High-Performance Computer Architecture and Algorithm Simulator

KENNETH E. HOGANSON
Kennesaw State University

This simulation tool allows the user to explore different computer architectures with hardware support at any or all of five levels of parallelism, from intrainstruction (pipeline) through distributed *n*-tier client/server systems. The tool supports the simulation of various user-configurable architectures and interconnection networks, running a user-configurable and variable workload. This allows the student and the instructor to observe how performance changes through the five levels of parallelism with changes in either the architecture or workload. The successful use of the simulation tool in a variety of undergraduate courses at the author's institution is presented, along with examples, and a set of experiments. The simulator is a Java applet, which can be used from a Web browser, allowing anyone with an Internet connection access to the tool, without concern about student licensing requirements. The simulator is hosted at the author's institution with funding provided by a recent grant. Its design as an applet also allows improvements and enhancements to the software to be implemented and instantly made available to all users of the product.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization - General**]: Modeling of Computer Architecture
General Terms: Design
Additional Key Words and Phrases: Computer architecture simulation, education, parallel processing, parallel speedup, unified parallel model

1. INTRODUCTION

The dramatically increasing performance of our personal computers is associated with an increase in hardware and system complexity that has impacted both undergraduate and graduate computer science education. Formerly, high-performance and advanced computer architecture techniques and technologies have migrated to our desktop computers, and consequently, into computer architecture courses in order to provide students with a solid understanding of the modern computer system. Students are now expected to learn more content at a faster rate, and it can be argued that the complexity of the content is increasing as well. In particular, recent work in high-performance and parallel computer systems have led to computer architectures that accommodate five distinct levels of parallelism. The importance of the interplay between these different levels, and the tradeoffs in efficiency and performance improvement between allocating hardware at different levels has recently been demonstrated [Hoganson 2000a; 1999]. This area is complex and not yet fully incorporated into textbooks, but responds well to exploration through simulation.

This research was supported by a grant from the Mentor-Protégé program at the College of Science and Mathematics at Kennesaw State University.
Authors' address: Department of Computer Science and Information Systems, Kennesaw State University, 1000 Chastain Road, Kennesaw, GA 30120

A simulator and lab experiment tool has been developed that can enhance the teaching of a number of computer architecture-related courses, including architecture, parallel systems and algorithms, distributed client/server systems, embedded systems, and operating systems. This simulation allows various user-configurable architectures and interconnection networks to be simulated, running a user-configurable and variable workload, which allows the student and the instructor to observe how performance changes through the different levels of parallelism with changes in either the architecture or workload. The simulator is a Java applet, which can be used from a Web browser, allowing anyone with an Internet connection access to the tool without concern about student licensing requirements. The simulator is online and is hosted at the author's institution with support funding provided by a recent grant. Its design as an applet also allows improvements and enhancements to the software to be implemented and instantly made available to all users of the product.

Prior to the creation of this simulator tool, the way to explore high-performance and parallel computer architectures was through study and analysis of existing architectures and their limitations and reasoning about the capabilities of architectures based on their theoretical performance from mathematical models, or the time-consuming development of an architecture-specific simulation. Using this tool, a computer architecture can be considered, set-up in the simulator, and tested and compared with alternatives in just minutes, making it appropriate for inclusion in undergraduate lectures, and very useful for homework and experiments.

The High-Performance Parallel Computer Architecture simulator is a tool that provides students and researchers with a lens into the previously unknown continuum of possibilities, which were defined and made tractable by the recent unified parallel processing speedup model [Hoganson 2000a; 2001]. For the first time, students have a way to explore the complex interaction of performance-enhancing computer components, including pipelines, cache, multiple processors, and clusters of processors, which previously had to be taught as separate and disjoint topics. The tool allows the student or researcher to explore alternative ways to allocate computer hardware in a system, i.e., should additional computing hardware be allocated to add more pipelines, more stages per pipeline, more processors in an architecture, or even more clusters of processors.

Many other computer architecture simulators have been created, with varying scope. Some simulate specific computers, or CPUs, or instruction sets, like those simulating historic or hypothetical machines. Others simulate specific functions of computer systems, like caching and memory systems, and can be used to explore performance effects and limitations. Still others simulate specific parallel processing levels and hardware implementations, like shared-memory multiprocessors, or pipelined processors, and can be used to explore the performance of a narrow range of architectures at one level of parallelism. The new simulation tool presented here is unique in modeling all five parallel levels in a single simulation system, which allows the user to investigate the interplay and tradeoffs among the five levels of parallelism. It is also the only system to specifically support *n*-tier client server architectures and allow their investigation as distributed parallel systems.

The simulator has been used successfully at the author's institution in a number of senior-level courses, including one on parallel processing architectures and algorithms. In that class, four 1.25 hour in-class periods were spent illustrating parallel speedup models and limitations with the simulator, and was well received by the students, as they

were able to immediately view the resulting changes in parallel speedup as the architecture, algorithm, and interconnect were varied. Homework exercises were assigned to illustrate important ideas about parallel speedup and high-performance computing through the use of the simulation that previously could only be discussed through conjecture and reasoning about various mathematical equations. Student course evaluation feedback pointed to the use of the simulator as one of the highlights of the course. The simulator was also used successfully in graduate and undergraduate courses on client-server architectures, which are distributed parallel systems, and used in an undergraduate course on computer organization and architecture.

Section 2 explains recent theoretical developments in the understanding of parallel mechanisms, which has magnified the need for a teaching and experimentation tool. Section 3 describes the simulation tool in detail, discusses modeling capabilities and the user interface, and illustrates the use of the tool with a simple experiment. Section 4 contains five problems or issues to explore in parallel processing theory and architectures that can be investigated with this simulation tool. Each problem has been assigned as a graded exercise for undergraduate computer science students at this author's institution. An experimental design and resulting data that explore the issue are included with each problem statement, along with a concluding paragraph that discusses the meaning and implication of the experimental results. These five problems/experiments demonstrate that, with this simulation tool, quite sophisticated issues can easily be explored by undergraduate students. Section 5 discusses the use of the simulation modeling system at the author's institution in a variety of computer science courses, the student acceptance of the system, and course topic areas where the tool can be of significant use. Section 6 summarizes conclusions and discusses planned enhancements to the simulation tool.

