Characterization of Scientific Workloads on Systems with Multi-Core Processors

Sadaf R. Alam, Richard F. Barrett, Jeffery A. Kuehn, Philip C. Roth, Jeffrey S. Vetter

Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN, USA 37831
{alamsr,rbarrett,kuehn,rothpc,vetter}@ornl.gov

Abstract. Multi-core processors are planned for virtually all next-generation HPC systems. In a preliminary evaluation of AMD Opteron Dual-Core processor systems, we investigated the scaling behavior of a set of micro-benchmarks, kernels, and applications. In addition, we evaluated a number of processor affinity techniques for managing memory placement on these multi-core systems. We discovered that an appropriate selection of MPI task and memory placement schemes can result in over 25% performance improvement for key scientific calculations. We collected detailed performance data for several large-scale scientific applications. Analyses of the application performance results confirmed our micro-benchmark and scaling results.

Keywords: Performance characterization, Multi-core processor, AMD Opteron, micro-benchmarking, scientific applications.

1 Introduction

The move by major microprocessor vendors toward processors containing multiple homogeneous processor cores is arguably the most important trend in contemporary computer architectures. Given the ability to produce chips with an ever-increasing number of transistors, the approach of duplicating existing cores is a straightforward way to address problems related to physical constraints (e.g., power, thermal, and signaling) and limited instruction-level parallelism. However, because all cores of a processor share the link between the processor's socket and memory, contention for this memory link can limit the achievable performance when using more than one core per processor. Applications can perform well on systems with these multi-core processors, but only if they expose enough parallelism to use the multiple cores within their collective memory bandwidth limitations.

The fundamental question for HPC scientific computing is whether multiple cores per processor can provide performance commensurate with initial expectations. Simply put, the shared memory and I/O (network) bandwidth of multiple cores in a socket draws into question both how efficiently an application can use multiple cores and what methods provide the highest efficiency.

In a preliminary evaluation of multi-core processors for scientific computing, we investigated the behavior of micro-benchmarks, macro-benchmarks, and applications running on systems with multi-core AMD Opteron processors [3]. We evaluated not only the processors' computation capabilities, but also the communication capabilities of symmetric multiprocessing (SMP) systems with multi-core processors that might be used as the building blocks of a large parallel system. We discovered that although systems with multi-core processors can be effective for scientific computation, approaches that are aware of the multi-core processors are necessary to achieve highest performance. We evaluated a number of processor affinity techniques for managing memory placement on multi-core systems and discovered that an appropriate selection of MPI task and memory placement schemes can result in over 25% performance improvement for key scientific calculations.

To evaluate computation characteristics of multi-core processors, we measured the performance of several low-level micro-benchmarks and a subset of the NAS Parallel Benchmarks [4] running on test systems with multi-core Opteron processors. We observed the expected effect that performance remained comparable to that of single-core processors when memory accesses were satisfied out of processor cache, but performance degraded when processes running on cores in the same processor socket contended for a common link to memory.

To evaluate the communication performance of multi-core processors, we investigated the intra-node communication behavior of several MPI benchmarks on our multi-core test systems [6, 11]. Although a hybrid programming model that uses OpenMP for parallelism within a node and MPI for parallelism among nodes is often proposed as the best way to use systems with multi-core processors, the traditional MPI programming model is likely to remain important for

portability reasons and to satisfy the requirements of the huge base of existing applications. To evaluate the traditional MPI programming model on systems with multi-core processors, we ran benchmarks, such as the Intel MPI Benchmark suite and the High Performance Computing Challenge (HPCC) benchmark [2] suite, on systems with at least two Opteron multi-core processors.

Based on our observations of the micro-benchmark and kernel results, we experimented with a set of representative applications from bio-molecular and climate simulations [7, 9, 10]. We confirmed that the memory and task placement configurations that result in an optimal performance for scientific kernels provide 10-20% performance improvement for full application runs. This validates our assertion that applications or the system software needs to be aware of multi-core resources and should be configured to exploit these resources.

2 Evaluation Systems

Because no definition of terms, such as "processor," has been widely accepted for multi-core processors, we use the following terminology. A **computing system** is a collection of nodes interconnected with a high-speed network. A **node** is a group of sockets that typically share main memory and perhaps other components of the memory hierarchy, such as cache, and communicate through one or more ports to the high-speed interconnect. A **socket** contains one or more cores, a memory link, and possibly an interconnect link and an IO link. A **core** is the fundamental execution unit in the system, containing functional units, registers, etc.

For our evaluation, we used several systems with AMD Opteron processors; the systems are summarized in Table 1.

Name	Opteron Model	Frequency (Ghz)	Cores per Socket	Sockets per Node	Total Cores per Node	Node Memory Size (GB)	Node Memory Type	OS
Tiger	248	2.2	1	2	2	8	DDR-400	Suse Linux
DMZ	275	2.2	2	2	4	4	DDR-400	RH Linux 2.6.9
Longs	865	1.8	2	8	16	32	DDR-400	RH Linux 2.6.13

Table 1: System Configurations

Longs system is an eight-socket Iwill H8501 server as shown in Figure 1. Each socket consists of a 1.8 GHz dual core AMD Opteron 865 and 4GBytes of dual channel DDR400 memory, connected via a 2x4 HyperTransport ladder topology network. Each core has a 64KB data cache, a 64 KB instruction cache, and a unified 1MB L2 cache. The system runs the Fedora Core 4 distribution of Linux, with a 2.6.13 SMP kernel. All code was compiled using GNU v4 compilers.

DMZ is a cluster of four nodes, each consisting of two dual core AMD 2.2 GHz Opteron 275 processors and 4 GBytes of shared memory, running Red Hat Enterprise Linux 4 update 3. The nodes are not connected by a high performance network, so we limit our experiments to a single node. All code was compiled using GNU v4 compilers.

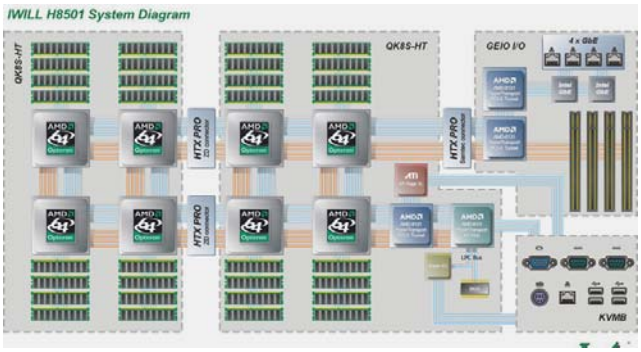


Figure 1: Iwill H8501 system architecture

In contrast to the dual core processor-based DMZ and Longs systems, the Cray XD-1, known as Tiger, consists of 144 single core 2.2 GHz AMD Opteron 248 processors. A node consists of two processors, each capable of 4.4 GFlop/s, with 8 GBytes of memory. The 72 nodes, connected by the Cray RapidArray fabric, run the Linux operating system; however, the compute nodes have a special kernel that allows them to synchronize with a global clock and co-schedule processes to avoid latency in global communication. All code was compiled using the PGI release 6.0.

