Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

USING ZPL TO DEVELOP A PARALLEL CHAOS ROUTER SIMULATOR

Wilkey Richardson

Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721, U.S.A.

Mary L. Bailey

Computer Science Department
University of Arizona
Tucson, AZ 85721, U.S.A.

William H. Sanders

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.

ABSTRACT

This paper reports on our experience in writing a parallel version of a chaos router simulator using the new data-driven parallel language ZPL. The simulator is a large program that tests the capabilities of ZPL. The (parallel) ZPL program is compared with the existing serial implementation on two very different architectures: a 16-processor Intel Paragon and a cluster of eight Alpha workstations. On the Paragon, the simulator performs best when simulating medium- to large-sized routers, and on the Alpha cluster, it performs best when simulating large routers. Thus a user can choose the parallel platform best suited to the router size.

1 INTRODUCTION

There are some simulators that are good candidates for parallel simulation. These simulators are generally too slow on a sequential machine and are much faster on a parallel or distributed machine. The first criterion is often much easier to judge than the second, since the second is only truly verified after the implementation of a (good) parallel simulator. A sequential simulator has been implemented to simulate a chaos router, a randomizing, non-minimal adaptive router for multicomputers (Bolding and Snyder 1992, Konstantinidou and Snyder 1994). It exhibits the first characteristic, slowness, when simulating large routers. The router simulator has been used extensively on a sequential machine, and one version of the router node has been implemented in VLSI (Boulding et al. 1994). While the sequential simulator proved sufficient for many different router configurations, it proved to be impractical for very large router sizes. Thus it becomes a candidate for a parallel simulation, assuming the parallel version can run much faster than the serial one and, in this case, can simulate larger router configurations than are practical using the sequential simulator.

There are several issues to consider when implementing a parallel simulation. First is whether the simulation is event-driven or time-driven. Event-driven strategies are most appropriate when the activity of the system being simulated is uneven. This is not true in the chaos router simulator under heavy loads. Almost all router nodes are active during each time interval, so a time-driven simulation is preferred, eliminating the overhead of event management.

The second issue is the implementation language. Ideally, the simulator will be portable, implying the use of some type of standard parallel language or communication library. There are many parallel languages that could be selected. We chose to use ZPL, a new data-driven portable parallel language that is a sublanguage of a more general family of parallel programming languages A-ZPL designed by Lin and Snyder (1992, 1993). ZPL is attractive for this parallel simulation for three reasons. First, a data-driven model is appropriate given the time-driven simulation model. Second, ZPL is portable, so users can run the parallel simulation on various platforms. Third, ZPL is a sublangague of A-ZPL. Many simulators are event-driven and thus cannot use a data-driven language but require a more general parallel language such as A-ZPL.

This paper reports on our experience writing a parallel version of the chaos router simulator using this new portable data-driven parallel language ZPL. We compare the resulting parallel simulator with the serial one on two very different platforms: the Intel Paragon and a network of Alpha workstations. The Intel Paragon performs best for small- to medium-sized problems but runs into memory problems for the largest sizes. The Alpha network performs poorly for small- to medium-sized problems but performs well for larger-sized problems. In fact, it can

simulate larger sized routers than can be simulated by the Paragon or by a single Alpha workstation. Thus we show that our original goal is met – the parallel simulator can run faster than the serial one on the Paragon for many router sizes, and we can simulate larger router nodes (mainly on the Alpha cluster due to memory limitations on the Paragon). Moreover, we believe that our experiences using ZPL are valuable to other parallel programmers who wish to use this new language. We believe that the parallel chaos router simulator is the largest ZPL program that has been written to date.

2 THE PARALLEL CHAOS ROUTER

The chaos router is a randomizing, nonminimal adaptive router for multicomputers. It dynamically selects communication paths depending on network traffic, thus bypassing network congestion. Moreover, messages do not necessarily take a minimal path between source and destination – they can be *derouted*, or steered off a minimal path, to avoid congestion. The chaos router uses randomization to eliminate the need for livelock protection, thus probabilistically ensuring that each message reaches its destination. We briefly describe the chaos router algorithm, focusing on those parts of the algorithm critical in the simulation; the router is fully described in Bolding and Snyder (1992).

