Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks

Haoqiang Jin and Rob F. Van der Wijngaart*
NASA Advanced Supercomputing Division
M/S T27A-1, NASA Ames Research Center, Moffett Field, CA 94035-1000
{hjin,wijngaar}@nas.nasa.gov

Abstract

We describe a new suite of computational benchmarks that models applications featuring multiple levels of parallelism. Such parallelism is often available in realistic flow computations on systems of grids, but had not previously been captured in benchmarks. The new suite, named NPB Multi-Zone, is extended from the NAS Parallel Benchmarks suite, and involves solving the application benchmarks LU, BT and SP on collections of loosely coupled discretization meshes. The solutions on the meshes are updated independently, but after each time step they exchange boundary value information. This strategy provides relatively easily exploitable coarse-grain parallelism between meshes. Three reference implementations are available: one serial, one hybrid using the Message Passing Interface (MPI) and OpenMP, and another hybrid using a shared memory multi-level programming model (SMP+OpenMP). We examine the effectiveness of hybrid parallelization paradigms in these implementations on three different parallel computers. We also use an empirical formula to investigate the performance characteristics of the multi-zone benchmarks.

1 Introduction

The NAS Parallel Benchmarks (NPB) [1] 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. Implementations of NPB in MPI [2] and OpenMP [5] take advantage of this fine-grain parallelism. However, many important scientific problems feature several levels of parallelism, and this property is not reflected in NPB. For example, in the NASA production flow solver program OVERFLOW [4], geometrically complex domains are covered by sets of partially overlapping discretization grids, called zones. Solutions on each zone can be computed independently, providing coarse-grain parallelism. Additionally, fine-grain, loop-level parallelism can be exploited within each zone.

To remedy the above deficiency, we created the NPB Multi-Zone (NPB-MZ) versions [9] which include three families of multi-zone benchmarks, derived from the NPB. These multi-zone benchmarks stress the need to exploit both levels of parallelism and to balance the computational load. In this paper, we describe three reference implementations of NPB-MZ – one serial and two hybrid parallel. The first hybrid implementation uses Message Passing Interface (MPI) to communicate data related to overlap regions of zones, and OpenMP to parallelize loops within each zone. It is fully portable and can run on shared and distributed-shared memory systems, as well as on clusters of symmetric multi-processors. The second

*Employee of Computer Sciences Corporation

hybrid implementation uses a shared-memory parallel library (SMPlib) [6] to exchange data related to overlap regions of zones, and OpenMP to parallelize loops within each zone. This version takes advantage of shared memory buffers for efficient data exchange between processes. It is an efficient approach for exploiting multi-level parallelism on shared memory systems. We will use an empirical formula to investigate the performance characteristics of the multi-zone benchmarks and also to estimate the best process-thread combination for running hybrid codes.

In the following, we briefly describe the multi-zone benchmarks in Section 2. We then discuss the three reference implementations of NPB-MZ, including a brief discussion of the programming paradigms used, in Section 3. Section 4 presents performance results and characteristics of the hybrid NPB-MZ benchmarks running on several parallel computers. We draw our conclusions in the last section.

2 The Multi-Zone Benchmarks

The application benchmarks of NPB as specified in [1] solve discretized versions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions in the forms of Lower-Upper symmetric Gauss-Seidel (LU), Scalar Penta-diagonal (SP), and Block Tri-diagonal (BT). 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. In the production code OVERFLOW [4], the flow equations are solved independently in each zone, and after each iteration, the zones exchange boundary values with their immediate neighbors with which they overlap.

We take the OVERFLOW approach [4] in creating the NPB Multi-Zone versions of LU, BT, and SP, namely LU-MZ, BT-MZ, and SP-MZ. Detailed specification of the multi-zone benchmarks is given in [9]. We will give a summary below: the general approach common for all three benchmarks in Section 2.1 and differences between individual benchmarks in the subsequent sections. We note that the selection of different NPB solvers for the new benchmarks is fairly arbitrary. The major difference between the three multi-zone problems lies in the way the zones are created out of the single overall mesh.

2.1 General Approach

For each benchmark problem a logically rectangular discretization mesh is divided into a two-dimensional horizontal tiling of three-dimensional zones. To avoid pathologically shaped zones after partitioning the overall mesh in the two horizontal directions (x and y), we change the aspect ratios of the meshes of the original NPB. For each problem class the total number of points in all zones is kept approximately the same as in the original NPB. Table 1 lists the aggregate problem sizes and the number of zones for different problem class in the multi-zone versions. Memory usage is estimated by assuming 24 double-precision variables for each mesh point.

Within all zones, the LU, BT, or SP problems are solved to advance the time-dependent solution, using exactly the same methods as described in [1, 8]. The mesh spacings of all

Table 1: Aggregate problem size, the number of zones, and the memory requirement for each problem class. G_x, G_y, and G_z are aggregate spatial dimensions.

Class	Aggregate Size	Zones ($x\_zones \times y\_zones$)			Memory
	($G_x \times G_y \times G_z$)	LU-MZ	SP-MZ	BT-MZ	
S	$24 \times 24 \times 6$	4×4	2×2	2×2	1 MB
W	$64 \times 64 \times 8$	4×4	4×4	4×4	6 MB
A	$128 \times 128 \times 16$	4×4	4×4	4×4	50 MB
B	$304 \times 208 \times 17$	4×4	8×8	8×8	200 MB
C	$480 \times 320 \times 28$	4×4	16×16	16×16	800 MB
D	$1632 \times 1216 \times 34$	4×4	32×32	32×32	12.8 GB

zones of a particular problem class are identical, and the overlap between neighboring zones is exactly one such spacing, so that discretization points in overlap regions coincide exactly.

