Performance Prediction of Large Parallel Applications Using Parallel Simulations

Rajive Bagrodia Ewa Deelman
Computer Science Department
University of California Los Angeles, CA 90024
1-310-825-0956 1-310-825-2091
rajive@cs.ucla.edu deelman@cs.ucla.edu

Steven Docy Thomas Phan
Computer Science Department
University of California Los Angeles, CA 90024
1-310-825-4885
sdocy@cs.ucla.edu phantom@cs.ucla.edu

ABSTRACT

Accurate simulation of large parallel applications can be facilitated with the use of direct execution and parallel discrete event simulation. This paper describes the use of COMPASS, a direct execution-driven, parallel simulator for performance prediction of programs that include both communication and I/O intensive applications. The simulator has been used to predict the performance of such applications on both distributed memory machines like the IBM SP and shared-memory machines like the SGI Origin 2000. The paper illustrates the usefulness of COMPASS as a versatile performance prediction tool. We use both real-world applications and synthetic benchmarks to study application scalability, sensitivity to communication latency, and the interplay between factors like communication pattern and parallel file system caching on application performance. We also show that the simulator is accurate in its predictions and that it is also efficient in its ability to use parallel simulation to reduce its own execution time which, in some cases, has yielded a near-linear speedup.

Keywords

Parallel Program Simulation, Application Scalability, MPI, MPI-IO, Parallel Discrete Event Simulation, Direct Execution.

1. INTRODUCTION

Accurate and efficient performance prediction of existing parallel applications on multiple target architectures is a challenging problem. Both analytical and simulation approaches have been used successfully for this purpose. Whereas analytical solutions have the advantage of efficiency, they also suffer from the limitation that many complex systems are analytically intractable. Although simulation is a widely applicable tool, its major limitation is its extremely long execution time for large-scale systems. A number of simulators, including Parallel Proteus [22], LAPSE [13], SimOS [30], Wisconsin Wind Tunnel [28], and MPI-SIM [26] have been developed to control the execution time of simulation models of parallel programs. The simulators typically use direct execution to reduce the cost of simulating sequential instructions and use parallel discrete event simulation to exploit parallelism within the simulator to reduce the impact of scaling up the target configuration being simulated. Most existing program simulators were designed to study CPU-intensive parallel programs. However, inadequate parallel I/O performance has become a significant deterrent to the overall performance of many applications and a number of solutions have been proposed to improve parallel I/O performance [10, 29.]. The ability to include I/O and CPU-intensive applications in a unified performance prediction environment thus appears to have significant value. We have developed COMPASS (**COM**ponent-based **PA**rallel **S**ystem **S**imulator), a portable, execution driven, asynchronous parallel discrete event simulator that can be used to predict the performance of large-scale parallel programs, including computation and I/O intensive applications, targeted for execution on shared-nothing and shared memory architectures, as well as SMP clusters.

In particular, simulation modules have been developed to predict the performance of applications as a function of communication latency, number of available processors on the machine of interest, different caching strategies for parallel I/0, parallel file system characteristics, and alternative implementations of collective communication and I/0 commands. The simulator is being used for detailed program simulations within the POEMS project [12]. POEMS (Performance Oriented End-to-end Modeling System) is a collaborative, multi-institute project whose goal is to create and experimentally evaluate a problem solving environment for end-to-end performance modeling of complex parallel/distributed systems. This paper describes the simulator and its use in evaluating the performance of large-scale, complex applications as a function of various system characteristics. As we demonstrate, the simulator is not only accurate, but is also fast due to its ability to run in parallel. Where we can, we use real world applications; however, in some cases we have used synthetic benchmarks to highlight a particular feature of the simulator. We show the simulator's portability and accuracy by validating the tool on two platforms (the distributed memory IBM SP and the shared memory SGI Origin 2000) for a range of synthetic and real world applications. For instance, we show that the predicted execution time of an ASCI kernel called Sweep3D [32] was within 5% of the measured execution time on both architectures.

Second, we demonstrate the scalability of the tool itself. A major impediment to widespread use of program simulators is their execution (in)efficiency. We show that COMPASS can effectively exploit parallel model execution to dramatically reduce the execution time of a simulation model, without sacrificing accuracy. In particular, we show that, for a configuration of an application kernel called Sweep3D and a target machine with 64 processors, the simulator reduces the slowdown factor from 35 using sequential simulation to as low as 2.5 using a parallel simulator running on 64 processors. Further, the larger amounts of memory available on a parallel platform allowed us to conduct scalability studies for target configurations that were at least two orders of magnitude larger than those obtained with a sequential machine. For instance, for the same Sweep3D application, memory constraints of a sequential simulator would have limited us to simulating a target architecture of at most 13 processors for a 150^3 problem size. Using the memory available to us on the 128-node SP, we were able to predict the performance of Sweep3D for up to 1600 processors.

Having established the simulator's accuracy and scalability, we demonstrate some of its capabilities.

1. We use the simulator to predict the scalability properties of an application using standard measures of scalability that include isoefficiency and scale-up as a function of the number of processors.
2. We analyze the behavior of an application as a function of the communication latency of the target architecture. We demonstrate that applications such as Sweep3D are not very sensitive to latency variations implying that executing such applications on a network of workstations rather than on a massively parallel system is a reasonable alternative.
3. We show how COMPASS can model the new architectures consisting of clusters of SMPs (such as the newest IBM SP). Even though the hardware for SMP cluster exists, the MPI software is not yet available to exploit the faster communication available among the processors of an SMP node. Using COMPASS we can show how an application would perform on the new architecture, if fast intra-node MPI communications were made available. In particular, using our synthetic benchmarks, we identify a type of application that can run 20% faster when using four 4-way SMPs rather than sixteen processors.
4. Using a synthetic benchmark, we demonstrate the sensitivity of different communication patterns to variations in the communication latencies of a target architecture.
5. Parallel file systems are becoming more complex, allowing both compute- and I/O-node caching. We demonstrate how various caching policies can affect the performance of a benchmark. In particular, for an I/O intensive benchmark, we see that as the network latency degrades, the gains from cooperative caching [10, 7] become negligible.

The next section gives a brief description of the simulator. Section 3 describes the benchmarks and the target and host architectures used for the performance study. Section 4 presents results on the validation and scalability of the simulator. Section 5 showcases some of features of the simulator as described in point 1 to 5above. Section 6 discusses related work and concludes with a discussion of our future research directions.

2. COMPASS

The goal of the simulator is to enable the simulation of large-scale parallel applications written using MPI and MPI-IO on a variety of high performance architectures. The application program to be simulated is referred to as the target program and the architecture on which its performance is to be predicted is referred to as the target architecture. The machine on which the simulator is executed is referred to as the host machine, which may be sequential or parallel.

The simulation environment is composed of several distinct, yet tightly coupled components—the *simulation kernel*, the *MPI communication library simulator (MPI-Sim), the parallel I/O simulator (PIO-Sim)* and the *parallel file system simulator (PFS-Sim)*. Each successive component builds upon and extends the capabilities of previous components, expanding the breadth and the depth of the performance issues, which may be investigated with the simulator. The simulation kernel provides the framework; it implements the simulation protocols and provides support for scheduling and execution of threads. MPI-Sim provides the capability to simulate individual and collective MPI communication routines. PIO-Sim extends MPI-Sim's capabilities to include I/O routines as well as providing several implementations of collective I/O, the ability to handle user defined data types that are needed to support complex I/O operations, and a simple I/O service time model. PFS-Sim completes the simulation environment by providing detailed simulation of the parallel file system and of multiple caching algorithms. The simulator itself is portable and runs on a variety of parallel platforms—the IBM SP, the Origin 2000, and the Intel Paragon.