The current router specification assumes that the communication structure is an n-dimensional torus, with a processor/router pair at each torus node. We further constrain the communication structure to be a two-dimensional torus, as in the hardware implementation. Each processor communicates with its corresponding router node; each router node communicates with both its corresponding processor and its four adjacent router nodes. A block diagram of a router node is shown in Figure 1.

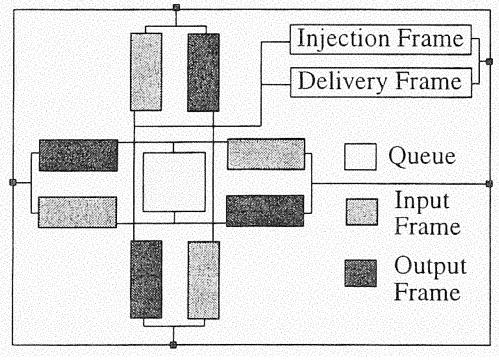

Figure 1: A Chaos Router Node

Messages enter the routing network through the Injection Frame and exit through the Delivery Frame. Both are actually FIFO buffers that connect the router to its corresponding processor. There are also two buffers associated with each of the four router communication channels: the Input Frame and Output Frame. When a message enters the router, either through the Injection Frame or an Input Frame, its header is read to determine which Output Frames and/or Delivery Frame can be used to profitably route the message. These desirable output buffers are then searched in a cyclical order, with the first free buffer receiving the message. If no desirable output buffer is free, then after a specified number of cycles the message is stored in a central Queue. Messages in the Queue have priority for accessing outgoing channel(s) when they become free. If a message is destined for the Queue, but the Queue is full, a message in the Queue is derouted. The message to be derouted is picked randomly.

3 THE ZPL IMPLEMENTATION

The parallel simulator uses the fact that there is a uniform torus of router nodes to create an array or ensemble of router nodes for the ZPL language. Ensembles in ZPL are automatically distributed across processors but are managed as a single entity in a ZPL program. Once the router ensemble was created, most of the ZPL simulation was a relatively straightforward implementation of the single-router node operation. Communication among the elements of an ensemble are facilitated by ZPL operators, including operators for "wrapping" data around the edges. It should be noted that even though the ZPL program is written in a purely SIMD style, the code is not executed in lock-step, as in an SIMD architecture. There are specific places in the code where synchronizations are performed; otherwise each processor runs its portion of the code asynchronously (Lin and Snyder 1993).

Our original implementation was quite slow; a 64-node router ran 16 times slower on a one-processor Paragon than the serial version! We thus focused initially on optimizing the ZPL code. There were four areas where we optimized: (1) communication, (2) memory usage, (3) procedure calls, and (4) statistics. The first two were needed mainly because of our naive translation of the serial router code into ZPL code. The third was needed because of the way that ZPL compiles procedure calls. The fourth was needed because of the way router statistics are computed, where global data are required. We will discuss each of these four areas, focusing on those aspects that might help other ZPL programmers.

In ZPL, there are two operators that are used to communicate data within an ensemble, the "wrap" and "@" (at) operators, both of which take an ensemble as their operand. In the chaos router, the data for each router node are encapsulated in a single ensemble, which we initially used as the operand for these operators. However, these operators invoke communication, so when we wrapped an ensemble node, the ZPL compiler packaged the entire router node (approximately 5KB) into a message and sent

it to another processor. This resulted in much larger messages than were necessary. By modifying the code to use only the portion of the node that was needed, we reduced the message size to 24 bytes, resulting in a 47% decrease in execution time for a 64-node router.

We had to pay close attention to memory usage in the router simulation because ZPL does not support dynamically allocated data structures, which the serial router simulator uses extensively. In some places, it is easy to avoid the dynamic structures by using ensembles; in other places, it is more difficult. For example, each node has a queue of messages that are being injected into the router by the processor associated with this node. This queue is typically quite small, but it can grow large when the router is congested. For the ZPL implementation, we statically fixed the sizes of these data structures to be the largest size supported in the serial simulator. In the case of the injection queue, this resulted in a 4KB queue per node. As the number of nodes grows, this becomes a significant amount of memory, which is often in short supply. Thus we more carefully determined the appropriate size for each data structure, shrinking them whenever possible. In the case of the injection queue, the amount of memory was reduced to 200 bytes per node with no loss of functionality. This resulted in a 17% savings in execution time for a 64-node router.

