Performance Evaluation: Techniques, Tools and Benchmarks

Author Info:

Prof. Lizy Kurian John
Electrical and Computer Engineering Department, ENS 143
The University of Texas at Austin
Austin, TX 78712
ljohn@ece.utexas.edu
Phone: 512 232 1455
Fax: 512 471 2893
http://www.ece.utexas.edu/~ljohn

Contents:

1. Introduction
    Classification of Performance Evaluation Techniques
2. Performance Measurement Techniques
    a. On-chip Performance Monitoring Counters
    b. Off-chip hardware measurement
    c. Software Monitoring
    d. Microcoded Instrumentation
3. Performance Modeling Techniques
    a. Simulation
        i. Trace Driven Simulation
        ii. Execution Driven Simulation
        iii. Complete System Simulation
        iv. Stochastic Discrete Event Driven Simulation
        v. Software Profiling Tools
    b. Analytical Modeling
4. Workloads and Benchmarks
    CPU, Embedded, Java, Transaction Processing, Web server, E-commerce, server and personal computer benchmarks

1. Introduction

State-of-the-art high performance microprocessors contain tens of millions of transistors and operate at frequencies close to 2GHz. These processors perform several tasks in overlap, employ significant amounts of speculation and out-of-order execution, and other microarchitectural techniques, and are true marvels of engineering. Designing and evaluating these microprocessors is a major challenge, especially considering the fact that one second of program execution on these processors involves several billion instructions and analyzing one second of execution may involve dealing with tens of billion pieces of information.

In general, design of microprocessors and computer systems involves several steps (i) understanding applications and workloads that the systems will be running (ii) innovating potential designs (iii) evaluating performance of the candidate designs, and (iv) selecting the best design. The large number of potential designs and the constantly evolving nature of workloads have resulted in designs being largely adhoc. In this article, we investigate major techniques used in the performance evaluation process.

It should be noted that performance evaluation is needed at several stages of the design. In early stages, when the design is being conceived, performance evaluation is used to make early design tradeoffs. Usually, this is accomplished by simulation models, because building prototypes of state-of-the-art microprocessors is expensive and time consuming. Several design decisions are made before any prototyping is done. Once the design is finalized and is being implemented, simulation is used to evaluate functionality and performance of subsystems. Later, performance measurement is done after the product is available in order to understand the performance of the actual system to various real world workloads and to identify modifications to incorporate in future designs.

Performance evaluation can be classified into performance modeling and performance measurement, as illustrated in Table 1. Performance measurement is possible only if the system of interest is available for measurement and only if one has access to the parameters of interest. Performance measurement may further be classified into on-chip hardware monitoring, off-chip hardware monitoring, software monitoring and microcoded instrumentation. Performance modeling is typically used when actual systems are not available for measurement or if the actual systems do not have test points to measure every detail of interest. Performance modeling may further be classified into simulation modeling and analytical modeling. Simulation models may further be classified into numerous categories depending on the mode/level of detail of simulation. Analytical models use probabilistic models, queueing theory, Markov models or Petri nets.

Table 1. A Classification of Performance Evaluation Techniques

Performance Measurement	Microprocessor On-chip Performance Monitoring Counters	
	Off-chip Hardware Monitoring	
	Software Monitoring	
	Micro-coded Instrumentation	
Performance Modeling	Simulation	Trace Driven Simulation
		Execution Driven Simulation
		Complete System Simulation
		Event Driven Simulation
		Software Profiling
	Analytical Modeling	Probabilistic Models
		Queuing Models
		Markov Models
		Petri Net Models

There are several desirable features that performance modeling/measurement techniques and tools should possess.

- They must be accurate. It is easy to build models that are heavily sanitized, however, such models will not be accurate.

- They must be non-invasive. The measurement process must not alter the system or degrade the system's performance.
- They must not be expensive. Building the performance measurement facility should not cost significant amount of time or money.
- They must be easy to change or extend. Microprocessors and computer systems constantly undergo changes and it must be easy to extend the modeling/measurement facility to include the upgraded system.
- They must not need source code of applications. If tools and techniques necessitate source code, it will not be possible to evaluate commercial applications where source is not often available.
- They should measure all activity including kernel and user activity. Often it is easy to build tools that measure only user activity. This was acceptable in traditional scientific and engineering workloads, however in database, web server, and Java workloads, there is significant operating system activity and it is important to build tools that measure operating system activity as well.
- They should be capable of measuring a wide variety of applications including those that use signals, exceptions and DLLs (Dynamically Linked Libraries).
- They should be user-friendly. Hard to use tools often are under-utilized. Hard-to-use tools also result in more user error.
- They should be fast. If a performance model is very slow, long-running workloads which take hours to run may take days or weeks to run on the model. If an instrumentation tool is slow, it can be invasive.
- Models should provide control over aspects that are measured. It should be possible to selectively measure what is required.
- Models and tools should handle multiprocessor systems and multithreaded applications. Dual and quad-processor systems are very common nowadays. Applications are becoming increasingly multithreaded especially with the advent of Java, and it is important that the tool handles these.
- It will be desirable for a performance evaluation technique to be able to evaluate the performance of systems that are not yet built.

Many of these requirements are often conflicting. For instance, it is difficult for a mechanism to be fast and accurate. Consider mathematical models. They are fast, however, several simplifying assumptions go into their creation and often they are not accurate. Similarly it is difficult for a tool to be non-invasive and user-friendly. Many users like graphical user interfaces (GUIs), however, most instrumentation and simulation tools with GUIs are slow and invasive.

Benchmarks and metrics to be used for performance evaluation have always been interesting and controversial issues. There has been a lot of improvement in benchmark suites since 1988. Before that computer performance evaluation has been largely with small benchmarks such as kernels extracted from applications (eg: Lawrence Livermore Loops), Dhrystone and Whetstone benchmarks, Linpack, Sorting, Sieve of Eratosthenes, 8-queens problem, Tower of Hanoi, etc. [1]. The Standard Performance Evaluation Cooperative (SPEC) consortium and the Transactions Processing Council (TPC) formed in 1988 have made available several benchmark suites and benchmarking guidelines to improve the quality of benchmarking. Several state-of-the-art benchmark suites are described in section 4.

Another important issue in performance evaluation is the choice of performance metric. For a system level designer, execution time and throughput are two important performance metrics. Execution time is generally the most important measure of performance. Execution time is the product of the number of instructions, cycles per instruction (CPI) and the clock period. Throughput of an application is a more important metric, especially in server systems. In servers that serve the banking industry, airline industry, or other similar business, what is important is the number of transactions that could be completed in unit time. Such servers, typically called transaction processing systems use transactions per minute (tpm) as a performance metric. MIPS (Millions of Instructions Per Second) and MFLOPS (Million of Floating Point Operations Per Second) have been very popular measures of performance in the past. Both of these are very simple and straightforward to understand and hence have been used often, however, they do not contain all three components of program execution time and hence are incomplete measures of performance. There are also several low level metrics of interest to microprocessor designers, in order to help them identify performance bottlenecks and tune their designs. Cache hit ratios, branch misprediction ratios, number of off-chip memory accesses, etc are examples of such measures.

