Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures

Jeffrey S. Vetter
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California, USA 94551
vetter3@llnl.gov

Frank Mueller
Department of Computer Science
North Carolina State University
448 EGRC, Raleigh, NC 27695
mueller@cs.ncsu.edu

Abstract

This paper examines the explicit communication characteristics of several sophisticated scientific applications, which, by themselves, constitute a representative suite of publicly available benchmarks for large cluster architectures. By focusing on the Message Passing Interface (MPI) and by using hardware counters on the microprocessor, we observe each application's inherent behavioral characteristics: point-to-point and collective communication, and floating-point operations. Furthermore, we explore the sensitivities of these characteristics to both problem size and number of processors. Our analysis reveals several striking similarities across our diverse set of applications including the use of collective operations, especially those collectives with very small data payloads. We also highlight a trend of novel applications parting with regimented, static communication patterns in favor of dynamically evolving patterns, as evidenced by our experiments on applications that use implicit linear solvers and adaptive mesh refinement. Overall, our study contributes a better understanding of the requirements of current and emerging paradigms of scientific computing in terms of their computation and communication demands.

1 Introduction

Historically, users have written scientific applications for large distributed memory computers using explicit communication as the programming model. This trend crystallized with the creation of the Message Passing Interface (MPI) specification [11, 22], which simplified numerous issues for both application developers and system designers. As a result, application developers stabilized on the MPI programming model and this has facilitated the ongoing development of a considerable number of applications based on MPI. Although MPI provides a common foundation for explicit communication, its wide range of functionality promotes a diverse set of application communication characteristics due to variations in application domain, algorithm, software design, and problem size.

Nevertheless, these communication characteristics [9] are critically important to the design of large scale computing systems for three reasons. First, design tradeoffs for any computer architecture hinge on specific properties of the system's proposed workload. Second, application developers must use algorithms appropriate for their target system architecture. Third, system software, such as the MPI library, must be optimized for the target architecture *and* the application workload.

1.1 Key Insights and Contributions

The main objective of our efforts is to quantify the communication characteristics of several scientific applications from the perspective of MPI and independent of the target architecture. In particular, for a wide range of existing scientific applications, we quantify their inherent behavioral characteristics: point-to-point communication, collective communication, and floating-point operations. To expose the key relationships among experiment parameters, we also study the effects of scaling both the problem size and the number of processors. Our experiments include applications that simulate radiation transport, turbulence, materials modeling, and fluid dynamics. We also compare and contrast an adaptive mesh refinement framework against traditional uniform mesh applications.

Earlier work [9] claimed a wide range of communication characteristics across a set of smaller applications. Our findings strengthen these results and we contribute several new observations for communication characteristics, such as small collective payload sizes, which is strikingly consistent across applications. In addition, we highlight the impact of adaptive methods on communication requirements.

MPI provides a unique opportunity to study these aspects. First, although applications can use a variety of communication routines to achieve similar types of communication, users typically strive to minimize the amount of communication. Second, MPI provides higher levels of abstraction that hide implementation complexity. This allows us to identify complex operations, such as reductions, which previous studies were not able to consider.

The core of this paper discusses these issues in more detail. In Section 2, we outline our experiment methodology. Following this, we introduce our applications in Section 3. Then, in Section 3.5, we present the results of our evaluation and describe our important observations. Section 5 describes related work. Finally, Section 6 concludes.

2 Methodology

We empirically evaluated five scientific applications on one platform; our results are not from simulation or analytical modeling. In order to obtain the results presented later in the evaluation section, we created a list of important characteristics that we wished to quantify. We then analyzed each application with a number of experiments to capture characteristics of interest, varying parameters, such as problem size, to explore relationships among characteristics.

We characterize our applications along four dimensions: point-to-point communication, collective communication, memory load operations, and floating point operations.

- For point-to-point communication, we measure distributions for number of messages, type, payload size, and size of destination clique.
- For collective communication, we determine the distributions for type, frequency, and payload size.
- To understand the amount of computation in the application, we measure the number of memory load operations and the number of floating point operations between significant MPI call sites.

In addition, we expose how these four dimensions scale with both input problem size and the number of tasks.

2.1 Platform

We ran our tests on an IBM SP system, located at Lawrence Livermore National Laboratory. This machine is composed of sixteen 222 MHz IBM Power3 8-way SMP nodes, totaling 128 CPUs. Each processor has three integer units, two floating-point units, and two load/store units. Its 64 KB L1 cache is 128 way associative with 32 byte cache lines and L1 uses a round-robin replacement scheme. The L2 cache is 8 MB in size, which is four-way set associative with its own private cache bus. At the time of our tests, the batch partition had 15 nodes and the operating system was AIX 4.3.3. Each SMP node contains 4GB main memory for a total of 64 GB system memory. A Colony switch--a proprietary IBM interconnect--connects the nodes. We compiled the various tests with the IBM XL and KAI Guide compilers using IBM's MPI library in user-space mode. Our test jobs ran on dedicated nodes, although other jobs were concurrently using the network.

2.2 Data collection

At the highest level, we empirically measure our data by tracing both the MPI and computation activity during execution. For communication, we record all MPI operations with their respective parameters. For computation, we use hardware counters on the microprocessor to capture specific data about each block of computation between significant MPI call sites. This strategy allows us to collect relevant yet limited information about application communication and computation.

Figure 1 depicts the five steps of our data collection process. During execution, the tracer on each task records fixed-sized events to a local memory buffer (steps ① and ②). When this memory buffer is filled, the tracer writes this information to a file stored on the node's local disk (step ③). Many of our applications never fill their local buffer, so they never spill to local disk. At the end of application execution, the tracer collects these events from each node and merges them into one trace file (step ④). We then analyzed the trace files offline (step ⑤). Most trace-based performance analysis systems, including PICL, Pablo, Tau, and Paraver [10, 16, 19, 21], use this approach.

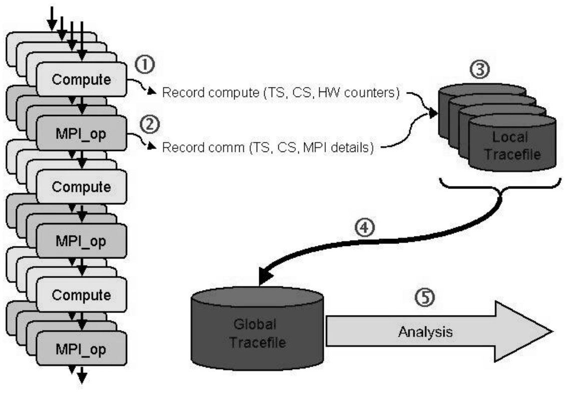

Figure 1: Tracing Infrastructure

For communication activity, our tracing system takes advantage of MPI's profiling layer by capturing information about each MPI call. For each MPI call site, the tracer captures the type of MPI call, parameters for that call, timestamp, call duration, and call site stacktrace. This provides sufficient information to identify different communication phases.