The third problem area involved procedure calls. There are two types of procedures in ZPL, parallel and promoted procedures. A parallel procedure uses a parallel construct, references an ensemble, or performs I/O. All other procedures are promoted procedures. Parallel procedures cannot be called from promoted procedures. When a promoted procedure is called from a parallel procedure, the promoted procedure is called in parallel, once for each element in the ensemble found in the parallel procedure. We originally assumed that the optimal performance would be obtained using parallel procedure calls, since it would allow the ZPL compiler to exploit the natural data parallelism. In fact, this is not true. Because ZPL is not recompiled for different numbers of processors, the ZPL compiler must insert additional code each time an ensemble is accessed in order to dereference ensemble elements, making it expensive to unnecessarily access ensembles. This is never a problem in promoted procedures, since the dereferencing is done before the procedure is called. We thus reinspected the code, creating as many promoted procedures as possible. In addition, we subdivided many parallel procedures to make additional promoted procedures, thus reducing ensemble references. This reduces the special code required for ensemble accesses, resulting in less overhead. These procedure changes resulted in a surprising 16% decrease in execution time.

The final optimization involved our statistics collection, which are collected to better understand the performance of the chaos router. The serial simulator computes statistics on a global basis while it is running, combining information from all nodes. We had two options for the parallel simulation: to collect statistics on a per-node basis and combine the data at some intervals, or to collect statistics on a per-processor basis. We began with the first option, to collect statistics on a per-node basis, combining the data at a "reasonable" interval. We immediately faced a trade-off. The longer the interval was between combining data, the more memory the simulation used to store the data it was collecting. On the other hand, the combination basically was a global sum, which is an expensive, serial operation (even with language support from ZPL). We were not able to find a good point in this space. If we combined data too frequently, we had communication bottlenecks, and if we waited too long we used up a large amount of memory, causing thrashing. Thus we considered the second alternative: collecting data on a per-processor basis. Unfortunately, there is no provision for this type of operation in ZPL. After talking with the ZPL designers, we wrote some C subroutines for data collection that were linked in with the ZPL code. This provided a "hook" for our data collection and improved the performance of the parallel simulation. While this optimization is more evident for larger configurations of the chaos router, we still obtained a 5% improvement for a 64-node router.

These four optimizations reduced the run-time of the ZPL implementation dramatically. The ZPL implementation now runs only 2.38 times slower than the serial implementation on a one-processor Paragon. Figure 2 illustrates the improvement for each of the four optimizations for a 64-node router on the Intel Paragon. Note that some optimizations such as statistics will show greater

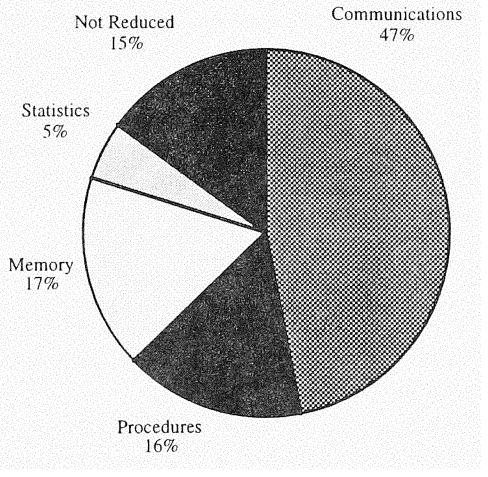

Figure 2: Improvement in Execution Time for ZPL Optimizations

improvement for larger router sizes. There is still room for improvement. For example, the ZPL compiler packages code that is passed between ensemble elements residing on the same processor as if it were being sent to another processor, increasing overhead. With additional compiler improvements and improvements in our code, we expect the execution time of the parallel code to continue to improve, although we don't expect the ZPL program to ever be as fast as the serial one.

In summary, our experience with ZPL has been quite positive. The language proved to be easy to use, with good documentation (Forman and Lin 1994, Lin 1994, Snyder 1994), and the compiler is reasonably efficient. The ZPL compiler was still under development during this project, and the compiler team was very helpful in fixing bugs that we found in the compiler and telling us when we were at fault.

4 EVALUATION OF THE SIMULATION

We now turn to the performance of the chaos router simulator on multiple processors. There are two main evaluation metrics that were used to measure the performance of the parallel simulator:

- The execution time of the parallel simulator and its relationship to the speed of the (existing) sequential simulator, and
- The portability of the parallel simulator and its ability to run on different architectures.

