High-Performance Computer Architecture and Algorithm Simulator

KENNETH E. HOGANSON, PH.D.
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 intra-instruction (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: computer architecture simulation, education, parallel processing
Additional Key Words and Phrases: 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 being 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 lead to computer architectures that accommodate five distinct levels of parallelism. The importance of the interplay between these different levels, and the trade-offs in efficiency and performance improvement between allocating hardware at different levels has recently been demonstrated [Hoganson 2000a; Hoganson 1999]. This area is complex and not yet fully incorporated into textbooks, but responds well to exploration through simulation.

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 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 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; Hoganson 2001]. For the first time, students have a way to explore the complex interaction of performance enhancing computer components including pipelines, cache, multiple processors, clusters of processors, etc., which previously had to be taught as separate and disjoint topics. The tool allows the student or researcher to explore the alternative ways to allocate computer hardware in a system, i.e. should additional computing hardware be allocated to adding 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. 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 trade-offs between 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 by 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, which 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 was varied. Homework exercises were assigned that 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 out the use of the simulator as one of the highlights of the course. The simulator has also been 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, discussing modeling capabilities, the user interface, and illustrates the use of the tool with a simple experiment. Section 4 contains 5 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. Included with each problem statement is an experimental design and resulting data that explore the issue under consideration, along with a conclusion 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 this author’s institution, in a variety of computer
science courses, and 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 future enhancements to the simulation tool.

2. RECENT UNIFIED THEORETICAL MODELS OF PARALLEL PROCESSING

Theories of parallel processing recently underwent a renewal of activity and interest 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 based previously on Amdahl’s law [Amdahl 67], 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] using disparate approaches, creating a pedagogical problem in requiring a large commitment of time (and textbook pages) to cover the subject. More recently, work toward consolidating the various approaches into a unified analytical model of parallel processing that incorporated 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; Hoganson 2000a; Hoganson 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].

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; Hoganson 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: a portion of work that can be done in parallel, and a portion of work that cannot, the ratio of synchronizing communication (graininess) degrades performance, and various implementation issues 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.
<table>
<thead>
<tr>
<th>Software Parallelism</th>
<th>Enabling Architecture Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Intra-Instruction</td>
<td>Multi-stage pipeline</td>
</tr>
<tr>
<td>2. Inter-Instruction</td>
<td>Superscalar architecture (multiple pipelines)</td>
</tr>
<tr>
<td>3. Algorithm</td>
<td>Multiple processor architectures</td>
</tr>
<tr>
<td>4. Multi-Program</td>
<td>Clustered computer systems</td>
</tr>
<tr>
<td>5. Client/Server applications</td>
<td>Distributed computing systems (incl. n-Tier systems)</td>
</tr>
</tbody>
</table>

Table 1. Algorithm and Machine Parallelism

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

These recent developments in parallel processing theory have multiplied the need for a simulation tool that is appropriate for student and researcher use, class demonstrations, and student explorations that makes possible an 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 of the topics, and the depth of understanding of the interactions, tradeoffs, and limitations.

3. DESCRIPTION OF THE SIMULATION TOOL

This tool is a high-level architecture simulator, that allows control over both 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 clusters of processors, 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 the 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 – 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 of the interconnection network performance disparity over the latency of 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, distributed system and server tier configuration. There is only one of each panel of the first two panel types, but there are up to 9 identical panels for each of up to 9 levels of servers in a 10-tier system. The first panel, Fig. 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.

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 systems 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.

Figure 1. Simulation Control and Results Panel.
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).

Figure 2. Machine/Client Configuration Panel
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.

![Figure 3. N-Tier Server Configuration Panel](image)

**Simple Example: Parallel Speedup from 4 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 4 processors, while 10% must be done in serial on a single processor).