Another major problem is the issue of reporting performance with a single number. A single number is easy to understand and easy to be used by the trade press. Use of several benchmarks also make it necessary to find some kind of a mean. Arithmetic Mean, Geometric Mean and Harmonic Mean are three ways of finding the central tendency of a group of numbers, however, it should be noted that each of these means should be used in appropriate conditions depending on the nature of the numbers which need to be averaged. Simple arithmetic mean can be used to find average execution time from a set of execution times. Geometric mean can be used to find the central tendency of metrics that are in the form of ratios (eg: speedup) and harmonic mean can be used to find the central tendency of measures that are in the form of a rate (eg: throughput). Cragon [2] and Smith [3] discuss the use of the appropriate mean for a given set of data. Cragon [2] and Patterson and Hennessy [4] illustrate several mistakes one could possibly make while finding a single performance number.

The rest of this article is organized as follows. Section 2 describes performance measurement techniques including hardware on-chip performance monitoring counters on microprocessors. Section 3 describes simulation and analytical modeling of microprocessors and computer systems. Section 4 presents several state-of-the-art benchmark suites for a variety of workloads. Due to limitations of space in this article, we describe some typical examples of tools and techniques and provide the reader with pointers for more information.

2. Performance Measurement

Performance measurement is used for understanding systems that are already built or prototyped. There are two major purposes performance measurement can serve: (i) tune this system or systems to be built (ii) tune the application if source code and algorithms can still be changed. Essentially, the process involves (i) understanding the bottlenecks in the system that has been built (ii) understanding the applications that are running on the system and the match between the features of the system and the characteristics of the workload, and, (iii) innovating design features that will exploit the workload features. Performance measurement can be done via the following means:

- Microprocessor on-chip performance monitoring counters
- Off-chip hardware monitoring
- Software monitoring
- Microcoded instrumentation

2.1 On-chip Performance Monitoring Counters

All state-of-the-art high performance microprocessors including Intel's Pentium III and Pentium IV, IBM's POWER 3 and POWER 4 processors, AMD's Athlon, Compaq's Alpha, and Sun's UltraSPARC processors incorporate on-chip performance monitoring counters which can be used to understand performance of these microprocessors while they run complex, real-world workloads. This ability has overcome a serious limitation of simulators, that they often could not execute complex workloads. Now, complex run time systems involving multiple software applications can be evaluated and monitored very closely. All microprocessor vendors nowadays release information on their performance monitoring counters, although they are not part of the architecture.

For illustration of on-chip performance monitoring, we use the Intel Pentium processors. The microprocessors in the Intel Pentium contain two performance monitoring counters. These counters can be read with special instructions (eg: RDPMC) on the processor. The counters can be made to measure user and kernel activity in combination or in isolation. A variety of performance events can be measured using the counters [50]. For illustration of the nature of the events that can be measured, Table 2 lists a small subset of the events that can be measured on the Pentium III. While more than 200 distinct events can be measured on the Pentium III, only 2 events can be measured simultaneously. For design simplicity, most microprocessors limit the number of events that can be simultaneously measured to 4 or 5. At times, certain events are restricted to be accessible only through a particular counter. These steps are necessary to reduce the overhead associated with on-chip performance monitoring. Performance counters do consume on-chip real estate. Unless carefully implemented, they can also impact the processor cycle time.

Table 2. Examples of events that can be measured using performance monitoring counters on an Intel Pentium III processor

EVENT	**Description of Event**	**Event Number in Hex**
DATA_MEM_REFS	All loads and stores from/to memory	43H
DCU_LINES_IN	Total lines allocated in the data cache unit	45H
IFU_IFETCH	Number of instruction fetches (cacheable and uncacheable)	80H
IFU_IFETCH_MISS	Number of instruction fetch misses	81H
ITLB_MISS	Number of Instruction TLB misses	85H
IFU_MEM_STALL	Number of cycles instruction fetch is stalled for any reason	86H
L2_IFETCH	Number of L2 instruction fetches	28H
L2_LD	Number of L2 data loads	29H
L2_ST	Number of L2 data stores	2AH
L2_LINES_IN	Number of lines allocated in the L2	24H
L2_RQSTS	Total number of L2 requests	2EH
INST_RETIRED	Number of instructions retired	C0H
UOPS_RETIRED	Number of micro-operations retired	C2H
INST_DECODED	Number of instructions decoded	D0H
RESOURCE_STALLS	Number of cycles in which there is a resource related stall	A2H
MMX_INSTR_EXEC	Number of MMX Instructions Executed	B0H
BR_INST_RETIRED	Number of branch instructions retired	C4H
BR_MISS_PRED_RETIRED	Number of mispredicted branches retired	C5H
BR_TAKEN_RETIRED	Number of taken branches retired	C9H
BR_INST_DECODED	Number of branch instructions decoded	E0H
BTB_MISSES	Number of branches for which BTB did not predict	E2H

There are several tools available to measure performance using performance monitoring counters. Table 3 lists some of the available tools. Intel's *Vtune* software may be used to perform measurements using the Intel processor performance counters [5]. The *P6Perf* utility is a plug in for Windows NT performance monitoring [6]. The Compaq DIGITAL Continuous Profiling Infrastructure *(DCPI)* is a very powerful tool to profile programs on the Alpha processors [7,8]. The performance monitor *perf-mon* is a small hack that uses the on-chip counters on UltraSPARC-I/II processors to gather statistics [9]. Packages like Vtune perform extensive post-processing and present data in graphical forms. However, some times, extensive post-processing can result in tools that are somewhat invasive. *PMON* [10] is a counter reading software written by Juan Rubio of the Laboratory for Computer Architecture at the University of Texas. It provides a mechanism to read specified counters with minimal or no perceivable overhead. All these tools measure user and operating system activity. Since everything on a processor is counted, effort should be made to have minimal or no other undesired process running during experimentation. This type of performance measurement can be done on binaries, and no source code is desired.

Table 3. Software packages for performance counter measurement

Tool	Platform	Reference
VTune	IA-32	http://developer.intel.com/software/products/vtune/vtune_oview.htm
P6Perf	IA-32	http://developer.intel.com/vtune/p6perf/index.htm
PMON	IA-32	http://www.ece.utexas.edu/projects/ece/lca/pmon
DCPI	Alpha	http://www.research.digital.com/SRC/dcpi/ http://www.research.compaq.com/SRC/dcpi/
Perf-mon	UltraSPARC	http://www.sics.se/~mch/perf-monitor/index.html

2.2 Off-chip hardware measurement

Instrumentation using hardware means can also be done by attaching off-chip hardware, two examples of which are described in this section.