Performance tests were run on (1) a 16-node Intel Paragon and (2) a small network of Digital Alpha workstations (eight workstations). These are two very different architectures, with different computation/communication ratios. Thus they should provide useful insight into any architecture-related issues in performance.

Three parameters were varied in these experiments:

- Load, the number of router messages generated as a percentage of the rate that would saturate the network;
- Processors, the number of processors used in the parallel simulation; and
- Nodes, the number of router nodes in the simulation.

We first discuss the performance of the parallel simulator on each of these platforms, and then we discuss the impact of the platform on the simulation.

4.1 The Intel Paragon

The Intel Paragon used in these experiments has 16 processors, each with 32MB of memory. Below are the parameters used for these experiments.

Load:	20%, 40%, 60%, 80%
Processors:	1, 2, 4, 8, 16
Nodes:	16, 64, 256, 1024, 4096, 16384

Not all possible permutations of these parameters were run; the larger-sized experiments (nodes greater than 1024) were measured only on the larger numbers of processor due to performance problems. These will be discussed later in this section.

Below is a graph of the execution times for all of the experiments with a load of 80%. This load was chosen since it is the heaviest load. In all cases, the one-processor parallel version is somewhat slower than the serial version, so additional processors are needed to "break-even" or to show an improvement. Not surprisingly, the 16-node instance fails to run faster than the serial implementation, but all other versions do outperform the serial implementation, given sufficient numbers of processors. The 4096 instance was not run on the Paragon for one and two processors, due to thrashing (all processors shared the same disk, so performance seriously degraded when much swapping occurred). For similar reasons, the 16384-node instance was only run on 16 processors. This version was too large to be run on the serial simulator. As the instances grow, the relative performance increases, and we see greater improvements over the serial version. Moreover, we are able to run larger experiments on the Paragon than can be run on the serial version (the 16384-node instance). We now discuss specific issues in the experiments to illustrate performance issues in the parallel simulator.

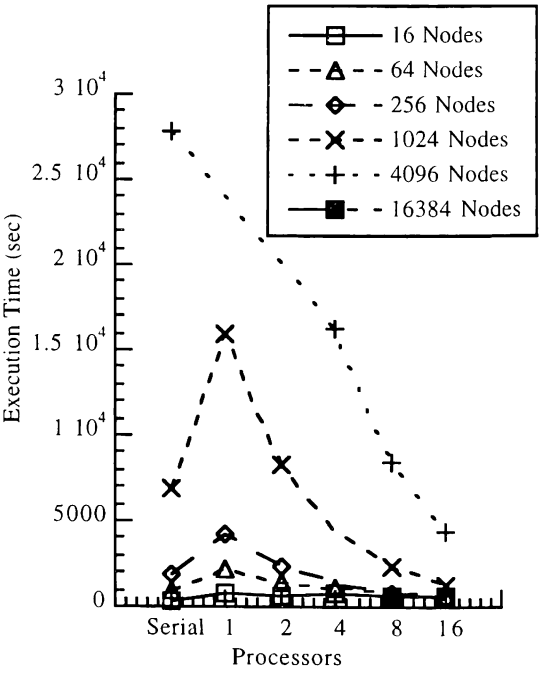

Figure 3: Execution Times on the Intel Paragon

The 16-node experiments were run to see how the communications affected the performance of the parallel simulator. Clearly a parallel machine is not necessary for this

sized experiment. In particular, this instance stresses the communications of the Paragon, since there is a small computation/communication ratio. We had, in fact, expected to see performance degrade as the number of processors was increased. To our surprise this didn't happen. The execution time actually decreased slightly when going from 8 to 16 processors.

The 64-node instances were again used to calibrate the "break-even" point for the parallel simulation. Here we observed some improvement over the serial version above two processors. It is in the 256-node instances where the performance of the parallel simulator begins to show reasonable improvement over the serial simulator. Here, we gain an improvement of approximately 3.5 over the serial version (and an improvement of 8 over the one-processor parallel version). This trend continues in the larger instances.

We now address speedup. Figure 3 shows the speedup of the parallel simulator vs. the serial simulator (effective speedup), and Figure 4 shows the speedup of the parallel simulator vs. its one-processor instance. Not surprisingly, the speedup curves are better for the larger problem sizes, and the relative speedups are much better than the effective speedups. For the 4096-node instance, the largest instance recommended for the serial router, the Paragon with 16 processors can improve the execution time by a factor of 6.4, a significant improvement since the serial version takes almost 8 hours to complete.The 16-processor Paragon is a good choice for 256 or more nodes.

