Microprocessor Simulators:
Fundamentals, Techniques and Validation

Roger Espasa
Dept. Arquitectura de Computadors
Universitat Politècnica de Catalunya
Abril-Juny 2000

© Roger Espasa, 2000

Goals

- **Understand the fundamentals of processor simulators**
- **Understand that all simulators are incorrect**
  - Inherent modeling error always present
  - Error margin must be known to interpret simulator outputs
  - User must choose an 'acceptable' error margin
- **Learn techniques to validate simulators**
  - Establish the error margin
  - Uncover bugs in ***your*** simulator

© Roger Espasa, 2000

Basic Bibliography

- **IEEE Micro, May/June 1999**
  - "*Challenges in Processor Modeling and Validation*",
    Pradip Bose, Thomas M. Conte, and Todd M. Austin
  - "*Environment for PowerPC Microarchitecture Exploration*"
    Mayan Moudgill, John-David Wellman, and Jaime H. Moreno
  - "*An Integrated Functional Performance Simulator*"
    C. Bechem, J. Combs, N. Utamaphethai, B. Black, R.D. Shawn Blanton, and J.P. Shen
  - "*Using Term Rewriting Systems to Design and Verify Processors*"
    Arvind and Xiaowei Shen
  - "*Verifying the FM9801 Microarchitecture*"
    Warren A. Hunt, Jr., and Jun Sawada
  - "*Performance Analysis and Verification of the picoJava Processor*"
    Sudheendra Hangal and Mike O'Connor

© Roger Espasa, 2000

Basic Bibliography

- **IEEE Computer, May 1998**
  - "*Performance Analysis and Its Impact on Design*",
    Pradip Bose, Thomas M. Conte
  - "*Performance Simulation of an Alpha Microprocessor*"
    Matt Reilly, John Edmondson
  - "*Calibration of Microprocessor Performance Models*"
    Bryan Black, John Paul Shen

© Roger Espasa, 2000

Other Resources

- **Conferences and Workshops**
  - Performance Analysis and Its Influence on Design (PAID)
  - Intl. Workshop on Microprocessor Test and Verification
  - Intl. High Level Design Validation and Test Workshop
- **John Paul Shen's Group at ece.cmu.edu**
  - Several Papers by Utamaphethai, Blanton and Shen:
    - "*Validation of Speculative and Out-of-order Execution Microarchitecture*"
    - "*Superscalar Processor Validation at the Microarchitecture Level*"
    - "*A Buffer-Oriented Methodology for Microarchitecture Validation*"
    - …

© Roger Espasa, 2000

Course Outline

- **Fundamentals of Microprocessor Simulators**
- **Developing a Simulator**
- **Validating a Performance Simulator**
- **Performance/Event Counters**

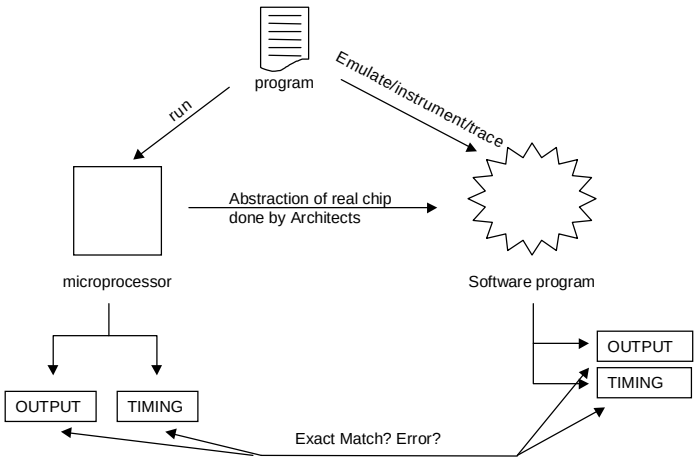
© Roger Espasa, 2000

Part 1: Fundamentals of Microprocessor Simulators

© Roger Espasa, 2000

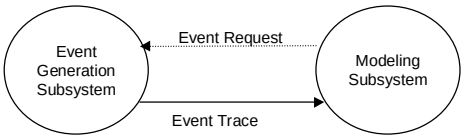
Simulator

- **A <u>software model</u> of the functional and timing behavior of a microprocessor**



© Roger Espasa, 2000

Simulator Components



- **A) Event generation subsystem (feeder)**
  - Generates the sequence of events used to feed the model
- **B) Interface between subsystems**
- **C) Modeling subsystem (simulator)**
  - Consumes the sequence of events and produces some statistic:
    - Execution cycles
    - cache hit rate
    - Coherency messages
    - Packet frequency
    - ….

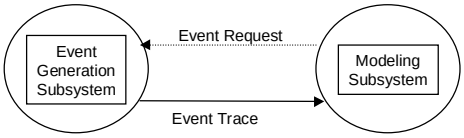
© Roger Espasa, 2000

A) Event Generation Subsystem

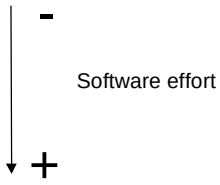
- **Typical goal for microprocessor simulations**
  - Generation of instruction trace from
    - A standalone program (user level trace)
    - A program + OS interaction (user+system trace)
    - A full system (user+system+interrupts generated by IO)
- **Other types of simulations**
  - Networking
    - Gather packets traveling on a network
    - Requests arriving at a router
    - TCP/IP stack
  - IO
    - Requests made to an IO device (disk/tape/graphic card)
    - Traffic on a PCI bus

© Roger Espasa, 2000

Event Generation Techniques



- **An instruction "trace" can be generated using**
  - Hardware probes
  - Microcode modification
  - Step-by-step execution
  - Program instrumentation
  - ISA emulation