2.1 Processor and Memory Affinity

A full evaluation of multi-core processors requires the use of *processor affinity*, the capability to specify that processes run only on a specific core or set of cores. Each of our test systems runs the Linux operating system, which provides a few mechanisms for controlling processor affinity. On systems with Non-Uniform Memory Access (NUMA) architectures, such as our AMD Opteron test systems, the `numactl` command controls processor affinity for a process and all of its children processes. It can also be used to control the operating system's memory page placement policy to ensure, for example, that a process' memory pages are always allocated in the memory that is directly attached to the socket that is running the process. Recent Linux kernels (version 2.6 and newer, and even some 2.4 versions) also contain system calls such as `sched_setaffinity` to set processor affinity. In our experiments, we used the `numactl` command to control processor and memory affinities.

3 Microbenchmarks and Kernels

Part of our evaluation of multi-core Opteron processors involved a study of the computation and memory access behavior of multi-core Opteron processors. In this section, we discuss our findings from the results from a collection of benchmarks used as part of the ongoing early systems evaluation effort at the Oak Ridge National Laboratory.

3.1 STREAM: Memory Latency & Bandwidth

STREAM is a benchmark used for determining a system's maximum memory throughput [14]. The LMbench benchmark suite includes an implementation of the STREAM benchmark. Figure 2 and Figure 3 show bandwidth scaling plots based on the LMbench3 STREAM-triad benchmark. Global bandwidth

increases almost linearly as the first core in each socket is activated; however, activating the second core in each socket generally provides flat or degraded performance. Interestingly, a significant difference in bandwidth is observed (especially clear at 1-2 cores, i.e., 1 or 2 sockets active) among the 8 socket system and systems with 2 or 4 sockets.

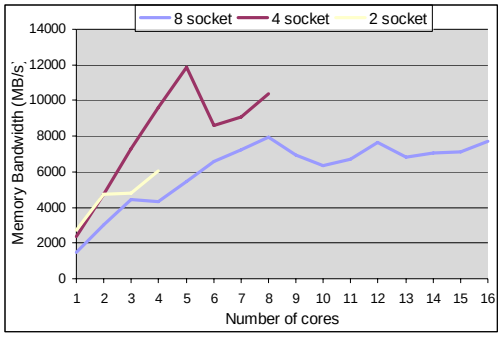


Figure 2: Memory bandwidth

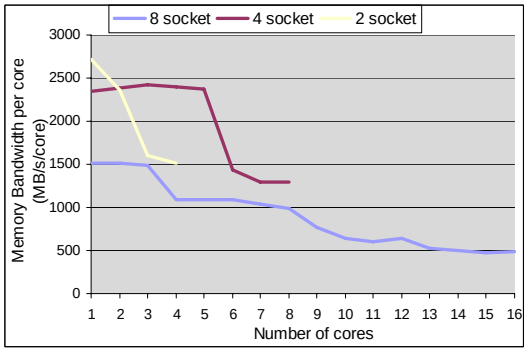


Figure 3: Memory bandwidth per core

3.2 BLAS Level 1 and 3 Operations

Many numerical operations common to scientific computing codes are represented by the Basic Linear Algebra Subprograms (BLAS) library [1]. Most vendors provide BLAS as an optimized system library, which provides a special opportunity for comparing the compiler's ability to optimize Fortran code (we refer to this as the "vanilla" version) against code that the vendor has invested significant effort. Here, we consider vector addition (DAXPY, with alpha=1) and matrix-matrix multiplication (DGEMM, with alpha=1 and beta=0). The high data reuse and cache-friendly nature of the DGEMM provides a significant opportunity for achieving strong performance relative to the theoretical peak capability.

We measured performance both with the vendor optimized implementations found in the AMD Math Core library (ACML) and with a vanilla implementation of the operation. The graphs shown in Figure 4 and Figure 6 compare the performance of DAXPY, and DGEMM operations provided with the ACML library on the DMZ system. These figures show the aggregated performance as well as the performance per core. For example, Total (*n* cores) is the aggregated performance and *n*T (core *x*) is performance of core *x* in an *n* processor run. Figure 5 and Figure 7 show the performance of "vanilla" implementations of the same operations, unoptimized for any particular processor. Here, we compare performance of single vs. two MPI tasks per socket.

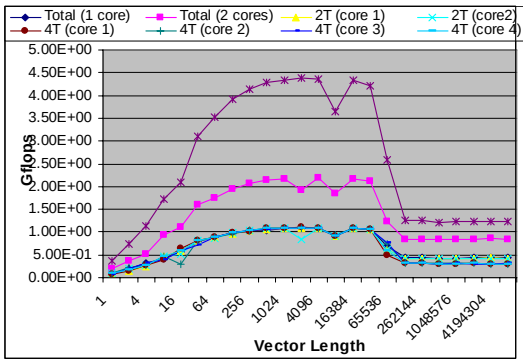


Figure 4: BLAS Level 1 (DAXPY) performance (ACML)

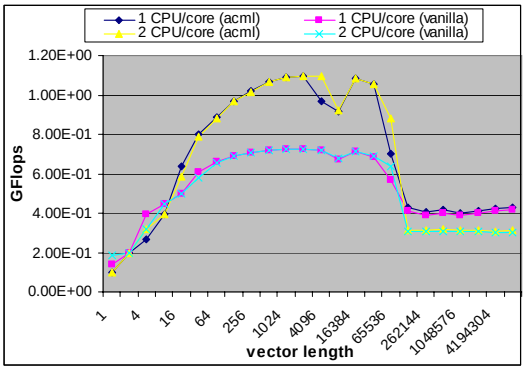


Figure 5: BLAS Level 1 (DAXPY) performance/ core

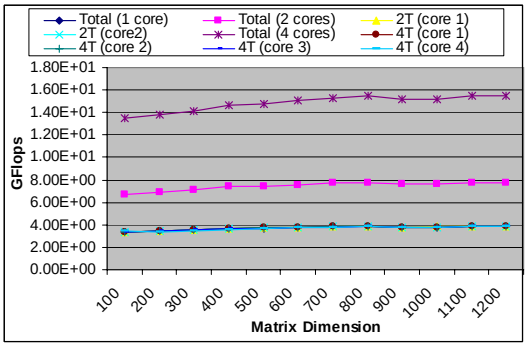


Figure 6: BLAS Level 3 performance (ACML)

