

# Performance Evaluation

J. Kelly Flanagan  
Computer Science Department  
Brigham Young University

## Outline

- ❑ Introduction
- ❑ Current projects and preliminary results
- ❑ General techniques
- ❑ Ideal and alternative techniques
- ❑ A new hybrid approach
- ❑ Future Directions
- ❑ Summary

## Introduction

### □ What do we do in the Performance Evaluation Lab?

- ➔ We identify and evaluate techniques for collecting data from existing systems and use the collected data to improve the performance of future designs.

### □ Who are we?

- ➔ Kelly Flanagan
- ➔ Jun Su
- ➔ Xiao-Hong Tu
- ➔ Chulkee Sung
- ➔ Niki Crockett
- ➔ Add your name here if you work hard and cost little! -- this is the definition of a grad student!

## Current Projects

- ❑ Power efficient memory hierarchy design
  - ➡ cache memory
  - ➡ main memory -- DRAM, EDO DRAM, etc.
  - ➡ disk drives
  
- ❑ Increasing the disk I/O performance in a Netware environment
  - ➡ disk rearrangement
  - ➡ prefetching
  - ➡ caching
  
- ❑ Evaluating and characterizing the network traffic in a Netware environment
  
- ❑ Generating accurate instruction traces from address traces

## General Techniques

- ❑ Measure existing systems that are made up of hardware and software components:
  - ☞ hardware
    - ✓ CPU
    - ✓ cache memory
    - ✓ main memory
    - ✓ disk I/O
  - ☞ applications
  - ☞ operating systems
  - ☞ compilers
  
- ❑ Identify architectural and implementation changes that may improve performance.
  
- ❑ Simulate or implement changes
  
- ❑ Measure impact and iterate

## Ideal Technique

- ❑ The best way to measure the impact of an architectural or implementation change is to construct two systems that differ by a single modification and measure the performance of both systems and compare the results.
  
- ❑ What makes this impractical?
  - ➡ if the systems are complex or expensive (they always are) or the design space one wishes to investigate is large (it always is).
  
  - ➡ How do we measure the performance of the systems
  
  - ➡ How do we know our comparisons are valid
  
- ❑ Examples
  - ➡ Low power memory hierarchy
  
  - ➡ Improving I/O performance for Netware

## Alternative Evaluation Techniques

### ❑ Back-of-an-envelope

- ➔ Some system designers have a pretty good feel for their profession and can guess what impact changes can have on a system. On the other hand, things change.

### ❑ Analytical models

- ➔ Queueing theory and other techniques can be used to develop mathematical or statistical models of computer systems.

### ❑ Simulation

- ➔ Programs representing various system components can be written with the desired amount of accuracy and used to determine the impact of changes.

## Simulation

- ❑ Using simulation, if we wanted to investigate the memory hierarchy performance of a given system we simply write programs that emulate the behavior of the memory hierarchy components:
  - ➡ CPU
  - ➡ all levels of cache
  - ➡ main memory
  - ➡ disk drive, etc.
  
- ❑ The user compiles code and executes it on the simulated CPU and uses the references produced to exercise the caches, memory, and disks.

## What's Wrong?

- ❑ Writing a simulator that emulates an entire system accurately is difficult.
- ❑ Making an accurate simulator that is flexible is even harder.
- ❑ The most difficult portion of the simulation previously described is producing a simulator of a CPU and executing code on it.
  - ➡ compile an OS -- UNIX
  - ➡ boot this OS on the simulator (this will be fast)
  - ➡ compile some interesting application on this platform and run it.
  - ➡ use the output of this mechanism to drive the cache and memory hierarchy of interest.
  - ➡ this process is tough and extremely slow!!!

## Trace-Driven Simulation -- TDS

- ❑ To solve these problems we use trace-driven simulation.
  - ➡ take an existing system that is similar to the system you would like to study
  - ➡ find some way to collect the stream of references generated by this processor while running an application of interest
  - ➡ use this collected data in your simulation to represent the CPU
  
- ❑ This eliminates the need to model the CPU and enables complex simulations to be performed in a timely manner.
  
- ❑ This technique is applicable in many situations and is used here to replace a CPU only as an example.