2. RECENT UNIFIED THEORETICAL PARALLEL PROCESSING MODELS

Theories of parallel processing recently underwent a renewal with the publication in 1988 of experimental results that seemed to indicate that it was possible to obtain greater parallel speedups than expected under current theory [Gustafson 1988]. Obtaining greater parallel speedups than expected (previously based on Amdahl's law [Amdahl 1967]) implied that computer architectures can be designed to efficiently utilize more processing elements than previously thought, resulting in much more powerful and scalable computer systems. Gustafson's paper generated considerable excitement (including publicity in the lay press), and attracted other researchers to the field, resulting in a flurry of papers on parallel speedup modeling [Carmona 91; Eager 1989; Flynn 1966; Karp and Flint 1990; Mabbs and Forward 1994; Mohaptra et al. 1994; Sun and Gustafson 1991; Van-Catledge 1989; Wang et al. 1995; Wood and Hill 1995], which took disparate approaches and created a pedagogical problem, in that a large commitment of time (and textbook pages) was required to cover the subject. More recently, work toward consolidating the various approaches into a unified analytical parallel processing model that incorporates Gustafson' speedup models, as well as others, has resulted in a single flexible analytical model, which illustrates that the different modeling approaches are simply different aspects of a single, more robust, model [Hoganson 2001; 2000a; 1999]. Two new levels of algorithm/software parallelism were identified, along with hardware mechanisms to realize potential performance improvements, resulting in a taxonomy of five clear and distinct levels of software/algorithm parallelism with supporting architectural structures [Hoganson 2000a].

Table I. Algorithm and Machine Parallelism

Software Parallelism	Enabling Architecture Element
1. Intrainstruction	Multistage pipeline
2. Interinstruction	Superscalar architecture (multiple pipelines)
3. Algorithm	Multiple processor architectures
4. Multiprogram	Clustered computer systems
5. Client/server applications	Distributed computing systems (including *n*-tier systems)

Each software level of parallelism is supported by an equivalent architectural level of parallel hardware which suffers from diminishing returns that limits scaling, and other constraints that bound their individual performance and efficiency [Hoganson 2000a; 2000b]. These five levels of parallelism so identified are conventional parallel mechanism for conventional digital/electronic computer systems. The simulation tool includes these five levels, but does not attempt to incorporate novel parallel methods including quantum or DNA computing.

Every level of parallelism shares a number of common elements: the portion of work that can be done in parallel and the portion of work that cannot; the ratio of synchronizing communication (graininess) that degrades performance; and the various implementation issues that limit performance to less than the ideal linear speedup and/or bound the realizable speedup [Hoganson 2000a]. Adjacent levels of parallelism often share similar characteristics, with algorithm examples and architectural mechanisms that could be argued for inclusion in more than one level. In a sense, there exists a single continuum of parallelism where algorithm work is divided for concurrent execution in different and nested groups. The discrete levels of parallelism that are recognized are imposed upon the continuum by the software mechanisms and architectural constructs that identify and accommodate the parallelism.

The High-Performance Computing Simulation tool currently supports five conventional levels of parallel processing. The architectural elements that realize the speedup potential of these five levels are integral parts of modern computing systems that should be understood by all computer science majors.

The above taxonomy of parallel speedup techniques does not include nonconventional and nonelectronic parallel mechanisms, including quantum computing and DNA computing, both of which seem to fall into level-3 algorithm parallelism [Hoganson 2000a].

The recent developments in parallel processing theory have multiplied the need for a simulation tool that is appropriate for students and researchers, class demonstrations, and student exploration of the broadened possibilities. Undergraduate computer architecture courses often do not have adequate time to cover all these levels of parallelism in sufficient detail, hence the motivation for the development of the simulation tool, intended to both increase the depth of coverage and the depth of understanding of interactions, tradeoffs, and limitations.

3. DESCRIPTION OF THE SIMULATION TOOL

This tool is a high-level architecture simulator, which allows control over the computer system and the workload to be run, simulation iterations and simulation control, and artifacts of the interactions between the architecture and workload. The tool is not a low-level design tool for specifying hardware systems in detail.

3.1 Architectural Modeling

The architecture can be configured in the following ways:

- number of stages per pipeline;
- number of pipelines (1 – 20);
- number of processors;
- number of processor clusters and the number of processors in each cluster;
- number of tiers in a distributed or *N*-tiered system (1 to 10);
- number of machines in each tier and configuration of the machines (as above);
- relative disparity between CPU operations and interconnection latency;
- interconnection network probability of acceptance of requests, a random variable, uniformly distributed between a specified maximum and minimum. The maximum and minimum can be set at the same values.

3.2 Workload Modeling

The workload to be run on the system can be configured in the following ways:

- percentage of the workload that can be distributed across *N* pipelines, where *N* varies between 1 and 20;
- granularity of the application, i.e., the ratio of computation to communication. The number of communications generated by a variable number of CPU operations can be specified. Each communication exacts a latency penalty for the disparity in interconnection network performance over the latency in CPU operations (which can be specified as an architecture configuration variable);
- workload distribution balance across a distributed or *n*-tiered system;
- parallel and serial fraction scaling factors;
- workload scaling factor.

3.3. Workload-Architecture Interaction Modeling

Artifacts of the interaction between the architecture and characteristics of the workload can be configured in the following ways:

- frequency of pipeline flushes, a random variable that is uniformly distributed between a selected maximum and minimum. The maximum and minimum can be set at the same value;
- frequency of pipeline stalls, a random variable that is uniformly distributed between a selected maximum and minimum. The maximum and minimum can be set at the same value.