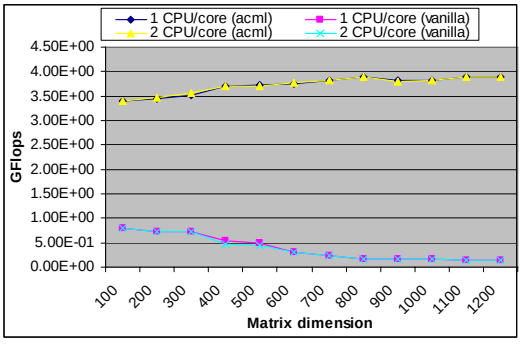


Figure 7: BLAS level 3 performance per core

3.3 HPC Challenge Benchmarks

HPC Challenge is a suite of benchmarks that do not focus tightly on a particular aspect of performance (i.e., computation or communication) but instead represent another step toward fidelity with respect to full scientific applications [2]. We performed this part of our evaluation on the Iwill system that contains eight sockets arranged in a 4x2 mesh topology (also referred to as a ladder), with a dual-core Opteron in each socket. The message passing capability was provided by LAM, version 7.7.1.

HPCC results were generated on 16 cores with a single binary, varying the NUMA memory placement and contrasting these variances with changes to the MPI communication layer. HPL results in Figure 8 demonstrate a variety of interactions between memory placement and the MPI communication layer. There are two memory placement schemes, *localalloc* (which forces pages to be allocated nearest the CPU where the allocation is performed) and *interleave* (which forces pages to be allocated round-robin across the CPUs). There are also several choices for the locking mechanism used in the MPI sub-layer, including SysV that uses System V semaphores, and USysV that uses spin locks. The memory placement schemes have a smaller impact than the selection of MPI sub-layer. Examination of the other HPCC benchmarks will shed further light on these results. Since Longs has a large NUMA domain, it is used to evaluate the NUMA related options (usysv and localalloc). In contrast, the HPCC benchmarks on the DMZ system are minimally affected by different NUMA options because it has a much simpler organization than the Longs system. Hence, we present HPCC benchmark results with six different runtime options on Longs but only one result for the DMZ system.

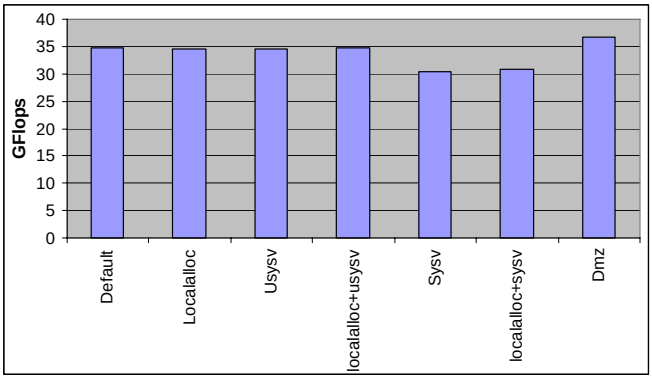


Figure 8: HPL performance with LAM/NUMA options

The computationally intensive kernels DGEMM (evaluated on the DMZ system) and FFT are extremely cache-friendly and are only slightly impacted by memory placement, the MPI sub-layer, and the number of cores actively executing the same algorithm, as shown in Figure 9. The figure shows results for several benchmarks from the HPCC benchmark suite, in both their *Single* mode (where exactly one processor runs the benchmark) and *Star* mode (also known as "embarrassingly parallel" mode, where all processors run the benchmark concurrently but without explicit communication).

Note that the Star DGEMM and Single DGEMM results are almost identical, whereas the somewhat less cache-friendly FFT shows slightly more impact going from Single FFT to Star FFT. The nearly 1:1 ratio between Single DGEMM and Star DGEMM equates to the second core effectively doubling the per socket performance.

The HPCC STREAM benchmark measures memory bandwidth, and thus is greatly impacted by choice of memory placement. Oddly, this benchmark is nominally independent of MPI performance; however, the results show some sensitivity to the choice of MPI sub-layer. Clearly, the MPI sub-layer is affecting page placement. Most surprising is that the impact of running the benchmark across all cores is greater than the expected factor of two in the cases of default placement and MPI sub-layer and localalloc with the USysV sub-layer as shown in Figure 10. With a Single to Star ratio of greater than 2:1, engaging the second core on this memory bandwidth intensive benchmark results in a net performance loss per socket. More disturbing is that the best achievable single core bandwidth on the 8 socket system is less than half of the more than 4 GBytes per second one would typically expect from an Opteron. More details will follow as we drill down on the other benchmarks.

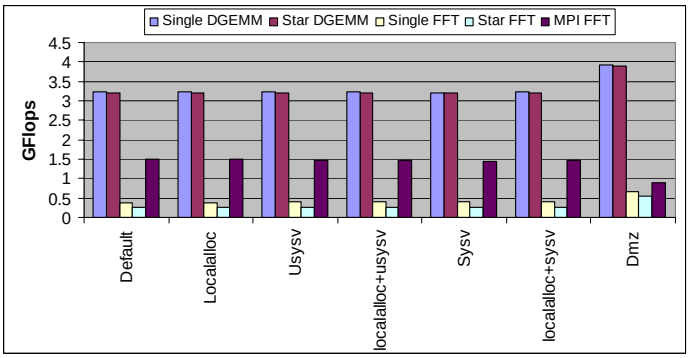


Figure 9: Processor performance with runtime options

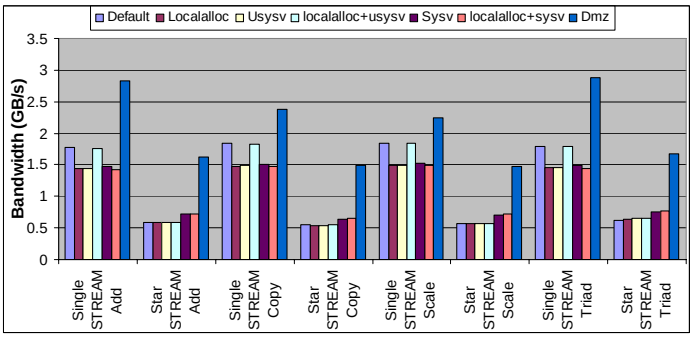


Figure 10: Impact of LAM and NUMA runtime options on memory performance

The RandomAccess (RA) benchmark is designed to measure the performance of the last level of hierarchy of the memory system. The messages sent by the MPI implementation of the RA benchmark are small. Thus the high MPI latency (see below), attributable to the high cost of the Linux implementation of the SystemV semaphore, results in

poor performance of this benchmark**.** Figure 11 demonstrates that the MPI sub-layer appears to impact memory performance for the Single RA benchmark, comparing localalloc to localalloc+USysV, and default to SysV. Notably on each of the benchmarks thus far, the selection of sysv seems to dominate choice of localalloc vs. interleave. Relative to STREAMS, Single and Star RA stronger dependence on memory latency than memory bandwidth impacts the relative performance of single and star RA, with the ratio (less than 2:1) from bringing the second core per socket online creates a net performance gain per socket for the second core.