The relative speedup is more impressive than the effective speedup, due to the fact that the one-processor ZPL program is around 2.4 times slower than the serial version. As the problem size increases, the speedup curves come closer to the ideal speedup; in the best case, we achieve a speedup of 1.89 in the 1024-processor instance when we go from one to two processors. As we "optimized" the ZPL program, the relative speedups tended to remain constant, while the effective speedups improved, implying that if we make additional optimizations in the ZPL code we can expect to see corresponding improvements in the effective speedup.

Thus far, we have fixed the load to be 80%; what effect does changing the load have on the execution times? We have run all of the different instances discussed above with various loads, ranging from 20% to 80%. For all instances, the execution times increase as the load increases, but the speedups also increase with increasing load, probably due to better balance among the processors. As an example, Figure 5 shows a plot of the effective speedup of the 256-node router with various loads. This plot is typical. The effective speedups are similar for each load, but they do increase with increased load. We had actually expected more of an impact from processor load, with the 20% load showing much poorer performance.

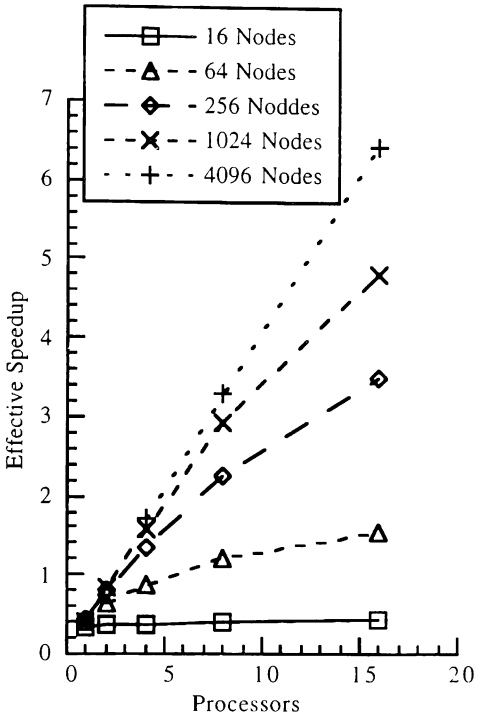

Figure 4: Effective Speedup of the Chaos Router Simulation on the Intel Paragon

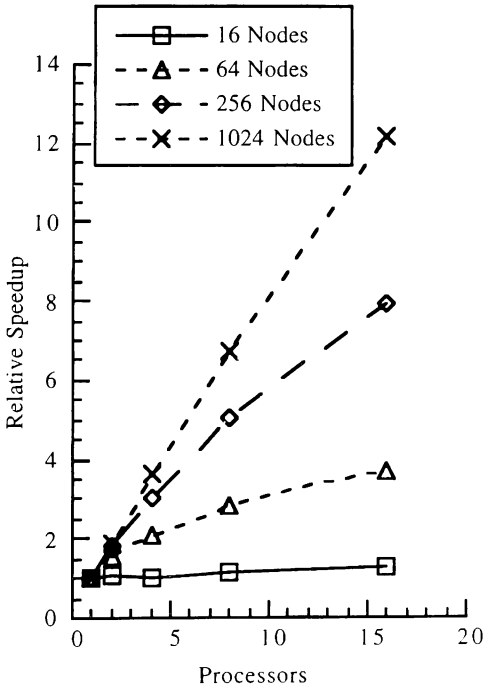

Figure 5: Relative Speedup of the Chaos Router Simulation on the Intel Paragon

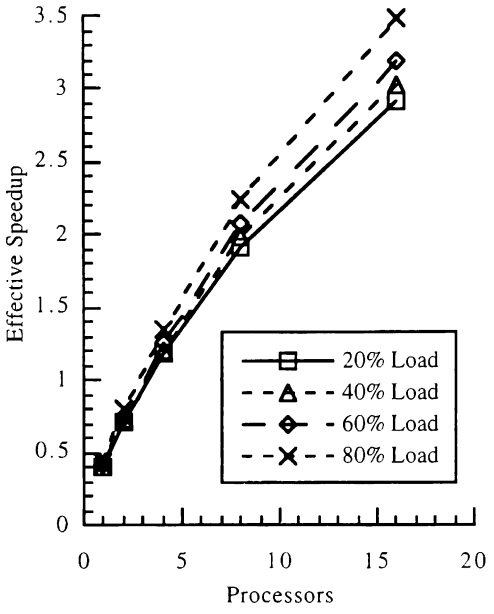

Figure 6: The Impact of Router Load on Effective Speedup Using 256 Nodes

4.2 The Alpha Cluster

The Digital Alpha cluster used in these experiments is a subset of the faculty workstation network of the Computer Science Department at the University of Arizona. The network is connected via an Ethernet and was not isolated from outside traffic during the runs, since we were not allowed to disturb the normal operation of the departmental system. However, we did make every effort to use idle workstations and run the experiments during "off hours." Due to the number and availability of workstations, we limited the experiments to eight workstations. All were identically configured with 64MB of memory. Below are the parameters used for these experiments.