Exchange of boundary values between zones takes place after each time step, which provides the fairly loose coupling of the otherwise independent solution processes within the zones. Solution values at points one mesh spacing away from each vertical zone face are copied to the coincident boundary points of the neighboring zone. The problem is periodic in the two horizontal directions, so donor point values at the extreme sides of the mesh system are copied to boundary points at the opposite ends of the system.

2.2 LU-MZ

For all problem classes the number of zones in each of the two horizontal dimensions equals four (i.e. 4×4). The overall mesh is partitioned such that the zones are identical in size, which makes it relatively easy to balance the load of the parallelized application.

2.3 SP-MZ

As in the case of LU-MZ, the overall mesh is partitioned such that the zones are identical in size. However, the number of zones in each of the two horizontal dimensions grows with the problem size (see Table 1). It is relatively easy to balance the load of the parallelized application.

2.4 BT-MZ

The number of zones in this benchmark grows with the problem size in the same fashion as in SP-MZ (see Table 1). However, the overall mesh is now partitioned such that the sizes of the zones span a significant range. This is accomplished by increasing sizes of successive zones in a particular coordinate direction in a roughly geometric fashion. Except for class S, the size ratio of the largest to smallest zone is approximately 20. This makes it harder to balance the load than for SP-MZ and LU-MZ if the implementation is to take advantage of multi-level parallelism. The BT-MZ benchmark is a more realistic case. Examples of uneven mesh tiling for the BT-MZ benchmark are shown in Figure 1.

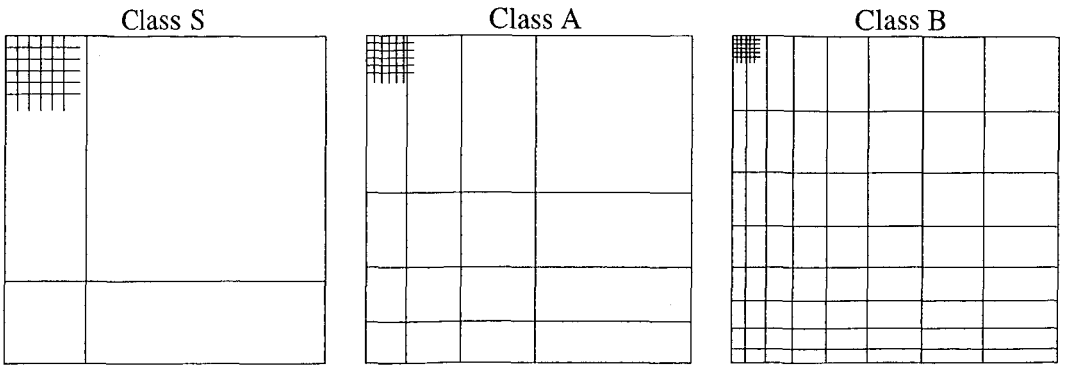

Figure 1: Examples of uneven mesh tiling (horizontal cut through mesh system) for three different classes of the BT-MZ benchmark.

3 Implementations

In this section, we describe three reference implementations of the multi-zone benchmarks – one serial and two hybrid parallel.

3.1 Serial Implementation

The serial implementation of NPB-MZ is based on the NPB3.0-SER release of NPB [5]. The data flow of the serial NPB-MZ version is shown in Figure 2. The original single-zone problem of LU, SP, and BT in NPB3.0-SER is first subdivided into multiple zones according to the benchmark specifications as prescribed in [9]. Solutions for each zone are then initialized. The benchmarking loop starts with a time step loop which contains a procedure (`exch_qbc`) to exchange boundary values of different zones. The same kernel PDE (partial differential equation) solvers for LU, SP, and BT to solve a set of system equations are used in the new LU-MZ, SP-MZ, and BT-MZ for obtaining solution updates within each zone. The solving stage includes procedures for performing right-hand-side (RHS) calculations and the Lower-Upper solver (for LU-MZ) or the Alternative Directional Implicit solver (for SP-MZ and BT-MZ). The solution is then verified for all zones for a given problem class.

3.2 Hybrid Implementation

Parallelism in the multi-zone benchmarks can be exploited with a two-level approach: a coarse grained parallelization among zones and a fine grained parallelization within each zone. The division (or data decomposition) of zones is natural except for exchanging overlapped boundary values since solutions within each zone can be performed concurrently. The main task is to balance the computational workload among processes.

In the following, we will first discuss the load balancing scheme. Our implementations of the two-level parallelization are in a hybrid form: we use either message passing or a shared-memory parallelization library to communicate data related to overlap regions of zones, and OpenMP to parallelize loops within each zone. The second hybrid model takes advantage of shared memory buffers and is an efficient way to exploit multi-level parallelism on shared memory systems.

3.2.1 Zone Grouping and Load Balancing

Proper load balancing is critically important for efficient parallel computing. There are a number of load balancing strategies for multi-zone overset grid applications (see [3] for an overview). We use a simple zone grouping strategy based on a bin-packing algorithm [10]. In a zone grouping strategy, the N_Z zones need to be clustered into N_G groups, where N_G is equal to the total number of processes, N_P. Each zone group is then assigned to a process for parallel execution. The goal is to distribute equal computational workloads among the zone groups while minimizing the inter-process communication.

In the bin-packing algorithm, we first estimate the computational workload of each zone by counting the number of grid points in the zone. The N_Z zones are then sorted by size in descending order. In the beginning, the zone groups are empty. Each zone in the sorted list is handled one at a time and is assigned to the smallest group that satisfies the connectivity test with other zones already in that group. The connectivity test examines any overlap between a pair of zones. We do not take into account the communication cost, as we will find out in the next section the communication cost is rather small in these benchmarks. The procedure finishes when all zones are assigned to groups.

