Name: Kehinde Oluwatoyin Martha
Matric no: 16/SCI01/023
Course code: CSC 410

Based on the perspectives of the operational laws you have studied, discuss the possibility of achieving an accurate performance report using the three major evaluation techniques.

Solution
The three Major performance evaluation techniques are:
1. Performance measurement techniques
2. Analytic performance modeling
3. Simulation performance modeling

Based on performance measurement techniques

- Microprocessor 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.

- Off-chip hardware monitoring: 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.

- Logic Analyzers: Poursepanj and Christie 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.

- Software Monitoring: Monitoring software observes and tracks the operations and activities of users, applications and network services on a computer or enterprise systems. This type of software provides a way to supervise the overall processes that are performed on a computing system, and provides reporting services to the system or network administrator. 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.

- Micro-coded 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.

Based on Analytic performance 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.

Based on simulation performance modeling

- 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 an 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. 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.

- 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. Mis-predicted branches can be accurately simulated as well. Thus, these simulators solve the two major problems faced by trace-driven simulators. The widely used Simple-scalar simulator 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.

- 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 9 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, a popular complete system simulator provides a simple pipelined processor model and an aggressive superscalar processor model. SimOS and SIMICS can simulate uniprocessor and multiprocessor systems.

- 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

- 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.