© Roger Espasa, 2000

1. Hardware Probes

- **Connect a logic analyzer to the microprocessor pins**
  - Gather the information at microprocessor speed
  - Buffer information gathered in some external memory area
  - Dump buffer periodically to disk/tape
- **Pros**
  - Very detailed
  - Covers both user and system instructions/memory requests
  - No slowdown!
- **Cons**
  - Can only analyze events that "reach" the external pins
    - i.e., the L1 cache events are not "visible" externally
    - Today's machines have "too much" on chip
  - Can only provide physical addresses
  - Not portable at all
- **Example: Analysis of VAX-11/780 [Cla83]**

© Roger Espasa, 2000

2. Microcode Modification

- **Only useful if microprocessor has microcode**
- **Example: VAX-8200, ATUM [ASH86]**
  - Microcode modified to gather address traces as program runs
  - Microcode would make a copy of every address generated in a reserved memory area.
- **PROS**
  - Good slowdown (only around 20)
  - Captures user+system traces
- **CONS**
  - Tailored to each machine
  - Requires microcode (not very useful nowadays)

© Roger Espasa, 2000

3. Step-by-Step execution

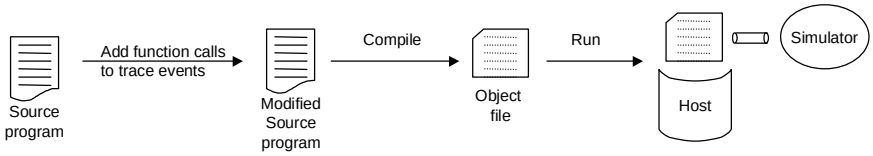
- **Most Unix implementations provide a "trace" bit**
  - Program executes 1 instruction and returns control to OS
  - Used by debuggers to trace program execution
  - OS offers interface to query the state of the traced program
- **PROS**
  - Flexible, portable, low cost
- **CONS**
  - Can only provide user level traces
  - No speculative execution trace possible
  - Very large slowdown (100-10.000) [Hol91]

© Roger Espasa, 2000

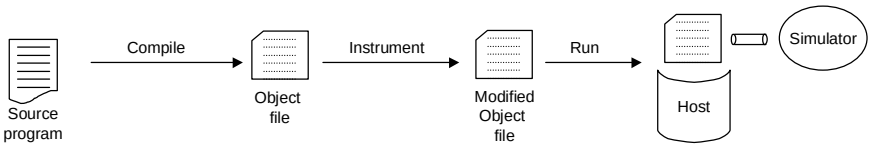
4. Program Instrumentation

- **Insert {instructions, function calls} in a program to**
  - Record those events interesting to the user

- **Source-code level (High Level or Assembler)**

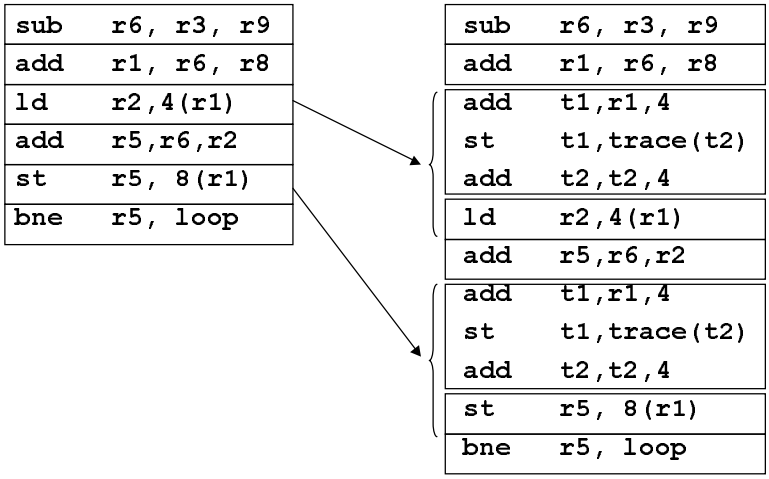


- **Object/Binary file level**



© Roger Espasa, 2000

Object Level Instrumentation



© Roger Espasa, 2000

Drawbacks of Instrumentation

- **Perturbations to original program**
  - Instruction positions are changed
    - Could affect branch prediction if 'pc' locations change
    - Easily compensated
  - If text area grows by more than a page…
    - Heap/Stack areas could be displaced
    - Data addresses change
  - A jump with a very long displacement might cause trouble
    - If text area grows, maybe an indirect jump is needed
- **No speculative execution trace**

© Roger Espasa, 2000

ATOM

- **Instrumentation of Alpha binaries**
- **Unlike most other instrumentation tools, ATOM provides a lot of flexibility on:**
  - What 'events' are to be traced
    - Function entry/exit
    - Instructions
    - Memory references
    - Branches
  - How exactly the instrumentation code has to be placed
    - Before/after
- **Gives access to all register values**

© Roger Espasa, 2000

Some Instrumentation Tools

Level	Name	Processor	Interface	Slowdown
Source Code	TRAPEDS	IPSC/2	Direct Call	20-30
	MPtrace	I386	File + processing	1000+
	AE	MIPS,SPARC	File	20-65
	Tangolite	MIPS	Mem. buffer	45
Object File	Epoxie	Titan	Global buffer	8-12
	Epoxie2	R3000	Global buffer	15
	ATOM	Alpha	Direct call	6-13
Binary File	Pixie	MIPS	File / Pipe	10
	Goblin	RS/6000	Direct Call	20
	IDTrace	I486	File / Pipe	12
	Qpt	MIPS,SPARC	File + processing	10-60