The simulation kernel is the heart of the simulation environment. In general, the number of processors in the host machine will be less than the number of processors in the target architecture being simulated, so the simulator must support multi-threading. The kernel on each processor schedules the threads and ensures that events on all processors are executed in their correct timestamp order. A target thread is simulated as follows: the local code is simulated by direct execution [8] and all communication and I/O commands are trapped by the simulator, which uses an appropriate model to predict the execution time for the corresponding activity on the target architecture. The corresponding communication or I/O commands are also executed for consistency with the target program, but the physical time taken for executing this operation is ignored. The use of direct execution for simulation of local code requires that the processors in the host and target machines be similar. However, the interconnection network, parallel I/O system, and file systems on the two architectures may be very different. COMPASS supports most of the commonly used MPI communication routines, such as point-to-point and collective communications. In the simulator, all collective communication functions are implemented in terms of point-to-point communication functions, and all point-to-point communication functions are implemented using a set of core non-blocking MPI functions [27]. The interconnection network model currently ignores contention in the network. More detailed models are being developed, but given the excellent validation obtained with the simpler model for a variety of benchmarks both here and in previous work [26], this was not considered to be a serious limitation.

The parallel I/O component of COMPASS simulates the individual and collective I/O constructs provided by MPI-IO. These constructs include creating, opening, closing and deleting a file; most data access (read/write) operations; and a local datatype constructor introduced as part of the MPI-IO specification. The file system component of COMPASS simulates the parallel file system used to service I/O requests generated by the MPI-IO programs. This component is self-contained and may be replaced by a simple disk access model in order to speed up the simulation whenever a detailed system model is not required. However, using the detailed model allows the study of a wide variety of parallel file system configurations. The basic structure and functionality of the file system component is taken from the Vesta parallel file system, a highly scalable, experimental file system developed by IBM [6]. The behavior of the physical disks is simulated by a set of disk models. We have included simple models based on seek time, rotational latency, and data transfer rate as well as a highly detailed model developed at Dartmouth [24].

Detailed system simulations are slow. Parallel simulators can potentially reduce execution time of the model, and provide greater amounts of memory, another necessity for large, detailed simulations. The simulation kernel provides support for sequential and parallel execution of the simulator. Parallel execution is supported via a set of conservative parallel simulation protocols [26]. When combined with the kernel's built-in multi-threading capabilities, this allows the simulator to effectively use however many host processors are available without limiting the size and type of experiments which may be run. The simulator also supports a number of optimizations that are based on an analysis of the behavior of the parallel application. Among the optimizations made available by program behavior analysis is a technique, which allows the simulation protocols described above to actually be turned off, eliminating the costly overhead of global synchronization [*in submission*].

3. Benchmarks and Systems

3.1 Real World Application Benchmarks

3.1.1 Sweep3D

Sweep3D is a solver for the three-dimensional, time independent, neutron particle transport. The computation calculates the flux of particles through a given region of space, where the flux in any region is dependent on the flux from all the neighboring cells. The three-dimensional space (XYZ) is discretized into three-dimensional cells (IJK). The computation progresses in a wavefront manner from all the eight octants of the space, with each octant containing six, independent angles. The angles correspond to the six, independent directions of the flux (one for each face of a cube-cell). Sweep3D uses a 2D domain decomposition onto a 2D array of processors in the I and J directions. In this configuration, the sweep progresses as a processor computes the flux through the column of cells, then sends the outgoing flux information to its two neighboring processors. In order to improve performance, the K dimension and the angles are divided into blocks, allowing a processor to calculate only part of the values in the dimension and only a few angles before sending the values to the neighboring processors.

3.1.2 NAS Benchmarks

The NAS Parallel Benchmarks (NPB) is a suite of parallel scientific benchmarks made available from the Numerical Aerodynamic Simulation (NAS) project at the NASA Ames Research Center [2]. The NAS suite contributes a strong core to our experimental set as it represents a number of well-known, different, real-world, non-vendor-specific codes that can be easily tailored to utilize the COMPASS system. We used the NPB 2.2 release of the software, and included a variety of applications, of which only four: BT, LU, MG, and SP were deemed stable by the NPB authors. BT, SP, and LU compute solutions to systems of discretized Navier-Stokes equations, while MG solves a three-dimensional scalar Poisson equation. The NPB distribution provides a preconfigured set of problem sizes (because of the F77 constraint with dynamic memory) with which these programs can operate. For each application the problem sizes are, in increasing order, S, A, B, and C. Furthermore, the programs can be run in parallel only with a specific number of processors: BT and SP run with 4, 9, and 16 processors, while LU and MG run with 4, 8, and 16. Both the NPB suite and SWEEP3D were originally programmed in Fortran. As MPI-SIM currently supports only C, the programs were first translated using f2c [14]. Subsequently, the translated code was automatically *localized*, to allow the simulator to simulate multiple simultaneous threads of the target program on a single processor of the host machine. The localizer also converts MPI and MPI-I0 calls to equivalent calls defined within the COMPASS library. The localizer is fully automated and has been used successfully with very large applications.

3.2 Synthetic Benchmarks—SAMPLE

Although real world applications or kernels like Sweep3D and NAS are useful benchmarks for simulators such as COMPASS, they have a major disadvantage in that their core algorithms are difficult to understand and it is impossible to modify them to evaluate the impact of alternative types of program structures including computation granularity and communication patterns. While each of these programs provided a means for parameter adjustment, the large granularity at which these changes could be made did not serve our need to measure their performance as a function of specific runtime behavior. Thus, in addition to using these real world benchmarks, we sought to write a synthetic application that allows for the explicit tuning of communication and computation parameters. This effort resulted in SAMPLE (Synthetic Application for Message-Passing Library Environments), a C program that performs precisely changeable amounts of calculation and message-passing inter-process communications suitable for experimental analysis. SAMPLE executes message passing via calls that can be targeted to either COMPASS or the actual MPI library, to facilitate validation. SAMPLE is a simple loop that contains two inner loops: the first is a pure computation loop whose duration can be varied by adjusting the number of floating point divisions executed, while the second is a communication loop that can implement multiple communication patterns by changing the frequency, size, and destination of messages sent (and received) by each process. Message distribution can take on a wide variety of patterns, as described in [17]. Using MPI's point-to-point capability, we implemented a number of these methods, such as wavefront, nearest neighbor, ring, one-to-all, and all-to-all communications. Using predefined metrics, the user can easily change the communication to computation ratio in the program.

3.3 I/O Benchmark

Since implementations of the MPI-I/O standard are not yet widely available, it is hard to find real world applications that stress the parallel I/O simulation capabilities of the simulator; hence a set of synthetic benchmarks were developed for this purpose. The benchmark uses N processes, each mapped to a unique compute node. Each process generates read and write requests for blocks of data of a given size. The interarrival times of the I/O requests are sampled from a normal random distribution with a given mean. The blocks are all in the same file, which is distributed across *M* I/O nodes, each with *D* disks (for a total of *M*D* disks). Each process issues *R* requests, where for a given c, the first *R/c* requests are used to warm up the caches. Each of these parameters can easily be modified.

3.4 Host and Target Architectures

The **SGI Origin** 2000 [19] is a multiprocessor system from Silicon Graphics, Inc. The Origin provides a cache-coherent NUMA distributed shared memory layout with two MIPS Rl0000 processors comprising a processing node multiplexed over a hub chip to reduce memory latency and increase memory bandwidth. Our Origin testbed is small, with only ten 180 MHz Rl0000 processors sharing 320 MB of memory. Due to its limited number of processors and memory, we could not completely perform the same number and size of experiments as we did on the IBM SP.

The **IBM Scalable Parallel (SP)** system is a scalable multiprocessor that condenses several complete RS6000 workstations into one system [9], forming a shared-nothing collection of processing nodes connected typically by a bidirectional 4-ary 2-fly multistage interconnection network that can achieve simultaneous any-to-any connections [21]. The packet-switched network can use, as an alternative to IP, a protocol named the User Space Communication Subsystem (US CSS) to provide near-constant latency and bandwidth. We have used US CSS as the baseline protocol in all our experiments on the SP2.