SpeedTracer from AMD: AMD developed this hardware tracing platform to aid in the design of their x86 microprocessors. When an application is being traced, the tracer interrupts the processor on each instruction boundary. The state of the CPU is captured on each interrupt and then transferred to a separate control machine where the trace is stored. The trace contains virtually all valuable pieces of information for each instruction that executes on the processor. Operating system activity can also be traced. However, tracing in this manner can be invasive, and may slow down the processor. Although the processor is running slower, external events such as disk and memory accesses still happen in real time, thus looking very fast to the slowed-down processor. Usually this issue is addressed by adjusting the timer interrupt frequency. Use of this performance monitoring facility can be seen in Merten [11] and Bhargava[12].

Logic Analyzers: Poursepanj and Christie [13] use a Tektronix TLA 700 logic analyzer to analyze 3D graphics workloads on AMD-K6-2 based systems. Detailed logic analyzer traces are limited by restrictions on sizes and are typically used for the most important sections of the program under analysis. Preliminary coarse level analysis can be done by performance monitoring counters and software instrumentation. Poursepanj and Christie used logic analyzer traces for a few tens of frames which covered a second or two of smooth motion [13].

2.3 Software Monitoring

Software monitoring is often performed by utilizing architectural features such as a trap instruction or a breakpoint instruction on an actual system, or on a prototype. The VAX processor from Digital (now Compaq) had a T-bit that caused an exception after every instruction. Software monitoring used to be an important mode of performance evaluation before the advent of on-chip performance monitoring counters. The primary advantage of software monitoring is that it is easy to do. However, disadvantages include that the instrumentation can slow down the application. The overhead of servicing the exception, switching to a data collection process, and performing the necessary tracing can slow down a program by more than 1000 times. Another disadvantage is that software monitoring systems typically only handle the user activity.

2.4 Microcoded Instrumentation

Digital (now Compaq) used microcoded instrumentation to obtain traces of VAX and Alpha architectures. The ATUM tool [14] used extensively by Digital in the late 1980s and early 1990s uses microcoded instrumentation. This is a technique lying between trapping information on each instruction using hardware interrupts (traps) or software traps. The tracing system essentially modified the VAX microcode to record all instruction and data references in a reserved portion of memory. Unlike software monitoring, ATUM could trace all processes including the operating system. However, this kind of tracing is invasive, and can slow down the system by a factor of 10 without including the time to write the trace to the disk.

3. Performance Modeling

Performance measurement as described in the previous section can be done only if the actual system or a prototype exists. It is expensive to build prototypes for early stage evaluation. Hence one needs to resort to some kind of modeling in order to study systems yet to be built. Performance modeling can be done using simulation models or analytical models.

3.1 Simulation

Simulation has become the defacto performance modeling method in the evaluation of microprocessor architectures. There are several reasons for this. The accuracy of analytical models in the past has been insufficient for the type of design decisions computer architects wish to make (for instance, what kind of caches or branch predictors are needed). Hence cycle accurate simulation has been used extensively by architects. Simulators model existing or future machines or microprocessors. They are essentially a model of the system being simulated, written in a high level computer language such as C or Java, and running on some existing machine. The machine on which the simulator runs is called the host machine and the machine being modeled is called the target machine. Such simulators can be constructed in many ways.

Simulators can be functional simulators or timing simulators. They can be trace driven or execution driven simulators. They can be simulators of components or that of the complete system. Functional simulators simulate functionality of the target processor, and in essence provide a component similar to the one being modeled. The register values of the simulated machine are available in the equivalent registers of the simulator. In addition to the values, the simulators also provide performance information in terms of cycles of execution, cache hit ratios, branch prediction rates, etc. Thus the simulator is a virtual component representing the microprocessor or subsystem being modeled plus a variety of performance information.

If performance evaluation is the only objective, one does not need to model the functionality. For instance, a cache performance simulator does not need to actually store values in the cache; it only needs to store information related to the address of the value being cached. That information is sufficient to determine a future hit or miss. While it is nice to have the values as well, a simulator that models functionality in addition to performance is bound to be slower than a pure performance simulator. Register Transfer Language (RTL) models used for functional verification may also be used for performance simulations, however, these models are very slow for performance estimation with real world workloads, and hence are not discussed in this article.

3.1.1 Trace Driven Simulation

Trace-driven simulation consists of a simulator model whose input is modeled as a trace or sequence of information representing the instruction sequence that would have actually executed on the target machine. A simple trace driven cache simulator needs a trace consisting of address values. Depending on whether the simulator is modeling a unified instruction or data cache, the address trace should contain addresses of instruction and data references.

Cachesim5 and Dinero IV are examples of cache simulators for memory reference traces. Cachesim5 comes from Sun Microsystems along with their Shade package [15]. Dinero IV [16] is available from the University of Wisconsin, Madison. These simulators are not timing simulators. There is no notion of simulated time or cycles, only references. They are not functional simulators. Data and instructions do not move in and out of the caches. The primary result of simulation is hit and miss information. The basic idea is to simulate a memory hierarchy consisting of various caches. The various parameters of each cache can be set separately (architecture, mapping policies, replacement policies, write policy, statistics). During initialization, the configuration to be simulated is built up, one cache at a time, starting with each memory as a special case. After initialization, each reference is fed to the appropriate top-level cache by a single simple function call. Lower levels of the hierarchy are handled automatically. One does not need to store a trace while using cachesim5, because Shade can directly feed the trace into cachesim5.

Trace driven simulation is simple and easy to understand. The simulators are easy to debug. Experiments are repeatable because the input information is not changing from run to run. However, trace driven simulation has two major problems:

1. Traces can be prohibitively long if entire executions of some real-world applications are considered. The storage needed by the traces may be prohibitively large. Trace size is proportional to the dynamic instruction count of the benchmark.
2. The traces do not represent the actual stream of processors with branch predictions. Most trace generators generate traces of only completed or retired instructions in speculative processors. Hence they do not contain instructions from the mispredicted path.

The first problem is typically solved using trace sampling and trace reduction techniques. Trace sampling is a method to achieve reduced traces. However, the sampling should be performed in such a way that the resulting trace is representative of the original trace. It may not be sufficient to periodically sample a program execution. Locality properties of the resulting sequence may be widely different from that of the original sequence. Another technique is to skip tracing for a certain interval, then collect for a fixed interval and then skip again. It may also be needed to leave a warm up period after the skip interval, to let the caches and other such structures to warm up [17]. Several trace sampling techniques are discussed by Crowley and Baer [18]. The QPT trace collection system [19] solves the trace size issue by splitting the tracing process into a trace record generation step and a trace regeneration process. The trace record has a size similar to the static code size, and the trace regeneration expands it to the actual full trace upon demand.