© Roger Espasa, 2000

5. ISA Emulation

- **ISA = Instruction Set Architecture**
  - The programmer visible state of a given microprocessor
  - Usually does not change across different implementations
    - Instructions may be added,
    - Usually, instructions are not removed

- **ISA examples**
  - Alpha (Compaq)
  - Mips (Mips)
  - IA32 and IA64 (Intel)
  - PowerPC (IBM, Motorola, Apple)
  - PA-RISC (HP)
  - Sparc V9 (Sun)

© Roger Espasa, 2000

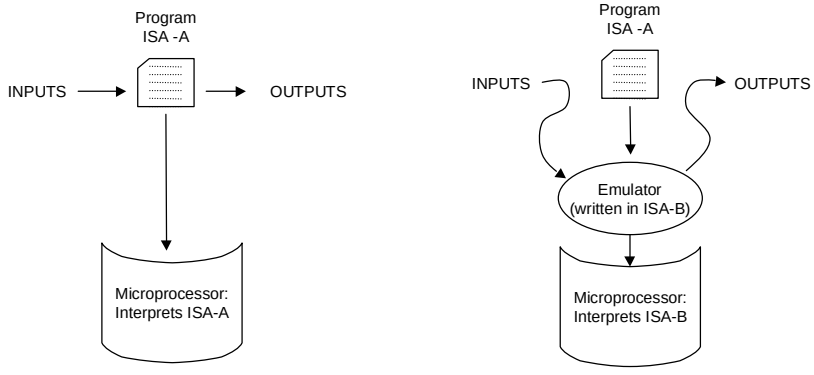
Emulation Concepts

- **A microprocessor is an emulator of a given ISA**
- **The microprocessor…**
  - accepts programs written in the ISA
  - Starts executing at a given PC in the program
  - Fetches, Decodes and executes one instruction at a time
- **A software program that performs the same tasks is called an emulator.**

© Roger Espasa, 2000

Software Emulators

- **An emulator executes/interprets programs written in ISA-A using a (possibly different) ISA-B**



© Roger Espasa, 2000

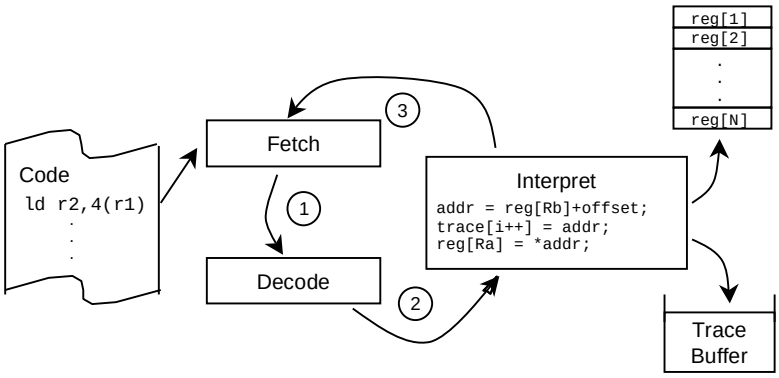
Emulator Skeleton

```
PC = get_entry_point();
While ( 1 ) {
 Service_Routine = decode_instruction(PC);
 PC = (*Service_Routine)(PC);
}
```

```
FUNC *Decode_instruction(UINT64 PC)
{
 opcode = text[PC] >> 26;
 switch ( opcode ) {
   case ADD: return &emulated_add();
   case SUB: return &emulated_sub();
   case MUL: return &emulated_mul();
   …
   case LOAD: return &emulated_load();
 }
}
```

© Roger Espasa, 2000

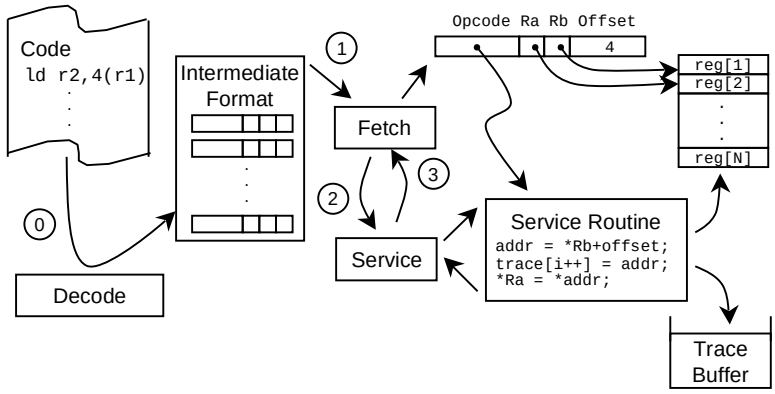
Iterative Interpretation



- **Mable [DLH94]**

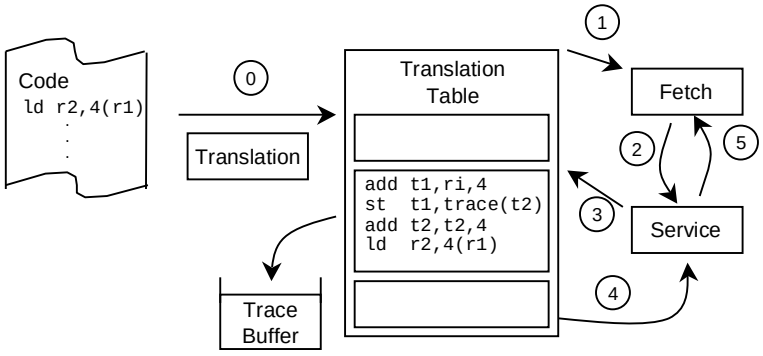
© Roger Espasa, 2000

Predecoding of Instructions



© Roger Espasa, 2000

Dynamic Translation



© Roger Espasa, 2000

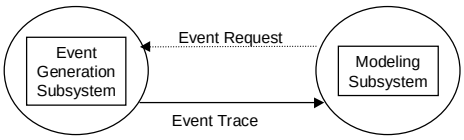
Emulators: Pros and Cons

- **PRO**
  - Can provide speculative (wrong-path) instructions
  - Can provide correct register values down wrong-path
  - Can be used to help validate semantic correctness of simulator (MINT/AINT)