MPI communication bandwidth is not particularly relevant to our understanding of the performance of dual-core processors; however, comparing the Ring and PingPong bandwidths clearly exposes the topology and congestion effects on the HT8501's HyperTransport ladder. As shown in Figure 12, the PTRANS benchmark demonstrates more extreme differences in performance for SysV and USysV MPI sub-layers, with USysV's spinlocks providing a clear performance advantage. Localalloc does well on its own, but degrades both SysV and USysV when combined. Again, localalloc and the choice of MPI sub-layer are interacting poorly.

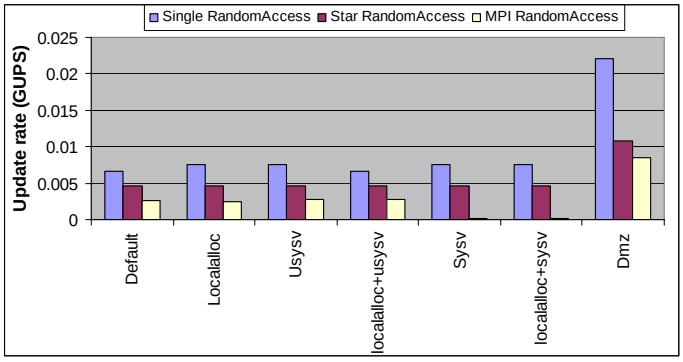


Figure 11: Another view of HPCC memory performance

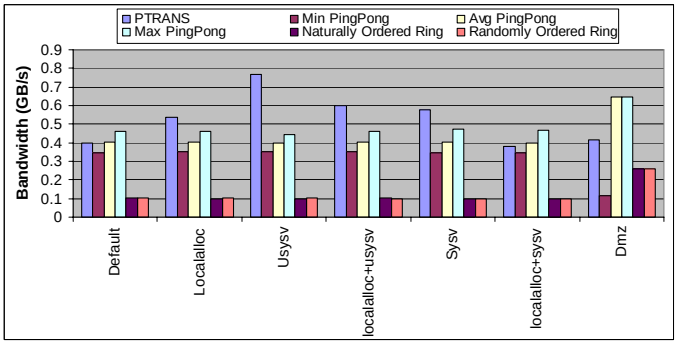


Figure 12: Impact of LAM and NUMA runtime options on communication performance

The MPI latency benchmarks expose information about the HT ladder interconnect. As expected ring latencies are higher than PingPong latencies (see Figure 13). However the differences between these are overwhelmed by the high latencies associated with the SysV MPI sub-layer. Comparing against the MPI-RA benchmark, the key conclusion is that the high SysV latencies have a strong negative impact on performance when the message size is small; however, with larger messages, the impact can be essentially negligible as in MPI-FFT.

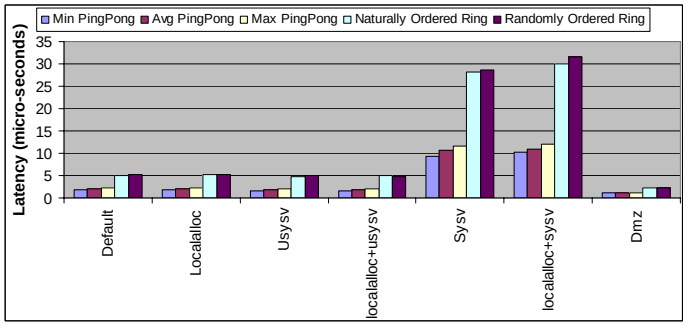


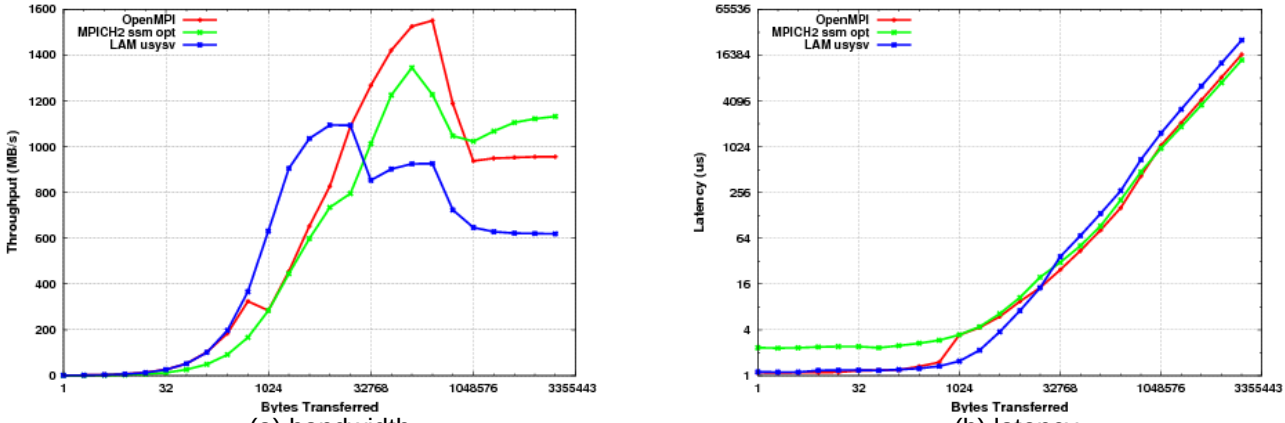
Figure 13: Communication performance

3.4 Intel MPI Benchmarks

On distributed memory systems with multi-core processors, a hybrid programming model consisting of MPI for inter-node communication and OpenMP (or a similar threading approach) is often proposed as the best match for such systems. However, a pure MPI model without intra-node threading is also likely to be used on such systems for the sake of application portability and due to the large base of existing applications written in MPI. Because of the continuing importance of the pure MPI programming model, we examined the behavior of MPI benchmarks and applications on systems similar to those found as nodes in a distributed memory parallel computer with multi-core processors. In this section we discuss the results from the Intel MPI Benchmarks.