For computation, we capture data from hardware counters periodically. This measurement paradigm provides precise information with low overhead and at a sufficient level of granularity.

To capture this data, we rely on eight hardware counters in the IBM Power3 and program them to count events of interest to our study. First, we capture the number of cycles and completed instructions. Second,

Table 1: Application Overview

Application	Language	Problem	Observed phase of application execution	Primary MPI Funcationality
sPPM	F77	3-D gas dynamics problem on a uniform Cartesian mesh using a simplified version of the Piecewise Parabolic Method	One double timestep.	MPI_Allreduce MPI_Isend MPI_Irecv MPI_Wait
SMG2000	C	Semicoarsening multigrid solver for linear systems.	Solve of one linear system including setup of linear system.	MPI_Allreduce MPI_Isend MPI_Irecv MPI_Wait MPI_Waitall
SPHOT	F77	2-D photon transport code using Monte Carlo transport	One timestep.	MPI_Barrier MPI_Irecv MPI_Reduce MPI_Send MPI_Waitall
Sweep3D	F77	Solver for the 3-D, time-independent, particle transport equation on an orthogonal mesh using a multidimensional wavefront algorithm	One timestep.	MPI_Allreduce MPI_Bcast MPI_Send MPI_Recv
Samrai	C++	3-D shock tube implemented with structured adaptive mesh refinement	One problem at two non-consecutive timesteps.	MPI_Allreduce MPI_Isend MPI_Irecv MPI_Test MPI_Wait

we capture the number of floating point operations, which are typically less sensitive to compiler optimization than other instructions. Third, we measure the number of memory loads. From this set of hardware events, we can calculate valuable measures that include cycles per instruction and flop to load ratio.

This accurate information has been carefully selected to allow us to reduce the size of our trace files while still allowing us to relate computation to communication. Furthermore, we can use this information to determine scaling effects for computation empirically. In this work, we define a block of computation as any work that occurs between two significant MPI call sites. We distinctly identify these blocks by using the call site stacktraces.

2.3 Application Phases

Virtually all scientific applications maintain a notion of *simulation time* and for many applications, the communication and computation activity for each timestep is static. For this reason, we focus our measurements on the activity for one timestep of each application. For those applications that have changing communication patterns [20], such as adaptive mesh refinement, we pay special attention, and report the communication characteristics for several different timesteps of the application.

3 Applications

For our investigation, we targeted a substantial number of very sophisticated scientific applications. Table 1 provides an overview of our applications. The *language* for the application represents the bulk of the languages used in the application source code, although most of these complex applications are mixed language. *Observed phase of application execution* identifies the specific phase of application's execution we measured. *Primary MPI functionality* shows the significant MPI calls detected during the observed phase. Table 2 summarizes the predominantly used MPI routines and illustrates that only a small subset of the rich MPI API is commonly used. The respective references provide more detail on each application. In addition, the source code for each application is also available from the ASCI Purple Benchmark website (www.llnl.gov/asci/platforms) with the exception of SAMRAI, which is available from CASC (www.llnl.gov/CASC).

Routines	sPPM	SMG2000	SPHOT	Sweep3D	Samrai
MPI_Allreduce	X	X		X	X
MPI_Barrier			X		
MPI_Bcast				X	
MPI_Irecv	X	X	X		X
MPI_Isend	X	X			X
MPI_Recv				X	
MPI_Reduce			X		
MPI_Send			X	X	
MPI_Test					X
MPI_Wait	X	X			X
MPI_Waitall		X	X		

Table 2: MPI Routines Used

3.1 sPPM

sPPM [18] solves a 3-D gas dynamics problem on a uniform Cartesian mesh, using a simplified version of the Piecewise Parabolic Method. The algorithm makes use of a split scheme of X, Y, and Z Lagrangian and remap steps, which are computed as three separate sweeps through the mesh per timestep. Message passing provides updates to ghost cells from neighboring domains three times per timestep.

3.2 SMG2000

SMG2000 [4] is a parallel semicoarsening multigrid solver for the linear systems arising from finite difference, finite volume, or finite element discretizations of the diffusion equation $\nabla \cdot (D\nabla u) + \sigma u = f$ on logically rectangular grids. The code solves both 2-D and 3-D problems with discretization stencils of up to 9-point in 2-D and up to 27-point in 3-D. Applications where such a solver is needed include radiation diffusion and flow in porous media. Our examination includes both the setup of the linear system and the solve itself. Note that this setup phase can often be done just once, thus amortizing the cost of the setup phase over many timesteps. This trait is relatively common in implicit timestepping codes.

3.3 Sphot

Sphot is a 2-D photon transport code. Photons are born in hot matter, and tracked through a spherical domain that is cylindrically symmetric on a logically