If the load cannot be well balanced with zone groups, we further try to adjust the number of OpenMP threads assigned to each process. An extra thread from the smallest group is moved to the largest group, provided the load balance is improved after such a movement and the number of threads in a group does not exceed a given limit. For a symmetric multi-processor node, the number of threads is limited to the number of CPUs in the node. The process stops when no more movement occurs.

As an extreme case, Figure 3 illustrates the use of threads to improve the load balance for the BT-MZ, Class C problem on $N_P = 256$, in which only one zone is assigned to one process. Even though load balancing of SP-MZ and LU-MZ is trivial, we still use the same algorithm to take into account the zone-to-zone connectivity.

3.2.2 MPI+OpenMP

Message Passing Interface (MPI) is a widely accepted standard for writing message passing programs and is supported in all modern parallel computers. The standard provides the user with a programming model where processes communicate by calling library routines to send to and receive messages from other processes. The programming model was designed for distributed memory systems, but also works on shared memory systems. As clusters of symmetric multi-processor machines 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 is summarized in Figure 4. The number of MPI processes is defined at compilation time in order to avoid dynamic memory allocation. Each process is first assigned with a group of zones and a given number of OpenMP threads based on the load balancing scheme described in Section 3.2.1. There is no dynamic load adjustment at run time. As in the sequential version (see Figure 2), solutions for the zones assigned to each process are then initialized, followed by the time

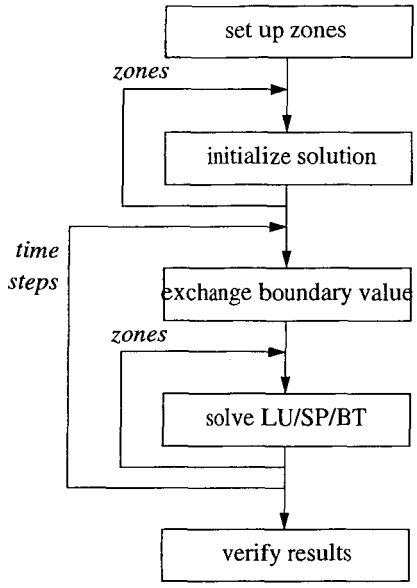

Figure 2: Schematic data flow graph of the multi-zone benchmarks in sequential execution.

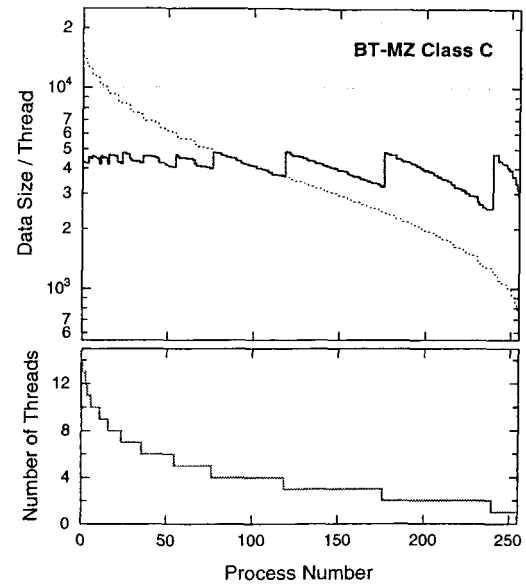

Figure 3: Uneven zone size distribution of the BT-MZ Class C before (dash line) and after (solid line) balancing with threads. The bottom figure indicates number of threads assigned to each process.

step loop. Process communication occurs inside the procedure `exch_qbc`, which exchanges boundary values of zones assigned to different processes. There is no communication during the LU, SP, or BT solving stage. The last stage of verification performs a reduction of solutions and residues from all zones for a given problem class.

The OpenMP parallelization within a zone is very similar to the OpenMP single-zone version of NPB [5]. A single level parallelization is used for the outermost parallel loops in SP-MZ and BT-MZ, mainly the K loops for the z dimension. The OpenMP parallelization of LU-MZ is on the J loops for the y dimension, mainly due to the required pipeline implementation for the LU solver and the fixed number of zones (thus, increased y size as the problem size grows) for this benchmark. The increased y size allows more efficient OpenMP parallelization in this dimension.

3.2.3 SMP+OpenMP

The second hybrid approach, SMP+OpenMP, is based on a multi-level parallel (MLP) programming model developed by Taft [7] at NASA Ames Research Center for achieving high levels of parallel efficiency on shared memory machines. It exploits two-level parallelism in applications: coarse-gained with forked processes and fine-grained (loop level) with OpenMP threads. (We refer to this coarse-gained parallelization with forked processes as the SMP, or shared memory parallel, model.) Communication between the forked processes is done by directly accessing data in a shared memory buffer. Coupled with the second level parallelism, MLP has demonstrated scalability on more than 500 processors for real CFD problems [7]. In the SMP model, a program starts with a single process (the master process) to perform initialization, such as reading input data from a file, and to set up necessary shared memory