The second problem can be solved by reconstructing the mispredicted path [20]. An image of the instruction memory space of the application is created by one pass through the trace, and thereafter fetching from this image as opposed to the trace. While 100% of the mispredicted branch targets may not be in the recreated image, studies show that more than 95% of the targets can be located.

3.1.2 Execution Driven Simulation

There are two meanings in which this term is used by researchers and practitioners. Some refer to simulators that take program executables as input as execution driven simulators. These simulators utilize the actual input executable and not a trace. Hence the size of the input is proportional to the static instruction count and not the dynamic instruction count. Mispredicted branches can be accurately simulated as well. Thus these simulators solve the two major problems faced by trace-driven simulators. The widely used Simplescalar simulator [21] is an example of such an execution driven simulator. With this tool set, the user can simulate real programs on a range of modern processors and systems, using fast execution-driven simulation. There is a fast functional simulator and a detailed, out-of-order issue processor that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction.

Some others consider execution driven simulators to be simulators that rely on actual execution of parts of code on the host machine (hardware acceleration by the host instead of simulation) [22]. These execution driven simulators do not simulate every individual instruction in the application. Only the instructions that are of interest are simulated. The remaining instructions are directly executed by the host computer. This can be done when the instruction set of the host is the same as that of the machine being simulated. Such simulation involves two stages. In the first stage or preprocessing, the application program is modified by inserting calls to the simulator routines at events of interest. For instance, for a memory system simulator, only memory access instructions need to be instrumented. For other instructions, the only important thing is to make sure that they get performed and that their execution time is properly accounted for. The advantage of execution driven simulation is speed. By directly executing most instructions at the machine's execution rate, the simulator can operate orders of magnitude faster than cycle by cycle simulators that emulate each individual instruction. Tango, Proteus and FAST are examples of such simulators [22].

3.1.3 Complete system simulation

Many execution and trace driven simulators only simulate the processor and memory subsystem. Neither I/O activity nor operating system activity is handled in simulators like Simplescalar. But in many workloads, it is extremely important to consider I/O and operating system activity. Complete system simulators are complete simulation environments that model hardware components with enough detail to boot and run a full-blown commercial operating system. The functionality of the processors, memory subsystem, disks, buses, SCSI/IDE/FC controllers, network controllers, graphics controllers, CD-ROM, serial devices, timers, etc are modeled accurately in order to achieve this. While functionality stays the same, different microarchitectures in the processing component can lead to different performance. Most of the complete system simulators use microarchitectural models that can be plugged in and out. For instance, SimOS [23], a popular complete system simulator provides a simple pipelined processor model and an aggressive superscalar processor model. SimOS and SIMICS [24,25] can simulate uniprocessor and multiprocessor systems. Table 4 lists popular complete system simulators.

Table 4. Examples of complete system simulators

Simulator	Information Site	Instruction Set	Operating System
SimOS	Stanford University http://simos.stanford.edu/	MIPS	SGI IRIX
SIMICS	Virtutech http://www.simics.com http://www.virtutech.com	PC, SPARC and Alpha	Solaris 7 and 8, Red Hat Linux 6.2 (both x86, SPARC V9, and Alpha versions), Tru64 (Digital Unix 4.0F), and Windows NT 4.0
Bochs	http://bochs.sourceforge.net	X86	Windows Windows 95, Windows NT, Linux, FreeBSD

3.1.4 Stochastic Discrete Event Driven Simulation

It is possible to simulate systems in such a way that the input is derived stochastically rather than as a trace/executable from an actual execution. For instance, one can construct a memory system simulator in which the inputs are assumed to arrive according to a Gaussian distribution. Such models can be written in general purpose languages such as C, or using special simulation languages such as SIMSCRIPT. Languages such as SIMSCRIPT have several built-in primitives to allow quick simulation of most kinds of common systems. There are built-in input profiles, resource templates, process templates, queue structures, etc. to facilitate easy simulation of common systems. An example of the use of event-driven simulators using SIMSCRIPT may be seen in the performance evaluation of multiple-bus multiprocessor systems in Kurian et. al [26,27].

3.1.5 Program Profilers

There are a class of tools called software profiling tools, which are similar to simulators and performance measurement tools. These tools are used to generate traces, to obtain instruction mix, and a variety of instruction statistics. They can be thought of as software monitoring on a simulator. They input an executable and decode and analyze each instruction in the executable. These program profilers can be used as the front end of simulators. A popular program profiling tool is Shade for the UltraSparc [15].

Shade

SHADE is a fast instruction-set simulator for execution profiling. It is a simulation and tracing tool that provides features of simulators and tracers in one tool. Shade analyzes the original program instructions and cross-compiles them to sequences of instructions that simulate or trace the original code. Static cross-compilation can produce fast code, but purely static translators cannot simulate and trace all details of dynamically linked code. One can develop a variety of 'analyzers' to process the information generated by Shade and create the performance metrics of interest. For instance, one can use shade to generate address traces to feed into a cache analyzer to compute hit-rates and miss rates of cache configurations. The shade analyzer **cachesim5** does exactly this.

Jaba

Jaba [46] is a Java Bytecode Analyzer developed at the University of Texas for tracing Java programs. While Java programs can be traced using shade to obtain profiles of native execution, Jaba can yield profiles at the bytecode level. It uses JVM specification 1.1. It allows the user to gather information about the dynamic execution of a Java application at the Java bytecode level. It provides information on bytecodes executed, load operations, branches executed, branch outcomes, etc. Use of this tool can be found in [47].

A variety of profiling tools exist for different platforms. In addition to describing the working of Shade, Cmelik et. al [15] also compares Shade to several other profiling tools for other platforms. A popular one for the x86 platform is Etch [51]. Conte and Gimarc [52] is a good source of information to those interested in creating profiling tools.

3.2 Analytical Modeling

Analytical performance models, while not popular for microprocessors are suitable for evaluation of large computer systems. In large systems where details cannot be modeled accurately for cycle accurate simulation, analytical modeling is an appropriate way to obtain approximate performance metrics. Computer systems can generally be considered as a set of hardware and software resources and a set of tasks or jobs competing for using the resources. Multicomputer systems and multiprogrammed systems are examples.

Analytical models rely on probabilistic methods, queuing theory, Markov models, or Petri nets to create a model of the computer system. A large body of literature on analytical models of computer exists from the 1970s and early 1980s. Heidelberger and Lavenberg [28] published an article summarizing research on computer performance evaluation models. This article contains 205 references, which cover all important work on performance evaluation until 1984. Readers interested in analytical modeling should read this article.