App	Number of MPI Tasks		Instructions (M)		Load Instructions (M)		Floating point operations (M)		Avg number of messages sent		Avg Send volume (MB)		Avg number of distinct destinations		Number of collectives issued		Collective Payload Size (bytes)	
sPPM	32	1.00	10554	2.91	2652	2.92	6887	2.93	30	0.88	26.3	1.87	5	0.88	2	1.00	14	1.00
sPPM	48	1.50	7081	1.95	1773	1.95	4617	1.97	32	0.94	20.2	1.43	5.33	0.94	2	1.00	14	1.00
sPPM	64	2.00	5356	1.48	1340	1.47	3483	1.48	33	0.97	17.2	1.22	5.5	0.97	2	1.00	14	1.00
sPPM	80	2.50	4317	1.19	1081	1.19	2800	1.19	33	0.97	15.3	1.09	5.6	0.99	2	1.00	14	1.00
sPPM	96	3.00	3630	1.00	909	1.00	2349	1.00	34	1.00	14.1	1.00	5.67	1.00	2	1.00	14	1.00
SMG2000	32	1.00	177	1.11	52	1.11	0.4	4.00	16722	1.09	2.2	0.76	23.5	0.37	15	1.00	8.11	0.99
SMG2000	48	1.50	171	1.08	50	1.06	0.3	3.00	16535	1.08	2.5	0.86	35.75	0.56	15	1.00	8.25	1.01
SMG2000	64	2.00	168	1.06	49	1.04	0.2	2.00	16444	1.07	2.7	0.93	41.88	0.65	15	1.00	8.17	1.00
SMG2000	80	2.50	164	1.03	48	1.02	0.2	2.00	15787	1.03	2.8	0.97	55.35	0.86	15	1.00	8.18	1.00
SMG2000	96	3.00	159	1.00	47	1.00	0.1	1.00	15306	1.00	2.9	1.00	64.33	1.00	15	1.00	8.19	1.00
Sphot	32	1.00	14031	0.87	2888	0.77	5676	1.00	4	1.00	360b	1.00	0.97	0.98	4	1.00	0	1.00
Sphot	48	1.50	14050	0.87	2896	0.78	5675	1.00	4	1.00	360b	1.00	0.98	0.99	4	1.00	0	1.00
Sphot	64	2.00	14841	0.92	3209	0.86	5676	1.00	4	1.00	360b	1.00	0.98	0.99	4	1.00	0	1.00
Sphot	80	2.50	14780	0.92	3185	0.85	5676	1.00	4	1.00	360b	1.00	0.99	1.00	4	1.00	0	1.00
Sphot	96	3.00	16151	1.00	3727	1.00	5677	1.00	4	1.00	360b	1.00	0.99	1.00	4	1.00	0	1.00
Sweep3D	32	1.00	1397	2.66	536	2.73	766	2.99	156	0.91	5.2	1.68	3.25	0.91	5	1.00	28.8	0.36
Sweep3D	48	1.50	956	1.82	366	1.87	511	2.00	164	0.95	4.1	1.32	3.42	0.96	5	1.00	41.6	0.52
Sweep3D	64	2.00	742	1.41	281	1.43	383	1.50	168	0.98	3.6	1.16	3.5	0.98	5	1.00	54.4	0.68
Sweep3D	80	2.50	607	1.15	230	1.17	307	1.20	170	0.99	3.3	1.06	3.55	0.99	5	1.00	67.2	0.84
Sweep3D	96	3.00	526	1.00	196	1.00	256	1.00	172	1.00	3.1	1.00	3.58	1.00	5	1.00	80	1.00
Samrai 4	32	1.00	1677	0.78	553	0.68	171	2.95	131	3.05	0.87	3.00	9.875	3.00	47	1.00	39.7	1.00
Samrai 4	48	1.50	1756	0.81	629	0.77	114	1.97	87	2.02	0.58	2.00	6.58	2.00	47	1.00	39.7	1.00
Samrai 4	64	2.00	2432	1.13	909	1.12	86	1.48	65	1.51	0.43	1.48	4.94	1.50	47	1.00	39.7	1.00
Samrai 4	80	2.50	3298	1.53	1259	1.54	70	1.21	52	1.21	0.35	1.21	3.95	1.20	47	1.00	39.7	1.00
Samrai 4	96	3.00	2158	1.00	815	1.00	58	1.00	43	1.00	0.29	1.00	3.29	1.00	47	1.00	39.7	1.00
Samrai 8	32	1.00	4370	0.60	1505	0.54	377	2.90	136	2.19	1.06	2.59	19.2	1.67	11	1.00	69.1	1.00
Samrai 8	48	1.50	5798	0.80	2123	0.77	256	1.97	106	1.71	0.81	1.98	20.9	1.82	11	1.00	69.1	1.00
Samrai 8	64	2.00	5794	0.80	2151	0.78	192	1.48	93	1.50	0.61	1.49	17.3	1.50	11	1.00	69.1	1.00
Samrai 8	80	2.50	4208	0.58	1569	0.57	154	1.18	74	1.19	0.49	1.20	13.8	1.20	11	1.00	69.1	1.00
Samrai 8	96	3.00	7244	1.00	2762	1.00	130	1.00	62	1.00	0.41	1.00	11.5	1.00	11	1.00	69.1	1.00

Table 3: Task scaling results with constant global problem size. Values are *per task.*

rectilinear, 2-D mesh. Monte Carlo transport solves the Boltzmann transport equation by directly mimicking the behavior of photons as they are born in hot matter, move through and scatter in different materials, are absorbed or escape from the problem domain. Particles are born with an energy and direction that are determined by using random numbers to sample from appropriate distributions. This code tracks particles through a logically rectangular, 2-D mesh that is internally generated.

3.4 Sweep3D

Sweep3D [13, 14] is a solver for the 3-D, time-independent, particle transport equation on an orthogonal mesh and it uses a multidimensional wavefront algorithm for "discrete ordinates" deterministic particle transport simulation. Sweep3D benefits from multiple wavefronts in multiple dimensions, which are partitioned and pipelined on a distributed memory system. The three dimensional space is decomposed onto a two-dimensional orthogonal mesh, where each processor is assigned one columnar domain. Sweep3D exchanges messages between processors as wavefronts propagate diagonally across this 3-D space in eight directions.

3.5 Samrai

The SAMRAI (Structured Adaptive Mesh Refinement Application Infrastructure) library [23] is an object-oriented C++ software framework for the development of computational physics applications using structured adaptive mesh refinement (AMR) technology. SAMR dynamically adapts its hierarchy of spatial and temporal refinement levels to follow interesting features in the evolving simulation, focusing computer resources on these localized regions of the computational domain. This hierarchy consists of several mesh levels where all cells at a particular level have the same mesh resolution. Each level is composed of a collection of patches, each of which is a logically rectangular collection of computational cells. A patch contains data that represent simulation quantities in the region of the simulation domain covered by the patch region. Because AMR problems are extremely sensitive to their input, we study problems at different time steps. Our initial problem is a sinusoidal shock wave traveling down a 3-D tube. The important point for this study is that the number of grid points remains relatively constant even though the mesh is refined and repartitioned as the shock wave travels down the tube. For this problem, we consider timesteps 4 and 8.

4 Evaluation and Implications

We present our evaluation along the dimensions described in Section 2. We try to preserve a realistic execution environment for our applications by running them with typical input parameters and at reasonable levels of concurrency. For example, we use a minimum of 32 tasks for our experiments.

First, Table 3 provides an overview of the effects of scaling the number of processors while holding the global problem size constant for each application. Next, Table 4 illustrates the effects of scaling the local problem size while holding the number of processors constant for each application. For each metric, we report the absolute numbers and normalized values in the left and right subcolumn, respectively.