![Figure 4. Simple Example Setup](image)

**Step 2:** Then run the workload using the **Control & Output** panel, leaving other values at their startup defaults.
Figure 5. Simple Example Simulation Run and Results

**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 4 processors:

\[
\text{Speedup} = \frac{1000}{0.10(1000) + \frac{0.90(1000)}{4}}
\]

This simple example does not utilize random variables, so the results generated are exactly what would result from using a calculator.

**Part 2:** To account for contention within the interconnection network, one 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 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, which the would be assigned to the students, followed by an example approach to a set of experimental runs which answer the problem statement, and a conclusion paragraph that discusses the simulation results and their meaning and further implications.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>Classic Scaling Problem: Demonstrating diminishing returns of performance increase as the number of processors in a system increase.</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>Process Scaling: Demonstrating the effect of scaling the workload at the process level.</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>Using “The Grid”.</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>Pipeline Performance.</td>
</tr>
<tr>
<td>Experiment 5</td>
<td>N-Tier Client Server Performance and Interconnection Latency</td>
</tr>
</tbody>
</table>

*Table 2. Five Student Experiments*
**Experiment 1: Demonstrating 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” observation of Amdahl’s speedup law that dictates that performance increases from adding additional processors diminish with each successive processor.

**One Example Solution:**

- Conducted runs with linearly increasing processors, starting at 4 processors, increasing by 4 additional processors.
- Recorded the speedup at each point.
- Calculating the efficiency (Speedup over #ofprocessors) at each point.
- Calculated 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%.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td>0.8695</td>
<td>0.741</td>
<td>0.645</td>
<td>0.571</td>
<td>0.513</td>
<td>0.465</td>
<td>0.020</td>
</tr>
<tr>
<td>Added Speedup</td>
<td>2.448</td>
<td>1.816</td>
<td>1.401</td>
<td>1.113</td>
<td>0.907</td>
<td>8.464</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3. Speedup and Efficiency with Scaling Processors**

**Conclusion:**

This experiment illustrates Amdahl’s diminishing returns through observation that the additional speedup by adding 4 processors steadily declines, as does the efficiency of each additional processor in adding speedup. This suggests that computer architectures...
with many processors will be inefficiency, a conclusion effectively disputed with more recent research by Gustafson and Hoganson.

**Experiment 2: Demonstrating 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%.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Speedup</th>
<th>Efficiency</th>
<th>Added Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3.478</td>
<td>0.8695</td>
<td>2.448</td>
</tr>
<tr>
<td>8</td>
<td>5.926</td>
<td>0.741</td>
<td>1.816</td>
</tr>
<tr>
<td>12</td>
<td>7.742</td>
<td>0.645</td>
<td>1.401</td>
</tr>
<tr>
<td>16</td>
<td>9.143</td>
<td>0.571</td>
<td>1.113</td>
</tr>
<tr>
<td>20</td>
<td>10.256</td>
<td>0.513</td>
<td>0.907</td>
</tr>
<tr>
<td>24</td>
<td>11.163</td>
<td>0.465</td>
<td>-</td>
</tr>
<tr>
<td>1000</td>
<td>19.627</td>
<td>0.020</td>
<td>-</td>
</tr>
</tbody>
</table>

| Parallel factor: 10  | 3.938   | 7.717      | 11.347        |
| 100                  | 3.994   | 7.971      | 11.931        |
| 1000                 | 3.999   | 7.997      | 11.993        |

**Process Scaling Speedups**

| Parallel factor: 10  | 14.835  | 18.190     | 21.421        |
| 100                  | 15.875  | 19.802     | 23.713        |
| 1000                 | 15.987  | 19.980     | 23.971        |

| Table 4. Process Scaling Speedup |
This experiment demonstrates that very high speedups are attainable for processes where only 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 back 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 (grain).

“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 parameters not specified.

- 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).