## Trace Collection Techniques

- ❑ Inlining -- software instrumentation added to programs that output desired information (10X dilation)
- ❑ Microcode modification -- microcode is added to microroutines that generate addresses (10X dilation)
- ❑ Single stepping -- interrupts are used to get instruction execution patterns (100X dilation)
- ❑ Processor simulation -- a simulator of the entire system is performed (1000X+ dilation)
- ❑ Hardware monitors -- collect signals from hardware (1X dilation)
- ❑ Hybrid techniques -- discussed later

## Hardware Monitors

### □ Promise to provide distortion free traces, BUT

- ☞ small trace buffers limit trace length
  - ✓ they are small because of density, power, and cost
- ☞ internal CPU caches filter what is seen by the hardware
  - ✓ if we turn them off then the processor no longer acts the same
- ☞ what is seen by the hardware is implementation specific
  - ✓ timing of events
  - ✓ prefetching activity
  - ✓ speculative execution
  - ✓ the rest of the system
- ☞ always one step behind in realization technology
  - ✓ my logic analyzer is too slow -- ALWAYS

## A Hybrid Approach

### □ The goals are

- ➔ collect accurate data from state-of-the-art processors with no dilation
- ➔ collect data with internal caches enabled
- ➔ collect data that contains no implementation specific information
- ➔ compress data for distribution
- ➔ do all of this using a processor simulator so any information of interest can be collected

### □ The following approach requires three items:

- ➔ an address trace as previously described
- ➔ a processor simulator
- ➔ several programs to translate data from one form to another

## Address Trace Collection

- ❑ An accurate address trace is necessary.
- ❑ The address trace may be collected anywhere in the memory hierarchy, but all caches above the collection point must be invalidated at the beginning of the trace collection process.
- ❑ A device driver of some sort must write the initial state of the processor to the address trace.
- ❑ The address trace should contain address, data, and control information and some timing indicator such as the number of instructions or branches executed.
- ❑ In addition, any hints that can be provided may make the task of generating instruction traces simpler and more accurate.

# Address Traces

Address, Data, Control, Timing

---

caches are initialized

CPU state is written

---

instruction fetches

data loads / stores

Interrupt event -- # instructions

data loads / stores

instruction fetches

DMA event -- # instructions

instruction fetches

data loads / stores

I/O event -- # instructions

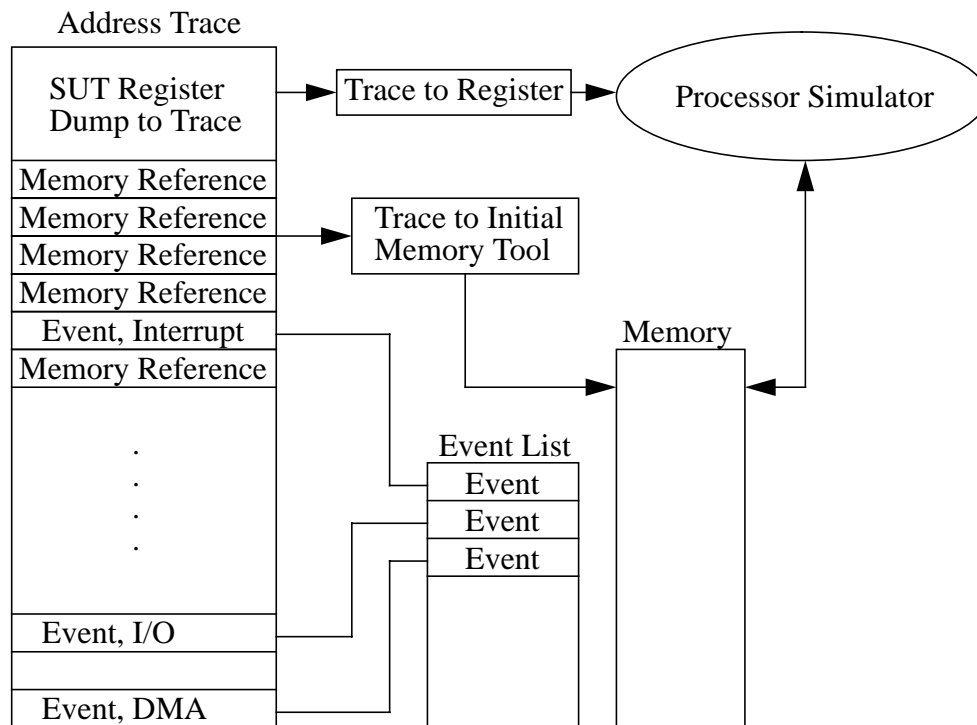
## Address Trace Processing

- ❑ The initial state of the processor can be extracted from the address trace.
  