3.4 Simulation Interface

The simulation tool interface is organized in three panel types: simulation control and universal variables; machine and client workstation configuration; and distributed system and server tier configuration. There is only one of each panel of the first two panel types, but there are up to nine identical panels for each of up to nine levels of servers in a ten-tier system. The first panel, Figure 1 (also the default startup panel), allows control of the number of simulation iterations (each of which run for a user selected number of CPU operations); displays speedup results; and allows the number of tiers to be specified for distributed and client/server systems and the workload balance across the tiers.

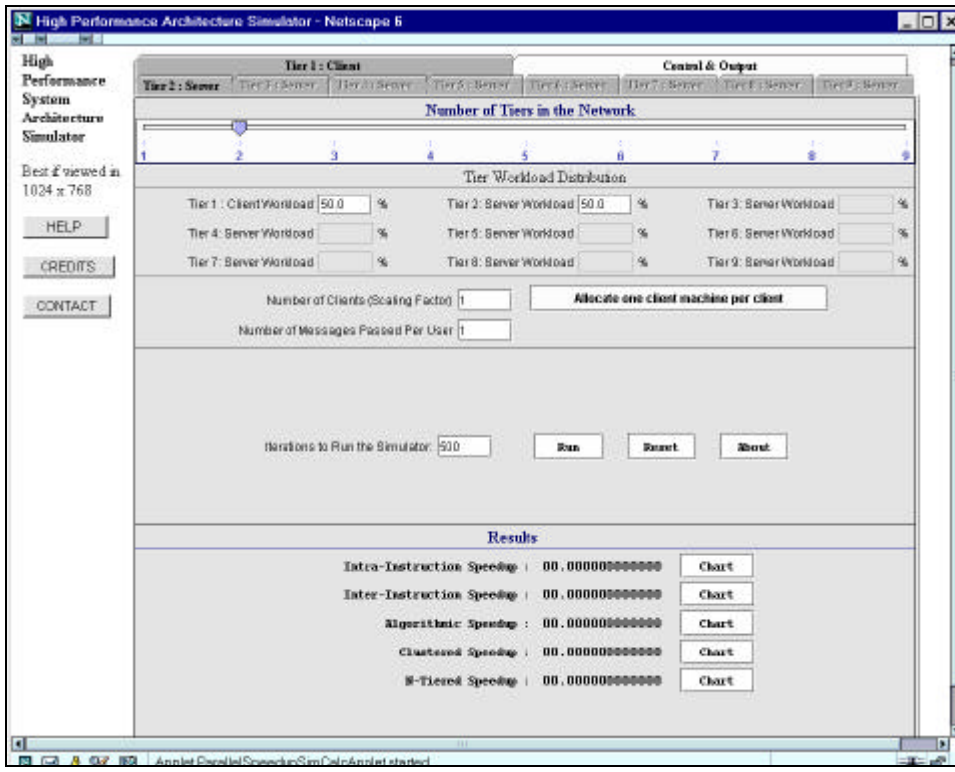

Fig. 1. Simulation control and results panel**.**

Performance results are reported as speedup, which allows values to be abstracted from hardware implementation details that affect performance and make direct comparisons difficult (processor and bus speeds and timings, manufacturing factors, etc.). Bus, memory, and interconnection network performance are measured as multiples of the average instruction processing latency. It is obvious that if the performance of all components of a computer system can be doubled without changing the system configuration (the "if" includes scaling limitations), then the system will exhibit doubled performance, so the model does not focus on timing issues other than contention on the interconnection networks. This allows the user and student to focus clearly on the performance effects of varying the system architecture.

The second panel (Fig. 2) allows specification of a single machine that can range from a single processor to a clustered multiprocessor. If the system being modeled is a client/server system, then this panel configures the client machines (all identical).

The third panel type allows the user to specify and configure distributed and *n*-tier systems by configuring the characteristics of the server machine (each of which could be a clustered multiprocessor). Functions to describe the characteristics of the interconnect at each tier can be used to approximate known behavior that varies with the load on the network.

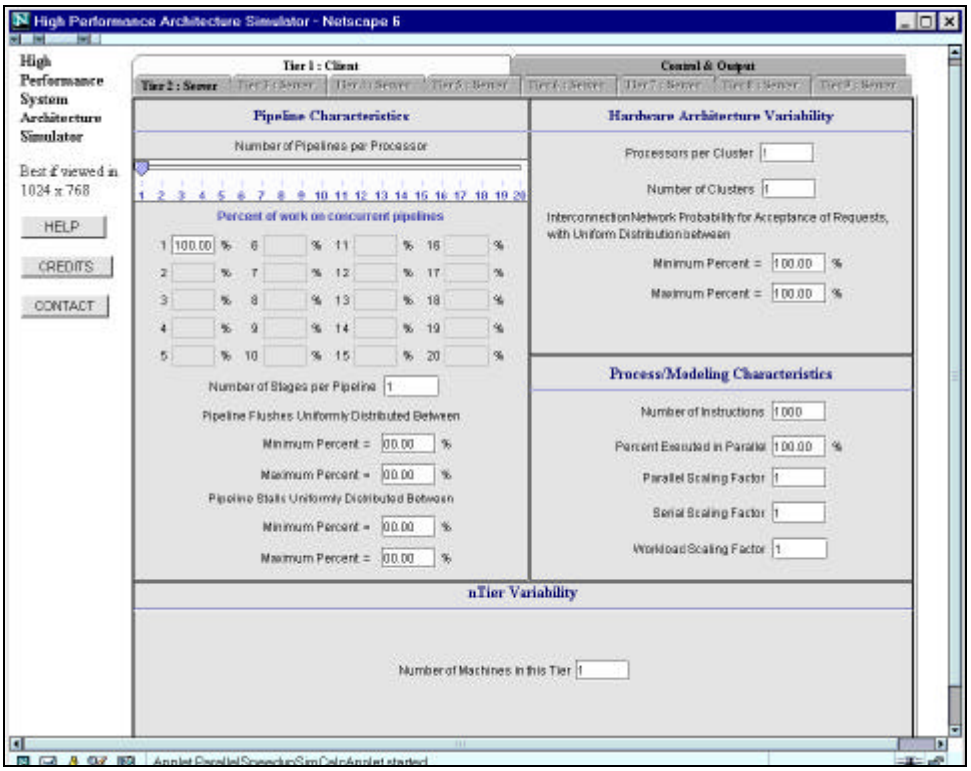

Fig. 2. Machine/client configuration panel.