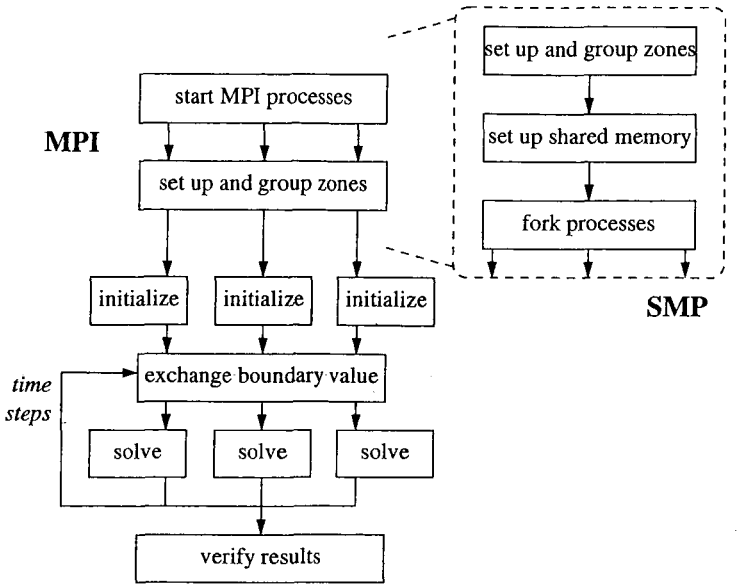

Figure 4: Coarse grained parallelization with zone groups for the multi-zone benchmarks using either MPI or SMP parallel programming model.

buffers (or arena) for communication. Additional processes are then created via the *fork* call. The forked processes have a private copy of the virtual memory of the master process except for the shared memory arena. Thus, broadcasting any input data is not necessary in this model as it would have been required in a message passing program. The master and its forked processes then work on the designated code segments in parallel and synchronize as needed.

The original MLP library (MLPlib) [7] consists of only three routines: `MLP_getmem` to allocate a piece of shared memory, `MLP_forkit` to spawn processes, and `MLP_barrier` to synchronize processes. The MLPlib application program interface (API) includes a special argument allowing thread-to-processor binding, or *pinning*, which has been shown to improve performance of hybrid codes on machines with non-uniform memory access. The main limitation of MLPlib is its lack of point-to-point synchronization primitives, which are usually required for more general classes of applications. The SMP library (SMPlib) [6] extends the MLPlib concept by including the `SMP_Signal` and `SMP_Wait` primitives for point-to-point synchronization between processes. A process may update a shared buffer and use `SMP_Signal` to inform another process the availability of the data; the other process can use `SMP_Wait` for the notification of the signal to copy data from the shared buffer. The Signal/Wait approach is very flexible and in general has less communication overhead than a global barrier. The complete description of the SMPlib API is given in [6].

The hybrid SMP+OpenMP implementation of the multi-zone benchmarks is very similar to the hybrid MPI+OpenMP. In particular, the OpenMP parallelization is identical in both versions. The main difference lies in the startup stage as shown in Figure 4. Before forking processes, the master process sets up zone groups based on the load balancing scheme described in Section 3.2.1 and allocates proper shared memory arena for later process communication. As in the MPI+OpenMP version, process communication occurs inside

the procedure `exch_qbc`, which copies boundary values of zones to/from the shared memory buffer with proper barrier synchronizations. Overall, it is slightly easier to develop the hybrid SMP+OpenMP codes than the MPI+OpenMP correspondents, mainly because the bookkeeping for process communication in the SMP version is simpler.

4 Results

In this section we report performance results and characteristics of the hybrid NPB-MZ benchmarks running on three different parallel machines. The sequential version of NPB-MZ serves as a baseline implementation for other parallel implementations and for parallel tools and compilers. We will not examine the performance of the sequentual codes in this report.

4.1 Testing Platforms

For running the multi-zone benchmarks we used three different parallel computers:

- SGI Origin 3000: a 128-node ccNUMA machine with 4 CPUs per node and a single-system image operating system,
- HP/Compaq Alpha SC45: a cluster of 348 shared memory nodes with 4 CPUs per node, and
- IBM pSeries: a cluster of 208 shared memory nodes with 16 CPUs per node.

The main performance characteristics of a node in each system is summarized in Table 2. Various compilers and compilation flags used in our tests are listed in Table 3. We used the `mpt.1.5.3.0` runtime system for running MPI programs on the SGI Origin 3000.

Table 2: Architectural specifications of the R12K, EV-68, and Power3 nodes in the three parallel systems.

Machine	CPU Type	CPU/ Node	Clock (MHz)	Peak (GFlops/s)	L1 (KB)	L2 (MB)	Memory/ Node (GB)
SGI Origin 3000	R12K	4	400	0.8	32	8	2
HP/Compaq SC45	EV-68	4	1000	2.0	64	8	2
IBM pSeries	Power3	16	375	1.5	128	8	2

Table 3: The compilers and compilation flags used in the tests.

Machine	O/S	Compiler	Compilation flag
SGI Origin 3000	IRIX 6.5	MIPSpro 7.4	-O3 -mp
HP/Compaq SC45	TRU64 5.1A	Compaq Fortran 5.5	-fast -omp
IBM pSeries	AIX 5.1	XL Fortran 7.1	-O3 -qsmp=omp

The SGI Origin 3000 (named Lomax), located at NASA Ames Research Center, is a distributed shared-memory system with the SGI NUMA 3 architecture (i.e., the third generation non-uniform memory access). The system contains 128 C-bricks (or nodes) that are connected by the NUMAlink3 interconnect network and are globally addressable through hardware cache-coherence protocol. The memory accessing time is about 175 ns within a local node and 470 ns to a remote node.

The HP/Compaq AlphaServer SC45 (named Halem) is located at NASA Goddard Space Flight Center. The system is a distributed-memory supercomputer that contains 348 AlphaServer ES45 nodes and uses high speed Intelligent Interconnects between nodes. Each ES45 node incorporates four Alpha-EV68 processors running at 1.0 GHz or 1.25 GHz that share 2 GB of local memory. Each EV68 processor has 64 KB of L1 cache and 8 MB of L2 cache. Our performance results were obtained on nodes running at 1.0 GHz.