- **CON**
  - Speed
    - If cross-emulating, speed will be a concern
    - On same architecture, the hardware can be used to speed up instruction execution
  - OS calls
    - Trivial if target and host environments are same
    - Very tricky for cross-emulation

© Roger Espasa, 2000

B) Interface between subsystems



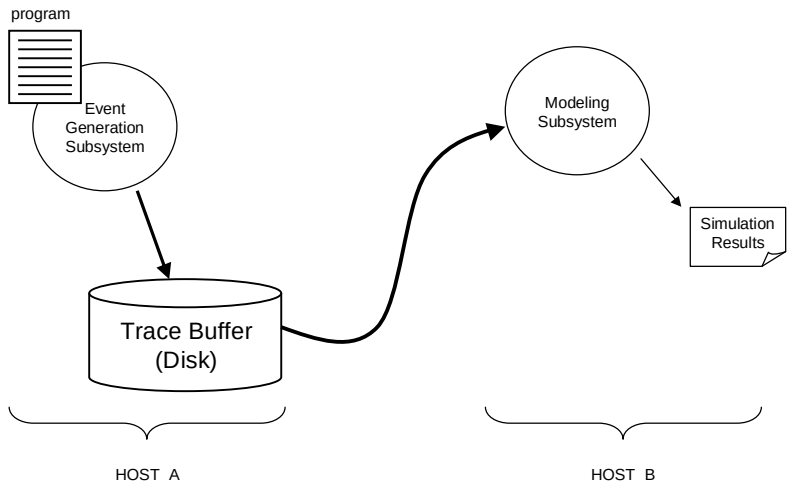
- **Important tradeoff between**
  - Cost of generating the trace
  - Cost of storing the trace
  - Cost/Type of communication

- **Three main styles**
  - Trace-driven simulation
  - Execution-driven simulation
  - Emulation-driven simulation

© Roger Espasa, 2000

Trace-Driven Simulation

Step 1: Generate trace **Step 2: Consume trace**

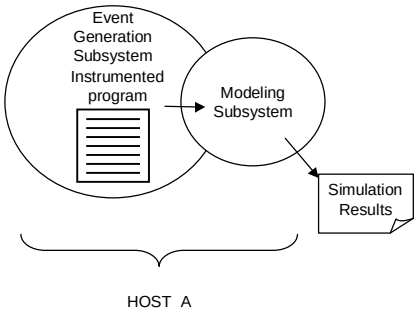


© Roger Espasa, 2000

Execution-Driven Simulation (1)

Step 1: Instrument Program

Step 2: Run Program, Generate Trace on-the-fly, Simulate



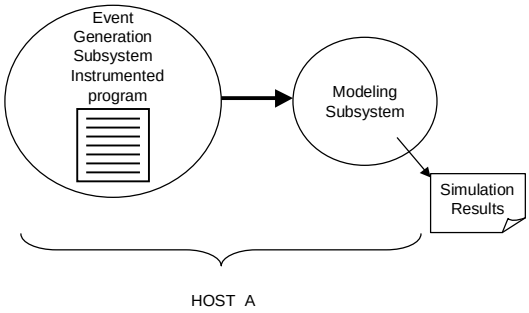
© Roger Espasa, 2000

✓ Direct Call Interface (Single Process)

Execution-Driven Simulation (2)

Step 1: Instrument Program

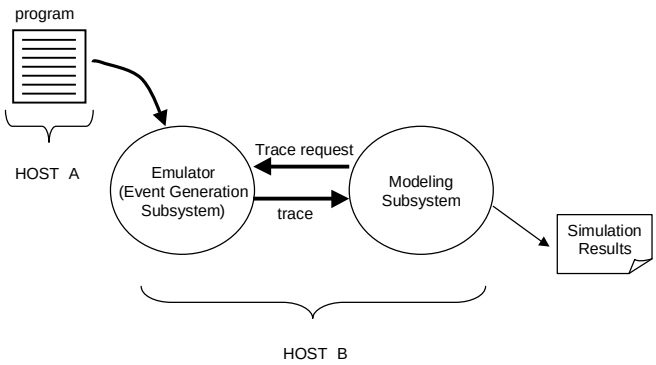
Step 2: Run Program, Generate Trace on-the-fly, Simulate



✓Multiple Processes using pipes/shared mem.