Analytical models are cost-effective because they are based on efficient solutions to mathematical equations. However, in order to be able to have tractable solutions, often, simplifying assumptions are made regarding the structure of the model. As a result, analytical models do not capture all the detail typically built into simulation models. It is generally thought that carefully constructed analytical models can provide estimates of average job throughputs and device utilizations to within 10% accuracy and average response times within 30% accuracy. This level of accuracy while insufficient for microarchitectural enhancement studies, is sufficient for capacity planning in multicomputer systems, I/O subsystem performance evaluation in large server farms, and in early design evaluations of multiprocessor systems.

There has not been much work on analytical modeling of microprocessors. The level of accuracy needed in trade off analysis for microprocessor structures is more than what typical analytical models can provide. However, some effort into this arena came from Noonburg and Shen [29] and Sorin et. al [30]. Those interested in modeling superscalar processors using analytical models should read Noonburg et. al's work [29] and Sorin et. al's work [30]. Noonburg et. al used a Markov model to model a pipelined processor. Sorin et. al used probabilistic techniques to processor a multiprocessor composed of superscalar processors. Queuing theory is also applicable to superscalar processor modeling, as modern superscalar processors contain instruction queues in which instructions wait to be issued to one among a group of functional units.

4. Workloads and Benchmarks

Benchmarks used for performance evaluation of computers should be representative of applications that are run on actual systems. Contemporary computer applications include a variety of applications, and different benchmarks are appropriate for systems targeted for different purposes. Table 5 lists several popular benchmarks for different classes of workloads.

Table 5. Popular benchmarks for different categories of workloads

Workload Category		**Example Benchmark Suite**
CPU Benchmarks	Uniprocessor	SPEC CPU 2000 [31]
		Java Grande Forum Benchmarks [32]
		SciMark [33]
		ASCI [34]
	Parallel Processor	SPLASH [35]
		NASPAR [36]
Multimedia		MediaBench [37]
Embedded		EEMBC benchmarks [38]
Digital Signal Processing		BDTI benchmarks [39]
Java	Client side	SPECjvm98 [31]
		CaffeineMark [40]
	Server side	SPECjBB2000 [31]
		VolanoMark [41]
	Scientific	Java Grande Forum Benchmarks [32]
		SciMark [33]
Transaction Processing	OLTP (On-Line Transaction Processing)	TPC-C [42]
		TPC-W [42]
	DSS (Decision Support Systems)	TPC-H [42]
		TPC-R [42]
Web Server		SPEC web99 [31]
		TPC-W [42]
		VolanoMark [41]
Electronic commerce	With commercial database	TPC-W [42]
	Without commercial database	SPECjBB2000 [31]
Mail-server		SPECmail2000 [31]
Network File System		SPEC SFS 2.0 [31]
Personal Computer		SYSMARK [43]
		Ziff Davis WinBench [44]
		3DMarkMAX99 [45]

4.1 CPU Benchmarks

SPEC CPU2000 is the industry-standardized CPU-intensive benchmark suite. The System Performance Evaluation Cooperative (SPEC) was founded in 1988 by a small number of workstation vendors who realized that the marketplace was in desperate need of realistic, standardized performance tests. The basic SPEC methodology is to provide the benchmarker with a standardized suite of source code based upon existing applications that has already been ported to a wide variety of platforms by its membership. The benchmarker then takes this source code, compiles it for the system in question. The use of already accepted and ported source code greatly reduces the problem of making apples-to-oranges comparisons SPEC designed CPU2000 to provide a comparative measure of compute intensive performance across the widest practical range of hardware. The implementation resulted in source code benchmarks developed from real user applications. These benchmarks measure the performance of the processor, memory and compiler on the tested system. The suite contains 14 floating point programs written in C/Fortran and 11 integer programs (10 written in C and 1 in C++). The SPEC CPU2000 benchmarks replace the SPEC89, SPEC92 and SPEC95 benchmarks.

The Java Grande Forum Benchmark suite consists of three groups of benchmarks, microbenchmarks that test individual low-level operations (eg: arithmetic, cast, create), Kernel benchmarks which are the heart of the algorithms of commonly used applications (eg: heapsort, encryption/decryption, FFT, Sparse matrix multiplication, etc), and applications (eg: Raytracer, MonteCarlo simulation, Euler equation solution, Molecular dynamics, etc) [48]. These are compute intensive benchmarks available in Java.

SciMark is a composite Java benchmark measuring the performance of numerical codes occurring in scientific and engineering applications. It consists of five computational kernels: FFT, Gauss-Seidel relaxation, Sparse matrix-multiply, Monte Carlo integration, and dense LU factorization. These kernels are chosen to provide an indication of how well the underlying Java Virtual Machines perform on applications utilizing these types of algorithms. The problems sizes are purposely chosen to be small in order to isolate the effects of memory hierarchy and focus on internal JVM/JIT and CPU issues. A larger version of the benchmark (SciMark 2.0 LARGE) addresses performance of the memory subsystem with out-of-cache problem sizes.

ASCI The Accelerated Strategic Computing Initiative (ASCI) of the Lawrence Livermore laboratories contain several numeric codes suitable for evaluation of compute intensive systems. The programs are available from [34].

SPLASH The SPLASH suite was created by Stanford researchers [35]. The suite contains six scientific and engineering applications, all of which are parallel applications.

The **NAS Parallel** Benchmarks (NPB) are a set of 8 programs designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications.

4.2 Embedded and Media Benchmarks

EEMBC Benchmarks

The EDN Embedded Microprocessor Benchmark Consortium (EEMBC - pronounced embassy) was formed in April 1997 to develop meaningful performance benchmarks for processors in embedded applications. EEMBC is backed by the majority of the processor industry and has therefore established itself as the industry-standard, embedded processor benchmarking forum. EEMBC establishes benchmark standards and provides certified benchmarking results through the EEMBC Certification Labs (ECL) in Texas and California. The EEMBC's benchmarks comprise a suite of benchmarks designed to reflect real-world applications, while it also includes some synthetic benchmarks. These benchmarks target the automotive/industrial, consumer, networking, office automation, and telecommunications markets. More specifically, these benchmarks target specific applications that include engine control, digital cameras, printers, cellular phones, modems, and similar devices with embedded microprocessors. The EEMBC consortium dissected these applications and derived 37 individual algorithms that constitutes the EEMBC's Version 1.0 suite of benchmarks.

BDTI Benchmarks

Berkeley Design Technology, Inc. (BDTI) is a technical services company that has focused exclusively on Digital Signal Processing since 1991. BDTI provides the industry standard BDTI Benchmarks™, a proprietary suite of DSP benchmarks. BDTI also develops custom benchmarks to determine performance on specific applications The benchmarks contain DSP routines such as FIR filter, IIR filter, FFT, dot-product, and Viterbi decoder.

MediaBench

The MediaBench benchmark suite consists of several applications belonging to the image processing, communications and DSP applications. Examples of applications that are included are JPEG, MPEG, GSM, G.721 Voice compression, Ghostscript, ADPCM , etc. JPEG is the compression program for images, MPEG involves encoding/decoding for video transmission, Ghostscript is an interpreter for the Postscript language, and ADPCM is Adaptive differential pulse code modulation. The MediaBench is an academic effort to assemble several media processing related benchmarks. An example of the use of these benchmarks may be found in [49].