The Power3 experiments reported in this paper were conducted on a single Nighthawk II node of the 208-node IBM pSeries system (named Seaborg) running AIX 5.1 and located at Lawrence Berkeley National Laboratory. The IBM Power3 was first introduced in 1998 as part of the RS/6000 series. Each 375 MHz processor contains two floating-point units (FPUs) that can issue a multiply-add (MADD) per cycle for a peak performance of 1.5 GFlops/s. The CPU has a 32KB instruction cache and a 128KB 128-way set associative L1 data cache, as well as an 8MB four-way set associative L2 cache with its own private bus. Each SMP node consists of 16 processors connected to main memory via a crossbar. Multi-node configurations are networked via the IBM Colony switch using an omega-type topology.

4.2 Timing and Scalability

Figure 5 shows the Gflops per second reported by the two hybrid versions of LU-MZ, SP-MZ and BT-MZ for the Class C problem size on the SGI Origin 3000. The number of CPUs (or processors) is indicated by the $N_p \times N_t$ combination, where N_p is the number of MPI or SMP processes and N_t is the number of threads per process. For LU-MZ and SP-MZ, N_t is the actual number of threads assigned to each process; for BT-MZ, N_t is the average number of threads per process. The $N_p \times N_t$ values in the figure are those combinations that produced the best performance for each benchmark for a given total number of CPUs. The best combination is further discussed in the next section. The thread-to-processor pinning was applied for all the runs on the SGI Origin 3000 reported in this section.

Overall, both MPI+OpenMP and SMP+OpenMP performed similarly, showing close to linear speed-up with increasing number of CPUs up to 256. Coarse grain parallelism in LU-MZ is limited to 16 processes due to the structure of the benchmark, and OpenMP threads are required for scaling beyond 16 CPUs. The SP-MZ benchmark poses no restriction on the coarse grain parallelism; in fact, the best performance is achieved with parallelism only on the coarse level. For the Class C problem of BT-MZ, the load can be balanced at the coarse grain level up to 64 processes and threads are required for load balancing more than 64 CPUs. Both SP-MZ and BT-MZ demonstrated over 60 Gflops/s performance on 256 CPUs.

Detailed comparison of the computation and communication times in the benchmarks is summarized in Table 4. The "Total" column indicates the total benchmark time, "RHS" is the time spent in the right-hand-side calculation, "Solver" is the solver time, and "exch_qbc"

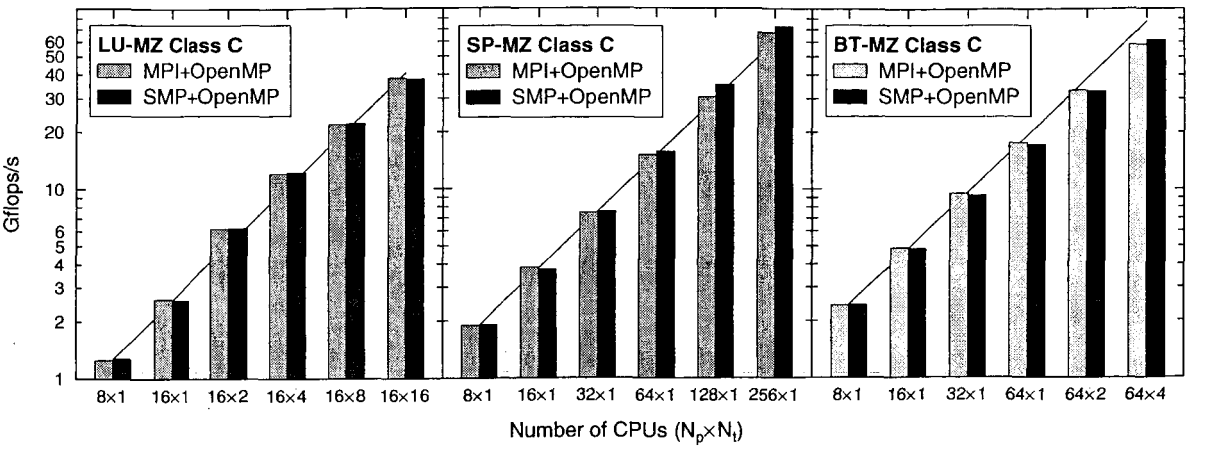

Figure 5: Scaling of the Class C problem of LU-MZ, SP-MZ and BT-MZ on the Origin 3000. The lines indicate linear speedup based on the 8 and 16 CPUs results.

indicates the communication time during boundary value exchange between zones. RHS and Solver are the two most important computing components; they were discussed in Section 3.1. The RHS and Solver times are closely matched between the MPI+OpenMP and SMP+OpenMP versions. The ratio of communication time and total time in both LU-MZ and BT-MZ increases slightly as the number of CPUs increases, but is about 5% or less. However, SP-MZ spent more time in communication, especially the MPI version shows 20% communication time on 128 CPUs. In general, the SMPlib used in the SMP+OpenMP version showed less communication overhead than the MPI library.

To examine the sensitivity of the process-thread combination on performance, we ran the hybrid versions of NPB-MZ on a single node of the IBM Power3 and the results from runs with a fixed number of 16 CPUs are shown in Figure 6. Again, both the MPI+OpenMP and SMP+OpenMP performed very similarly. In all cases, performance drops as the number of threads per process increases. Results from the Class A problem of NPB-MZ obtained on the HP/Compaq SC45 are shown in Figure 7. Due to the limited number of 4 CPUs in one shared memory node of this machine, we only ran the MPI+OpenMP versions. As one can see, the performance of BT-MZ does not improve much beyond 16 CPUs even though LU-MZ and SP-MZ still scale up. In contrast, there is no such a limitation on the SGI Origin 3000 due to the globally shared address space with a single-system image.

