# task_per_node	execution time
8	1	491.68
4	2	500.27
2	4	512
1	8	514.83

Table 21. DataStar p655, BT-MZ, Class C results with 32 processors

32 processors BT-MZ Class C		
# nodes	# task_per_node	execution time
32	1	122.97
16	2	124.83
8	4	127.76
4	8	131.02

Table 22. DataStar p655, BT-MZ, Class C results with 64 processors

64 processors BT-MZ Class C		
# nodes	# task_per_node	execution time
64	1	63.96
32	2	65.85
16	4	66.24
8	8	68.1

The first analysis is done on DataStar p655 for problem classes B and C and benchmarks BT-MZ and SP-MZ. The following tables show the results for these runs.

Table 23. DataStar p655, SP-MZ, Class B results with 8 processors

8 processors SP-MZ Class B		
# nodes	# task_per_node	execution time
8	1	92.88
4	2	93.34
2	4	96.79
1	8	104.62

Table 24. DataStar p655, SP-MZ, Class B results with 32 processors

32 processors SP-MZ Class B		
# nodes	# task_per_node	execution time
32	1	23.23
16	2	24.38
8	4	24.81
4	8	26.02

Table 25. DataStar p655, SP-MZ, Class B results with 64 processors

64 processors SP-MZ Class B		
# nodes	# task_per_node	execution time
64	1	11.75
32	2	11.77
16	4	12.63
8	8	13.15

Table 26. DataStar p655, SP-MZ, Class C results with 8 processors

8 processors SP-MZ Class C		
# nodes	# task_per_node	execution time
8	1	400.21
4	2	401.29
2	4	423.03
1	8	427.32

Table 27. DataStar p655, SP-MZ, Class C results with 32 processors

32 processors SP-MZ Class C		
# nodes	# task_per_node	execution time
32	1	93.37
16	2	98.78
8	4	99.87
4	8	104.97

Table 28. DataStar p655, SP-MZ, Class C results with 64 processors

64 processors SP-MZ Class C		
# nodes	# task_per_node	execution time
64	1	46.68
32	2	47.31
16	4	50.24
8	8	60.2

The pattern of these results shows that using less processors per node as possible is the most efficient configuration. Next we will show the results from the same analysis on Hydra.

Table 29. Hydra, BT-MZ, Class B results with 16 processors

16 processors BT-MZ Class B		
# nodes	# task_per_node	execution time
16	1	52.66
8	2	52.54
4	4	52.62
2	8	52.79
1	16	44.37

Table 30. Hydra, BT-MZ, Class B results with 32 processors

32 processors BT-MZ Class B		
# nodes	# task_per_node	execution time
32	1	48.25
16	2	48.20
8	4	48.24
4	8	48.24
2	16	48.37

Table 31. Hydra, BT-MZ, Class B results with 64 processors

64 processors BT-MZ Class B		
# nodes	# task_per_node	execution time
64	1	X
32	2	48.34
16	4	48.29
8	8	48.31
4	16	48.57

Table 32. Hydra, BT-MZ, Class C results with 8 processors

16 processors BT-MZ Class C		
# nodes	# task_per_node	execution time
16	1	202.17
8	2	203.12
4	4	202.42
2	8	203.10
1	16	204.77

Table 33. Hydra, BT-MZ, Class C results with 32 processors

32 processors BT-MZ Class C		
# nodes	# task_per_node	execution time
32	1	101.50
16	2	101.65
8	4	102.65
4	8	101.94
2	16	104.11

Table 34. Hydra, BT-MZ, Class C results with 64 processors

64 processors BT-MZ Class C		
# nodes	# task_per_node	execution time
64	1	X
32	2	53.90
16	4	53.92
8	8	54.46
4	16	54.64

Table 35. Hydra, SP-MZ, Class B results with 16 processors

16 processors SP-MZ Class B		
# nodes	# task_per_node	execution time
16	1	42.54
8	2	42.54
4	4	42.56
2	8	42.54
1	16	43.64

Table 36. Hydra, SP-MZ, Class B results with 32 processors

32 processors SP-MZ Class B		
# nodes	# task_per_node	execution time
32	1	21.10
8	2	21.10
4	4	21.15
2	8	21.27
1	16	21.68

Table 37. Hydra, SP-MZ, Class B results with 64 processors

64 processors SP-MZ Class B		
# nodes	# task_per_node	execution time
64	1	X
32	2	10.51
16	4	10.53
8	8	10.57
4	16	10.99

Table 38. Hydra, SP-MZ, Class C results with 16 processors

16 processors SP-MZ Class C		
# nodes	# task_per_node	execution time
16	1	171.43
8	2	171.60
4	4	171.59
2	8	171.72
1	16	176.14