4.3 Java Benchmarks

SPECjvm98 The SPECjvm98 suite consists of a set of programs intended to evaluate performance for the combined hardware (CPU, cache, memory, and other platform-specific performance) and software aspects (efficiency of JVM, the JIT compiler, and OS implementations) of the JVM client platform [31]. The SPECjvm98 uses common computing features such as integer and floating point operations, library calls and I/O, but does not include AWT (window), networking, and graphics. Each benchmark can be run with three different input sizes referred to as S1, S10 and S100. The 7 programs are compression/decompression (compress), expert system (jess), database (db), Java compiler (javac), mpeg3 decoder (mpegaudio), raytracer (mtrt) and a parser (jack).

SPECjbb2000 (Java Business Benchmark) is SPEC's first benchmark for evaluating the performance of server-side Java. The benchmark emulates an electronic commerce workload in a 3-tier system. The benchmark contains business logic and object manipulation, primarily representing the activities of the middle tier in an actual business server. It models a wholesale company with warehouses serving a number of districts. Customers initiate a set of operations such as placing new orders and checking the status of existing orders. It is written is Java, adapting a portable business oriented benchmark called pBOB written by IBM. Although it is a benchmark that emulates business transactions, it is very different from the Transaction Processing Council (TPC) benchmarks. There are no actual clients, but they are replaced by driver threads. Similarly, there is no actual database access. Data is stored as binary trees of objects.

The CaffeineMark 2.5 is the latest in the series of CaffeineMark benchmarks. The benchmark suite analyses Java system performance in eleven different areas, nine of which can be run directly over the internet. It is almost the industry standard Java benchmark. The CaffeineMark can be used for comparing appletviewers, interpreters and JIT compilers from different vendors. The CaffeineMark benchmarks can also be used as a measure of Java applet/application performance across platforms.

VolanoMark is a pure Java server benchmark with long-lasting network connections and high thread counts. It can be divided into two parts: server and client, although they are provided in one package. It is based on a commercial chat server application, the VolanoChat which is used in several countries world-wide. The server accepts connections from the chat client. The chat client simulates many chat rooms and many users in each chat room. The client continuously sends messages to the server and waits for the server to broadcast the messages to the users in the same chat room. VolanoMark creates two threads for each client connection. VolanoMark can be used to test both speed and scalability of a system. In speed test, it is run in an iterative fashion on a single machine. In scalability test, the server and client are run on separate machines with high speed network connection.

SciMark, see CPU Benchmarks, section 4.1

Java Grande Forum Benchmarks, see CPU Benchmarks, section 4.1

4.4 Transaction Processing Benchmarks

The Transaction Processing Council (TPC) is a non-profit corporation founded in 1988 to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. The term transaction is often applied to a wide variety of business and computer functions. Looked at it as a computer function, a transaction could refer to a set of operations including disk read/writes, operating system calls, or some form of data transfer from one subsystem to another. TPC regards a transaction as it is commonly understood in the business world: a commercial exchange of goods, services, or money. A typical transaction, as defined by the TPC, would include the updating to a database system for such things as inventory control (goods), airline reservations (services), or banking (money). In these environments, a number of customers or service representatives input and manage their transactions via a terminal or desktop computer connected to a database. Typically, the TPC produces benchmarks that measure transaction processing (TP) and database (DB) performance in terms of how many transactions a given system and database can perform per unit of time, e.g., transactions per second or transactions per minute. The TPC benchmarks can be classified into 2 categories, On Line Transaction Processing (OLTP) and Decision Support Systems (DSS). OLTP systems are used in day-to-day business operations (airline reservations, banks), and are characterized by large number of clients who continually access and update small portions of the database through short running transactions. Decision support systems are primarily used for business analysis purposes, to understand business trends, and for guiding future business directions. Information from the OLTP side of the business is periodically fed into the DSS database and analyzed. DSS workloads are characterized by long running queries that are primarily read-only and may span a large fraction of the database. There are four benchmarks that are active, TPC-C, TPC-W, TPC-R and TPC-H. These benchmarks can be run with different data sizes, or scale factors. In the smallest case (or scale factor =1), the data size is approximately 1 GB. The earlier TPC benchmarks, namely TPC-A, TPC-B, and TPC-D have become obsolete.

TPC-C

TPC-C is an OLTP benchmark. It simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered around the principal activities (transactions) of a business similar to that of a world-wide wholesale supplier. The transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment, but, rather represents any industry that must manage, sell, or distribute a product or service. TPC-C involves a mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution. There are multiple on-line terminal sessions. The benchmark can be configured to use any commercial database system such as Oracle, DB2 (IBM) or Informix. Significant disk input and output are involved. The databases consist of many tables with a wide variety of sizes, attributes, and relationships. The queries result in contention on data accesses and updates. TPC-C performance is measured in new-order transactions per minute. The primary metrics are the transaction rate (tpmC) and price per transaction ($/tpmC).

TPC-H

The TPC Benchmark™H (TPC-H) is a decision support system (DSS) benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark is modeled after decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. There are 22 queries in the benchmark. The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric (QphH@Size), and the TPC-H Price/Performance metric, $/QphH@Size. One may not perform optimizations based on apriori knowledge of queries in TPC-H.

TPC-R

The TPC Benchmark™R (TPC-R) is a decision support benchmark similar to TPC-H, but which allows additional optimizations based on advance knowledge of the queries. It consists of a suite of business oriented queries and concurrent data modifications. As in TPC-H, there are 22 queries. The performance metric reported by TPC-R is called the TPC-R Composite Query-per-Hour Performance Metric (QphR@Size), and the TPC-R Price/Performance metric, $/QphR@Size.

TPC-W

TPC Benchmark™ W (TPC-W) is a transactional web benchmark. The workload simulates the activities of a business oriented transactional web server in an electronic commerce environment. It supports many of the features of the TPC-C benchmark and has several additional features related to dynamic page generation with database access and updates. Multiple on-line browser sessions and on-line transaction processing are supported. Contention on data accesses and updates are modeled. The performance metric reported by TPC-W is the number of web interactions processed per second (WIPS). Multiple web interactions are used to simulate the activity of a retail store, and each interaction is subject to a response time constraint. Different profiles can be simulated by varying the ratio of browsing and buying i.e. simulating customers who are primarily browsing and those who are primarily shopping.