4.3 Performance Model

Speedup of a run with N_p processes and N_t threads per process can roughly be estimated by

$$P_{N_p \times N_t} = S_t/(S_{mg} \cdot f_{N_t}) \tag{1}$$

where S_t is the total problem size, S_{mg} is the maximum size of a zone group assigned to a process divided by the number of threads used, and f_{N_t} is a factor weighted by the number of threads. Assuming a static scheduling in the OpenMP parallelization, we have an empirical form

$$f_{N_t} = [1 + (N_t - 1)\,(0.005 + 0.5/S_p)] \cdot f_{ib} \tag{2}$$

Table 4: Timing profile in seconds of the three benchmarks obtained on the SGI Origin 3000 and with 1/10th of the benchmark iterations.

CPUs	MPI+OpenMP				SMP+OpenMP			
	Total	RHS	Solver	exch_qbc	Total	RHS	Solver	exch_qbc
LU-MZ, Class C								
8	150.34	41.98	107.13	1.21 (0.8%)	147.33	41.21	104.85	1.26 (0.9%)
16	73.02	20.72	51.61	0.69 (0.9%)	73.76	20.17	53.12	0.47 (0.6%)
32	30.61	10.19	19.44	0.54 (1.8%)	30.47	10.03	19.31	0.26 (0.9%)
64	15.65	5.46	9.46	0.28 (1.8%)	15.49	5.33	9.37	0.30 (1.9%)
128	8.61	3.04	4.71	0.29 (3.4%)	8.53	3.01	4.73	0.21 (2.5%)
256	4.93	1.67	2.39	0.25 (5.0%)	4.97	1.68	2.39	0.27 (5.3%)
SP-MZ, Class C								
8	64.43	29.49	31.15	3.77 (5.9%)	63.72	29.31	30.80	3.61 (5.7%)
16	31.78	13.89	15.39	2.49 (7.8%)	32.56	14.93	15.75	1.88 (5.8%)
32	16.26	7.34	7.87	1.04 (6.4%)	16.14	7.58	7.84	0.72 (4.5%)
64	8.05	3.24	3.77	1.03 (12.9%)	7.75	3.47	3.75	0.52 (6.7%)
128	3.98	1.36	1.82	0.80 (20.1%)	3.44	1.36	1.80	0.28 (8.0%)
256	1.79	0.68	0.91	0.20 (10.8%)	1.69	0.66	0.90	0.13 (7.6%)
BT-MZ, Class C								
8	99.53	15.14	83.02	2.07 (2.1%)	98.97	14.65	83.34	1.65 (1.7%)
16	49.15	7.32	41.01	1.15 (2.3%)	49.51	7.26	41.50	1.07 (2.2%)
32	25.15	3.62	21.30	0.39 (1.5%)	25.62	3.65	21.75	0.39 (1.5%)
64	13.50	1.79	11.39	0.41 (3.0%)	13.86	1.80	11.87	0.28 (2.0%)
128	7.16	0.99	5.86	0.36 (5.0%)	7.25	0.99	6.10	0.21 (2.8%)
256	4.11	0.62	3.17	0.36 (8.7%)	3.89	0.62	3.15	0.14 (3.7%)

where $f_{ib} = \max(\lceil S_p/N_t \rceil, 1) N_t/S_p$ is an estimate of the load balancing factor from the OpenMP parallelization and S_p is the typical dimension size in which the OpenMP parallelization is applied. For BT-MZ and SP-MZ, $S_p = G_z$; for LU-MZ, $S_p = G_y/y\_zones$. Equation 1 does not take into account the cost of process communication which, as indicated from the previous section, is relatively small. The thread weighting factor in equation 2 takes into account the scaling degradation caused by the increased number of threads and the load imbalance resulting from thread scheduling.

The left panel of Figure 8 compares the calculated speedup using equation 1 with the measured values on the SGI Origin 3000 for the BT-MZ, Class A problem. As one can see from the figure, the calculated values match very well with those measured, not only the best $N_p \times N_t$ combination (as indicated by the dash lines) but also the shape as a function of thread numbers.

In the right panel of Figure 8 we plot the speedup from the best $N_p \times N_t$ combination for the SGI Origin 3000 and the HP/Compaq SC45. The Origin 3000 places no limitation on the number of threads for each process, while the SC45 cluster allows only the maximum of 4 threads for each process due to the node limitation, which in fact limits the scalability

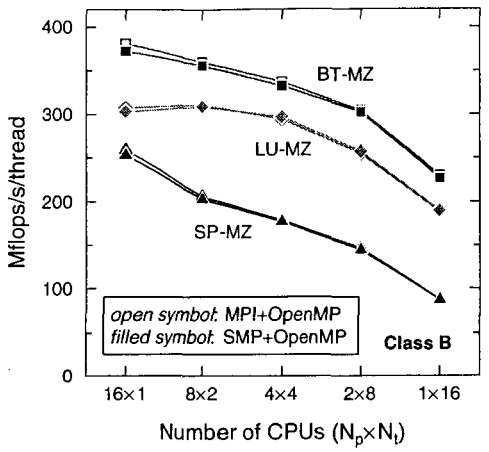

Figure 6: Performance of NPB-MZ on IBM Power3 from different combinations of $N_p \times N_t$ for a given total number of 16 CPUs.