© Roger Espasa, 2000

Emulation-Driven Simulation



© Roger Espasa, 2000

Emulation-Driven Simulation

- **Interface can be either**
  - Direct Call
  - Using pipes/shared memory
- **The Modeling Subsystem DRIVES the simulations**
  - Speculative execution IS POSSIBLE!
- **The modeling subsystem**
  - Requests an instruction at a particular PC
  - The emulator must be able to detect it is going down a wrong path
  - The emulator must be able to recover from wrong path

© Roger Espasa, 2000

Brief List of Simulation Tools

- **ATOM [SE94]: Instruments alpha binaries and provides an execution-driven environment for simulation.**
- **G88 [Bed90]: Motorola 88000 simulator. Accepts user and system traces. Dynamically translates instructions to the native host (68000, 88000 or Sparc)**
- **MINT [VF94]: Simulator for memory references in a multiprocessing environment. Emulates using a hybrid between interpretation and dynamic translation**
- **Pixie [Smi91]: Instrumentation tool that works for Mips and Alpha binaries. Generates instruction and data reference traces**
- **Shade [CK94]: Emulator that heavily uses dynamic translation techniques with a cache for translated instructions. Accepts as inputs Sparc and Mips binaries and runs on Sparc.**
- **SimOS [RHW96]: Simulator for a full computer (including processor, memory, IO devices and OS). Runs on Alpha, Mips.**

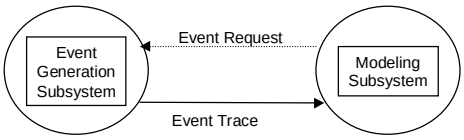
© Roger Espasa, 2000

SimpleScalar [BAB96]

- **Emulation-driven simulation with direct call interface**
- **Accepts two ISAs (v3.0)**
  - PISA (Portable ISA, a MIPS derivative)
  - Alpha
- **Runs on multiple platforms (x86, Sparc, Alpha, Mips)**
- **Provides gcc compiler suite that generates PISA code**
- **Models a very simplistic OoO superscalar**
  - 5-stage pipeline
  - Reorder Buffer
  - Idealized L1 cache

© Roger Espasa, 2000

C) Modeling Subsystem



- **The topic for part (2) of this course**
- **Some examples**
  - Cache/Memory system simulator
  - Processor pipeline simulator
- **Major types**
  - Cycle-by-Cycle vs. Event-List
  - Structural vs. Procedural

© Roger Espasa, 2000

Part 2:
Developing a Simulator

© Roger Espasa, 2000

Psychological Challenges

- **Common Misbelieves:**
  - "A *true* Ph.D. student in Computer Architecture must have his/her own simulator!"
  - "Simulator XXX is really baaad! I'll do my own, which, of course, will be waaaay better, since I am the best programmer in the world"
  - "I can have a simulator ready in 3 weeks"
  - "Simulator XXX is used in every paper I read… Therefore it must be a good simulator"
  - "As soon as I'm done with this paper I''ll forget about this simulator"
  - "MY SIMULATOR DOES NOT HAVE BUGS!" ← Worst of All!

© Roger Espasa, 2000

Simulators last many years

- **More years than you might believe today**
- **At least, as long as your thesis (>5 years)**
- **And, if your thesis is any good, it will outlast you!**

- **Therefore, start obsessing today about…**
  - Code Readability
  - Code Quality
  - Software Reuse
  - Version Control
  - TESTING

- **Take some time to choose your simulation technology**

© Roger Espasa, 2000

Software Engineering Challenges

- **Large number of parameters**
- **Code readability**
- **Code extensibility**
- **Hardware is parallel, software is not!**
- **Matching simulation results and software version**
- **Deterministic behavior (reproducibility)**
- **Modeling Style**
  - Structural vs. Procedural
- **Implementation Style**
  - Cycle-by-Cycle vs. Event-List
  - Implicit time flow vs. Explicit time flow

© Roger Espasa, 2000

Large Number of Parameters

- **Even a modest simulator has many parameters to tune**
- **Run-time or compile-time?**
  - Run-time requires a variable per parameter
    - Usually easier to code
    - Single binary used across all simulations
    - SLOWER
  - Compile-time requires either
    - A really ugly command line
    - A pre-processor to “specialize” the simulator
    - FASTER
- **Behavior changes substantially across parameter values**
- **Verification…**
  - For which parameter set?
  - Coverage Analysis should be an important tool

© Roger Espasa, 2000

Code Readability

- **Most important attribute of a long-lasting program?**
  - READABILITY!!!!
  - Code must be easy to read
  - Code must easily “show” the hardware being modeled
- **“What-if” experiments can destroy your simulator**
  - What if I add 2 more ports to my cache?
  - What if I have a second level TLB?
  - …. (and countless more) …
- **Lot of effort goes into abstracting and re-coding**
  - Don’t be afraid to re-implement a piece of your simulator to make it look nicer
- **If you ever share your simulator in the future**
  - You do not want to be embarrassed
  - You want your work to be useful to other people

© Roger Espasa, 2000

Code Extensibility

- **"What-if" experiments can destroy your simulator**
- **How can one extend the simulator…**
  - From in-order to out-of-order
  - From non-multithreaded to multithreaded
  - Adding a detailed memory system
- **… without loosing its previous code?**
  - Very difficult
  - Requires lot of software abstraction
  - CASE tools would be useful (but we don't know them)
- **How should we "deprecate" pieces of code?**
  - Old models that are no longer interesting
  - Experiments that ended up making no sense

© Roger Espasa, 2000

Hardware is parallel, software is not!

- **CAM structures are very common in HW**
  - Instruction Queue wakeup & select
  - TLB lookup
  - Cache lookup
  - Killing instructions in flight
- **Hardware CAMs are parallel!**
- **Software is not parallel**
  - Carefully choose CAM simulation algorithms
    - Fast-but-complicated better than slow-but-simple
  - Invest in sophisticated data structures
    - It will pay off in simulation time

© Roger Espasa, 2000

Matching Results and Simulator

- **Four months after a paper has been accepted**
  - Will you be able to reproduce the experiments?
  - Will you be able to simply read the simulator code you used?
- **Make sure the simulator output file**
  - Has a meaningful name
  - Has a date and time stamp
  - Has each and every parameter value listed
  - Identifies each and every source file used in the simulator
    - List every source file and its RCS id string
    - If a source file is not up-to-date,
      - mark output file as DANGEROUS
- **Use some versioning software (CVS)**