The instruction frequency measurements illustrate similarities and differences for our choice of a variety of scientific applications. On average, every third to fifth instruction is a load reference, regardless of problem and task scaling. This indicates a good breakdown of large-grain parallelism by the applications while the potential for instruction parallelism remains constant during scaling experiments. The varying degree of floating-point intensity during execution illustrates our choice of a wide variety of applications, ranging from three to one floating point operation per load (sPPM, Sphot to Sweep3d) over only a fraction of floating ops per fixed op (Samrai) to largely fixed-point intensive applications (SMG2000).

The adaptive application Samrai also exhibits changing ratios with a decrease in float ops relative to loads for an increasing number of tasks. For this application, dynamic changes over timesteps resulted in proportional increases in computational overhead for each task but the ratios between instruction types remained constant. This illustrates the challenge of increasing demand for adaptive methods, which should be met by dynamically changing support to meet these resource requirements.

For an increasing number of tasks (Table 3), a decrease in computational work can be observed for most applications (sPPM, Sphot, Sweep3D). SMG2000 only exhibits this decrease for the number of floating point operations during task scaling. For Samrai, the adaptive application, an increase in computation was observed for the total number of instructions. Loads fluctuated for timestep 4 and increased for timestep 8 with increasing tasks. Most notably, float ops decreased, as in most other applications, which shows the effectiveness of task parallelism for adaptive methods.

The increase in adaptation overhead drives this increase in overall instructions. This causes more loads on the adaptation phase while loads decrease for the floating-point intensive calculations.

For an increase in problem size (Table 4), all instruction categories increase at the same rate for all tested applications, except for float ops in the case of

App	Task Problem size (length of one dimen	Relative Problem Size	Instructions (M)		Load Instructions (M)		Floating point operations (M)		Avg number of messages		Avg Send volume (MB)		Avg number of distinct destinations		Number of collectives issued		Avg Collective Payload Size (bytes)	
sPPM	64	1.00	1926	1.00	493	1.00	1226	1.00	33	1.00	8.8	1.00	5.5	1.00	2	1.00	14	1.00
sPPM	80	1.95	3695	1.92	924	1.87	2394	1.95	33	1.00	13.5	1.53	5.5	1.00	2	1.00	14	1.00
sPPM	96	3.38	6272	3.26	1584	3.21	4066	3.32	33	1.00	19.3	2.19	5.5	1.00	2	1.00	14	1.00
sPPM	112	5.36	9864	5.12	2473	5.02	6441	5.25	33	1.00	26	2.95	5.5	1.00	2	1.00	14	1.00
sPPM	128	8.00	14565	7.56	3743	7.59	9429	7.69	33	1.00	33.8	3.84	5.5	1.00	2	1.00	14	1.00
SMG2000	2	1.00	65	1.00	19.5	1.00	0.01	1.00	5996	1.00	1.1	1.00	41.88	1.00	16	1.00	8.2	1.00
SMG2000	3	3.38	126	1.94	37.7	1.93	0.07	7.00	11231	1.87	1.7	1.55	55	1.31	16	1.00	8.1	0.99
SMG2000	4	8.00	159	2.45	47.3	2.43	0.19	19.00	15446	2.58	2.5	2.27	41.88	1.00	16	1.00	8.1	0.99
SMG2000	5	15.63	264	4.06	78.7	4.04	0.51	51.00	25636	4.28	3.8	3.45	55	1.31	17	1.06	8.1	0.99
SMG2000	6	27.00	292	4.49	89.9	4.61	0.87	87.00	27004	4.50	4	3.64	47	1.12	17	1.06	8	0.98
Sphot	5	1.00	16906	1.00	3842	1.00	5722	1.00	4	1.00	4E-04	1.00	0.98	1.00	4	1.00	0	1.00
Sphot	10	4.00	25542	1.51	5659	1.47	9231	1.61	4	1.00	4E-04	1.00	0.98	1.00	4	1.00	0	1.00
Sphot	15	9.00	34552	2.04	7451	1.94	13091	2.29	4	1.00	4E-04	1.00	0.98	1.00	4	1.00	0	1.00
Sphot	20	16.00	40807	2.41	8768	2.28	15644	2.73	4	1.00	4E-04	1.00	0.98	1.00	4	1.00	0	1.00
Sphot	25	25.00	53187	3.15	11649	3.03	20017	3.50	4	1.00	4E-04	1.00	0.98	1.00	4	1.00	0	1.00
Sweep3D	50	1.00	12	1.00	4	1.00	3	1.00	84	1.00	0.25	1.00	3.5	1.00	5	1.00	54.4	1.00
Sweep3D	75	3.38	35	2.92	12	3.00	12	4.00	126	1.50	0.57	2.28	3.5	1.00	5	1.00	54.4	1.00
Sweep3D	100	8.00	75	6.25	27	6.75	30	10.00	168	2.00	1.01	4.04	3.5	1.00	5	1.00	54.4	1.00
Sweep3D	125	15.63	136	11.33	49	12.25	58	19.33	210	2.50	1.58	6.32	3.5	1.00	5	1.00	54.4	1.00
Sweep3D	150	27.00	227	18.92	82	20.50	100	33.33	210	2.50	1.89	7.56	3.5	1.00	5	1.00	54.4	1.00

Table 4: Problem size scaling results at 64 tasks. Values are *per task*.

SMG2000. SMG2000 results in dramatic increases in float ops for problem size scaling but the overall ratio to other operations is still relatively insignificant. (We had to limit the problem sizes for our SMG2000 experiments because, in our existing experimental framework, tracefile sizes grew unmanageable.)

4.1 Point-to-Point (P2P) Communication

The majority of applications in our study use point-to-point communication for sending the lion's share of their data. Even though all of the applications use similar MPI functionality, we see a diverse set of characteristics with respect to the patterns these applications exploit in their utilization of point-to-point communication.

The *average number of messages sent* shows the number of point-to-point messages sent by a task while the *average send volume* quantifies the amount of data sent by one task during the observed phase. For task scaling in Table 3, the majority of the applications show a relationship between processor scaling and the number of messages sent. Figures 2 and 3 illustrate this behavior.