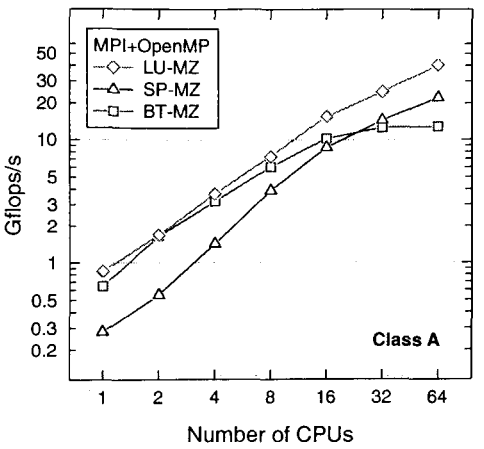

Figure 7: Performance of NPB-MZ on HP/Compaq Alpha SC45.

of BT-MZ, Class A problem beyond 16 processors. The calculation shows the correct trend although the values are underestimated, primarily because cache effects are not considered in equation 1. Clearly equation 1 can be used to find out the best combination of N_p and N_t for a given total number of processors.

Table 5 summarizes the calculated maximum number of processors up to which each problem of different benchmarks could run and achieve more than 50% of the parallel efficiency for a given machine. This table could guide the running of a particular problem on a selected machine. We would like to emphasize that these estimates do not take into account the cost of process communication.

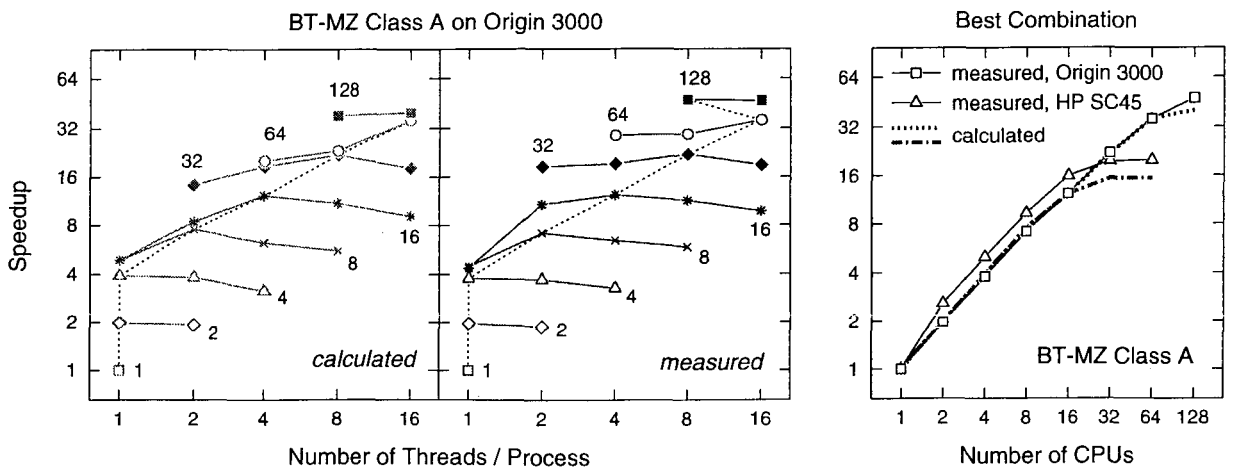

Figure 8: Left panel: Comparison of calculated speedup versus measured values from different $N_p \times N_t$ combinations. The best values are indicated by the dash lines. The total number of CPUs is also marked for each group. Right panel: Speedup from the best $N_p \times N_t$ combination on the SGI Origin 3000 and HP SC45.

Table 5: Calculated maximum number of processors up to which each NPB-MZ could run and achieve more than 50% of parallel efficiency for two different machines.

	SGI Origin 3000			HP/Compaq SC45		
Class	LU-MZ	SP-MZ	BT-MZ	LU-MZ	SP-MZ	BT-MZ
A	512	256	64	64	64	16
B	512	1024	256	64	256	64
C	512	8192	1024	64	1024	256
D	1024	> 16384	4096	64	4096	1024

4.4 Other Performance Issues

The hybrid implementation of NPB-MZ has demonstrated good scalability. We would like to point out a few factors that could potentially limit the performance of NPB-MZ. First, the coarse grained parallelization limits the maximum number of processes to the number of zones for a given problem. For instance, LU-MZ allows a maximum of 16 processes. Second, the number of OpenMP threads on the fine grain level is limited by the underlying hardware as well as the dimension size to which OpenMP is applied. On the HP/Compaq SC45, LU-MZ cannot scale beyond $16 \times 4 = 64$ processors. Third, the load balancing with OpenMP threads for BT-MZ is also limited on the HP/Compaq SC45 due to only 4 CPUs in one shared memory node of this machine. The shared memory architecture of the SGI Origin 3000 poses fewer constraints on the use of threads.

We also observed the performance impact of using thread-to-processor pinning on the hybrid codes on the SGI Origin 3000. Application performance on NUMA architectures like the Origin depends on memory placement of data and thread placement onto CPUs. Improper initial data placement or unwanted migration of threads between CPUs can increase memory access time, thus, degrading performance. SGI provides a number of system calls to bind threads to Origin CPUs. We tested the effect of pinning versus no pinning on the hybrid SMP+OpenMP NPB-MZ codes and the results are shown in Figure 9. We noticed that pinning improves performance substantially in the hybrid mode when more than one threads are used for one process. The impact is even more profound as the number of CPUs increases. Pure process mode (like 16×1) or pure thread mode (like 1×16) is not influenced by pinning. The little impact of pinning on the BT-MZ benchmark is likely due to a more dominant factor from the load imbalance in the Class A problem on 16 CPUs as evident from the 16×1 run. A detailed analysis of these results will be presented in a subsequent paper.