© Roger Espasa, 2000

Deterministic Behavior

- **Same simulator and same input**
  - Should produce exactly the same output file
    - Exact instruction count
    - Exact number of cache misses
    - Exact cycle count
  - Regardless of
    - Host where simulation was run (abrell vs. kemet)
    - Day when simulation was run (Monday vs. Tuesday)
    - OS version (Tru64 4.0D vs. 4.0F)
- **Yet, this is very difficult to achieve**
  - In trace-driven simulators: relatively easy (for user-level trace)
  - In execution-driven simulators: almost impossible
  - In emulation-driven simulators: very hard, but not impossible

© Roger Espasa, 2000

Modeling and Implementation Style

- **Modeling Style**
  - Structural vs. Procedural
- **Implementation Style**
  - How does time flow?
  - Where does time flow?
- **Style chosen has impact on final accuracy**
  - Avoid cheating in your simulator
    - Decode should not be allowed to “peek” at the Queue Internals
      - I.e. communication can NOT happen faster than speed-of-light
  - Each style has advantages and drawbacks
    - Speed vs Accuracy
    - Unfortunately, the classical “procedural,cycle-by-cycle” style allows too much cheating

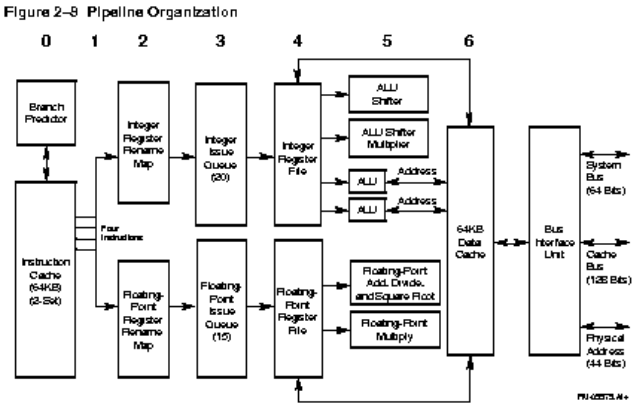
© Roger Espasa, 2000

Procedural Modeling Style

- **Typically, single file for most pipeline stages**
- **A procedure corresponds to a pipe stage**
- **Passing information between pipe stages is**
  - “ad-hoc”
  - Difficult to modify
- **As programmers, we tend to unify similar things**
  - Same code for FP and INT queue
  - Yet, they are seldom the same!
    - If we care about accuracy, we will pay a high price
- **Difficult to extend machine with new pipe stages**

© Roger Espasa, 2000

21264 Pipe Diagram



© Roger Espasa, 2000

An example of Procedural Style

```
main()
{
 while ( end_of_sim == 0 ) {
  do_commit();
  do_memory();
  do_issue();
  do_decode();
  do_fetch();
  now++;
 }
}
```

```
do_decode()
{
 if ( iqueue.full ) return;

 insn = . . . decode instruction . . .

 iqueue.data[iqueue.idx++] = insn;
 if ( iqueue.idx == iqueue.max ) {
  iqueue.full = TRUE;
 }

 . . .
```

© Roger Espasa, 2000

Comments

- **Procedures help in cheating because**
  - Can modify each others data structures
  - Can look at the state of far away data structures
- **Adding a pipe stage is difficult**
  - Either need a new function
    - Performance suffers
  - Either you can add time stamps in the ‘iqueue’

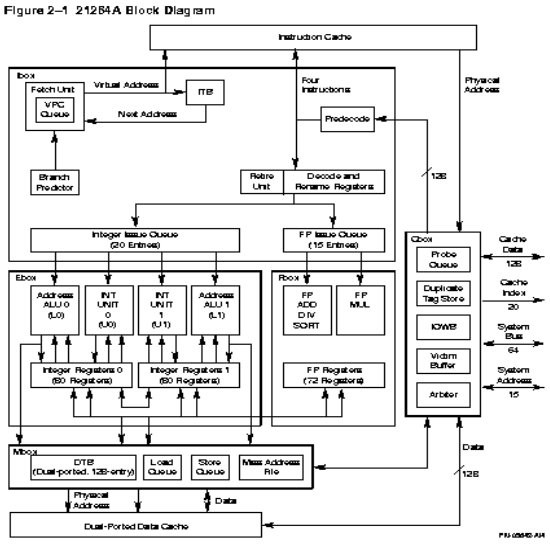
© Roger Espasa, 2000

Structural Modeling Style

- **A procedure/file unit closely matches a hardware structure**
- **A software box is isolated from other boxes**
  - As much as possible
  - Defines clear input/output interface
  - Inputs/outputs are “models” of real hardware buses/wires
  - A box CAN NOT look into other boxes code
- **Resembles a “block diagram”**
- **Better for**
  - Modeling BW bottlenecks
  - Adding new features
  - Accuracy (more real)

© Roger Espasa, 2000

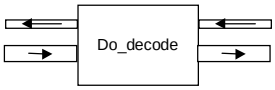
21264 Block Diagram



© Roger Espasa, 2000

An example of Structural Style

```
main()
{
 while ( end_of_sim == 0 ) {
  do_commit();
  do_memory();
  do_issue();
  do_decode();
  do_fetch();
  now++;
 }
}
```




```
do_decode()
{
 get_signal_val(IQFull,&iqf);
 while ( get_signal_val(Icache, &insn[I]) > 0) {
  nin++
 }

 if ( iqf ) set_signal_val(DecFull,TRUE);
 else {
  . . . Decode insn[i] for i=0..N  . . .
  set_signal_val(DecodeOut[i], insn[i])
 }
```

© Roger Espasa, 2000

Comments

- **do_decode is independent of all other code**
- **“Signal” interface improves**
  - Modularity
  - Adding pipe stages
  - Replacing this decode routine by a different one

© Roger Espasa, 2000

Implementation Style

- **Two major decisions:**
  - How does time flow?
  - Where does time flow?