The **new generation IBM SP** showcases a cluster architecture where each of nodes of the machine is a 4-way SMP. An example of such a machine is the new IBM SP at Lawrence Livermore National Laboratory. Currently, this machine includes 158 compute nodes each with four 332 MHz 604e processors, sharing 512 MB of memory and attached to 1GB disks. The inter-node communications of the SP give a bandwidth of l00MB/second and a latency of 35 microseconds with the use of SP High Performance Switch TB3 (currently, this performance is possible when the application is running only on one of the four processors of the node). We have simulated the behavior of such a system running MPI applications. The inter-node communications are handled the same way as for the shared-nothing architecture, by modeling the communications between processors as using the high-performance switch. However, the intra-node communications are modeled as using shared memory. As information on the implementation of the MPI constructs designed to exploit shared memory were not yet available for the IBM SP (in fact, in the current implementation processors on a node communicate with each other using the much slower IP!), the COMPASS model is based on the MPI implementation on the SGI Origin 2000 [22]. Certainly, the performance of the application will depend on the exact implementation, but it allows us to demonstrate the capability of the tool in enabling such studies.

4. Validation and Performance of COMPASS

4.1 Validation

Our first set of experiments was aimed at validating the predictions of COMPASS for the IBM SP and SGI Origin 2000. Figure 1 is a graph of the execution time of the measured Sweep3D program compared to the execution time predicted by COMPASS. The curves are a function of the number of processors used by Sweep3D and the number of target processors simulated by COMPASS; validation is thus limited to the number of physical processors (on the Origin we have only **10).** The COMPASS data is taken as the average of the running times of all multithreaded combinations of the target processor number. (For instance, for eight target processors, the average running time was taken from executions with 1, 2, 4, and 8 host processors.) From the graph it is seen that COMPASS is indeed accurate, correctly predicting the execution time of the benchmark within 5% for the IBM SP and 3% for the O2K, even with multithreaded operation.

When conducting scalability studies, it is often the case that the number of available host processors is significantly less than the number of target processors. This results in several simulation threads running on the same processor. Since multithreading might affect the results of the simulation because threads might affect each other's runtime, it is important to study whether such effects exists. To quantify the effect of multithreading on the ability of the simulator to correctly predict the runtime of the application, we simulated Sweep3D using a wide range of host and target processors. As can be seen from Figure 2, even with a relatively high degree of multithreading of 8 target MPI processes to a single host processor, the variation in the predicted runtime is very small (below 2%).

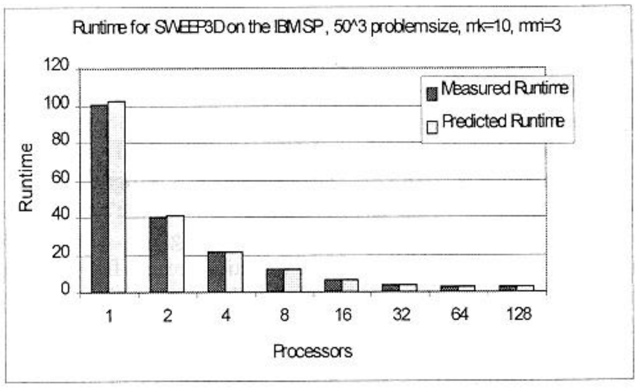

Figure 1(a): Validation of COMPASS for Sweep3D (IBM SP).

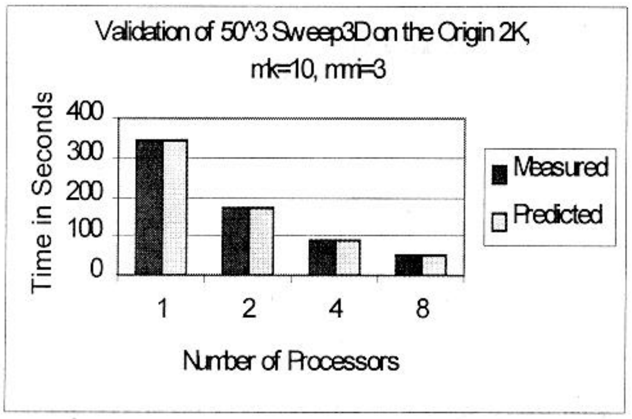

Figure 1(b): Validation of COMPASS for Sweep3D (O2K).

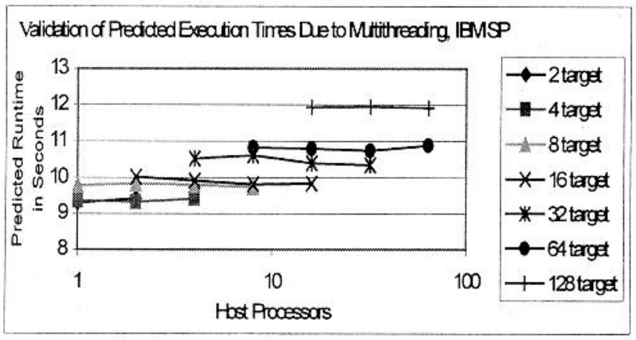

Figure 2: Effect of COMPASS' Multithreading on Predicted Performance (IBM SP).

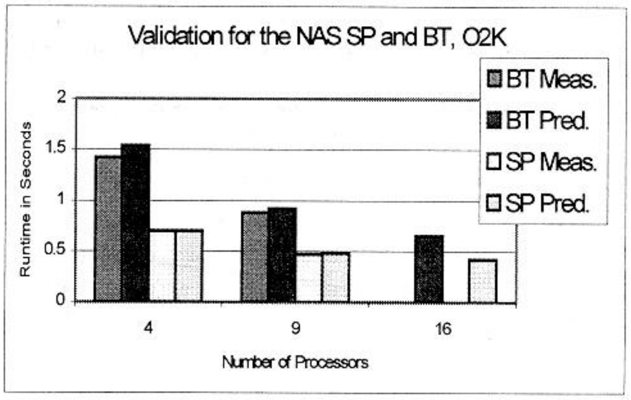

Figure 3: Validation for NAS benchmarks (O2K).

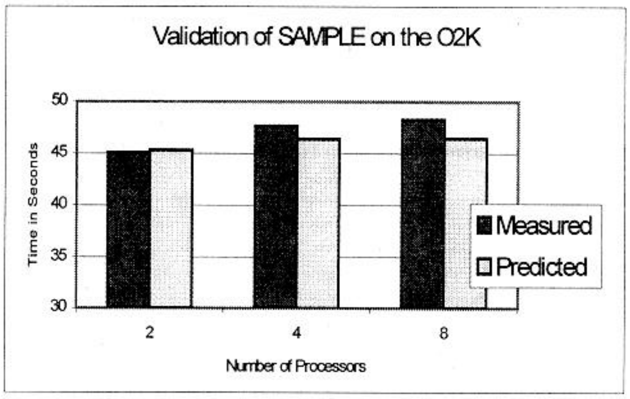

Figure 4: Validation for SAMPLE (O2K).

COMPASS was also validated with the suite of NAS benchmarks. Here we present the results for the SP and BT benchmarks for the Origin 2000. As mentioned earlier, the NAS programs come configured to run in parallel only on a predetermined number of processors and a predetermined set of problem sizes. The processor and memory constraints of our relatively small 02K restricted us only to the S size of these benchmarks. Figure 3 shows the results of the validation experiments for BT and SP (both class S). They show good validation, with accuracy to within 8.5% and 2.1%, respectively, at all points. For 16 processors, the graph shows only the predicted performance since only 8 host processors are available on the machine.