5 Conclusions

In summary, we have described the implementations of three multi-zone benchmarks that are derived from the NAS Parallel Benchmarks. These multi-zone benchmarks are suitable for exploiting multi-level parallelism which exists in many important scientific problems. The sequential implementation is a good candidate for parallelization tools and compilers to exploit multi-level parallelization strategies. The two hybrid implementations, MPI+OpenMP

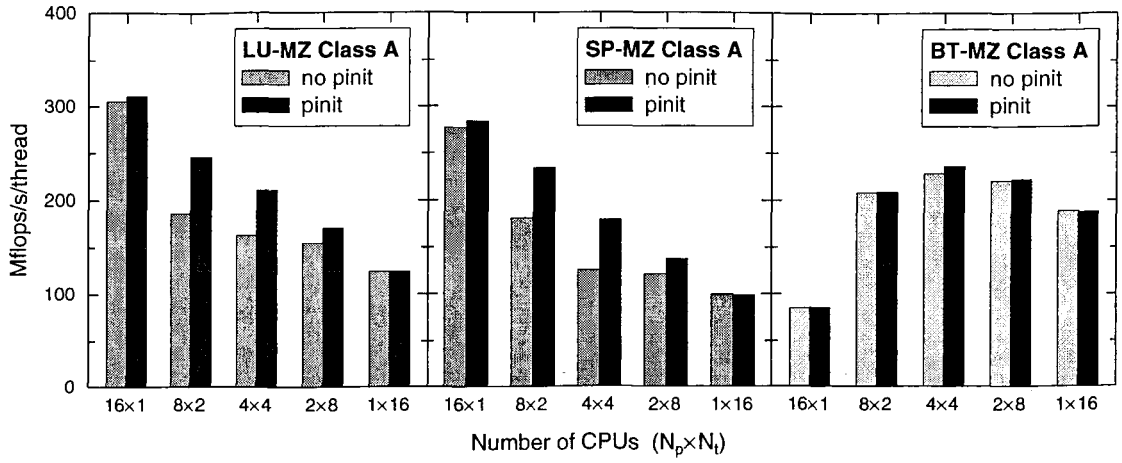

Figure 9: Effects of pinning versus no pinning on the SGI Origin 3000.

and SMP+OpenMP, of the NPB-MZ benchmarks have been tested on three different parallel machines and demonstrated great potential of the hybrid programming model on these machines. We used an empirical formula to study the performance characteristics of the benchmarks and estimate the best $N_p \times N_t$ combination for running the hybrid codes. The load balance of the BT-MZ benchmark, which contains non-uniform size zones, is crucial for good performance. A simple bin-packing algorithm together with the use of threads in load balancing has presented satisfactory results. For additional performance improvement on a large processor counts, further examination of the communication cost in the load balancing algorithm may be needed.

Acknowledgment
The authors would like to thank many colleagues in the Algorithms, Tools, and Architectures group, in particular Rupak Biswas, Mohammad Djomehri, Robert Hood, and Gabriele Jost for their valuable discussion and inputs. This work was partially supported by NASA contracts DTTS59-99-D-00437/A61812D with Computer Sciences Corporation/AMTI.

References

[1] D. Bailey, E. Barscz, J. Barton, D. Browning. R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, S. Weeratunga. *The NAS Parallel Benchmarks.* NAS Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA, 1994.

[2] D.H. Bailey, T. Harris, W.C. Saphir, R.F. Van der Wijngaart, A.C. Woo, M. Yarrow. *The NAS Parallel Benchmarks 2.0.* NAS Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[3] M.J. Djomehri, R. Biswas, N. Lopez-Benitez. *Load Balancing Strategies for Multi-BLock Overset Grid Applications.* Proceedings of the 18th International Conference on Computers and Their Applications, (2003) 373.

[4] M.J. Djomehri, R. Biswas, M. Potsdam, R.C. Strawn. *An Analysis of Performance Enhancement Techniques for Overset Grid Applications.* Proceedings of the 17th International Parallel and Distributed Processing Symposium, Nice, France, 2003.

[5] H. Jin, M. Frumkin, J. Yan. *The OpenMP Implementation of NAS Parallel Benchmarks and its Performance.* NAS Technical Report NAS-99-011, NASA Ames Research Center, Moffett Field, CA, 1999.

[6] H. Jin, G. Jost. *Performance Evaluation of Remote Memory Access Programming on Shared Memory Parallel Computers.* NAS Technical Report NAS-03-001, NASA Ames Research Center, Moffett Field, CA, 2003.

[7] J. Taft, *Achieving 60 GFLOP/s on the Production CFD Code OVERFLOW-MLP.* Parallel Computing, 27 (2001) 521.

[8] R.F. Van Der Wijngaart. *NAS Parallel Benchmarks, Version 2.4.* NAS Technical Report NAS-02-007, NASA Ames Research Center, Moffett Field, CA, 2002.

[9] R.F. Van Der Wijngaart, H. Jin. *NAS Parallel Benchmarks, Multi-Zone Versions.* NAS Technical Report NAS-03-010, NASA Ames Research Center, Moffett Field, CA, 2003.

[10] A.M. Wissink and R. Meakin, *Computational Fluid Dynamics with Adaptive Overset Grids on Parallel and Distributed Computer Platforms.* Proceeding of International Conference on Parallel and Distributed Processing Techniques and Applications, (1998) 1628.