4.5 Web server Benchmarks

SPECweb99 is the SPEC benchmark for evaluating the performance of World Wide Web Servers. It measures a system's ability to act as a web server. The initial effort from SPEC in this direction was SPECweb96, but it contained only static workloads, meaning that the requests were for simply downloading web pages that do not involve any computation. But if one examines the use of the web, it is clear that many downloads involve computation to generate the information the client is requesting. Such web pages are referred to as dynamic web pages. SPECweb99 includes dynamic web pages. The file accesses are made to closely match today's real-world web server access patterns. The pages also contain dynamic ad rotation using cookies and table lookups.

VolanoMark See Java Benchmarks, section 4.3.

TPC-W See Transaction Processing Benchmarks, section 4.4

4.6 E-commerce benchmarks – see SPECjbb2000 in Java Benchmarks (section 4.3) and TPC-W in Transaction processing benchmarks (section 4.4)

4.7 Mail server benchmarks

SPECmail2001 is a standardized mail server benchmark designed to measure a system's ability to act as a mail server servicing email requests. The benchmark characterizes throughput and response time of a mail server system under test with realistic network connections, disk storage, and client workloads. The benchmark focuses on the ISP as opposed to Enterprise class of mail servers, with an overall user count in the range of approximately 10,000 to 1,000,000 users. The goal is to enable objective comparisons of mail server products.

4.8 File Server Benchmarks

System File Server Version 2.0 (SFS 2.0) is SPEC's benchmark for measuring NFS (Network File System) file server performance across different vendor platforms. It contains a workload that was developed based on a survey of more than 1,000 file servers in different application environments.

4.9 PC Benchmarks

A variety of benchmarks are available, primarily from Ziff Davis, and Bapco to benchmark the Windows based personal computer. Table 6 lists the most common PC benchmarks. Ziff Davis Winstone and Bapco SYSMARK are benchmarks that measure overall performance while the other benchmarks are intended to measure performance of one subsystem such as video or audio or one aspect such as power.

Table 6. Popular personal computer benchmarks

Benchmark	**Description**
Business Winstone [44]	A system-level, application-based benchmark that measures a PC's overall performance when running today's top-selling Windows-based 32-bit applications. It runs real 32-bit business applications through a series of scripted activities and uses the time a PC takes to complete those activities to produce its performance scores. The suite includes five Microsoft Office 2000 applications (Access, Excel, FrontPage, PowerPoint, and Word), Microsoft Project 98, Lotus Notes R5, NicoMak WinZip, Norton AntiVirus, and Netscape Communicator.
WinBench99 [44]	A subsystem-level benchmark that measures the performance of a PC's graphics, disk, and video subsystems in a Windows environment.
3DwinBench [44]	Tests the bus used to carry information between the graphics adapter and the processor subsystem. Hardware graphics adapters, drivers, and enhancing technologies such as MMX/SSE are tested.
CD WinBench99 [44]	Measures the performance of a PC's CD-ROM subsystem, which includes the CD drive, controller, and driver, and the system processor
Audio WinBench 99 [44]	Measures the performance of a PC's audio subsystem, which includes the sound card and its driver, the processor, the DirectSound and DirectSound 3D software, and the speakers.
Battery Mark [44]	Measures battery life on notebook computers.
I-bench [44]	A comprehensive, cross-platform benchmark that tests the performance and capability of Web clients. The benchmark provides a series of tests that measure both how well the client handles features and the degree to which network access speed affects performance.
Web Bench [44]	Measures Web server software performance by running different Web server packages on the same server hardware or by running a given Web server package on different hardware platforms.
NetBench [44]	A portable benchmark program that measures how well a file server handles file I/O requests from clients. NetBench reports throughput and client response time measurements.
3Dmark MAX 99 [45]	From Futuremark Corporation. Is a nice 3D Benchmark which measures 3D gaming performance. Results are dependent on CPU, memory architecture, and the 3D Accelerator employed.
SYSMARK [43]	Measures a system's real-world performance when running typical business applications. This benchmark suite comprises the retail versions of eight application programs and measures the speed with which the system under test executes pre-determined scripts of user tasks typically performed when using these applications. The performance times of the individual applications are weighted and combined into both category-based performance scores as well as a single overall score. The application programs employed by SYSmark 32 are: Microsoft Word 7.0 and Lotus WordPro 96 for word processing, Microsoft Excel 7.0 (for spreadsheet), Borland Paradox 7.0 (for database), CorelDraw 6.0 (for desktop graphics), Lotus Freelance Graphics 96 and Microsoft Powerpoint 7.0 (for desktop presentation) and Adobe Pagemaker 6.0 (for desktop publishing).

Techniques and tools for performance evaluation improve year by year. For instance, performance monitoring counters were not available to the public until 1997. Benchmarks get updated almost every year. Those interested in experimental performance evaluation should continuously monitor the state-of-the-art. Table 7 provides sources for the benchmarks described in this article. The references at the end can provide new information on tools and benchmarks. Microprocessor vendors are inclined to show off their products in the best light, to projecting results for benchmarks that run well on their system, developing special optimizations within their compilers just for the sake of improving benchmark scores, and stretching the benchmark's behavior while staying within the 'legal' limits of the benchmark guidelines. It is extremely important to understand benchmarks, their features and metrics used for performance evaluation to really understand the performance results.

Table 7. Benchmark Web sites

Example Benchmark Suite	**Web site for more information**
SPEC CPU 2000	http://www.spec.org
Java Grande Forum Benchmarks	http://www.epcc.ed.ac.uk/javagrande/
SciMark	http://math.nist.gov/scimark2
ASCI	http://www.llnl.gov/asci_benchmarks/asci/asci_code_list.html
NASPAR	http://www.nas.nasa.gov/Software/NPB/
MediaBench	http://www.cs.ucla.edu/~leec/mediabench/
EEMBC benchmarks	http://www.eembc.org
BDTI benchmarks	http://www.bdti.com/
SPECjvm98	http://www.spec.org
CaffeineMark	http://www.pendragon-software.com/pendragon/cm3
SPECjBB2000	http://www.spec.org
VolanoMark	http://www.volano.com/benchmarks.html
TPC-C	http://www.tpc.org
TPC-W	http://www.tpc.org
TPC-H	http://www.tpc.org
TPC-R	http://www.tpc.org
SPEC web99	http://www.spec.org
SPECmail2000	http://www.spec.org
SPEC SFS 2.0	http://www.spec.org
SYSMARK	http://www.bapco.com/
Ziff Davis Benchmarks	http://www.zdnet.com/etestinglabs/filters/benchmarks
3DMarkMAX99	http://www.pcbenchmarks.com

References

[1] Reinhold P. Weicker, An Overview of Common Benchmarks, IEEE Computer, December 1990, pp. 65-75

[2] H. Cragon, Computer Architecture and Implementation, Cambridge University Press, 2000

[3] J. E. Smith, Characterizing Computer Performance with a Single Number, Communications of the ACM, October 1988.