Simple Example. Parallel speedup from four processors.
This simple example illustrates how easy the system is to use and achieve interesting results by executing a workload on a 4-processor machine. The simulation is run twice to compare the effect of the interconnection network on realized speedup.

Part 1

Step 1. On the tier 1:client panel, set the *processors per cluster* at 4, and the *percent executed in parallel* at 90% (90% of the work can be done in parallel on the four processors, while 10% must be done in serial on a single processor).

Step 2. Then run the workload using the control & output panel, leaving other values at their startup defaults.

Results. The resulting speedup of 3.077 is in agreement with the speedup formula results expected for 1000 operations with 10% on a single processor and 90% parallel on four processors:

Part 2. To account for contention within the interconnection network, we can utilize the probability of acceptance of the interconnection network.

Step 5. On the tier 1: client panel, set the *minimum probability of acceptance of requests* at 80%, and leave the maximum at the default of 100%. A random variable

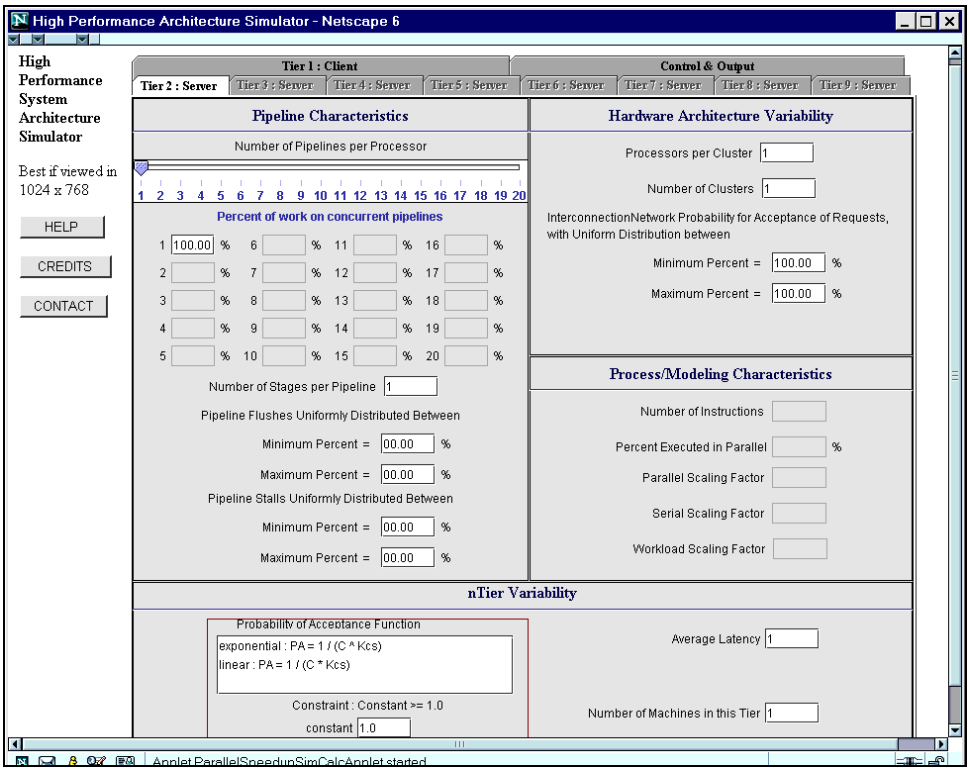

Fig. 3. *N*-tier server configuration panel.

is now interjected into the simulation runs, which varies uniformly between 80% and 100%.

Result 2. Individual simulation runs will vary, but the speedup realized will converge at around 2.85 as the number of simulation runs is increased.

4. ILLUSTRATIVE EXPERIMENTS

These hands-on assignments ask students to design their own experiments to explore an aspect of parallel speedup. Not only do these exercises reinforce the principles governing parallel processing and parallel speedup under consideration, but they also teach scientific reasoning and experiment design. The following five experiments are organized as a problem statement to be assigned to students, followed by an example approach to a set of experimental runs that answer the problem statement, and a concluding paragraph that discusses the simulation results and their meaning and further implications.

Experiment 1. Demonstrates diminishing returns of performance enhancement as the number of processors in a system increase. *Appropriate for a course on parallel computer architectures and/or parallel programming.*

Objective*:* Conduct a set of simulation runs to demonstrate the "discouraging" obser-vation of Amdahl's speedup law that dictates that increases in performance due to adding additional processors diminish with each successive processor.