Load:	20%, 40%, 60%, 80%
Processors:	1, 2, 4, 8
Nodes:	16, 64, 256, 1024, 4096, 16384

Not all possible permutations of these parameters were run; the largest-sized experiments (16384 nodes) were measured only on four and eight processors due to the length of time taken for the simulation.

The execution times for the larger experiments (256 nodes and greater) using 80% loads are shown in Figure 6. We don't show the 16- and 64-node instances here; they exhibited fairly large slowdowns due to the small number of nodes per processor (Richardson 1995). Note that we only gain performance improvements over the serial simulator in the largest two instances (4096 and 16384 nodes). This is likely due to several factors. First, the network is connected via an Ethernet bus, which only allows one message to be transmitted at a time. Thus all messages are serialized here, while in the Paragon, multiple messages can be transmitted simultaneously. Second, ZPL uses PVM to interface to the network. This means that it uses the TCP/IP protocol to send messages. While the TCP/IP protocol is robust, it incurs a nontrivial software overhead cost that must be borne by the parallel program. Finally, we cannot be completely sure that there was no other network traffic competing for the limited Ethernet bandwidth, although we believe that this effect was minimal.

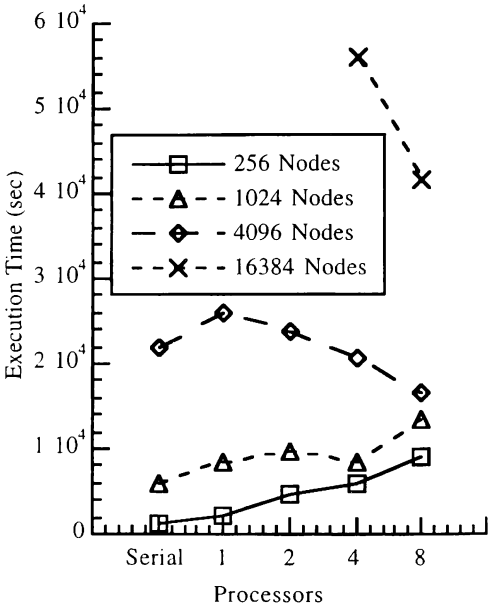

Figure 7: Execution Times on the Digital Alpha Cluster

The effective and relative speedups are shown in Figures 7 and 8, respectively. We do not include the 16384-node instances because we have no serial or one-processor baseline, since the problem size is too large for a uniprocessor. It is clear from the effective speedup curve that only the 4096-node instance has any speedup, and that the speedup here is minimal. Perhaps more interesting is that fact that this is true for the relative speedup too. It is not the slowdown of the parallel implementation that appears to be hurting performance, but rather the computation/communication ratio. For good performance on an Ethernet with this high-performance processor, we need lots of computation per processor.