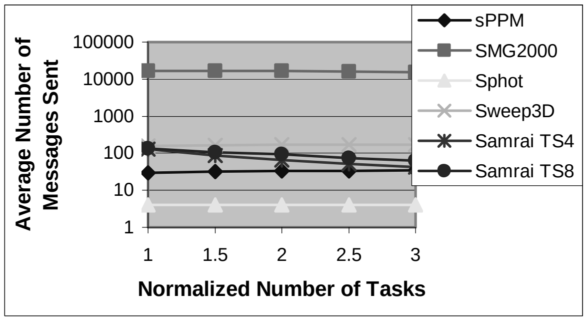

Figure 2: Number of Messages for Task Scaling

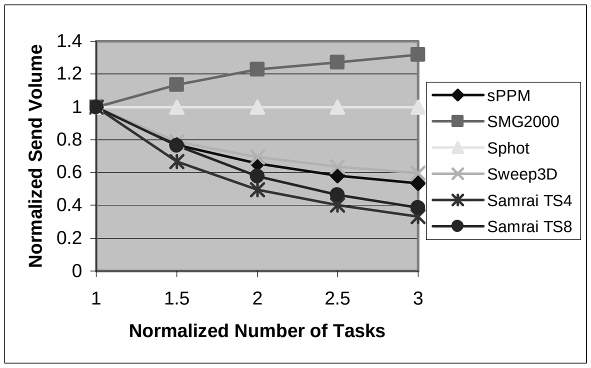

Figure 3: Send Volume for Task Scaling

The number of messages decreases sharply for Samrai as the number of processors increases. In contrast, sPPM and Sweep3D appear to be growing yet reaching an asymptotic limit as the task count increases. The number of messages for SMG2000 declines as the number of tasks increases, but the trend is relatively slow. Sphot remains constant at 4 messages per task. The send volume for sPPM, Sweep3D, and Samrai decreases as processor count grows; this indicates that the amount of data sent is tied to the local problem size as revealed by the decrease in floating point operations. SMG2000 send volume increases slightly as the number of tasks expands. Interestingly, we believe that SMG2000 is suffering from the fact that it must send more data because the decomposition becomes more fragmented at higher numbers of processors, requiring additional communication to converge to a solution [12], even though the amount of local work decreases. Not surprisingly, Sphot has a constant send volume.

The *average number of distinct destinations* approximates the number of distinct recipients of point-to-point sends for a task, also illustrated in Figure 4.. Sphot tasks always send all data to a single master task (0.98). Predictably, sPPM has an average number of distinct destinations that approach six for the 3-D mesh structure of sPPM's data decomposition. Likewise, Sweep3D approaches four due to its 2-D mesh decomposition. On the other hand, the number of destinations for a SMG2000 task appears to grow in proportion with the task count. The average number of destinations for a Samrai task decreases as the task count increases. More important are the differences between timestep 4 and timestep 8. At timestep 8, Samrai has two to three times as many destinations as at timestep 4 on average.

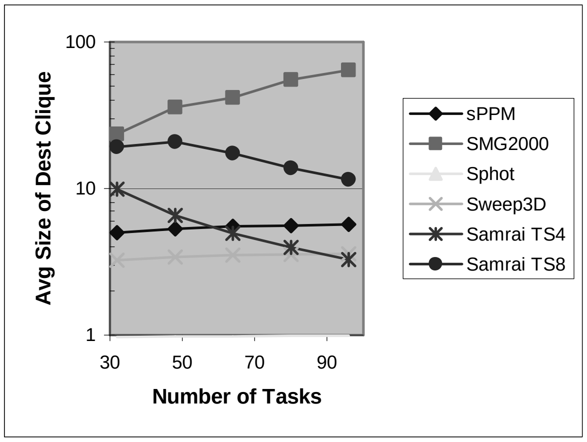

Figure 4: Avg. Number of Distinct Destinations

Table 4 shows the impact on changing problem sizes on each application. Either the number of messages or the message volume (or even both of them), depending on the algorithms, increases at the same growth rate as the input. A graphical presentation of message and send volumes is given in Figures 5 and 6, respectively. For example, as the input size increases by a factor of 8 (from 64^3 to 128^3 for sPPM), the send volume increases at approximately one-half the rate (factor 3.84) while the number of messages stays constant. For SMG2000 and Sweep3D, both volume and number of messages increase with the input. In contrast, Sphot exhibits constant overheads independent of the problem size. In general, the referenced end-points remain constant (except for insignificant variations for SMG2000) with a fixed number of tasks.

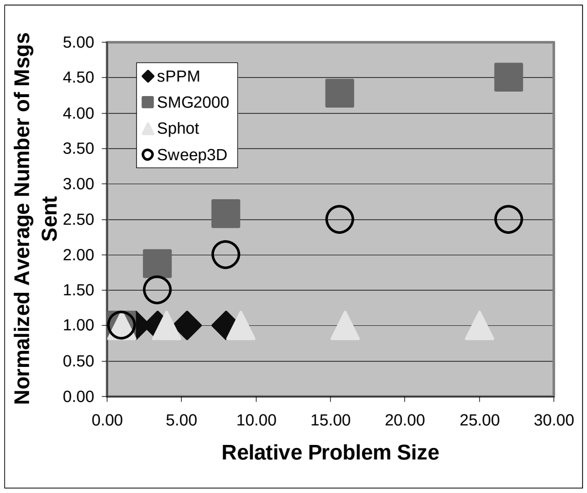

Figure 5: Number of Messages for Problem Scaling

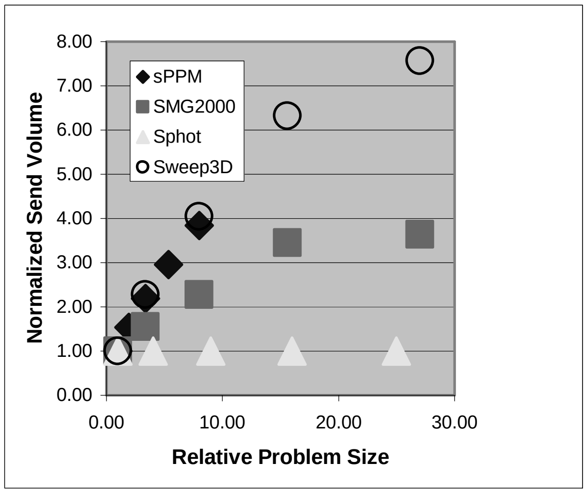

Figure 6: Send Volume for Problem Scaling

In summary, these tables show that varying the number of processors or the problem size alters the size of messages sent by each application. As Figure 7 (with the corresponding numerical values in Table 5) illustrates, there is a wide range of message sizes for these applications when running at 64 tasks. sPPM and Sweep3D have large messages that reflect their data decomposition structure while SMG2000 and Samrai have smaller messages. Traditionally, communication overhead within the communication library dominates performance for smaller messages. Our results show that with this trend toward smaller messages, communication libraries should improve support for these messages. For example, small messages can capitalize on eager protocols, and suffer when buffer management algorithms use ill-suited allocation strategies.