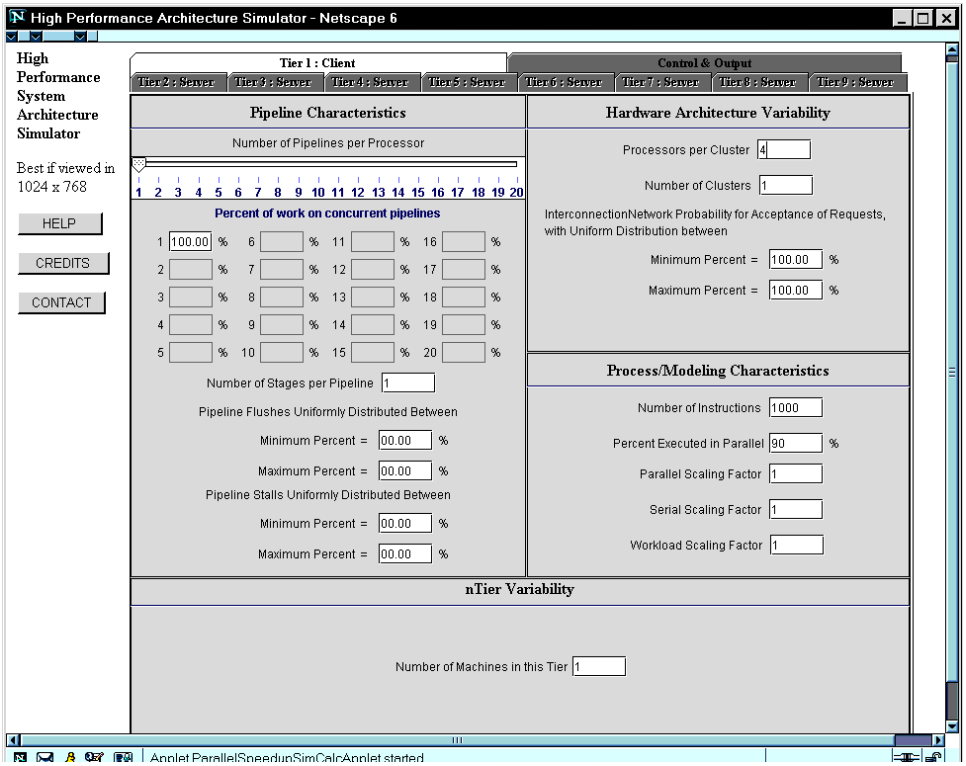

Fig. 4. Simple example setup.

One Example Solution:

- conduct runs with linearly increasing processors, starting at four processors, increasing by four additional processors;
- record the speedup at each point;
- calculate the efficiency (speedup over # of processors) at each point;
- calculate the change in speedup;
- constant: all variables other than the number of processors; the fraction of the program that can run in parallel is 95%.

Conclusion

This experiment illustrates Amdahl's finding of diminishing returns by observing that the additional speedup steadily declines by the addition of four processors, as does the efficiency of each additional processor in adding speedup. This suggests that computer architectures with many processors will be inefficient, a conclusion effectively disputed by more recent research by Gustafson and Hoganson.

Experiment 2. Demonstrates the effect of scaling the workload at the process level (Gustafson's process scaling).
Conduct a set of simulation runs to illustrate that the speedup of a process increases as the process is scaled beyond what would be expected by Amdahl's law. *Appropriate for a course on parallel computer architectures and/or parallel programming.*

One Example Solution:

- conduct a set of simulation runs with a fixed process and increasing numbers of processors;
- collect a baseline set of values for a process without scaling;
- collect a set of data points with different scaling factors;
- constant: all variables other than the number of processors; the fraction of the program that can run in parallel is 95%.

This experiment demonstrates that very high speedups are attainable for processes where only a very small fraction of work cannot be done in parallel. High efficiencies can also be obtained when the process can be scaled by very large factors. A small but identifiable subset of computing algorithms can be effectively scaled. [Notice that it is assumed that the amount of memory and other resources also scale with the number of processors, as well as the degree of parallelism.]

Experiment 3. Using "the grid" (an advanced experiment). *Appropriate for a course on parallel computer architectures and/or parallel programming.*
The grid is a system that allows a researcher to utilize unused compute cycles on high-performance machines that are linked over the Internet. It attempts to make supercomputing cycles available as if they were a public service utility, hence "the grid." An application will be dynamically distributed and migrated across the net taking advantage of available resources. Results, of course, will be returned to the originating workstation. Because of the time and work required to migrate applications across the grid, it is appropriate only for very highly parallel applications with very high computation vs communication (grains).

"The grid" can be simulated using the *n*-tier client/server modeling structure, where each machine in a tier is a highly-parallel system. Try the following example. (Note that the degree of multiprocessing is always assumed to be large enough to utilize all processing resources allocated.) Use default values for all unspecified parameters.

- Tier 1: the user's workstation – 0.001% of the work occurs at this machine (user input and results collating and display).
- Tier 2: a highly parallel supercomputer, represented by a system with 20 pipelines each with 1000 stages (20,000 processing elements). 40% of the overall workload is executed on this machine. This machine's workload is evenly distributed across the 20 pipelines (100% on 20 pipelines – default value).
- Tier 3: a cluster of parallel machines, consisting of 4 multiprocessors clustered together, each with 64 processors (256 processors total). Each processor consists of 8 pipelines with 16 stages per pipeline (32,768 processing elements total). 59.9% of the workload is executed on this machine. This machine's workload is evenly distributed across the clusters, processors, and 8 pipelines per processor (100% on 8 pipelines – default value).

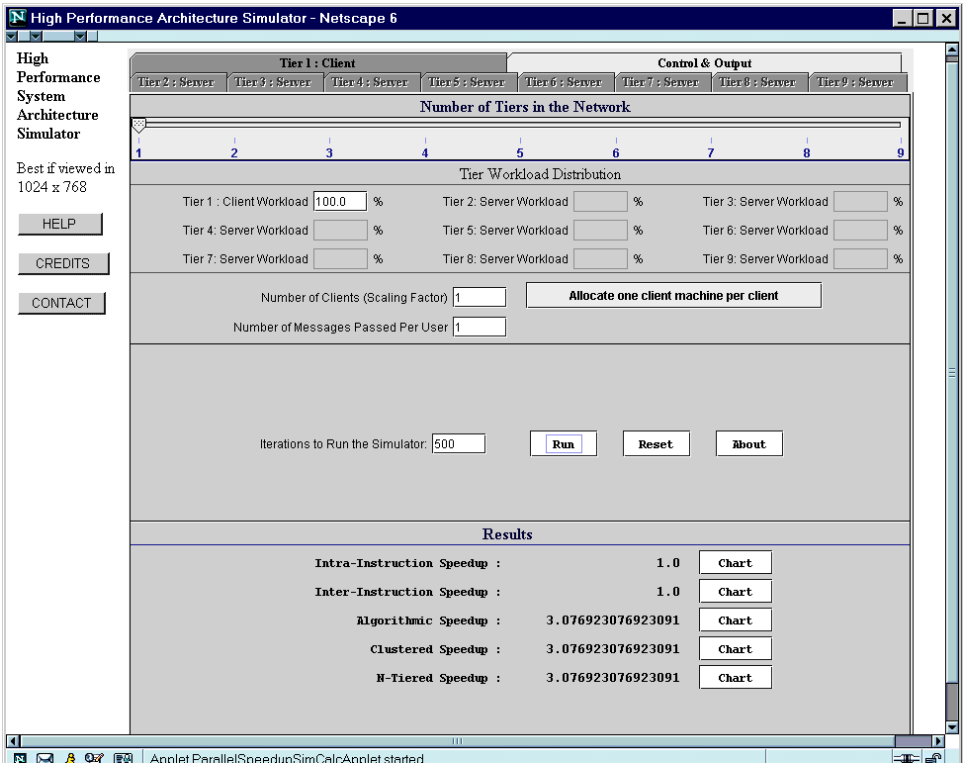

Fig. 5. Simple example simulation run and results.