© Roger Espasa, 2000

How does time flow?

- **In Increments of 1 cycle (a.k.a. Cycle-by-Cycle)**
  - Most common in microprocessor simulators
- **In Increments of N cycles (a.k.a. Event-List)**
  - N varies
  - VHDL and logic simulators

© Roger Espasa, 2000

Cycle-by-Cycle

```
main()
{
 while ( end_of_sim == 0 ) {
  do_commit();
  do_memory();
  do_issue();
  do_decode();
  do_fetch();
  now++;
 }
}
```

© Roger Espasa, 2000

Event-Driven

- **Central event list ordered by "time of event"**
- **When simulator wants to perform an action**
  - Creates a new event
  - Sets a particular time for that event
  - Enqueues (ordered) in the event list
- **Simulator core is a loop that advances from event to event**
- **Time can "jump" in increments of N cycles depending on distance to next event**
- **Not very common in pipeline simulations**
  - Because there is always an event in next cycle

© Roger Espasa, 2000

Where does time flow?

- **Three major abstractions**
  - Implicit through variables (Simplescalar)
  - Explicit through Communication Links (ASF)
  - Event Queue (for event-based simulators)

© Roger Espasa, 2000

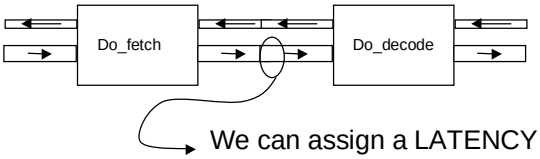
Implicit Time flow

```
main()
{
 while ( end_of_sim == 0 ) {
  do_commit();
  do_memory();
  do_issue();
  do_decode();
  do_fetch();
  now++;
 }
}
```

```
do_decode()
{
 if ( iqueue.full ) return;

 insn = . . . decode instruction . . .

 iqueue.data[iqueue.idx++] = insn;
 if ( iqueue.idx == iqueue.max ) {
  iqueue.full = TRUE;
 }

 . . .
```

© Roger Espasa, 2000

Explicit Time Flow



We can assign a LATENCY to this bus
We can also assign a BANDWIDTH to it

```
do_decode()
{
 get_signal_val(IQFull,&iqf);
 while ( get_signal_val(Icache, &insn[I]) > 0) {
  nin++
 }

 if ( iqf ) set_signal_val(DecFull,TRUE);
 else {
  . . . Decode insn[i] for i=0..N  . . .
  set_signal_val(DecodeOut[i], insn[i])
 }
```

We do not know when these instructions will reach their destination (and its none of our business)

© Roger Espasa, 2000

Lab Session 1:
A “Wire” Package

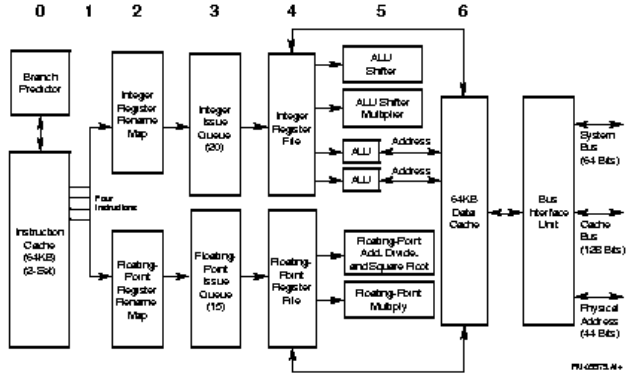
© Roger Espasa, 2000

Lab Session 2:
Structural Model of 21264

© Roger Espasa, 2000

21264 Pipe Diagram

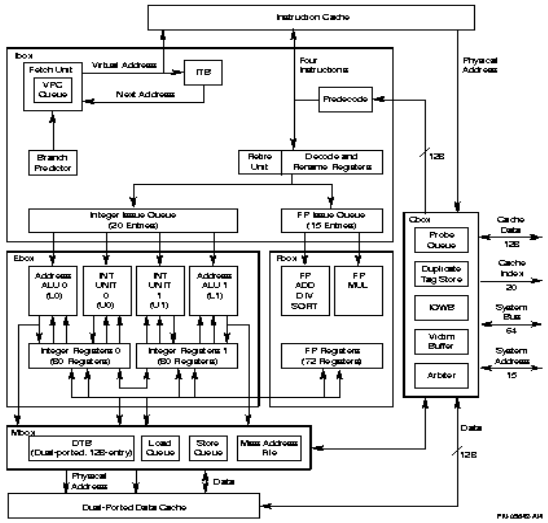
Figure 2–8 Pipeline Organization



© Roger Espasa, 2000

21264 Block Diagram

Figure 2–1 21264A Block Diagram



© Roger Espasa, 2000

Lab Session 3:
Structural Model of 21264
(cont)

© Roger Espasa, 2000

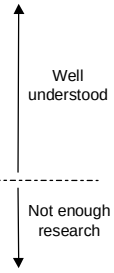
Part 3:
Validating a Performance Simulator

© Roger Espasa, 2000

Microprocessor Validation

- **General Goal**
  - Demonstrate that a given chip truly *implements* the architecture as described in the Reference Manual

- **Many levels of validation**
  - Transistor-level, EM checks, signal integrity, etc…
  - Simplified gate-level validation (0/1 model)
  - Gate-level vs RTL
  - RTL vs functional specification
  - RTL vs performance model
  - Performance model vs what?