Since both Sweep3D and the NAS benchmarks are computationally intensive, we also used the communication-intensive synthetic benchmark (SAMPLE) to validate the communication models. The measured and predicted execution times for the SAMPLE benchmark also showed excellent validation of COMPASS for a variety of configurations. Figure 4 shows a sample run using the wavefront communication pattern and a computation-to-communication ratio of 11 to 1. As seen from the figure, COMPASS accurately predicts running time to within 3 percent; the results were similar for the other patterns and have been omitted for brevity.

4.2 Scalability of the Simulator

We present a number of results to demonstrate the relative improvement in the performance of the simulator that can be obtained with parallel execution. Figure 5(a) shows the performance of COMPASS when simulating the execution of Sweep3D for problem sizes of 50^3 and 100^3 cells on 64 target processors of the IBM SP. As seen from the figure, the simulator can effectively use additional processors; the parallel simulation on 64 processors achieves a speedup of almost 35 for the 100^3 problem size as compared with the sequential execution time of he simulator. The speedup for the 50^3-problem size is smaller, because, ultimately, the performance of the simulator is bound by the performance of the application.

Another metric commonly used to evaluate the performance of a simulator is the slowdown of the simulator relative to the target architecture. We define *slowdown(S,T) as (time to simulate the application using S host processors / time to execute the application on T processors).*

Figure 5(b) shows the slowdown of COMPASS when simulating a target problem size of 50^3. When the number of host processors is equal to the number of target processors, the simulator has a slowdown factor of less than 3. If the host architecture has fewer available processors than the target machine, the slowdown does get worse, but the overall performance is reasonable. Thus with an I-ratio (number of target processors / number of host processors) of 16 (64 target processors and 4 host processors), the slowdown factor is only 10.

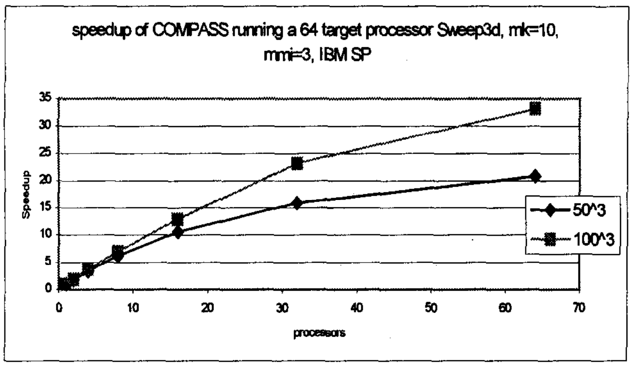

Figure 5 (a): Speedup of COMPASS on the IBM SP (Sweep3D).

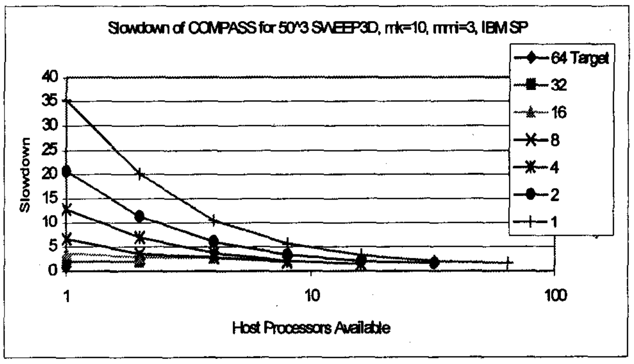

Figure 5(b): Slowdown of COMPASS on the IBM SP(Sweep3D).

The largest configuration studied with 1600 target processors, using only 64 host processors (I-ratio of 25) yielded a slowdown of only 18. This is considerably better than the slowdown factors that have been reported for other program simulators like WWT [23] and LAPSE [13], where the slowdown factors reported have been as high as 100 for computationally intensive applications.

In Figure 6(a), we show the speedup attained by COMPASS for the Origin 2000 while it simulates 32 target processors for two problem sizes of Sweep3D. For the Origin 2000, COMPASS achieves near-linear speedup as the number of host processors is increased, reaching a speedup of 7 when 8 host processors are used. The slowdown graph for an 8-target processor configuration is shown in Figure 6(b) and shows that for an I-ratio of 1, the simulator has a slowdown of 2. The slowdown with 4 host processors is slightly above 2, which shows that even if just half of the desired number of processors is available, the simulator runs only about twice slower the application on all target processors would.

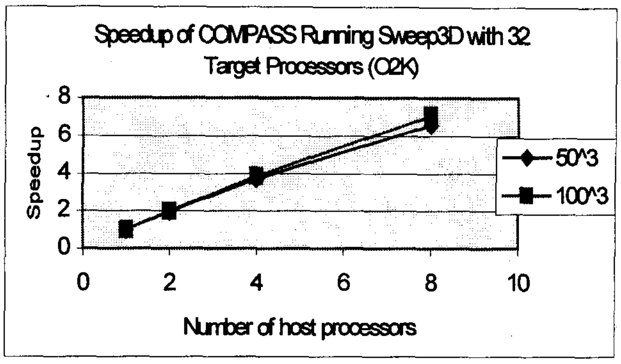

Figure 6(a): Speedup of COMPASS on the SGI Origin 2000 (Sweep3D).

The speedup and slowdown of COMPASS simulating the NAS benchmarks also show improvements with parallel execution, albeit to a lesser degree. Figures 7(a) and 8(a) show the speedup of the BT and SP applications, respectively. We see that the speedup of the simulator increases progressively as the number of host processors is increased, but the rate of increase as well as the final speedup attained with 8 hosts is lower than those seen with the previous benchmark. The simulator produces a speedup as high as 5.45 for the BT benchmark and 4.38 for the SP benchmark. Similarly, the slowdown curves reach a low of 1.42 and 1.67, respectively, for each application (see Figures 7(b) and 8(b)). Further investigation indicated that these applications did not scale as well as SWEEP3D, and hence the differences in the performance of COMPASS are directly related to the performance of the target program being simulated. The speedup and slowdown experiments show that COMPASS can exploit the parallelism available in the application without adding any considerable overhead.

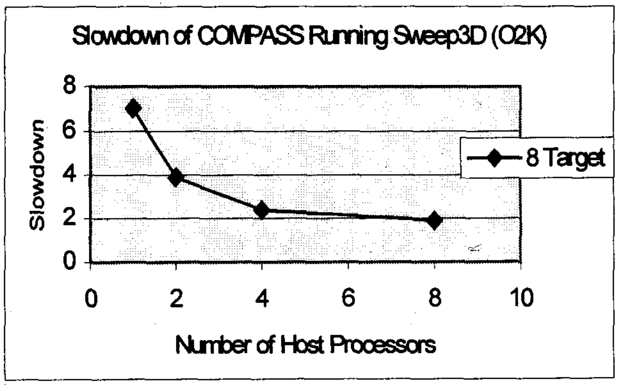

Figure 6(b): Performance of COMPASS on the SGI Origin 2000 (Sweep3D).

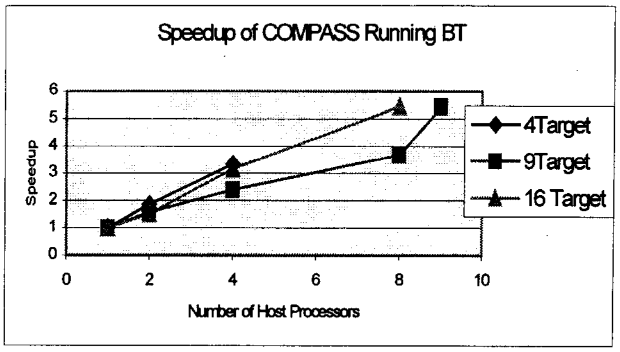

Figure 7(a): Speedup of COMPASS on the SGI Origin 2000 (BT).