Table II. Five Student Experiments

Experiments	Topics
Experiment 1	Classic scaling problem: Demonstrates diminishing returns of performance as the number of processors in a system increases.
Experiment 2	Process scaling: Demonstrates the effect of scaling the workload at the process level.
Experiment 3	Using "the grid."
Experiment 4	Pipeline performance.
Experiment 5	*N*-tier client server performance and interconnection latency.

Table III. Speedup and Efficiency with Scaling Processors

Measuring speedup with increasing processors 1000 operations, 500 runs							
Number of processors	4	8	12	16	20	24	1000
Speedup	3.478	5.926	7.742	9.143	10.256	11.163	19.627
Efficiency	0.8695	0.741	0.645	0.571	0.513	0.465	0.020
Added speedup		2.448	1.816	1.401	1.113	0.907	8.464

Table IV. Process Scaling Speedup

Measuring speedup with increasing processors. 1000 operations, 500 runs							
Number of processors	4	8	12	16	20	24	1000
Speedup	3.478	5.926	7.742	9.143	10.256	11.163	19.627
Efficiency	0.8695	0.741	0.645	0.571	0.513	0.465	0.020
Added speedup		2.448	1.816	1.401	1.113	0.907	-
Process Scaling Speedups							
Parallel factor: 10	3.938	7.717	11.347	14.835	18.190	21.421	160.504
100	3.994	7.971	11.931	15.875	19.802	23.713	655.517
1000	3.999	7.997	11.993	15.987	19.980	23.971	950.050

1. run the simulation with 1,000,000,000 instructions and a communication latency of 1, with a single communication event per machine and a parallel fraction of 99.999%;
2. run the simulation again, this time with a communication latency of 1000 at both Tier 2 and Tier 3;
3. run the simulation again, with a communication latency of 1000 and 500 communication events;

- record and explain your observations;
- repeat the experiments with a parallel fraction of 95.0% - 5% on the workstation, 40% on tier 2, 55% on tier 3; record and explain your observations.

Experiment 3: ***Results***

Table V shows that very large speedups are possible for an application with a very small serial fraction (0.001%) and with very small communication requirements relative to computation. It also shows the extreme sensitivity of these large-scale applications to the serial fraction (at 5%), and the "graininess" with a larger communication frequency and penalty.

In each case, with the number of parallel processing elements exceeding 50,000, the parallel speedup is limited by the theoretical maximum values of 100,000 and 20 for the case with 99.999% parallel and 95% parallel, respectively. The realized speedup is reduced by the pipeline load time on the super-machines, even with no stalls or control hazards.

Experiment 4. Pipeline performance. *Appropriate for a first course on computer archi-tecture.*

The object of this experiment is to explore the performance of a processor that incorporates an idealized pipeline against one without a pipeline, and against a pipelined processor with real-world flushes and stalls. Compare systems with pipelines of 4, 8, and 16 stages against the performance of a processor without a pipeline. Try runs with 0% stalls and 0% flushes, 10% stalls and 0% flushes, and 0%stalls and 10% flushes. Does the system conform to the theoretical model of pipeline speedup?

One Example Solution:

- conduct runs with a single processor with 4, 8, and 16 stages in a single pipeline, and a nonpipelined processor at 0% stalls and 0%flushes, 10% stalls and 0% flushes, and 0%stalls and 10% flushes, recording the speedup at each point;

Table V. Performance of "The Grid"

"The Grid"			
Parallel Fraction	**COMMUNICATION PARAMETERS**		
	Latency=1, Events=1	Latency=1000, Events=1	Latency=1000, Events=500
99.999%	20,692.46	19,112.13	488.204
95%	19.99	19.98	19.22

Table VI. Pipeline Speedup by Scaling Stages

Measuring Speedup with Increasing Stages. 1000 operations, 500 runs					
		Number of Stages			
		1	4	8	16
0% stalls, 0% flushes	Speedup	1	3.988	7.944	15.764
	Efficiency	1	99.7	99.3	98.5
10% stalls, 0% flushes	Speedup	1	3.626	7.227	14.350
	Efficiency	1	90.7	90.3	89.7
0% stalls, 10% flushes	Speedup	1	3.070	4.687	6.362
	Efficiency	1	76.8	58.2	39.8

Table VII. Comparison with Theoretical Pipeline Performance

Theoretical Expected Speedup with Increasing Stages. 1000 operations					
		Number of Stages			
		1	4	8	16
0% stalls, 0% flushes	Speedup	1	3.988	7.944	15.764
	Efficiency	1	99.7	99.3	98.5
10% stalls, 0% flushes	Speedup	1	3.626	7.227	14.350
	Efficiency	1	90.7	90.3	89.7
0% stalls, 10% flushes	Speedup	1	3.070	4.687	6.362
	Efficiency	1	76.8	58.2	39.8

- exactly 10% of stalls is selected by setting both the minimum and maximum fraction of stalls to 10% (no random variability in fraction of stalls). This sets 10% of pipeline operations as stalled for a single pipeline stage time;
- calculate the efficiency (speedup over # of stages) at each point;
- calculate the change in speedup;
- constant: all variables other than the number of processors.