The effect of nodes on speedup was not as clear for the Alpha as for the Paragon. The general trend still held – that an increased load tended to increase the effective parallelism. However, there were instances where this trend did not hold. For example, the effective speedup decreased from 0.71 to 0.43 when going from a load of

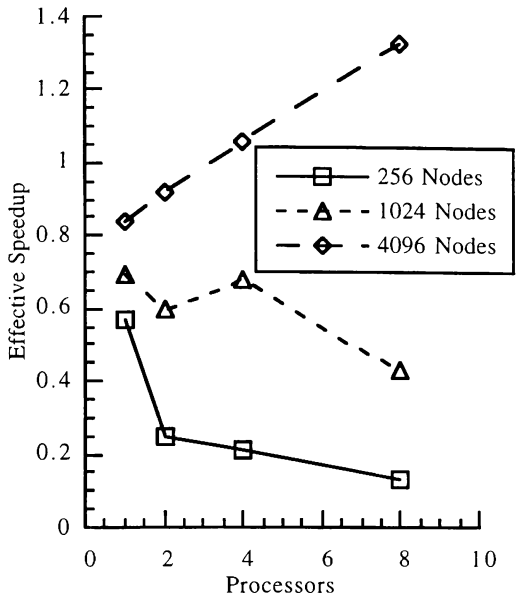

Figure 8: The Effective Speedup of the Chaos Router Simulation on the Digital Alpha

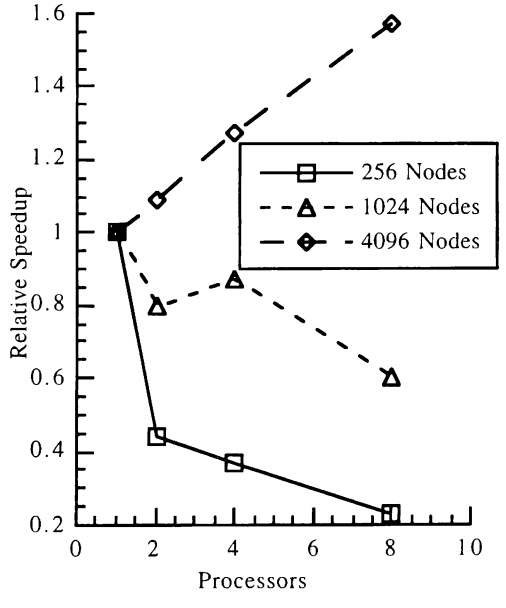

Figure 9: The Relative Speedup of the Chaos Router Simulation on the Digital Alpha

60% to 80% using 1024 nodes on eight processors. It is not clear whether these experiments were anomalous; we have not yet had enough access time to replicate the experiments sufficiently to insure that the measured differences are due to program effects rather than to interference from other Ethernet traffic. For the experiments where we achieved positive speedup, node sizes of 4096 and above, we saw none of the anomalies.

4.3 Comparing the Performance Results

The Alpha and Paragon provided very different architectural platforms on which to test the effectiveness of the parallel chaos router. The processor on the Paragon, the i860, is much slower than the Alpha processor, while the Paragon's interconnection network is much faster. Moreover, the Paragon's interconnection network essentially "matches" the communication patterns of the router, while the Ethernet serializes all communication.

These differences manifested themselves in the performance of the parallel simulation. On the Paragon, thrashing occurred more quickly than we had expected from the serial results. For example, we should have been able to run the 2048-node parallel simulator on a single processor without undue thrashing. One effect that we had failed to consider originally is that the parallel simulator uses substantially more memory than the serial version, because in the serial version memory could be dynamically allocated and pointers were often used to pass information. These were not possible in the parallel simulator because pointers cannot be used when the data are spread among multiple processors. Because each Alpha processor has twice as much memory as a Paragon processor, thrashing was not as evident on the Alpha cluster. Moreover, each Alpha has its own disk, while the Paragon processors all share a single disk array, which further exacerbates thrashing on the Paragon.

Overall, the two architectures complemented each other quite nicely. For small- to medium-sized problems, the Paragon had the best performance. This is likely due to its communication network. The mesh interconnection network maps well to this problem, and its speed is also faster than the Ethernet. This makes interprocessor communication much more efficient. However, for large-sized problems, the Paragon ran into memory limitations. In these cases, the Alpha cluster performed better. In fact, the Alpha cluster can run larger-sized router nodes than can run on a single Alpha. Each Alpha processor has more memory, and the computation/communication ratio is more favorable for the Ethernet for large router sizes. Thus the user can choose the best platform for the router simulation based on the router size, using the Paragon for medium-sized problems and the Alpha cluster for the larger problems.