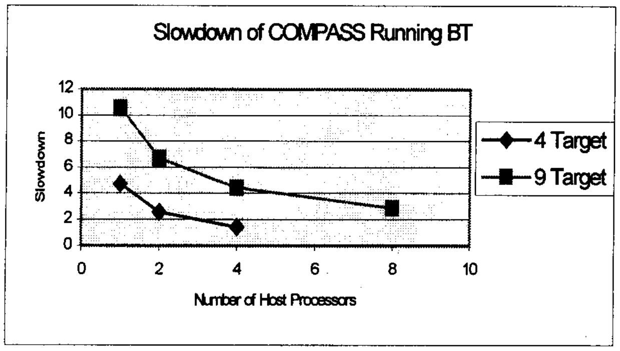

Figure 7(b): Slowdown of COMPASS on the SGI Origin 2K (BT).

5. Results and Features of COMPASS

Scalability of Sweep3DThe performance study first evaluated the scalability of Sweep3D as a function of various parameters including the size of the problem, the number of processors and as a function of network latency. We have performed this study on the IBM SP using up to 64 host processors. Figure 9(a) demonstrates the scalability of Sweep3D for three problem sizes: 50^3,100^3 and 150^3. For large problems, the study showed that their performance scales well as the number of processors is increased to almost 1600, although the relative improvement in performance drops beyond 256 processors. For the largest problem size, the runtime of the application was shown to be 125 times smaller running on 1,600 processors as compared to running the application on 4 processors. For the smaller problem size with elements, the performance appears to peak at about 1024 processors and subsequently gets worse. This observation was strengthened by the isoefficiency analysis, where the efficiency is defined as speedup *(S)/p* (number of processors).

The isoefficiency function determines at what rate the problem size needs to be increased with respect to the number of processors to maintain a fixed efficiency [16]. A system is highly scalable if the problem size needs to be increased linearly as a function of the number of processors. The total work W is the time to run the algorithm on a single processor, and T_p is the time to run the algorithm on p processors. $T_p= (W+T_o)/p$ (T_o= sum of overhead on all processors) giving the efficiency $E = 1/(1+T_o/W)$. If W needs to grow as $fE(p)$ to maintain efficiency $E, fE(p)$ is defined as the isoefficiency function.

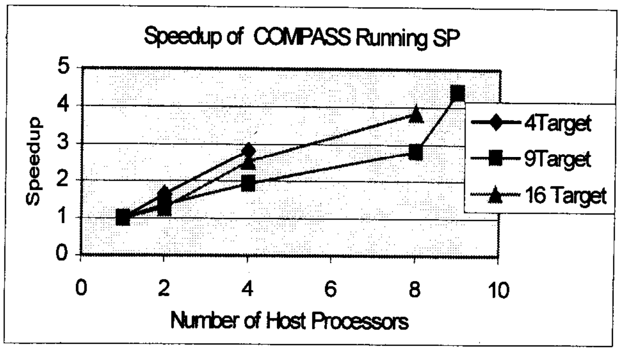

Figure 8(a): Speedup of COMPASS on the Origin 2K (SP).

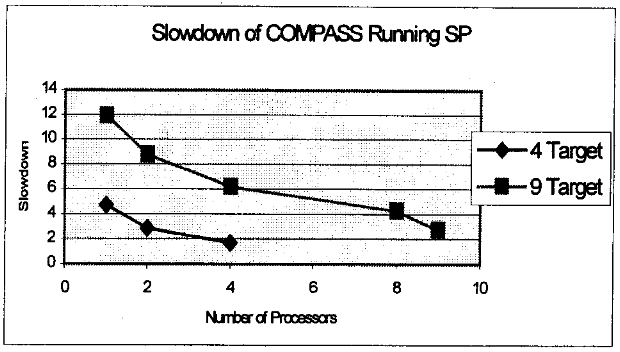

Figure 8(b): Slowdown of COMPASS on the Origin 2K (SP).

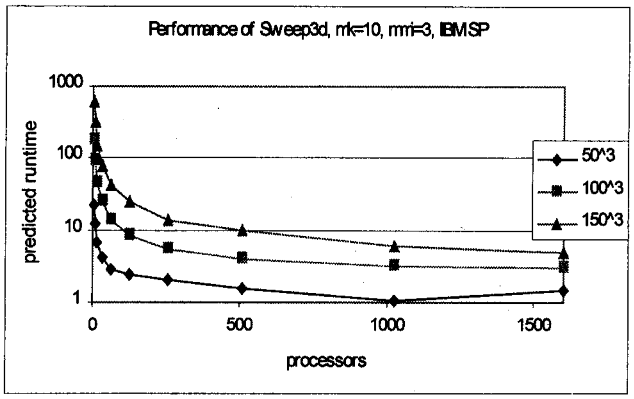

Figure 9(a): Scalability of Sweep3D on the IBM SP.

Figure 9(b) shows the isoefficiency function for Sweep3D for various numbers of efficiencies. The graph shows what problem size needed to maintain a given efficiency (20,40,60 or 90%) on a given number of processors. First, we observe that maintaining 90% or even 60% efficiency is hard. However 40% is more manageable. Second, using a large number of processors for a given problem size is not efficient. For example, for the 500,000 (about 22x22x1000) problem size, using less than 16 processors gives the best efficiency (about 90%), using 100 processors results in only 20% efficiency. Since running the problem on only 16 processors might result in slow runtime, a tradeoff between time and efficiency can be made and 36 processors can be used resulting in 60% efficiency. Figure 9(b) also demonstrates that isoefficiency is hard to capture with simple extrapolation. For example, the 40% isoefficiency curves flattens out for the 1.6 million problem size, implying that giving more processors to the application does not improve efficiency.

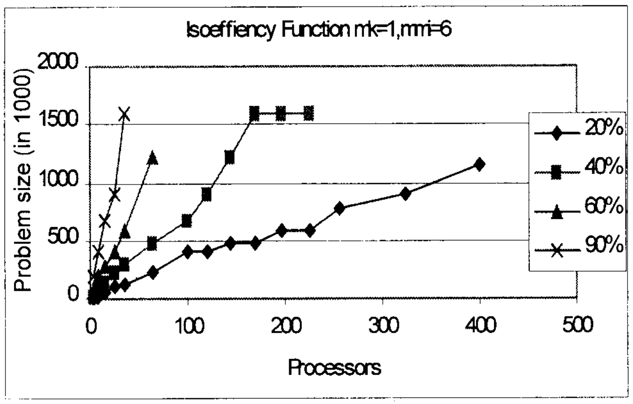

Figure 9(b): Isoefficiency for Sweep3D on the IBM SP.

5.1 Impact of Latency Variation on Performance

We have also studied the effect of communication latency on performance. Figure 10 shows the performance of Sweep3D as the latency in the network is varied, for problem sizes 50^3 and 150^3. As seen from the figure, a faster communication switch will not have a significant impact on this application—the performance changes by at most 5% for variations in latency between 0 and 10x the current switch latency. With more than 32 processors (128 for the larger problem), the difference is negligible. However, the performance does appear to suffer significantly if the latency is increased by more than a factor of 50, which might be the case if the application is ported to a network of workstations. Latency impacts are much more significant with a small number of processors, because each processor contains a larger portion of the computational region, causing messages to become large and more sensitive to latency.

5.2 Modeling SMP Cluster Architectures.

The preceding experiments evaluated application performance on the distributed memory architecture. New architectures, such as the IBM SP cluster architecture use 4-way SMP nodes as described in section 3.4 to exploit both the fast memory access of shared memory systems and the scalability of distributed memory machines. The next set of experiments projects improvements in the execution time of our benchmarks obtained by migrating to this architecture. Since the previous experiments showed that the NAS and Sweep3D benchmarks were relatively insensitive to the communication latency, it was hardly surprising that they did not appear to benefit noticeably from fast intra-node communication (for brevity, we omit these results). However, as demonstrated by the SAMPLE benchmark, for applications that have a higher percentage of communication, the new architecture appears to offer some benefits.