The theoretically predicted values for pipeline performance in Table VII are derived using the analytical model of pipeline performance from Hoganson [2001], given in equation 1 – Pipeline speedup, and agree exactly.

S = Number of pipeline Stages
n = Number of instructions
s = Frequency of pipeline stalls
f = Probability that the an instruction causes a pipeline flush

$$Speedup = \frac{nS}{S - 1 + n + fn(S-1) + sn}$$

Conclusion

This experiment illustrates that the pipeline load time, which increases with the number of pipeline stages, impacts the resulting speedup and efficiency negatively, but by a small amount. Increasing the frequency of stalls from 0 to 10% detracts from speedup and efficiency. Increasing the frequency of pipeline flushes impacts speedup and efficiency even more, with dramatically decreasing performance with larger numbers of stages, due to the increasing miss penalty. [This suggests that unless the fraction of pipeline flushes can be forced close to zero, the scalability of pipelines, by increasing the number of stages, is limited.]

The simulation results agree exactly with the theoretically calculated values when using the simulation tool as a calculator; that is, when parameters like the frequency of pipeline flushes and pipeline stalls are not allowed to vary randomly. To use these parameters as uniform random variables, the maximum and minimum values would be set differently, and the parameter will then be a random variable uniformly distributed between the maximum and minimum.

Experiment 5. *N*-tier client server performance and interconnection latency. *Appropri-ate for a course on client/server systems or distributed systems.*

This experiment observes the sensitivity of *N*-tier client/server architectures to communication latency and frequency. Set up the following simulation values and run the simulator noting the speedup that results.

1. *Simulation Run* 1: Observe and record the speedup:
   - 3 tiers
   - 33% work on client
   - 33% work on each server tier
   - number of instructions at 1000
   - 10 machines at tier 1, one per client
   - 2 machines at tier 2
   - 2 machines at tier 3

2. *Simulation Run* 2: Increased frequency of communication and increased latency. Observe and record the speedup:
   - all settings the same as in run 1 except:
   - number of messages passed per client=10
   - communication latency at tier 2 =10
   - communication latency at tier 3 is 10

Table VIII. *N*-Tier Speedup and Interconnect Latency

***N*-Tier Client/Server Interconnection Network Sensitivity 1000 operations, 500 runs**					
Run	**Msg Frequency**	**Msg Latency**	**CPUs per Server**	**Servers per Tier**	**Speedup**
Run 1	1 per client	1 CPU-Op	1	2	2.700
Run 2	10 per client	10 CPU-Op	1	2	1.304
Run 3	1 per client	1 CPU-Op	4	5	14.150
Run 4	10 per client	10 CPU-Op	4	5	2.143

Table IX. Student Acceptance of the Simulation Tool

Course	**Number of Students**	**Average Evaluation**	**Positive Comments**	**Negative Comments**	**Number of Experiments**	**Term**
CSIS 3510 Org&Arch	32	4.26	7	0	3	Summer 2001
CSIS 4490 N-Tier Architectures	29	4.83	8	0	4	Summer 2001
CSIS 4310 Parallel Systems	8	4.50	7	0	4	Spring 2001

3. *Simulation Run* 3: Rerun the settings for run 1, but with 4 processors in each server instead of 1, and 5 servers at each tier instead of 2.
4. *Simulation Run* 4: Rerun the settings for run 1, but with 4 processors in each server instead of 1, and 5 servers at each tier instead of 2.
5. Explain the disparity in performance between the two runs.

***Example Solution*:** Run 1 provides a modest speedup, even thought there are 14 machines operating in parallel, because only 1/3rd of the work is distributed across the clients, while 2/3rd is allocated to tiers with only 2 machines operating in parallel at each tier.

Run 2 provides significantly smaller speedup due to the performance effect of communication latency. Each client communicates with the server 10 times, experiencing the latency effect 10 times per client, while the latency of the communication itself is 10 times that of Run 1. A greater number of communications between client and server occur (more fine-grained, and more tightly coupled).

Run 3 and 4 illustrate that the server tiers are a performance bottleneck, and better speedups will result with more computation power allocated to the servers. Run 4 shows that an application whose performance is communication-bound will see only modest performance improvements from increasing computation power.

Conclusion

The performance of both processors and networks will continue to improve as technology methods improve, but the performance disparity between the two will likely continue to be significant, which has implications for the design of *n*-tier client/servers and other distributed systems: the more coarse-grained the better (few communication events

between much processing). Some applications may not be appropriate for these types of architectures due to the pattern of communication between components.

Also, the performance of the servers can easily be a performance bottleneck it a significant portion of the processing occurs at the server side. The above systems perform significantly better with more computatioal power on the server side. The adage "buy the largest server you can afford" in designing client/server systems seems to be correct.

5. USING THE SIMULATION TOOL FOR UNDERGRADUATE COURSES

A prototype of the simulation tool has been used very successfully in an undergraduate course, CSIS 4130 Parallel Architectures and Algorithms, during the spring semester of this year. Student reaction to the tool was universally very positive. Students who heard about the simulator actually requested that it be used in the CSIS 4490 *n*-Tier Client/Server Architectures course, which ran during the summer 2001 semester, and was also used in teaching the CSIS 3510 Computer Organization and Architecture course, summer 2001, as well. All three classes were attended by information systems majors as well as computer science majors, demonstrating that the tool is useful to and understood by students with different levels of preparation and interest in the theoretical underpinnings of computer architecture.

5.1 Student Feedback

The following table summarizes student feedback on the use of the simulation tool in three recent courses at the author's institution. The "Average Evaluation" column allows students to subjectively evaluate the usefulness of the online resources supporting the course, which include the simulation tool as a prominent component (scale 1-5). The "Positive Comments" column is a tally of how many students mentioned the simulation tool specifically (without prompting) in the unstructured comments section of the student evaluations of the course. No students mentioned the tool as a negative that detracted from the course.