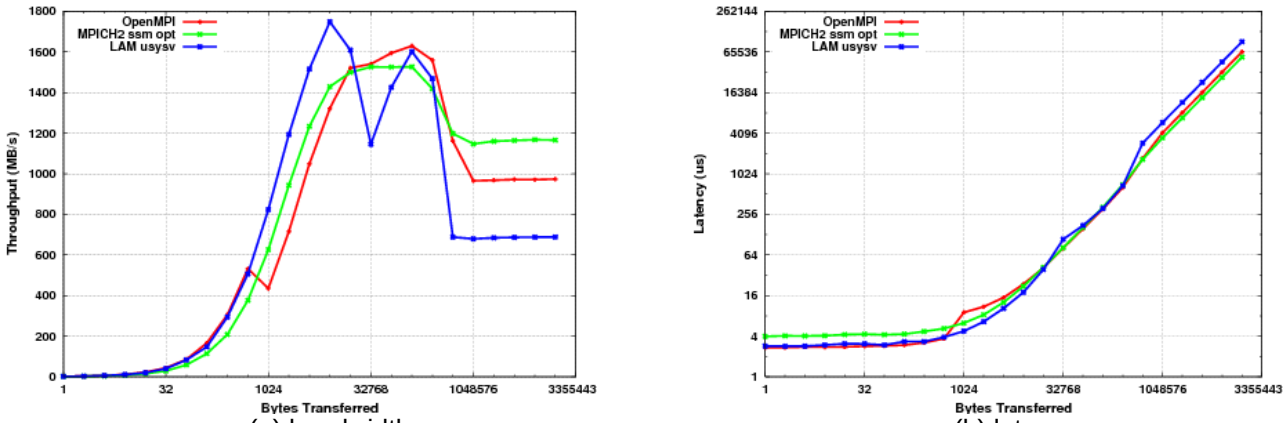
We considered two popular MPI implementations, MPICH2 version 1.0.3 [13] and LAM version 7.1.2 [12]. We also considered OpenMPI version 1.0.1, because it appears to have the potential for widespread adoption going forward. Experiments executed on a DMZ node. Although this system is not part of a larger distributed memory system, this type of system could serve as a node in a cluster or MPP. Ideally, the MPI implementation on a system like this uses an optimized communication mechanism for intra-node communication such as shared memory buffers; each of the MPI implementations we considered has this capability.

Based on these PingPong and Exchange benchmark results (Figure 14 and Figure 15), there is no clear consensus on which MPI implementation makes best use of the shared memory approach for intra-node communication. MPICH2 seems to have a high latency overhead compared to the others for small message lengths, but becomes comparable with the others with messages of approximately 16KB. The bandwidth results also show no clear picture with respect to which MPI implementation best took advantage of the dual-core processors. LAM showed superior performance for messages smaller than 16KB, OpenMPI showed the best performance for intermediate-sized messages, and MPICH was superior for large messages.



(a) bandwidth (b) latency

Figure 14: Intra-node Intel MPI Benchmark PingPong benchmark performance.



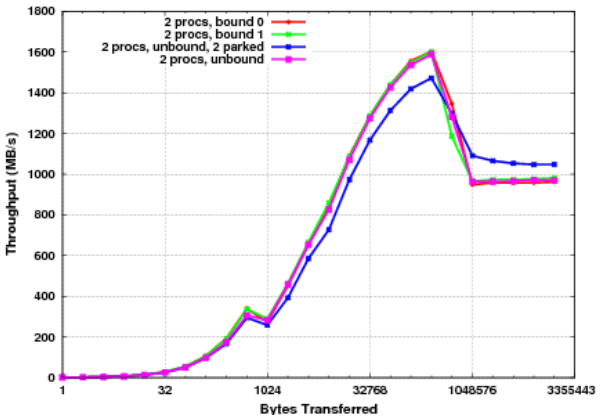
(a) bandwidth (b) latency

Figure 15: Intra-node Intel MPI Benchmark Exchange benchmark performance

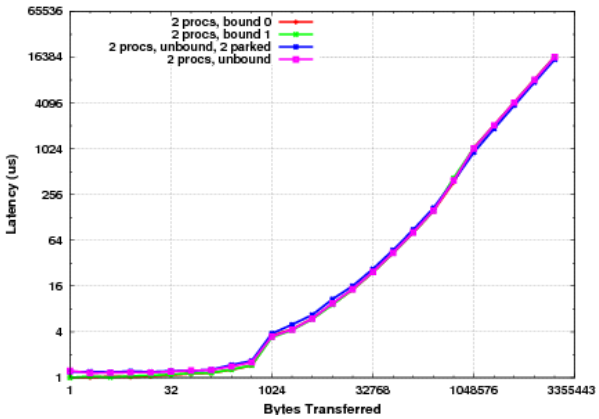
The second part of our investigation focused on the effects of multi-core processors on the intra-node communication behavior of a single MPI implementation. We chose the OpenMPI implementation for these experiments because it exhibited good performance for large messages using its default configuration. These results are shown in Figure 16 and Figure 17 for the PingPong and Exchange benchmarks, respectively. For each benchmark, we report the bandwidth and latency of the operation when run in several different configurations intended to provide insight into the effect of the multi-core processors on intra-node communication. In each plot, the curves labeled "2 procs, bound *n*" indicate a configuration where the benchmark was limited to two processes, and the Linux `numactl` command was used to set the processor affinity and memory allocation policies so that the processes ran only on one or the other of the system's dual-core processors and always allocated memory locally to that processor. The "2 procs, unbound" curve represents a configuration where the benchmark was limited to two processes but used the default processor affinity and memory allocation policy. The "2 procs, unbound, 2 parked" configuration created four processes but only two were involved in the MPI communication operation. Finally, for the Exchange benchmark, a "4 procs" configuration was used that measured performance of a four-processor IMB Exchange operation.

Based on these results, for both benchmarks there is a small but non-negligible bandwidth benefit (approximately 10 to 13%) from confining communication within a multi-core processor. A latency benefit also appears to be present for small messages.

The experiments described in this section show that one must consider the topology of a node with more than one multi-core processor to achieve the highest communication performance. We observed better bandwidth and latency between processes running on the same multi-core processor than between processes running on distinct multi-core processors connected via a coherent HyperTransport link. We also observed the possibility of non-negligible communication degradation on systems with a relatively large number of multi-core processors (eight dual-core processors in our case) arranged in a 2D mesh topology, suggesting that cache coherence traffic imposes a limit on the number of processor sockets each node should contain for best performance.