5 CONCLUSIONS

In this paper, we discussed our experiences in using ZPL for a parallel chaos router simulator. ZPL was easy to use, although we did have to understand some aspects of the compiler in order to obtain more efficient performance. The lack of dynamic memory, especially for use with a single node, proved somewhat difficult, but we

were able to use static memory with little difficulty. The ability to write C routines to be called from ZPL proved invaluable for gathering statistics about the performance of the chaos router. Moreover, ZPL's portability allowed us to test the parallel simulator on different architectures without rewriting the simulator.

While we achieved our original goal of a faster parallel simulator, the parallel simulation running on a single processor is still slower than the optimized serial simulator by a factor of approximately 2.4. We expect some improvement in this factor as the ZPL compiler matures, but the one-processor parallel implementation will remain slower than the serial one. This is because the parallel program has inherent overheads (as compared to C) due to the fact that pointers cannot be used for data that can be partitioned across multiple processors in non-shared memory architectures.

The two architectures tested here complemented each other nicely. The Paragon performed best for medium- to large-sized routers, likely due to the lower computation/communication ratio (it has a slower processor and a faster interconnection structure than the workstation cluster), and the fact that the interconnection structure mapped well to this problem. However, it was susceptible to thrashing for large routers. The workstation cluster appears to be limited by the Ethernet and PVM for medium-sized routers, but it could simulate really large routers that are not viable for either the Paragon or a single Alpha. With a faster interconnection such as an ATM network, we expect the performance of the workstation cluster to improve for medium-sized problems.

REFERENCES

Bolding, K., M. Fulgham, and L. Snyder. 1994. The Case for Chaotic Adaptive Routing. Technical Report UW-CSE-94-02-04, University of Washington.

Bolding, K., and L. Snyder. 1992. Mesh and Torus Chaotic Routing. In *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conferences*, 222–247.

Forman, G., and C. Lin. 1994. ZPL Tutorial. Technical Report, Computer Science and Engineering Department, University of Washington.

Konstantinidou, S., and L. Snyder. 1994. The Chaos Router. *IEEE Trans. on Computers* 43(12):1386–1397.

Lin. C. 1994. ZPL Language Reference Manual. Technical Report UW-CSE-94-10-06, Univ. of Washington.

Lin, C., and L. Snyder. Data Ensembles in Orca C. 1992. *In 5th Workshop on Languages and Compilers for Parallel Computing*, New Haven, CT.

Lin, C., and L. Snyder.1993. ZPL: An Array Sublanguage. In *Languages and Compilers for Parallel Computing*, U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, 96–114. Springer-Verlag.

Snyder. L. 1994. A ZPL Programming Guide. Technical Report UW-CSE-94-12-02, University of Washington.

Richardson. G. W. 1995. *Evaluation of a Parallel Chaos Router Simulator*, Master's Thesis, Dept. Of Electrical and Computer Engineering, University of Arizona.

AUTHOR BIBLIOGRAPHIES

WILKEY RICHARDSON is a Member of the Technical Staff at Hughes Missile Systems Company working on real time simulation. He has previously been an engineer at Kaman Corp., Raytheon Co., and Sperry Corp. He received a BS in Computer Science and Mathematics from the University of Kentucky in 1982, and an MS in Electrical and Computer Engineering from the University of Arizona in 1995.

MARY L. BAILEY is a Senior Member of the Technical Staff at Rincon Research Corporation. She was previously an Assistant Professor in the Computer Science Department at the University of Arizona. She received a B.A. in Mathematics and Physics from Vanderbilt University, and an M.A. in Mathematics and M.S. and Ph.D. in Computer Science and Engineering from the University of Washington. She was Program Chair of PADS'95 and General Chair of PADS'96. She also served on the executive committee of ICCAD (1993-1996). Her research interests include parallel and distributed simulation, logic simulation, and special-purpose architectures.

WILLIAM H. SANDERS is an Associate Professor of Electrical and Computer Engineering and Research Associate Professor at the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. His research interests include methods for performance, dependability, and performability evaluation, computer networks and protocols, and fault-tolerant computing. He is the developer of two tools for assessing the performability of systems represented as stochastic activity networks: METASAN and UltraSAN. UltraSAN has been distributed widely to industry and academia, and is currently being used at more than 100 universities, five companies, and NASA. His research awards include the Digital Equipment Corporation Incentives for Excellence Faculty Award 1989-1991. He is a member of the IFIP Working Group 10.4 on Dependable Computing and is also a member of Sigma Xi and Eta Kappa Nu academic honor societies.