- ❑ The initial (needed) state of main memory can be extracted from the address trace:
  - ✓ The address is used to perform a lookup in a simulated memory image
  - ✓ If this location has not been initialized place the data associated with this address into the simulated memory image
  - ✓ Repeat this process until the address trace is exhausted or the memory is completely initialized
  
- ❑ From the address trace I/O, DMA, and interrupt requests can be identified and placed on an event list. The timing of these events can be determined from the time information included in the trace data.

## Processor Simulator

- ❑ A processor simulator of the target CPU must be available or created. It is not necessary to model the entire system of interest.
- ❑ This simulator must be accurate at the instruction level. It need not simulate implementation specific details such as pipelining, caches, TLBs, etc.
- ❑ The simulator must be modified to accept the initial state, initial memory image, and event list generated in the previous step.

# Putting It All Together



## Generating Instruction Traces

- ❑ Collect a complete address trace as described.
- ❑ Using special tools generate an initial processor state, initial memory image, and an event list from the collected address trace.
- ❑ Initialize the processor simulator with the initial state generated in the previous step.
- ❑ The processor simulator fetches an instruction from the initial memory image and executes it.
- ❑ The processor simulator checks the event list to determine if it is time for the first event. If not execution continues otherwise the event is initiated and then execution continues.
- ❑ The simulator outputs desired information.

## How Good Is It?

- ❑ This process results in implementation independent instruction traces instead of implementation dependent address traces.
  - ➡ no prefetching activity
  - ➡ no timing due to system under test
  
- ❑ Instruction traces can be used with a pipeline model to simulate future designs or make future design choices.
  - ➡ examples will be presented shortly
  
- ❑ These traces can be used to simulate processor internals.
  
- ❑ The output traces are not cache filtered.
  
- ❑ This process yields significant data compression.

## Speculative Execution

- All modern processors are or will use speculative execution to improve performance.
  - ☞ Each time we execute a branch in a program we can guess which way the program flow will go before we know the outcome of the branch.
  - ☞ If we guess correctly we can be executing many of the instructions down that path before the branch outcome is determined.
    - ✓ In a superscalar processor this may be tens of instructions.
  - ☞ If we are wrong we have changed the state of the processor in an incorrect manner and all of these changes must be unrolled.
  - ☞ The key is guessing correctly which can be done about 80 to 90 percent of the time.

## Speculative Execution and Caches

- ❑ What happens to cache performance if we use speculative execution?
- ❑ If the processor speculatively executes down the wrong path we bring many instructions into the cache that would not normally be fetched.
- ❑ This may remove useful information fetched earlier that will be used again.
- ❑ The newly fetched information may be a complete waste of cache space or it may be used in the near future when this path is actually taken.
- ❑ Will cache performance suffer or be improved?  
Can we quantify the impact?

## Speculative Execution and Caches

- ❑ Trace gathering techniques that rely on software have a difficult time obtaining speculatively executed references since they are run on nonspeculative machines.
- ❑ Hardware monitors can collect this data if the machine already exists. If the machine exists it would not be very interesting to study!
- ❑ The hybrid technique can collect data from a current machine and generate an instruction trace.
- ❑ This trace and the addition of the initial memory image can be read and executed by a simulator of a machine that does speculative execution and an address trace can be generated and fed to a cache simulator.

## Future Directions

- ❑ Instruction level parallelism of important workloads:
  - ➡ multiprocessing workloads
  - ➡ database workloads (OLTP, etc.)
  
- ❑ Energy efficient CPU design
  - ➡ VLIW, superscalar, pipelined, etc.
  - ➡ internal cache configurations
  - ➡ energy efficient instruction set design
  
- ❑ Trace collection on a parallel machine
  - ➡ This technique may have some promise for collecting traces on one parallel machine and saying something about the design of another.

## Summary

- ❑ We are developing new techniques to monitor current systems to learn what should be done in the future.
  
- ❑ We are system oriented, concerned with the performance of the entire system
  - ✓ applications
  - ✓ compilers
  - ✓ operating systems
  - ✓ hardware -- architecture and implementation
  
- ❑ We are currently pushing forward in four areas:
  - ✓ low power memory hierarchy design
  - ✓ improving the disk I/O performance of a Novell Netware server
  - ✓ improving network I/O simulation models
  - ✓ hybrid trace collection techniques described in this presentation