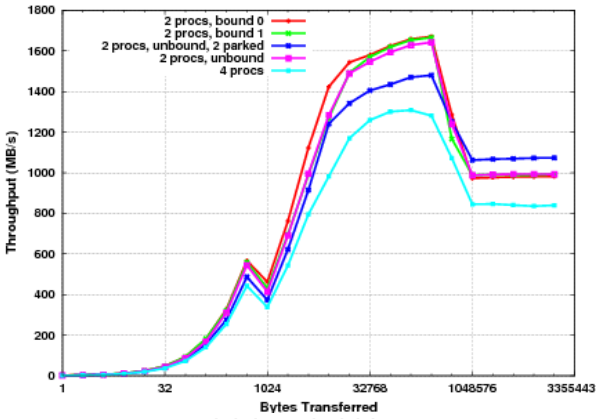


(a) bandwidth

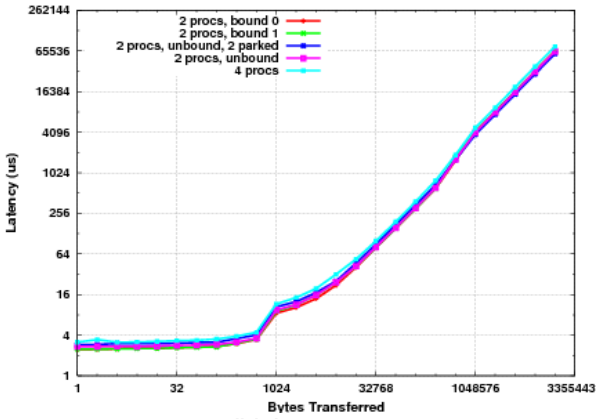


(b) latency

Figure 16: Intra-node OpenMPI PingPong benchmark performance with scheduler affinity



(a) bandwidth



(b) latency

Figure 17: Intra-node OpenMPI Exchange benchmark performance with scheduler affinity.

Combining these two observations suggests that for best performance we must view systems that use multi-core processors as having not two classes of communication channels, but three: the system interconnect, the links between processor sockets within an SMP node, and the links within each multi-core processor. A programming model using OpenMP only within each multi-core processor, and MPI for communication both between processor sockets and between system nodes might be a high-performance alternative that best exploits the three classes of communication performance found in systems with SMP nodes and multi-core processors.

3.5 NAS Parallel Benchmarks

The NAS Parallel Benchmark (NPB) Suite consists of several small programs derived from computational fluid dynamics applications [4]. Using the MPI version from NAS benchmark distribution 3.2, the experiments reported here were performed using the class B problem sets. The code was compiled using gnu4 on DMZ and Longs and PGI on Tiger. The MPICH2 library provided the message passing capability on all platforms.

Table 4 shows the speedup for two representative kernels (CG and FT) relative to single processor performance. Because a goal of this work is to examine intra-node communication behavior, and there are four total cores in our DMZ test system, we used at most four MPI tasks for the DMZ tests. Note that this benchmark suite tests the strong scaling capabilities of a computer, and thus we can (and do here) see speedups greater than 1.0.

As anticipated, the scaling within a system largely depends on the underlying characteristics of the algorithms. Scaling of three different dual-core Opteron systems are consistent. At the same time, the HT ladder in the Longs system prevents the two benchmarks from sustaining speedup on 8 and 16 processors. Since CG and FFT calculations are widespread in a number of scientific applications, we investigated the effects of processor and memory affinity techniques for these two calculations on the Longs and the DMZ system.

Number of MPI tasks	Kernel	Default	One MPI + Local Alloc	One MPI + Membind	Two MPI + Local Alloc	Two MPI + Membind	Interleave
2	CG	162.81	162.68	162.72	172.08	170.79	190.18
4	CG	98.51	88.21	111.02	102.94	99.54	109.93
8	CG	50.93	51.15	109.11	49.24	115.87	67.23
16	CG	54.17	—	—	54.45	121.87	72.62
2	FFT	118.97	118.56	123.15	129.18	129.12	137.79
4	FFT	79.96	67.72	91.84	74.38	92.79	84.89
8	FFT	42.32	39.96	69.79	62.80	81.95	47.13
16	FFT	30.77	—	—	31.36	63.39	41.48

Table 2: Effect of *numactl* options on NAS CG and FT benchmarks on the Longs system. Times listed in seconds.

Number of MPI tasks	Kernel	Default	One MPI + Local Alloc	One MPI + Membind	Two MPI + Local Alloc	Two MPI + Membind	Interleave
2	CG	106.8	106.24	125.87	111.17	111.20	115.02
4	CG	59.22	—	—	68.16	86.93	66.74
2	FFT	93.58	100.84	115.42	108.30	101.18	105.13
4	FFT	57.05	—	—	57.03	75.50	63.67

Table 3: Impact of *numactl* options on NAS CG and FT benchmarks performance. Times listed in seconds

Table 5 lists a combination of memory and processor affinity schemes that are used in running the NAS parallel benchmark and scientific applications experiments.

Benchmark	System	2 cores	4 cores	8 cores	16 cores
CG	DMZ	1.07	0.86	—	—
CG	Longs	1.07	0.73	0.52	0.25
CG	Tiger	1.01	—	—	—
FT	DMZ	0.82	0.64	—	—
FT	Longs	0.85	0.69	0.62	0.42
FT	Tiger	0.88	—	—	—

Table 4: Multi-core speedup for NAS benchmarks

Name	Description
Default	Default (no numactl)
One MPI+Local Alloc	One MPI task per socket and local allocation policy
One MPI+Membind	One MPI task per socket with explicit memory binding per core
Two MPI+Local Alloc	Two MPI tasks per socket and local allocation policy
Two MPI+Membind	Two MPI tasks per socket with explicit memory binding per core
Interleave	Interleaved memory allocation

Table 5: *numactl* options used for experiments

The runtime performance for CG and FFT (Class B) benchmarks for Longs and DMZ is listed in Table 2 and Table 3 respectively. Note that as the number of cores increases (one MPI task per core) the workload per core decreases and the total number of inter-process communication messages increases. On the Longs system experiments are run so as to minimize the effect of the HT ladder (or the number of communication hops) on up to four cores. For example, we have used nodes 2, 3, 4, and 5 to run four single task/core experiments and 8 two tasks/core experiments. Overall, the FT benchmark is found to be more sensitive to the memory placement techniques as compared to the CG benchmark. On the Longs system, one task/core with localalloc option provides the best performance for the two benchmarks. Memory interleaving and forcing membind (allocate memory from nodes) and cpubind (execute process on CPU of nodes) processors result in worst-case performance for almost all test cases.

4 Applications

4.1 Molecular Dynamics Simulations

Molecular dynamics (MD) simulations enable the study of complex, dynamic processes that occur in biological systems [8]. MD methods are now used routinely to investigate the structure, dynamics, functions, and thermodynamics of biological molecules and their complexes. The types of biological activity that have been investigated using MD simulations include protein folding, enzyme and DNA, and biological membrane complexes. Biological molecules exhibit a wide range of time and length scales over which specific processes occur, hence the computational complexity of an MD simulation depends greatly on the time and length scales considered. With an explicit solvation model, typical system sizes of interest range from 20K atoms to more than 1 million atoms; if the solvation is implicit, sizes range from a few thousand atoms to about 100K. The simulation time period can range from pico-seconds to the a few micro-seconds or longer.