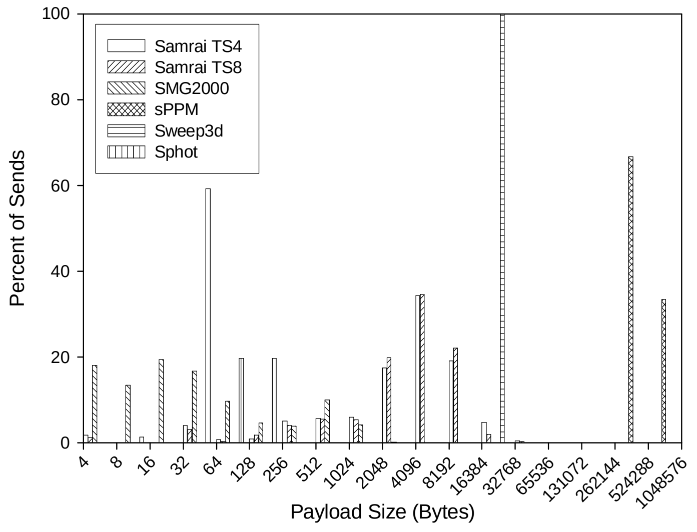

Figure 7: Payload size distribution for P2P messages (64 tasks).

Payload	Samrai TS4	Samrai TS8	SMG2000	SPPM	Sweep3d	SPhot
4	1.7	1.2	18.0	0.0	0.0	0.0
8	1.7	1.2	31.4	0.0	0.0	1.3
16	1.7	1.2	50.8	0.0	0.0	1.3
32	5.8	4.3	67.6	0.0	0.0	60.5
64	6.5	4.6	77.3	0.0	0.0	80.2
128	7.5	6.3	81.9	0.0	0.0	100.0
256	12.5	10.4	85.7	0.0	0.0	100.0
512	18.1	15.9	95.6	0.0	0.0	100.0
1024	24.1	21.3	99.8	0.0	0.0	100.0
2048	41.5	41.0	100.0	0.0	0.0	100.0
4096	75.8	75.7	100.0	0.0	0.0	100.0
8192	94.8	97.7	100.0	0.0	0.0	100.0
16384	99.5	99.7	100.0	0.0	100.0	100.0
32768	100.0	100.0	100.0	0.0	100.0	100.0
65536	100.0	100.0	100.0	0.0	100.0	100.0
131072	100.0	100.0	100.0	0.0	100.0	100.0
262144	100.0	100.0	100.0	66.7	100.0	100.0
524288	100.0	100.0	100.0	100.0	100.0	100.0

Table 5: Cumulative distribution of payload sizes for P2P messages (64 tasks).

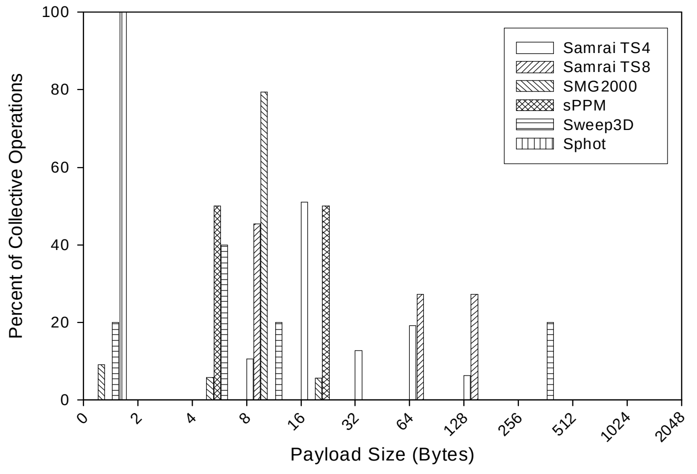

Figure 8: Payload size distribution for collective communication (64 tasks).

Payload	Samrai TS4	Samrai TS8	SMG2000	SPPM	Sweep3d	SPhot
0	0.0	0.0	9.1	0.0	20.0	100.0
2	0.0	0.0	9.1	0.0	20.0	100.0
4	0.0	0.0	15.0	50.0	60.0	100.0
8	10.6	45.5	94.3	50.0	80.0	100.0
16	61.7	45.5	100.0	100.0	80.0	100.0
32	74.5	45.5	100.0	100.0	80.0	100.0
64	93.6	72.7	100.0	100.0	80.0	100.0
128	100.0	100.0	100.0	100.0	80.0	100.0
256	100.0	100.0	100.0	100.0	100.0	100.0

Table 6: Cumulative distribution of payload sizes for collective (64 tasks).

4.2 Collective Communication

All of our applications use collective operations. Many applications that simulate physical systems must make several calculations across the domain at every timestep to preserve the integrity of the physical system and to determine the length of the next timestep. Although these calculations are global, the payload size is typically only a few double precision numbers.

In this regard, we found that virtually all of the collective operations have very small payloads that change neither with the number of tasks nor with the problem size. As Figure 8 illustrates, most collective operations send data payloads of less than 256 bytes. One exception is Sweep3D, which it is the only outlier in Figure 8 (with the corresponding numerical values in Table 6): it has one broadcast operation whose payload size scales linearly with the number of tasks.

All of the communicator groups were the width of the MPI_COMM_WORLD. Although several of the applications did create new communicators, they did not partition the space of the original communicator. The collective operations that perform an operation on the data, such as a reduction, were limited to MAX and SUM.

New architectures with tens of thousands or even millions of processors [1] must have special support for these types of global operations, whether this support draws on either hardware assistance or new algorithms for collectives, such as MPI_Allreduce. Our evidence demonstrates that these applications rely on a very limited region of the design space: simple reduction operators and very small data payloads. Improved performance of collectives may also encourage their use in applications.

4.3 Computation

To correlate the communication activity with computation, we counted several types of events between significant MPI call sites. As Tables 3 and 4 illustrate, the number of floating point operations is closely tied to the problem size. The execution overhead (both instructions and floating-point only) decreases at the same rate that the number of tasks increases, which indicates good scaling at the local task level. Samrai presents an exception as it exhibits increased integer overhead for more tasks (Table 3) that results from additional overhead of the mesh refinement between time steps.