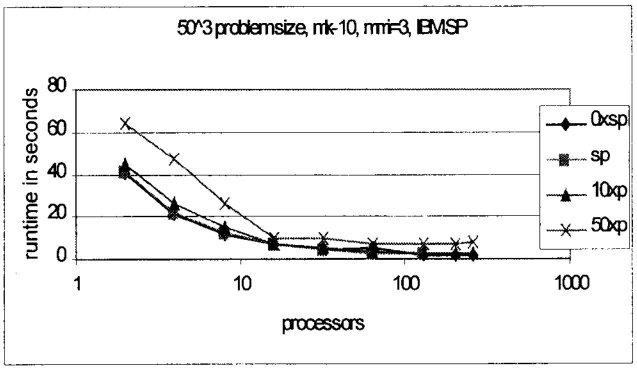

Figure 10(a): Sensitivity of Sweep3D to Latency (Small Problem Size).

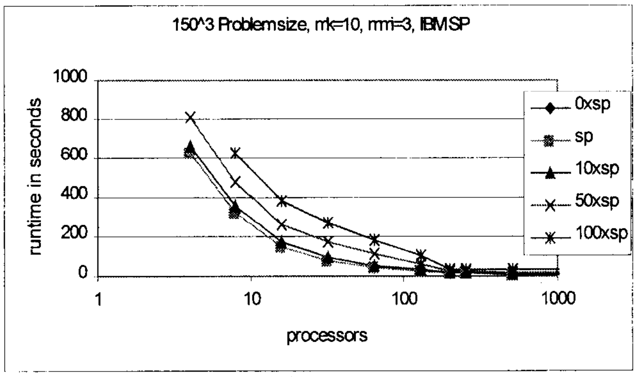

Figure 10(b): Sensitivity of Sweep3D to Latency (Large Problem Size).

Figure 11(a) shows the performance of SAMPLE for a fixed problem size per processor. We can see that the simulator validates well for the one processor per node case ("Meas. Non-SMP" and "COMPASS"). We also notice that we can predict a slightly better performance when running on an SMP node which would have support for fast intra-node communications ("COMPASS for SMP"), even though the current implementation of MPI communications on the SMP nodes has poor performance ("Meas. SMP"). Similarly, Figure 11(b) shows the performance of the SP running SAMPLE as a function of the number of computational iterations. Here, the time for communications is 37% of the total runtime. As the number of iterations increases, the ratio of computation to communication is kept constant. Again, we see that the predicted SMP performance improves on average by 20% as compared to the single processor per node performance, and we see clear drawbacks to using the intra-node communications as supported currently ("Meas. Current SMP").

Even though MPI on the SP does not support fast intra-node communications, the processors of the SMP do share the same main memory. This might tempt application developers to redesign existing MPI application to use main memory between processors of a node and MPI between nodes. A simulator like COMPASS can help make the decision where such an investment of time and effort would result in better performance.

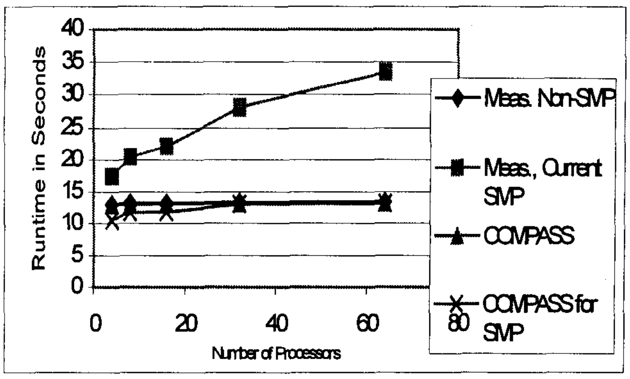

Figure 11(a): SMP Performance on the IBM SP (SAMPLE with Constant Computation Per Processor).

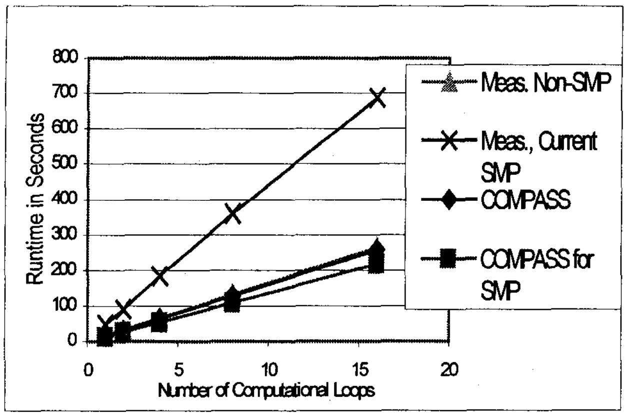

Figure 11(b): SMP Performance on the IBM SP (SAMPLE with Increasing Computation Per Processor).

5.3 Simulating Common Communication Patterns

Another set of experiments involved investigating the impact of different communication patterns on program performance through the use of our synthetic benchmark SAMPLE. Scientific programs can produce a wide variety of traffic patterns depending on the algorithm being used, and we sought to understand how these different types of message dispersal affected application performance. The SAMPLE benchmark was used to generate a number of such message-passing schemes for study. The *wavefront* pattern involves a 2-dimensional mesh with the 0-th processor, residing on the upper-left-hand corner, initializing a communication wave towards the lower-right-hand corner. Using the same mesh layout, the *nearest-neighbor* dispersal has each processor sending (and receiving) a message from each of its four logically adjacent processors. The *ring* pattern forms a cycle where a single message token is sent around a logical "ring" of processors. Finally, for the *one-to-all* pattern, a processor broadcasts a message, that is routed using a broadcast tree to all others. The performance of the various communication patterns was evaluated as a function of communication latency and the number of processors. The host machine selected for the experiments was the Origin 2000 with 8 processors. Figure 12(a) shows the performance of SAMPLE as a function of latency for a target O2K architecture with 16 processors and Figure 12(b) shows the performance as a function of number of processors in the target architecture. As expected, the ring pattern was most sensitive to the latency and processor count as the message traverses sequentially through a ring. The somewhat surprising result was the relative insensitivity of the wavefront and on-to-all communications; however, note that both these patterns do not block the initiator processor. Immediately after initiating the communication, the corresponding process executes the next iteration, which is hence reasonably well overlapped with the communication, producing the observed insensitivity. The slight jump in the predicted execution time with increasing processors was attributed to a change in the depth of the broadcast tree (Figure 12(b)).

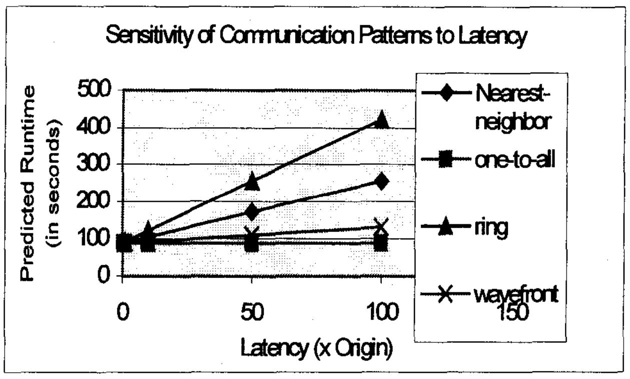

Figure 12(a): Performance of Communication Patterns as a Function of Latency.