Table 39. Hydra, SP-MZ, Class C results with 32 processors

32 processors SP-MZ Class C		
# nodes	# task_per_node	execution time
32	1	98.22
8	2	98.27
4	4	98.34
2	8	98.29
1	16	100.95

Table 40. Hydra, SP-MZ, Class C results with 64 processors

64 processors SP-MZ Class C		
# nodes	# task_per_node	execution time
64	1	X
32	2	42.50
16	4	42.64
8	8	42.96
4	16	44.66

These results are not consistent and do not show a clear pattern. The reason for this is that the nodes in Hydra are shared, meaning that if your application is not using all processors per node another application can take advantage of the other processors in the nodes sharing the memory of the node too. This causes an unclear result for the execution time. Unlike Hydra, DataStar has dedicated nodes so its results are more precise.

6 PROHESY PERFORMANCE MODELING

Prophesy [9] is an infrastructure for analyzing and modeling the performance of parallel and distributed applications. The core component of Prophesy is a relational database that allows for the recording of performance data, system features and application details. As a result, a Prophesy system can be used to develop models based upon significant data, identify the most efficient implementation of a given function based upon the given system configuration, explore the various trends implicated by the significant data, and predict the performance on a different system.

The Prophesy framework consists of three major components: data collection, data analysis, and three central databases.

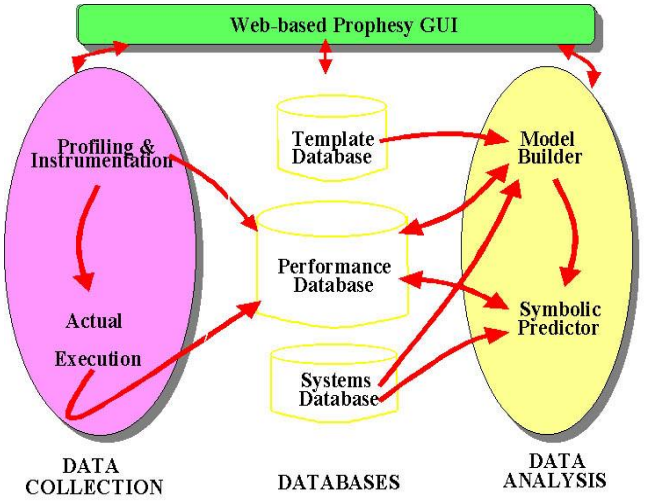


Fig 19. Prophesy framework diagram.

The data collection component focuses on the automatic instrumentation of codes at the level of basic blocks, procedures, or functions. The default mode consists of instrumenting the entire code at the level of basic loops and procedures. A user can specify that the code be instrumented at a finer granularity than that of loops or identify the particular events to be instrumented. The resultant performance data is automatically placed in the performance database and is used by the data analysis component to produce an analytical performance model with coefficients, at the granularity specified by the user. The models are developed based upon performance data from the performance database, model templates from the template database, and system characteristics from the systems database. The interface uses web technology to allow users to access Prophesy from anywhere. An application goes through three stages (instrumentation of the application, performance data collection of many runs, and model development using optimization techniques) to generate an analytical performance model.

Prophesy allows for the development of linear as well as nonlinear models. These models, when combined with data from the system database, can be used by the prediction engine to predict the performance on a differ-

ent compute platform. These models can then be used to give insight into which machine may perform the best for the given implementation of the kernel and what happens when one changes different features of the system [10].

For this research we uploaded the performance results into the Prophesy database and used the Prophesy predictor modeler for predict the performance as shown in figures 20, 21 and 22.

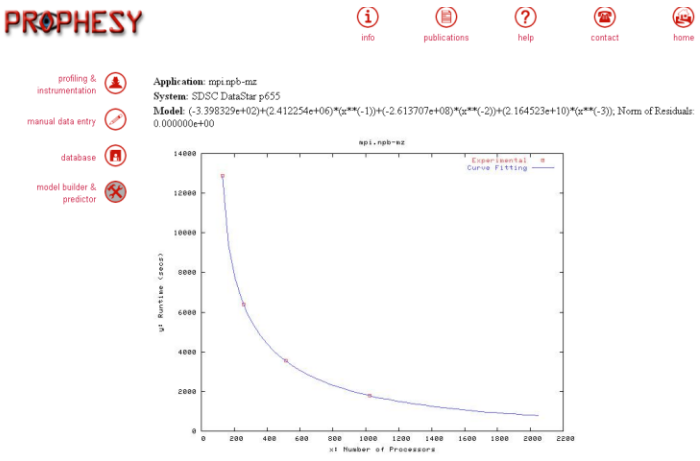


Fig 20. Prediction Performance modeler for MPI, problem class E, for 2048 processors on DataStar p655.