In an effort to determine the distribution of computation relative to communication activity, we analyzed the number of floating point operations performed between communication operations as Table 7 depicts. Many of the applications execute few floating-point operations, if any, between two communication operations. This situation often appears when multiple communication operations occur in a series, usually following a computational time step.

Both sPPM and Sphot show that 5-8% of their computational blocks are very large, containing over 536M floating-point operations. In contrast, Samrai and SMG2000 perform modest amounts of floating point computation between communication operations. Compared to these other applications, Sweep3D executes over 50% of its floating-point operations in multiple blocks of 1024 or greater.

These results indicate that the dynamic and implicit applications tend to communicate more frequently relative to its number of floating point operations. That is, Samrai and SMG2000 do no more than 8M and 1024 floating-point operations, respectively, between significant communication operations.

Flops, Blk	Samrai TS4	Samrai TS8	SMG2000	SPPM	Sweep3d	Sphot
0	89.2	93.5	87.2	94.0	38.3	85.2
2	89.3	93.5	87.5	94.0	38.3	85.2
4	89.3	93.5	90.7	94.0	38.6	85.2
8	89.3	93.5	93.1	94.0	38.6	85.2
16	89.5	93.9	95.7	94.0	38.6	85.2
32	90.5	94.7	98.1	94.8	38.6	92.1
64	95.5	95.8	99.2	95.5	38.6	92.1
128	96.2	96.4	99.7	95.5	38.6	92.1
256	97.2	97.2	99.8	95.5	38.6	92.1
512	97.8	97.7	99.9	95.5	38.6	92.6
1024	98.2	98.1	100.0	95.5	44.6	92.6
2048	98.7	98.5	100.0	95.5	44.6	92.6
4096	98.9	98.9	100.0	95.5	62.1	92.6
8192	99.1	99.1	100.0	95.5	69.9	92.6
16384	99.2	99.3	100.0	95.5	71.3	92.6
32768	99.3	99.3	100.0	95.5	71.3	92.6
65536	99.4	99.3	100.0	95.5	71.3	92.6
131072	99.7	99.6	100.0	95.5	71.3	93.0
262144	99.7	99.7	100.0	95.5	71.3	93.0
524288	99.7	99.7	100.0	95.5	71.6	93.0
1048576	99.8	99.7	100.0	95.5	78.9	93.0
2097152	99.8	99.8	100.0	95.5	78.9	93.0
4194304	99.9	99.8	100.0	95.5	100.0	93.0
8388608	99.9	99.9	100.0	95.5	100.0	93.0
16777216	100.0	100.0	100.0	95.5	100.0	93.0
33554432	100.0	100.0	100.0	95.5	100.0	93.0
67108864	100.0	100.0	100.0	95.5	100.0	93.0
1.34E+08	100.0	100.0	100.0	95.5	100.0	93.0
2.68E+08	100.0	100.0	100.0	95.5	100.0	93.0
5.37E+08	100.0	100.0	100.0	100.0	100.0	93.0
1.07E+09	100.0	100.0	100.0	100.0	100.0	100.0

Table 7:Cumulative distribution of blocks of floating point operations between communication points (64 tasks).

4.4 Observations

First, we found contemporary, large-scale scientific applications have a wide range of characteristics, which range from small, frequent messages to large, infrequent messages. As similar findings were reported for previous studies of scientific applications [9], it is remarkable that our results not only strengthen them but also provide novel characteristics, as discussed earlier.

Second, our experiments revealed that collective communication operations are used by all the applications. Further, the payload size of these collective operations is very small and this size remains practically invariant with respect to the problem size or the number of tasks. Our results show that Allreduce and Bcast have very small payloads. This result clearly shows that all of the applications in our study could benefit from improvements in the performance of collective communications, whether those improvements come in hardware or software. Historically, collective communication often suffered from high performance overhead due to a lack of scalability, which often forced application programmers to hand-code collectives with a series of point-to-point messages. Once these legacy communication patterns are transformed into collectives, the importance of collectives is most likely to grow.

Third, we also note a substantial difference in algorithms in terms of their increasing message and computation activities over consecutive time steps: implicit versus explicit methods, and uniform mesh versus adaptive mesh. Sweep3D and sPPM use explicit methods and uniform meshes, which lead to easily predicted communication patterns. On the other hand, Samrai's adaptive mesh refinement can make both the communication patterns and computational load difficult to predict as Table 3 shows. Likewise, the implicit techniques used in SMG2000 have considerably different communication requirements than the explicit techniques.

5 Related Work

Characterization of applications and architectures is an ongoing and important process as evidenced by the considerable amount of previous work [3, 5, 7, 9, 15, 24-26]. With the broad range of design parameters for today's computer systems and the fact that both applications and architectures evolve, these quantitative evaluations help focus attention on important design points.

In the past, synthetic kernel benchmarks were often used to evaluate and compare architectures, e.g., using Linpack on parallel machines [2]. The NAS parallel benchmarks [3] consist of small kernels and applications; they have been used by a large number of groups for performance evaluation of architectures. These benchmarks have been adapted to a wide range of platforms and programming models [5, 6]. The SPLASH-2 suite of parallel applications is another example of widely used benchmarks [25], which are targeted toward centralized and distributed shared-address-space multiprocessors but does not capture the challenges of parallelism in cluster computing. Worley [26] presents a detailed comparison of a climate modeling application that uses explicit communication on two different platforms. Prior work has also focuses on the differences between commercial and scientific workloads [8, 17]. Our choice of scientific applications for benchmarks specifically considers appropriate programming paradigms for clusters with an emphasis on message passing, large scientific codes and a diversity in application characteristics as well as domains.

The two most closely related papers to the work that we present here are work by Wong and associates [24] and by Cypher and colleagues [9]. Wong and associates [24] studied the effect of different architectural parameters on the NAS parallel benchmarks using a methodology similar to ours. They captured information about the message and instruction behaviors of these much smaller benchmarks to understand communication and simulate caching behavior on different architectures. Cypher and colleagues [9] quantitatively characterize the behavior of numerous scientific applications that use explicit communication. In particular, they report on floating-point operations, memory size, I/O, and communication in order to help design well-balanced architectures. More importantly, they demonstrate the effects of scaling problem size and the number of processors for these application characteristics. Our results strengthen these previous results in showing their validity for larger scientific applications on contemporary clusters and indicate new trends in application behavior well beyond previous work.