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 5 shows that very large speedups are possible with an application with a very small serial fraction (0.001%) 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 larger communication frequency and penalty.
\[ \begin{array}{|c|c|c|c|} \hline \text{Parallel Fraction} & \text{Latency=1, Events=1} & \text{Latency=1000, Events=1} & \text{Latency=1000, Events=500} \\ \hline 99.999\% & 20,692.46 & 19,112.13 & 488.204 \\ \hline 95\% & 19.99 & 19.98 & 19.22 \\ \hline \end{array} \]

Table 5. Performance of "The Grid"

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 architecture.]

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:**

- Conducted runs with a single processor with 4, 8, and 16 stages in a single pipeline, and a non-pipelined processor, at 0% stalls and 0% flushes, 10%
stalls and 0% flushes, and 0% stalls and 10% flushes, recording the speedup at each point.

- Exactly 10% 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 are stalled for a single pipeline stage time.
- Calculating the efficiency (Speedup over #ofstages) at each point.
- Calculated the change in speedup.
- Constant: all variables other than the number of processors.

<table>
<thead>
<tr>
<th>Measuring Speedup with Increasing Stages.</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of Stages</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>0% stalls, 0% flushes</td>
<td>Speedup</td>
<td>3.988</td>
<td>7.944</td>
<td>15.764</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>99.7</td>
<td>99.3</td>
<td>98.5</td>
</tr>
<tr>
<td>10% stalls, 0% flushes</td>
<td>Speedup</td>
<td>3.626</td>
<td>7.227</td>
<td>14.350</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>90.7</td>
<td>90.3</td>
<td>89.7</td>
</tr>
<tr>
<td>0% stalls, 10% flushes</td>
<td>Speedup</td>
<td>3.070</td>
<td>4.687</td>
<td>6.362</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>76.8</td>
<td>58.2</td>
<td>39.8</td>
</tr>
</tbody>
</table>

Table 6. Pipeline Speedup by Scaling Stages

<table>
<thead>
<tr>
<th>Theoretical Expected Speedup with Increasing Stages.</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of Stages</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>0% stalls, 0% flushes</td>
<td>Speedup</td>
<td>3.988</td>
<td>7.944</td>
<td>15.764</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>99.7</td>
<td>99.3</td>
<td>98.5</td>
</tr>
<tr>
<td>10% stalls, 0% flushes</td>
<td>Speedup</td>
<td>3.626</td>
<td>7.227</td>
<td>14.350</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>90.7</td>
<td>90.3</td>
<td>89.7</td>
</tr>
<tr>
<td>0% stalls, 10% flushes</td>
<td>Speedup</td>
<td>3.070</td>
<td>4.687</td>
<td>6.362</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>76.8</td>
<td>58.2</td>
<td>39.8</td>
</tr>
</tbody>
</table>

Table 7. Comparison with Theoretical Pipeline Performance.
The theoretically predicted values for pipeline performance in Table 7 are derived using the analytical model of pipeline performance from [Hoganson 2001], given in Equation 1., and agree exactly.

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

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

\textbf{Equation 1. Pipeline Speedup}

\textbf{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. By increasing the frequency of stalls from 0 to 10% detracts from the 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

[Appropriate for a course on client/server systems or distributed systems.]

This experiment observes the sensitivity of N-Tier Client/Server architectures to the 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 obtained.
   - 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 obtained.
   - 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.

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:

<table>
<thead>
<tr>
<th>Run</th>
<th>Msg Frequency</th>
<th>Msg Latency</th>
<th>CPUs per Server</th>
<th>Servers per Tier</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>1 per client</td>
<td>1 CPU-Op</td>
<td>1</td>
<td>2</td>
<td>2.700</td>
</tr>
<tr>
<td>Run 2</td>
<td>10 per client</td>
<td>10 CPU-Op</td>
<td>1</td>
<td>2</td>
<td>1.304</td>
</tr>
<tr>
<td>Run 3</td>
<td>1 per client</td>
<td>1 CPU-Op</td>
<td>4</td>
<td>5</td>
<td>14.150</td>
</tr>
<tr>
<td>Run 4</td>
<td>10 per client</td>
<td>10 CPU-Op</td>
<td>4</td>
<td>5</td>
<td>2.143</td>
</tr>
</tbody>
</table>