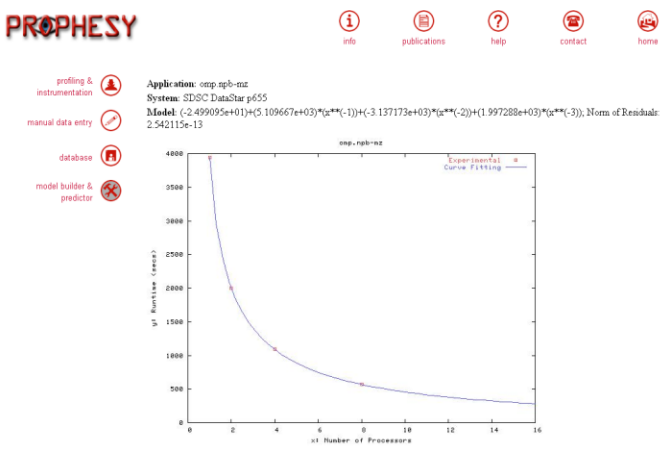


Fig 21. Prediction Performance modeler for OpenMP, problem class C, for 16 processors on DataStar p655.

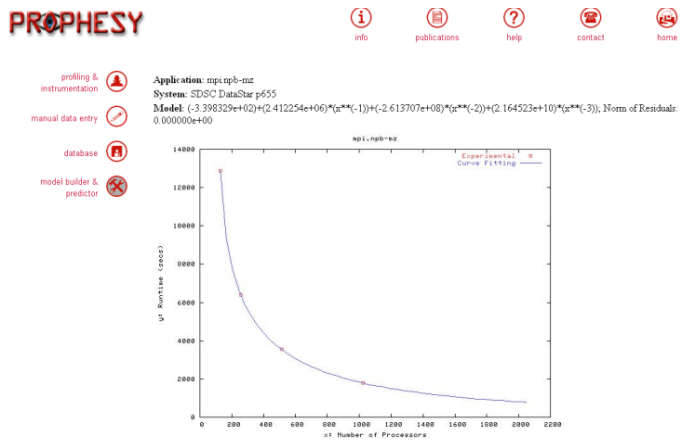


Fig 22. Prediction Performance modeler for Hybrid, problem class D, for 2048 processors on DataStar p655.

7 CONCLUSION

Through this research is shown that MPI overperfoms OpenMP for most cases. Also is shown that Hybrid overperforms MPI performance. It can be concluded that Hybrid programming paradigms such as the combination of MPI and OpenMP, MPI for inter node communication and OpenMP for inner node communication, can efficiently exploit the potential offered by CMP clusters.

For future work I would work on a hardware-level performance analysis with hpmcount for deeper understanding of these results. Also apply this kind of performance analysis method on other HPC and scientific computing applications.

Acknowledgement

I want to thank my mentors Dr. Valerie E. Taylor and Dr. Xingfu Wu for their support and all their help through the summer of 2008. This research was supported by Texas A&M Computer Science REU program and the DMP program for the summer of 2008.

REFERENCES

[1] SDSC DataStar, http://www.sdsc.edu/user_services/datastar/

[2] TAMU Hydra, "TAMU Supercomputing Facility," http://sc.tamu.edu/systems/hydra/hardware.shtml

[3] F. Ian, "Designing and Building Parallel Programs," http://www-unix.mcs.anl.gov/dbpp/ .

[4] OpenMP, http://openmp.org/.

[5] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Fredrickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga, "The NAS Parallel Benchmarks," *RNR Technical Report*, RNR-94-007, March 1994.

[6] R.F. Van der Wijngaart, H. Jin, "NAS Parallel Benchmarks,

Multi-Zone Versions," *NASA Advanced Supercomputer (NAS) Division NASA Ames Research Center*, NAS Technical Report NAS-03-010, Moffet Field, CA.

[7] R.F. Van der Wijngaart, H. Jin, "Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks," *NASA Advanced Supercomputer (NAS) Division NASA Ames Research Center*, M/S T27A-1, Moffet Field, CA.

[8] Xingfu Wu and Valerie Taylor, Processor Partitioning: An Experimental Performance Analysis of Parallel Applications on SMP Cluster Systems, *the 19th International Conference on Parallel and Distributed Computing and Systems (PDCS 2007)*, November 19-21, 2007, Hotel@MIT, Cambridge, MA.

[9] Valerie Taylor, Xingfu Wu, and Rick Stevens, Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications, *ACM SIGMETRICS Performance Evaluation Review*, Volume 30, Issue 4, March 2003. Also see http://prophesy.cs.tamu.edu.