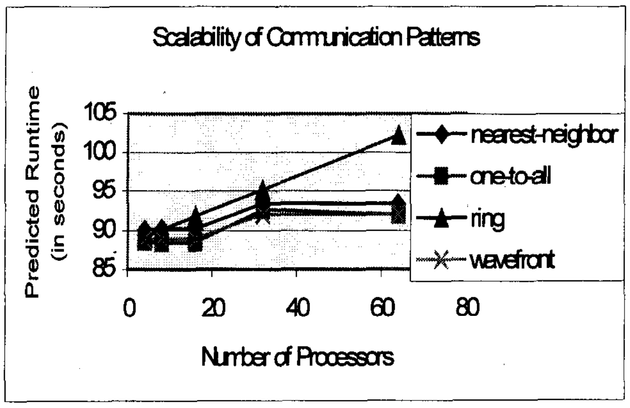

Figure 12(b): Performance of Communication Patterns as a Function of Number of Processors (O2K)

5.4 Effect of Latency on Parallel File System Caching Strategies

The last experiment demonstrates the use of the simulator in evaluating the impact of architectural features on I/O intensive programs. Cooperative caching techniques were proposed to improve the performance of applications with large I/O requirements [7,10] by suggesting that the caches be, at least partially, managed globally rather than in an entirely local manner. In all cases, compute node (cnodes) and I/O nodes (ionodes) have caches. Base caching simply allows each node to manage only its own cache. Greedy forwarding allows an ionode that has a cache miss to check if any other node is caching the required data (before going to fetch it from the disk). In centrally coordinated caching, portions of the cnode caches are collectively managed by the ionodes. The remaining portion of the cnode cache is managed locally by the cnode. The percentage of coordinately managed cache can be varied (as it is in our experiment). Globally managed caching is similar to 100% coordinate caching, except the strategy for block placement in caches is modified to allow the ionode caches to hold data evicted from the cnode caches. As these caching techniques depend on having efficient access to remote memory in order to improve cache hits rates and application performance, their performance should be dependent on the communication latency in the network.

Figure 13 shows the results from a set of experiments designed to measure the impact of changing network latencies of the IBM SP on the cooperative caching techniques supported by COMPASS. In this benchmark, 16 processes on separate compute nodes randomly read and write 512 byte blocks of data. The blocks are all in the same file, which is distributed across 2 I/O nodes, each with 2 disks (for a total of 4 disks). Each process issues 10,000 requests, with the first 5,000 requests being used to warm up the caches and with 80 of the requests being read requests. The graph plots the predicted execution time of the benchmark as the network latency is increased. Caching performance for base caching (no cooperation), greedy forwarding, centrally coordinated (with 40, 80 and 100 percent coordination) and globally managed caching are shown for network latencies of 0, 1, 10, and 100 times the latency of the SP2 interconnect.

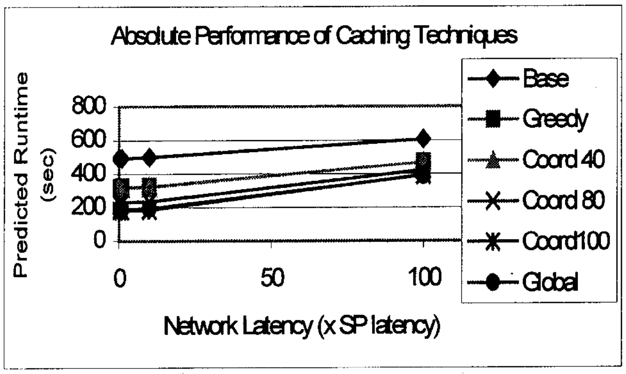

Figure 13(a): Comparison of Caching Techniques on the IBM SP.

Understandably, as the network latency is increased, the predicted execution time of the benchmark also increases. However, this experiment also hints at the extreme sensitivity of the cooperative caching techniques to increased network latency. While it may appear that all caching techniques (even base caching) are equally affected by the increasing network latency, this was not found to be the case. While the absolute difference in predicted execution time diminishes only slightly as the latency is increased, the relative difference between different caching techniques decreases markedly, as shown in Figure 13(b). In effect, as the network becomes slower, the benefit of using cooperative caching is lost and performance degrades to only slightly better than that of base caching. This result has important implications for the use of this technique in large networks of workstations and in the design of hybrid strategies, where the caches are managed cooperatively over small regions of the network rather than over the entire network.

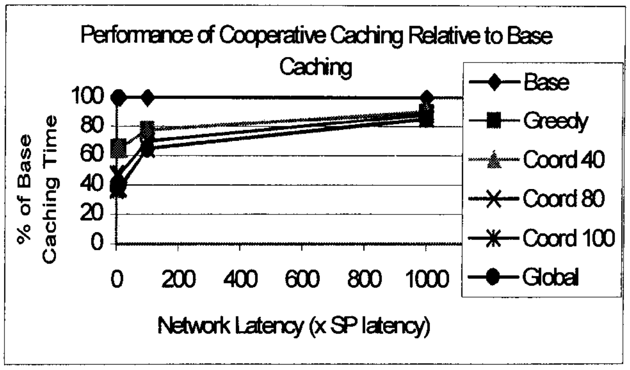

Figure 13(b): Performance of Caching Techniques Relative to Base Caching on the IBM SP.

6. Related Work

Accurate and efficient performance prediction of existing parallel applications on target machines with thousands of processors is a challenging problem. The first generation of simulators like Proteus [4] used sequential simulation, which were very slow with slowdown factors ranging from 2 to 35 *for each process in the target program*. This led to many efforts in improving the execution time of program simulators: DP-Sim [25] LAPSE [13], Parallel Proteus [20], SimOS [30], Wisconsin Wind Tunnel [28], Tango [11], and MPI-SIM [26,27] have all been designed for this purpose. The simulators typically use direct execution of portions of the code to reduce the cost of simulating sequential instructions and typically use a variation of the conservative parallel discrete-event simulation [5] algorithm to exploit parallelism within the simulator to reduce the impact of scaling up the target machine size.

Many parallel simulators use the synchronous approach to simulation where simulation processes synchronize globally at fixed time intervals in order to maintain program correctness. The interval or quantum is taken to be no larger than the communication latency of the network being simulated. This guarantees that a message sent in one quantum cannot be received until the next interval. This also implies that messages will be processed in a correct order. Some of the synchronous simulators are Proteus, a parallel architecture simulation engine, Tango, a shared memory architecture simulation engine, Wisconsin Wind Tunnel (WWT), a shared memory architecture simulation engine and SimOS, a complete system simulator (multiple programs plus operating system). In terms of simulation of communications, two simulation engines, which use approaches, similar to ours are Parallel Proteus and LAPSE. A distinguishing feature of COMPASS is that it is portable, in part due to being implemented with the use of MPI. Since MPI is readily available on any parallel or distributed system, the simulator is able to use it for data movement and synchronization. On the other hand, LAPSE was designed specifically to run on the Intel Paragon, using the Paragon's native communication primitives. This made LAPSE broad usefulness limited. COMPASS is also fast, having slowdowns of around 2, where Proteus' typical slowdowns are in the range of 35-100[4].

A number of simulators have also been designed to simulate I/O operations, although most of these have tended to use sequential simulators. A set of collective I/O implementations was compared using the STARFISH [18] simulator, which is based on Proteus. In [3], a hybrid methodology for evaluating the performance of parallel I/O subsystems was described. PIOS, a trace-driven I/O simulator, is used to calculate the performance of the I/O system for a subset of the problem to be evaluated, while an analytical model was used for the remainder. Scalability of distributed memory machines was examined in [31], which used application kernels to investigate network performance and contention. Libraries have also been developed. PPFS [15] is a portable parallel file system library designed to sit on top of multiple UFS instances and provide a wide variety of parallel file system capabilities, such as caching, prefetching, and data distribution.

The COMPASS environment described in this paper used the parallel I/O system simulator detailed in [1] and is perhaps the only simulator that combines the ability to do integrated interconnection network, I/O and file system, and scalability studies. It has also been used for the simulation of data parallel programs compiled to message-passing codes [25]. Additionally, the simulator itself is highly scalable, with slowdown factors in the single digits for large target applications and architectures.