Table 8. N-Tier Speedup and Interconnect Latency.
Run 1 provides a modest speedup, even thought there are 14 machines operating in parallel, because only $1/3^{rd}$ of the work is distributed across the clients, while $2/3^{rd}$ 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 the communication latency. Each client communications 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 & 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 illustrates 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/Server and other distributed systems: the more coarse-grained the better (few communications 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 if a significant portion of the processing occurs at the server side. The above systems perform significantly better with more computation 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 IN UNDERGRADUATE

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 have heard about the simulator actually requested that it be used in the CSIS 4490 n-Tier Client/Server Architectures course which ran summer semester 2001, and it was also used in the teaching of CSIS 3510 Computer Organization and Architecture summer 2001 as well. All three class contain both Information Systems majors as well as Computer Science majors, demonstrating that the tool is useful and understandable by students with different levels of preparation and interest in theoretical underpinnings of computer architecture.

5.1 Student Feedback

The following table summarizes the 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.
<table>
<thead>
<tr>
<th>Course</th>
<th>Number of Students</th>
<th>Average Evaluation</th>
<th>Positive Comments</th>
<th>Negative Comments</th>
<th>Number of Experiments</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIS 3510 Org&amp;Arch</td>
<td>32</td>
<td>4.26</td>
<td>7</td>
<td>0</td>
<td>3</td>
<td>Summer 2001</td>
</tr>
<tr>
<td>CSIS 4490 N-Tier Architectures</td>
<td>29</td>
<td>4.83</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>Summer 2001</td>
</tr>
<tr>
<td>CSIS 4310 Parallel Systems</td>
<td>8</td>
<td>4.50</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>Spring 2001</td>
</tr>
</tbody>
</table>

Table 9. Student Acceptance of the Simulation Tool.

The consistently positive student feedback is indicates that the students found the tool both illuminating and easy enough to understand and use. It was also observed by the instructor, that the use of the tool seemed to spark student interest and interactive participation 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, then configure the simulation tool for the proposed architecture and run and experiment. Using a projection system, the class as a group was able to experiment with the system, and discuss the observed results. This often lead to another configuration/experiment suggested by students, that would either confirm or discred the proposed explanation for the observed performance.

5.2 Pedagogy Uses for the Simulation Tool.

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

Courses where the tool will be 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

Courses where the simulation tool will be 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 and required by the Computer Science Accreditation Commission of the Computer Sciences Accreditation Board, where the simulation tool will be useful, indicates that virtually every undergraduate computer science program could potentially benefit from this new educational tool, thereby improving the quality of undergraduate education both in this country and internationally.

6. CONCLUSION

The High-Performance Computer Architecture and Algorithm Simulation tool is a unique and useful tool, appropriate and useful for teaching computer science courses having to do with hardware/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 2 semesters of on-line use. It supports the five levels of parallelism and a wide variety of architectures and software/application characteristics, allowing real-time 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 nested interconnections between the tiers. Rather than add categories of interconnects, the approach used is to develop a modest programming/modeling interface that allows users to define the capabilities and behavior of the interconnection network as a function. This will take the form of a simple specification language that will be implemented and executed to provide dynamic evaluations of performance that will factored into the system and parallel speedup calculations. To accommodate dynamic behavior, the programming interface will provide access to the underlying simulation variables, both dynamic and static. The functional definition of the interconnection network behavior will be defined in a user entry box, similar to that used in spreadsheets to enter formulas, and then will be interpreted when the simulation runs.

- The interconnection network modeling/specification interface will be propagated across parallel levels 3 (multiprocessor and multi-computer), 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 of multiple levels of caching systems, from levels of caching within a machine to include different caching strategies used in multiprocessors and multi-computers.

• Investigate whether unconventional parallel mechanisms (DNA, quantum) can be modeled with this tool.

ACKNOWLEDGEMENTS

I would like to acknowledge 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