Several commercial and open source MD software frameworks are in use by a large community of biologists, including AMBER and LAMMPS. These packages differ in the form of their potential function and also in their force-field calculations. Some of them are able to use force-fields from other packages as well. The version of LAMMPS used in our evaluation does not use the energy minimization technique.

Benchmark	dhfr	factor_ix	gb_cox2	gb_mb	JAC
Number of atoms	22,930	90,906	18,056	2,492	23,558
MD technique	PME	PME	GB	GB	PME

Table 6: Description of AMBER benchmarks

MPI tasks	System	Default	One MPI + Local Alloc	One MPI + Membind	Two MPI + Local Alloc	Two MPI + Membind	Interleave
2	Longs	3.13	2.76	3.13	3.3	3.31	3.50
4	Longs	1.83	1.45	1.78	1.48	1.77	1.75
8	Longs	0.81	0.82	1.17	0.77	1.01	0.85
16	Longs	0.63	—	—	0.57	1.32	2.22
2	DMZ	1.81	1.77	2.39	2.25	2.25	1.96
4	DMZ	1.03	—	—	1.08	1.51	1.09

Table 7: FFT Performance in the JAC benchmark. Times listed in seconds.

Number of cores	System	dhfr	factor_ix	gb_cox2	gb_mb	JAC
2	DMZ	1.90	1.91	1.98	1.98	1.96
4	DMZ	3.45	3.35	3.92	3.94	3.63
2	Longs	1.95	1.89	1.98	2.06	1.93
4	Longs	3.63	3.43	3.92	4.07	3.78
8	Longs	6.02	5.94	7.63	7.96	6.22
16	Longs	7.24	7.35	14.29	14.93	7.97

Table 8: AMBER PME multi-core speedup with no `numactl` option

AMBER consists of about 50 programs that perform a diverse set of calculations for system preparation, energy minimization (EM), molecular dynamics (MD), and analysis of results [9]. AMBER's main module for EM and MD is known as *sander* (for *s*imulated *a*nnealing with *N*MR-*d*erived *e*nergy *r*estraints). We used sander to investigate the performance characteristics of EM and MD techniques using the Particle Mesh Ewald (PME) and Generalized Born (GB) methods. Table 6 lists five benchmarks that are part of AMBER 8.0 release. The PME benchmarks use FFT calculations as part of the reciprocal PME calculations, while GB calculations are more computation-intensive as compared to the PME calculations. Table 8 lists the AMBER PME benchmark speedup across multiple cores for DMZ and Longs. Both MD techniques manage to utilize the dual-core resources efficiently as we observe near linear scaling on up to 4 cores systems for PME method and on up to 16 cores for the GB calculations.

Since we observed a notable difference in the NAS FFT calculation using processor and memory affinity techniques, we anticipated a performance improvement on AMBER PME simulation runs, not only in its FFT calculations but also in overall simulation runtime. Table 7 lists the variations in FFT performance and Table 9 lists the overall runtime as a function of different processor and memory placement techniques on the JAC benchmark on the DMZ and Longs systems. As we noted in the NAS benchmark runs, the processor and memory affinity techniques significantly influence performance on the Longs system. We observed similar behavior on a full-scale application run. Likewise we demonstrate that the default option on the DMZ system is sufficient to obtain near optimal runtimes for NAS FFT benchmark. This is also true for AMBER benchmark runs.

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a classical MD code. LAMMPS models an ensemble of particles in a liquid, solid or gaseous state and can be used to model atomic, polymeric, biological, metallic or granular systems [10]. We used the latest (2006) C++ and MPI version and ran the following three benchmarks: Lennard-Jones (LJ), Polymer (chain), and Metal (eam).

The above benchmarks contain 32,000 atoms each and run for 100 simulation time steps. We ran the experiments on the DMZ, Longs and Tiger systems. These results are listed in Table 10. The LAMMPS benchmarks scale linearly on multiple cores and the scaling behavior is increasingly different for different classes of computation. For instance, the chain calculations perform local, point-to-point interactions, while the LJ benchmark calculates the overall energy effect using the non-local calculations. The scaling behavior is consistent across different dual-core Opteron system configurations. Table 11 lists the impact of different processor and memory affinity techniques on the LJ calculations that perform FFT operations. The performance impacts are similar to what we observed in AMBER.

4.2 Parallel Ocean Program (POP)

POP is the ocean component of Community Climate System Model (CCSM) [5, 7]. The code is based on a finite-difference formulation of the three-dimensional flow equations on a shifted polar grid. In its high-resolution configuration, 1/10-degree horizontal resolution, the code resolves eddies for effective heat transport and the locations of ocean currents.

MPI tasks	System	Default	One MPI + Local Alloc	One MPI + Membind	Two MPI + Local Alloc	Two MPI + Membind	Interleave
2	Longs	38.08	35.21	35.63	35.91	36.75	36.99
4	Longs	20.18	18.70	19.72	18.83	19.63	19.97
8	Longs	11.47	11.39	13.85	11.12	13.42	12.06
16	Longs	8.96	—	—	8.95	14.71	14.99
2	DMZ	27.05	26.30	28.08	28.01	27.59	27.27
4	DMZ	14.38	—	—	14.44	16.08	14.74

Table 9: Overall performance of the JAC benchmark. Times listed in seconds.

Number of cores	System	LJ	Chain	EAM
2	DMZ	1.79	2.13	1.96
4	DMZ	3.61	4.41	3.60
2	Longs	1.89	2.23	1.82
4	Longs	3.51	5.53	3.45
8	Longs	6.63	11.52	6.74
16	Longs	10.65	19.95	12.54
2	Tiger	1.92	2.13	1.87

Table 10: LAMMPS benchmark: Multi-core speedup (no *numactl*)

MPI tasks	System	Default	One MPI + Local Alloc	One MPI + Membind	Two MPI + Local Alloc	Two MPI + Membind	Interleave
2	Longs	3.82	3.6	3.76	3.73	3.73	3.93
4	Longs	1.95	1.87	1.99	2.52	2.99	2.03
8	Longs	1.03	1.02	1.11	1.97	1.067	1.05
16	Longs	0.63	—	—	0.63	0.77	0.64
2	DMZ	3.07037	2.89618	3.10457	3.00691	3.00305	2.96663
4	DMZ	1.55389	—	—	1.53995	1.73746	1.58052

Table 11: LAMMPS benchmark: Impact of *numactl* options on overall performance

POP performance is characterized by the performance of two phases: baroclinic and barotropic. The baroclinic phase is three dimensional with limited nearest-neighbor communication and typically scales well on all platforms. In contrast, runtime of the barotropic phase is dominated by the iterative solution of a two-dimensional, implicit system using a conjugate gradient method. The performance of the barotropic solver is very sensitive to network latency and typically scales poorly on all platforms.