The consistently positive student feedback indicates that they found the tool both illuminating and easy enough to understand and use. It was also observed by the instructor that using the tool seemed to spark student interest and interactivity in exploring the material. A technique that worked well was to pose an example system architecture, discuss what the expected performance (or change in performance) would be, and then configure the simulation tool for the proposed architecture. Using a projection system, the class as a group was able to experiment with the system and discuss the results. This often led to another configuration/experiment suggested by students, which would either confirm or discredit the proposed explanation for the observed performance.

5.2 Pedagogical Uses for the Simulation Tool

The following list of course topics is segregated into categories of primary and ancillary use. Primary use courses are those in which the simulation tool represents significant and heavy use in teaching core concepts in the course knowledge areas. In these courses it is expected that, for multiple hours, the tool will be the primary lecture technology with extensive use by students in homework or research problem experiments. Courses where the tool can be used to demonstrate secondary concepts and is not useful for more than a few demonstrations are considered ancillary courses.

Courses where the tool is of primary or significant use:

- computer organization and architecture;
- parallel algorithms and systems;
- high performance computing systems;
- *n*-tier client-server architectures;
- distributed systems;
- embedded and real-time systems.
-

Courses where the simulation tool is a useful ancillary supplement:

- operating systems;
- data communications and networking.

This wide range of courses, including core foundation computer science courses common to all CS programs (required by the Computer Sciences Accreditation Board), in which the simulation tool will be useful, indicates that virtually every undergraduate computer science program could benefit from this new educational tool, and thereby improve the quality of undergraduate education both in this country and abroad.

6. CONCLUSION

The High-Performance Computer Architecture and Algorithm Simulation tool is unique and useful, appropriate for teaching computer science courses on hardware and system architecture and interactions that effect performance. The High-Performance Computing Simulation tool is highly useful in a variety of undergraduate and graduate courses, and has supported many hours of instructor and student use over two semesters of online use. It supports five levels of parallelism and a wide variety of architectures and software/application characteristics, allowing realtime and live experiments and demonstrations. These knowledge areas are integral parts of modern computing systems that should be understood by all computing students.

The following list of planned specific enhancements to the simulation tool will make the tool even more flexible and useful.

- Enhance the *n*-tier client/server modeling capabilities to support a wider range of nes interconnections between the tiers. Rather than add categories of interconnects, the approach is develop a modest programming/modeling …interface that allows users to define the capabilities a behavior of the interconnection network as a function. This will take the form of a sim specification language implemented and executed to provide dynamic performance evaluations t will be factored into the system and parallel speedup calculations. To accommodate dynar behavior, the programming interface will provide access to the underlying simulation variables, b dynamic and static. The functional definition of the interconnection network behavior will defined in a user entry box, similar to that used in spreadsheets to enter formulas, and will then interpreted when the simulation runs.
- The interconnection network modeling/specification interface will be propagated across parallel levels: 3 multiprocessors and multicomputers; 4 clustered computing architectures; and 5 distributed client/server systems. Each level of an *n*-tier system can be interconnected with a different network, with different behavior.

- Enhance the level of modeled detail in multiple levels of caching systems, from levels of caching within a machine, in order to include different caching strategies used in multiprocessors and multicomputers.
- Investigate whether unconventional parallel mechanisms (DNA, quantum) can be modeled with this tool.

ACKNOWLEDGMENTS

I would like to acknowledge the following Kennesaw State University undergraduate researchers: Anthony Aquilio, Chris Moran, and John Scragg, for their work on the High-Performance Computer Architecture and Algorithm Simulator.

REFERENCES

AMDAHL, G. 1967. Validation of the single processor approach to achieving large-scale computing capabilities. In *Proceedings of the AFIPS Conference*, 30, 483.

CARMONA, E. A. 1991. Modeling the serial and parallel fractions of a parallel program. *J. Parallel Distributed Comput*. *13*, 286-298.

EAGER, D. L., SAHORJAN, J., AND LAZOWSKA, E. D. 1989. Speedup vs efficiency in parallel systems. *IEEE Trans. Computers 38*, 3 (March).

FLYNN, M. J. 1966. Very high-speed computing systems. *Proc. IEEE 54*, 12 (Dec.).

GUSTAFSON, J. L. 1988. Reevaluating Amdahl's law. *Commun. ACM 31*, 5 (May).

HOGANSON, K. E. 2001. The unified parallel speedup model and simulator.In *Proceedings of the Southeast Regional ACM Conference* (Athens, GA, March).

HOGANSON, K. E. 2000. Alternative mechanisms to achieve parallel speedup. In *Proceedings of the First IEEE Online Symposium for Electronics Engineers* (Nov.), IEEE Society Press.

HOGANSON, K. E. 2000. Mapping parallel application communication topology to rhombic overlapping-cluster multiprocessors. *J. Supercompuing 17* (Aug.), 67-90.

HOGANSON, K. E. 1999. Workload execution strategies and parallel speedup on clustered computers. *IEEE Trans. Computers 48*, 11 (Nov.).

KARP, A. H. AND FLATT, H. P. 1990. Measuring parallel processor performance. *Commun. ACM 33*, 5 (May).

MABBS, S. A. AND FORWARD, K. E. 1994. Performance analysis of MR-1, a clustered shared-memory multiprocessor. *J. Parallel Distributed Comput. 20*, 158-175.

MOHAPTRA, P., DAS, C. R., AND FENG, T. 1994. Performance analysis of cluster-based multiprocessors. *IEEE Trans. Computers 43*, 1 (Jan.).

SUN, X. H. AND GUSTAFSON, J. L. 1991. Toward a better parallel performance metric. *Parallel Comput. 17*, 1093-1109.

VAN-CATLEDGE, F. A. 1989. Toward a general model for evaluating the relative performance of computer systems. *Int. J. Supercomputer Appl. 3*, 2 (Summer),100-108.

WANG, H., JIAN, Y., AND WU, H. 1995. Performance analysis of cluster-based PPMB multiprocessor systems. *Computer J. 38*, 5, 1995.

WOOD, D. M., AND HILL, M. D. 1995. Cost-effective parallel computing, *Computer 28*, 2 (Feb.).

Received September 2001; revised January 2002.