6 Conclusions

In this paper, we evaluated explicit communication characteristics across a set of diverse, large-scale scientific applications, primarily from the perspective of message passing via MPI *and* independent of the target architecture. By focusing on the MPI activity of these applications along with coarse-grain measurements of the computation, we separate the application behavior from the architecture behavior and present the inherent communication signatures of these diverse applications.

Our results do not only strengthen findings of studies with smaller applications and reinforce differences in application behavior. We also uncovered striking similarities, such as the trend of small payload sizes for collective operations, which are significant due to the increasing acceptance of more efficient implementation of collectives. Collectives with competitive scaling capabilities should ensure that collectives become more widely used. We also highlight novel applications parting with regimented, static communication patterns in favor of dynamically evolving patterns as evidenced by our experiments on applications that use implicit linear solvers and adaptive mesh refinement. Clearly, these investigations will continue to be important as new applications, architectures, and software becomes available.

Overall, our study contributes a better understanding of the demands for current and emerging paradigms of scientific computing in terms of their computation and communication demands.

tation and communication demands.

Acknowledgements

We thank Mark Seager, Bob Lucas, and the anonymous reviewers for their useful comments. We also thank Andy Wissink for creating the shock tube problem for SAMRAI. This work was performed under the auspices of the U.S. Dept. of Energy by University of California LLNL under contract W-7405-Eng-48. LLNL Document Number UCRL-JC-143483.

References

[1] Almasi, G.S., C. Cascaval et al., "Demonstrating the scalability of a molecular dynamics application on a Petaflop computer," Proc. Int'l Conf. Supercomputing, 2001, pp. 393-406.

[2] Anderson, E., A. Benzoni et al., "LAPACK for distributed memory architectures: progress report," Proc. Fifth SIAM Conference Parallel Processing Scientific Computing, 1991, pp. 625-30.

[3] Bailey, D.H., E. Barszcz et al., "NAS parallel benchmark results," *IEEE Parallel & Distributed Technology: Systems & Applications*, 1(1):43-51, 1993.

[4] Brown, P.N., R.D. Falgout, and J.E. Jones, "Semicoarsening multigrid on distributed memory machines," *SIAM Journal on Scientific Computing*, 21(5):1823-34, 2000.

[5] Cappello, F. and D. Etiemble, "MPI versus MPI+OpenMP on IBM SP for the NAS Benchmarks," Proc. SC2000: High Performance Networking and Computing Conf. (electronic publication), 2000.

[6] Clémençon, C., K.M. Decker et al., "HPF and MPI implementation of the NAS Parallel Benchmarks supported by integrated program engineering tools," Proc. 8th Int'l Conf. on Parallel and Distributed Computing and Systems (PDCS'96), 1996.

[7] Culler, D.E., J.P. Singh, and A. Gupta, *Parallel computer architecture: a hardware software approach*. San Francisco: Morgan Kaufmann Publishers, 1999.

[8] Cvetanovic, Z. and R.E. Kessler, "Performance analysis of the Alpha 21264-based Compaq ES40 system," Proc. 27th Annual Int'l Symp. Computer Architecture, 2000, pp. 192 - 202.

[9] Cypher, R., A. Ho et al., "Architectural requirements of parallel scientific applications with explicit communication," Proc. 20th annual Int'l Symp. Computer Architecture, 1993, pp. 2-13.

[10] Geist, G.A., M.T. Heath et al., "A Users' Guide to PICL - A Portable Instrumented Communication Library," Oak Ridge National Laboratory, P.O.Box 2009, Bldg. 9207-A, Oak Ridge, TN 37831-8083 1991.

[11] Gropp, W., E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*, 2nd ed. Cambridge, MA: MIT Press, 1999.

[12] Gropp, W.D., D.K. Kaushik et al., "Performance Modeling and Tuning of an Unstructured Mesh CFD Application," Proc. SC2000: High Performance Networking and Computing Conf. (electronic publication), 2000.

[13] Hoisie, A., O. Lubeck et al., "A General Predictive Performance Model for Wavefront Algorithms on Clusters of SMPs," Proc. ICPP 2000, 2000.

[14] Koch, K.R., R.S. Baker, and R.E. Alcouffe, "Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor," *Trans. Amer. Nuc. Soc.*, 65(198), 1992.

[15] Kumar, V., A. Grama et al., *Introduction to parallel computing: design and analysis of algorithms*. Redwood City, Calif.: Benjamin/Cummings Pub. Co., 1994.

[16] Labarta, J., S. Girona et al., "DiP: A Parallel Program Development Environment," CEPBA, Barcelona, Spain 1996.

[17] Lee, D.C., P.J. Crowley et al., "Execution characteristics of desktop applications on Windows NT," Proc. 25th annual Int'l Symp. Computer architecture (ISCA), 1998, pp. 27-38.

[18] Mirin, A.A., R.H. Cohen et al., "Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System," Proc. SC99: High Performance Networking and Computing Conf. (electronic publication), 1999.

[19] Reed, D.A., R.A. Aydt et al., "An Overview of the Pablo Performance Analysis Environment," Department of Computer Science, University of Illinois, 1304 West Springfield Avenue, Urbana, IL 61801 1992.

[20] Shan, H., J.P. Singh et al., "A Comparison of Three Programming Models for Adaptive Applications on the Origin2000," Proc. SC2000: High Performance Networking and Computing Conf. (electronic publication), 2000.

[21] Shende, S., A.D. Malony et al., "Portable profiling and tracing for parallel, scientific applications using C++," Proc. SIGMETRICS Symp. Parallel and Distributed Tools (SPDT), 1998, pp. 134-45.

[22] Snir, M., S. Otto et al., Eds., *MPI--the complete reference*, 2nd ed. Cambridge, MA: MIT Press, 1998.

[23] Wissink, A.M., R.D. Hornung et al., "Large Scale Parallel Structured AMR Calculations Using the SAMRAI Framework," Proc. SC2001 High-Performance Computing and Networking Conf, 2001.

[24] Wong, F., R. Martin et al., “Architectural Requirements and Scalability of the NAS Parallel Benchmarks,” Proc. SC99: High Performance Networking and Computing Conf. (electronic publication), 1999.

[25] Woo, S.C., M. Ohara et al., “The SPLASH-2 programs: characterization and methodological considerations,” Proc. 22nd Annual Int'l Symp. Computer Architecture, 1995, pp. 24-36.

[26] Worley, P.H., “Performance evaluation of the IBM SP and the Compaq AlphaServer SC,” Proc. ACM Int'l Conf. Supercomputing (ICS), 2000, pp. 235-44.