For our evaluation we used version 1.4.3 of POP and a POP benchmark configuration called x1,'which represents a relatively coarse resolution similar to that currently used in coupled climate models. The horizontal resolution is roughly one degree (320×384). The vertical coordinate uses 40 vertical levels.

Number of cores	System	Baroclinic	Barotropic
2	DMZ	2.04	2.07
4	DMZ	3.87	3.99
2	Tiger	1.97	1.93
2	Longs	2.02	2.002
4	Longs	4.08	4.07
8	Longs	8.26	8.28
16	Longs	16.11	14.85

Table 12: POP multi-core speedup.

Using the embedded timers, we measured performance and scaling of baroclinic and barotropic calculation phases. All timings are presented in seconds, for a 50 time-step or 2-day simulation run. Table 12 lists the speedups across multiple cores for POP runs on the DMZ, Tiger, and Longs systems. Although the baroclinic process is relatively more computationally expensive than the barotropic process, both phases of calculations scale almost linearly on dual-core Opteron systems.

Conjugate Gradient (CG) calculations are performed as part of the Barotropic calculations in POP. Like FFT, NAS CG benchmarks show sensitivity to the processor and memory affinity schemes, therefore, we conducted experiments on the DMZ and Longs systems with combinations of `numactl` options. Table 13 shows the impact of the `numactl` options on the Baroclinic calculations and Table 14 lists barotropic runtimes on the DMZ and Longs system. Note that the number of MPI invocations and message volume are much higher in the barotrpic phase as compared to the baroclinic phase. Therefore, the effect of the HT ladder in the Longs system results in higher performance when there are 2 MPI tasks per core as compared to one MPI task per core. On 2 and 4 MPI tasks runs, the CG options that provided higher performance number on the NAS CG benchmark, shows a similar effect on the barotropic calculations.

MPI tasks	System	Default	One MPI + Local Alloc	One MPI + Membind	Two MPI + Local Alloc	Two MPI + Membind	Interleave
2	Longs	358.57	332.29	343.89	354.01	354.62	408.66
4	Longs	177.64	163.37	191.78	169.08	275.91	194.99
8	Longs	87.58	86.61	118.87	84.5	184.33	98.09
16	Longs	44.93	—	—	44.9	75.96	57.08
2	DMZ	301.82	284.53	326.43	316.36	305.34	306.05
4	DMZ	150.15	—	—	154.03	199.51	156.79

Table 13: Impact of `numactl` on POP baroclinic execution time (in seconds).

MPI tasks	System	Default	One MPI + Local Alloc	One MPI + Membind	Two MPI + Local Alloc	Two MPI + Membind	Interleave
2	Longs	36.13	34.35	35.12	37.28	37.37	41.41
4	Longs	17.75	17.08	20.3	17.51	34.92	19.29
8	Longs	8.74	10.06	10.41	8.96	21.99	9.31
16	Longs	4.87	—	—	4.23	4.55	4.36
2	DMZ	29.78	26.18	29.68	30.40	28.21	29.84
4	DMZ	13.76	—	—	13.94	17.55	14.33

Table 14: Impact of `numactl` on POP barotropic execution time (in seconds).

5 Summary and Conclusions

We considered multi-core processors using computation and communication micro-benchmarks, and using higher-level benchmark suites that reach toward full scientific applications. We observed a significant benefit (approximately 8% to 12%) when communicating between processes running within a multi-core processor as opposed to between cores on different processors, indicating an opportunity for a programming model (or an implementation of an existing programming model) that is multi-core processor-aware. Most disappointing were the scalability issues inherent in our eight-socket dual core Opteron system Longs, suggesting that the current cache coherence scheme is not sufficient to sustain the full memory bandwidth capability of the memory parts or the Opteron's memory interface. We expect that this is a function of the maturity level of the architecture and/or compilers and that a combination of improved prefetching and latency hiding in future compilers and improvements in future Opteron products will improve the scalability. The DGEMM, FFT, and RandomAccess results show that while adding a second core to a socket decreases the per core performance (varying depending on cache hit rate), it does increase the overall per socket performance (again, varying depending on the cache hit rate). The impact on STREAM performance, however, was significantly worse than expected, in that adding the second core resulted in an overall decrease not only in per core performance, but also in per socket (overall) performance. From our initial results, we conclude that dual core processors are generally worth the investment in 1, 2, and 4 socket configurations, but that current 8 socket configurations should be reserved to those application classes which exhibit extremely high cache locality as exemplified by DGEMM. The odd interactions between the page placement algorithms and the MPI sublayers are only observable on the 8 socket configurations, leading us to believe that the cache coherence issues noted on the Longs platform negatively impact the benchmark behavior and predictability. Investigation and understanding of the micro-benchmarks and scientific kernels results subsequently informed analysis of large-scale applications that are discussed in this paper.

References

1. Basic Linear Algebra Subprograms (BLAS), http://www.netlib.org/blas/
2. HPC Challenge Website, http://icl.cs.utk.edu/hpcc
3. "Software Optimization Guide for AMD Athlon™ 64 and AMD Opteron™ Processors," Technical Manual 25112, 2004.
4. D. Bailey, E. Barszcz *et al.*, "The NAS Parallel Benchmarks (94)," NASA Ames Research Center, RNR Technical Report RNR-94-007, 1994, http://www.nas.nasa.gov/Pubs/TechReports/RNRreports/dbailey/RNR-94-007/RNR-94-007.html
5. M.B. Blackmon, B. Boville *et al.*, "The Community Climate System Model," *BAMS*, 82(11):2357--76, 2001.
6. P. Ekman and P. Mucci, "Design Considerations for Shared Memory MPI Implementations on Linux NUMA Systems: An MPICH/MPICH2 Case Study," AMD, 2005
7. P.W. Jones, P.H. Worley *et al.*, "Practical performance portability in the Parallel Ocean Program (POP)," *Concurrency and Computation: Experience and Practice*, 2005; 17:1317-1327.
8. M. Karplus and G.A. Petsko, "Molecular dynamics simulations in biology," *Nature*, 347, 1990.
9. D.A. Pearlman, D.A. Case *et al.*, "AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules," *Computer Physics Communication*, 91, 1995.
10. S.J. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," in *Journal of Computational Physics*, vol. 117, 1995.
11. M. Snir, W.D. Gropp *et al.*, Eds., *MPI--the complete reference (2-volume set)*, 2nd ed. Cambridge, Mass.: MIT Press, 1998.
12. LAM, http://www.lam-mpi.org
13. MPICH, http://www-unix.mcs.anl.gov/mpi/mpich
14. STREAM benchmarks, http://www.cs.virginia.edu/stream