[4] Patterson and Hennessy, Computer Architecture: The Hardware/Software Approach, by Hennessy and Patterson, Morgan Kaufman Publishers, 2nd edition, 1998, ISBN 1558604286

[5] Vtune profiling software, http://developer.intel.com/software/products/vtune/vtune_oview.htm

[6] P6perf utility, http://developer.intel.com/vtune/p6perf/index.htm

[7] DCPI Tool home page, http://www.research.digital.com/SRC/dcpi/) and http://www.research.compaq.com/SRC/dcpi/

[8] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Chrysos, “Profile Me: Hardware Support for Instruction Level Profiling on Out of Order Processors”, MICRO-30 proceedings, 1997, pp. 292-302.

[9] Perf-monitor for UltraSparc, http://www.sics.se/~mch/perf-monitor/index.html

[10] PMON http://www.ece.utexas.edu/projects/ece/lca/pmon

[11] M. C. Merten, A. R. Trick, E. M. Nystrom, R. D. Barnes, and W. W. Hwu, "A hardware-driven profiling scheme for identifying hot spots to support runtime optimization", Proceedings of the 26th International Symposium on Computer Architecture, pp. 136-147, May 1999.

[12] R. Bhargava, J. Rubio, S. Kannan, L. K. John, D. Christie, and L. Klaes, "Understanding the Impact of x86/NT Computing on Microarchitecture", Book Chapter in Characterization of Contemporary Workloads, pages 203- 228, Kluwer Academic Publishers, 2001, ISBN 0-7923-7315-4

[13] Ali Poursepanj and David Christie, "Generation of 3D Graphics Workload for System Performance Analysis", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[14] A. Agarwal, R. L. Sites and M. Horowitz, "ATUM: A New Technique for Capturing Address Traces Using Microcode"" Proceedings of the 13th International Symposium on Computer Architecture, June 1986, pp. 119-127.

[15] B. Cmelik and D. Keppel, "Shade: A Fast instruction-set simulator for execution profiling", Chapter 2 in "Fast Simulation of Computer Architectures", by T. M. Conte and C. E. Gimarc, Kluwer Academic Publishers, 1995.

[16] Dinero IV cache simulator, www.cs.wisc.edu/~markhill/DineroIV

[17] P. Bose and T. M. Conte, Performance Analysis and Its Impact on Design, IEEE Computer, May 1998, pp. 41-49

[18] P. Crowley and J-L Baer, "On the Use of Trace Sampling for Architectural Studies of Desktop Applications", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, ISBN 0-7695-0450-7, edited by John and Maynard, IEEE CS Press, 1999, pp. 15-24.

[19] J. R. Larus, Efficient Program Tracing, IEEE Computer, May 1993, pp. 52-61

[20] Ravi Bhargava, Lizy K. John, and Francisco Matus, Accurately Modeling Speculative Instruction Fetching in Trace-Driven Simulation, Proceedings of the IEEE Performance, Computers and Communications Conference (IPCCC), Feb. 1999, pp. 65-71.

[21] The Simplescalar simulator suite, http://www.simplescalar.org or http://www.cs.wisc.edu/~mscalar/simplescalar.html

[22] B. Boothe, Execution Driven Simulation of Shared Memory Multiprocessors", Chapter 6 in "Fast Simulation of Computer Architectures", by T. M. Conte and C. E. Gimarc, Kluwer Academic Publishers, 1995.

[23] The SimOS complete system simulator, http://simos.stanford.edu/

[24] SIMICS www.simics.com

[25] SIMICS, VIRTUTECH http://www.virtutech.com

[26] L. Kurian, Performance Evaluation of Prioritized Multiple-Bus Multiprocessor Systems, M. S. Thesis, University of Texas at El Paso, Dec 1989.

[27] L. K. John, Yu-cheng Liu, A Performance Model for Prioritized Multiple-Bus Multiprocessor Systems, IEEE Transactions on Computers, Vol.45, No.5, pp.580-588, May 1996.

[28] P. Heidelberger and S. S. Lavenberg, Computer Performance Evaluation Methodology, IEEE Transactions on Computers, Dec 1984, pp. 1195-1220

[29] D. B. Noonburg and J. P. Shen, A Framework for Statistical Modeling of Superscalar Processor Performance, Proceedings of the 3rd International Symposium on High Performance Computer Architecture (HPCA), 1997pp. 298-309.

[30] D. J. Sorin, V. S. Pai, S. V. Adve, M. K. Vernon, and D. A. Wood, "Analytic Evaluation of Shared Memory Systems with ILP Processors, " Proceedings of the International Symposium on Computer Architecture, 1998, pp. 380-391.

[31] SPEC Benchmarks, www.spec.org

[32] Java Grande Benchmarks, http://www.epcc.ed.ac.uk/javagrande/

[33] SciMark, http://math.nist.gov/scimark2

[34] ASCI Benchmarks, http://www.llnl.gov/asci_benchmarks/asci/asci_code_list.html

[35] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations”, Proceedings of the 22nd International Symposium on Computer Architecture, pages 24-36, June 1995.

[36] NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/

[37] MediaBench benchmarks, http://www.cs.ucla.edu/~leec/mediabench/

[38] EEMBC, www.eembc.org

[39] BDTI, http://www.bdti.com/

[40] The Caffeine benchmarks, http://www.pendragon-software.com/pendragon/cm3

[41] VolanoMark, http://www.volano.com/benchmarks.html

[42] Transactions Processing Council, www.tpc.org

[43] SYSMARK, http://www.bapco.com/

[44] Ziff Davis Benchmarks, www.zdbop.com or www.zdnet.com/etestinglabs/filters/benchmarks

[45] PC Benchmarks, www.pcbenchmarks.com

[46] The Jaba profiling tool, http://www.ece.utexas.edu/projects/ece/lca/jaba.html

[47] R. Radhakrishnan, J. Rubio and L. K. John, Characterization of Java Applications at Bytecode and Ultra-SPARC Machine Code Levels, Proceedings of IEEE International Conference on Computer Design, pp. 281-284

[48] J. A. Mathew, P. D. Coddington, and K. A. Hawick, Analysis and Development of the Java Grande Benchmarks, Proceedings of the ACM 1999 Java Grande Conference, June 1999

[49] C. Lee, M. Potkonjak, W. H. M. Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems, Proceedings of the 30th International Symposium on Microarchitecture, pp. 330-335

[50] D. Bhandarkar and J. Ding, “Performance Characterization of the Pentium Pro Processor”, Proceedings of the 3rd High Performance Computer Architecture Symposium, 1997, pp. 288-297.

[51] Ted Romer, Geoff Voelker, Dennis Lee, Alec Wolman, Wayne Wong, Hank Levy, Brian Bershad and Brad Chen. "Instrumentation and Optimization of Win32/Intel Executables Using Etch", USENIX, 1997.

[52] T. M. Conte and C. E. Gimarc, “Fast Simulation of Computer Architectures”, Kluwer Academic Publishers, 1995.