7. Conclusions and Future Research

We have demonstrated that COMPASS can be used to study a wide range of applications as a function of a variety of architectural characteristics ranging from standard scalability studies through network stress test and parallel I/O properties. We have shown that not only is COMPASS accurate (having validated it on multiple applications and architectures to within a few percent of the physical measurements), but it is also fast achieving excellent performance both on the IBM SP as well as on the SGI Origin 2000. It achieves near-linear speedups for highly parallel applications and suffers only from moderate slowdowns. It has been shown to be useful for a wide range of architectural performance studies that combine the separate areas of I/O and parallel file system performance with interconnection network and communication library simulators. COMPASS is being used for detailed program simulations within the POEMS project. In collaboration with other "POETS" we are working on developing hybrid performance models which combine analytical and simulation modeling techniques. Also, as part of the project, COMPASS will be integrated with a detailed memory and processor model. This will allow us to break away from the dependency of requiring a host processor architecture that is similar to the target processor architecture for direct execution simulation. This will also provide an opportunity to extend the use of parallel simulation techniques for processor and memory simulations

8. ACKNOWLEDGMENTS

This work was supported by the Advanced Research Projects Agency, DARPA/CSTO, under Contract F-30602-94-C-0273, "Scalable Systems Software Measurement and Evaluation" and by DARPA/ITO under Contract N66001-97-C-8533, "End-to-End Performance Modeling of Large Heterogeneous Adaptive Parallel/Distributed Computer/Communication Systems."

Thanks to the Office of Academic Computing at UCLA and to Paul Hoffman for help with the IBM SP2 as well as to the Lawrence Livermore National Laboratory for use of their IBM SP on which many of these experiments were executed.

9. REFERENCES

[1] R. Bagrodia, S. Docy and A. Kahn. "Parallel Simulation of Parallel File Systems and I/O Programs," *SuperComputing'97*, 1997.

[2] D. Bailey, T. Harris, W. Shaphir, R. van der Wijngaart, A. Woo, and M. Yarrow. "The NAS Parallel Benchmarks 2.0," Report NAS-95-090, NASA Ames Research Center, 1995.

[3] S.J. Baylor, C. Benveniste and L.J. Beolhouwer. "A Methodology for Evaluating Parallel I/O Performance for Massively Parallel Processors." In *Proceedings of the 27th Annual Simulation Symposium*, 1994, pp.31-40.

[4] E.A. Brewer, C.N. Dellarocas, A. Colbrook and W.E. Weihl. "Proteus: A High-Performance Parallel Architecture Simulator," MIT Technical Report MIT/LCS/TR-516, 1991.

[5] M. Chandy and J. Misra. "Distributed Simulation: A Case Study in Design And Verification Of Distributed Programs," *IEEE Trans. on Software Engineering*, Sept. 1979, pp.440-452.

[6] P. F. Corbett and D. G. Feitelson. "The Vesta parallel file system," *ACM Transactions on Computer Systems*, 14(3):225-264, August 1996.

[7] T.Cortes, S.Girona and J.Labarta. "Avoiding the Cache-Coherence Problem in Parallel/Distributed File System," in Proceedings of the High-Performance Computing and Networking Conference, 1997, pp. 860-869.

[8] R.G. Covington, S. Madala, V. Mehta, J.R Jump and J.B. Sinclair. "The Rice parallel processing testbed." In *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.*

[9] D. Culler, J.P. Singh, with A. Gupta. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, Inc. 1999.

[10] M.H. Dahlin, R.Y. Wang, T.E. Anderson and D.A. Patterson. "Remote Client Memory to Improve File System Performance." In *Proceedings of the 1994 Symposium on Operating Systems.*

[11] H.Davis, S.R. Goldschmidt and Hennessey. "Multiprocessor Simulation and Tracing Using Tango." In *Proceedings of ICPP'91*, pp. 99-107, August 1991.

[12] E. Deelman, A. Dube, A. Hoisie, Y. Luo, R. Oliver, D. Sundaram-Stukel, H. Wasserman, V.S. Adve, R. Bagrodia, J.C. Browne, E. Houstis, O. Lubeck, J. Rice, P. Teller, and M. K. Vernon. "POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational Systems." In *Proceedings of the First International Workshop on Software and Performance '98* - WOSP '98, October 12-16, 1998, Santa Fe, New Mexico.

[13] P.M. Dickens, P. Heidelberger, and D.M. Nicol. "Parallel Direct Execution Simulation of Message-Passing Parallel Programs," *IEEE Transactions on Parallel and Distributed System 1996.*

[14] *f2c (Fortran to C converter)*, http://www.netlib.org/f2c/

[15] J.Huber, C.L.Elford, D.A.Reed, A.A.Chien, and D.S.Blumenthal. "PPFS: A High Performance Portable Parallel File System." In *Proceedings of the 9th ACM International Conference on Supercomputing*, pp.385-394, July 1995.

[16] V. Kumar and A. Gupta. "Analysis of Scalability of Parallel Algorithms and Architectures: A Survey," International Conference on Supercomputing, 1991, pp.396-405.

[17] V. Kumar, A. Grama, A. Gupta, and G. Karypis. Introduction to Parallel Computing: Design and Analysis of Algorithms. The Benjamin/Cummings Publishing Company, Inc. 1995.

[18] D. Kotz. "Tuning STARFISH," *Technical Report PCS-TR96-296.* Department of Computer Science, Dartmouth College, October 1996.

[19] J. Laudon and D. Lenoski. "The SGI Origin: A ccNUMA Highly Scalable Server," The 24th Annual International Symposium on Computer Architecture, May 1997.

[20] "ASCI Blue-Pacific IBM RS/6000 TR System at Lawrence Livermore National Laboratory," http://www.llnl.gov/asci/platforms/bluepac/tr.hwtable.html.

[21] U. Legedza and W.E. Weihl. "Reducing Synchronization Overhead in Parallel Simulation," *10th Workshop on Parallel and Distributed Simulation, PADS'96*, pp. 86-95.

[22] Y. Luo. "MPI Performance Study on the SGI Origin 2000," *Pacific Rim Conference on Communications, Computers and Signal Processing*, 1997, pp.269-272.

[23] S.S. Mukherjee, S.K. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M.D. Hill, J.R. Larus, and D.A. Wood. "Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator," *Workshop on Performance Analysis and Its Impact on Design (PAID), 1997.*

[24] N. Nieuwejaar and D. Kotz. "The Galley Parallel File System." In *Proceedings of the 10th ACM International Conference on Supercomputing*, 1996, pp.188-195.

[25] S. Prakash and R. Bagrodia. "Parallel Simulation of Data Parallel Programs," Proceedings of the 8th Workshop on Languages and Compilers for Parallel Computing, Columbus, Ohio, August 1995

[26] S. Prakash. "Performance Prediction of Parallel Programs," Computer Science Dept, UCLA, Ph.D. thesis, 1996.

[27] S. Prakash and R. Bagrodia. "Using Parallel Simulation to Evaluate MPI Programs." In *Proceedings of the 1998 Winter Simulation Conference*, Dec. 12-13, 1998 in Washington D.C.

[28] S.K. Reinhardt, Mark D. Hill, J.R. Larus, A.R. Lebeck, J.C. Lewis and D.A. Wood. "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers." In *Proceedings of the 1993 ACM SIGMETRICS Conference*, May 1993.

[29] J.M. del Rosario, R. Bordawekar and A. Choudhary. "Improved Parallel I/O via a Two-Phase Runtime Access Strategy." In *Proceedings of the IPP'93 Workshop on I/O in Parallel Computer Systems*, 1993, pp. 56-70.

[30] M. Rosenblum, E. Bugnion, S. Devine and S.A. Herrod. "Using the SimOs Machine Simulator to Study Complex Computer Systems," *ACM Trans. On Modeling and Computer Simulation*, Vol.7, No. 1, January 1997, pp. 78-103.

[31] A.S. Sivasubramaniam, A.Singla, U.Ramachandran and H.Venkateswaran. "A Simulation Based Scalability Study of Parallel Systems," *Journal of Parallel and Distributed Computing*, 22:411-426, 1994.

[32] "The ASCI sweep3d Benchmark Code," http://www.llnl.gov/asci_benchmarks/.