© Roger Espasa, 2000

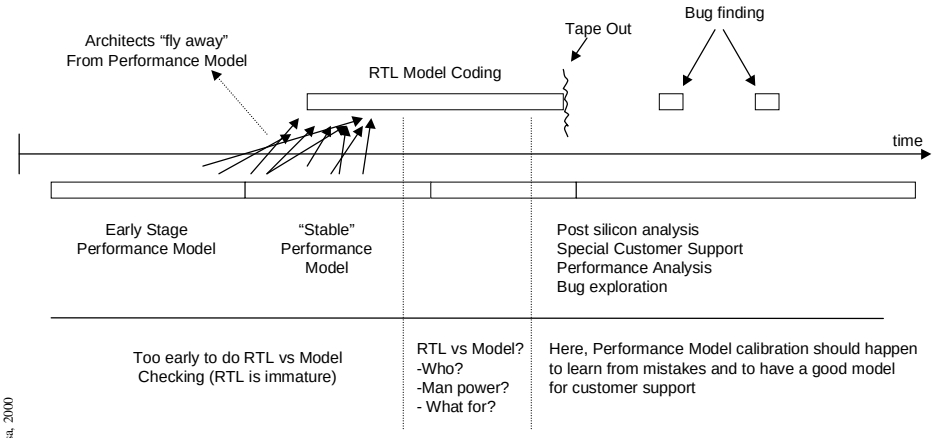
Performance Simulator

- **Initial model of a future architecture**
  - A bunch of architects get together and
    - Develop new ideas
    - Code them in the performance model
    - Evaluate performance on benchmarks
    - Go back to step 1

- **The performance model is a software program**
  - To get these early stage evaluations done
  - Timing is more important than “correctness”
  - The verification team is NOT working with the architects at this stage of the project
    - Probably busy debugging the previous chip

© Roger Espasa, 2000

Performance Model Lifetime



© Roger Espasa, 2000

This course

- **We will focus on validating a <u>Performance</u> Model**
  - Different from validating the RTL/VHDL model
- **Two major aspects must be validated**
  - SEMANTICS
    - Does the performance model implement the architecture correctly?
  - TIMING/PERFORMANCE
    - Does the performance model predict correctly the number of cycles that a given program takes?

© Roger Espasa, 2000

Validating Semantics of a Performance Model

© Roger Espasa, 2000

Validating PM Semantics

- **Usually PM does not get the "royal treatment"**
  - The verification team works on RTL
    - Hardly ever on the PM
  - Although the test suite for RTL is available
    - Not used against PM
    - Most PMs only are an abstraction of timing
      - Test suite would only be useful if PM core-dumps!
      - How can semantics be checked?

© Roger Espasa, 2000

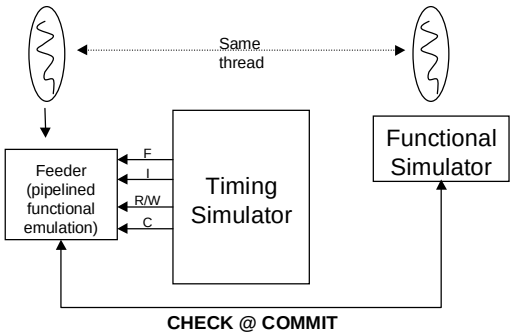
Approaches to Validating Semantics

- **Add some semantic information to timing model**
  - Model reading/writing register values
  - Model reading/writing data caches
  - …
  - Then check against functional simulator
- **Indirectly by measuring "coverage" of the model**
  - Coverage = Given a set of tests, what fraction of all the simulator code is exercised?
  - Assume that high coverage means "few bugs"
- **Model processor as a set of FSM-like buffers**
  - Add checking code to measure coverage of the FSM transitions
  - Use coverage of FSM transitions to drive ATPG tool

© Roger Espasa, 2000

Adding Semantic Actions to PM

- **"Feeder" is MINT-like (an emulator)**
  - Emulation broken into phases
    - Fetch, Issue, Read/Write, Commit
- **If timing model drives feeder incorrectly**
  - Errors will show up when checking against Functional Simulator



© Roger Espasa, 2000

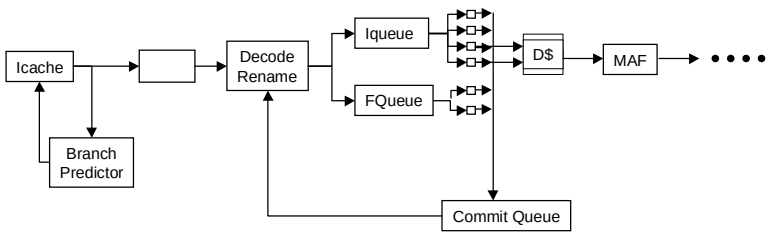
Software Coverage

- **Most compilers have a flag to generate coverage info**
  - Counts number of times each source code line is executed
- **Feed "test programs" to the simulator until 100% coverage is achieved**
  - Tests can come from
    - Popular benchmarks: spec, oracle, specjvm, mpeg, …
    - Random test generator
- **100% coverage doesn't guarantee much…**
  - But, at least, model is fully exercised
  - Useful to get rid of "embarrassing" core dumps ;-)
  - Useful also to check combinations of parameters

© Roger Espasa, 2000

Buffer-Oriented Validation Methodology

- **Proposed by Utamaphethai, Blanton and Shen**
  - Model architecture as set of performance-critical buffers
  - Entries in buffers modeled as FSM



© Roger Espasa, 2000

Model Buffer Entries as FSM

- **An entry usually contains**
  - Data (instruction, value, miss request, etc.)
  - Status Bits (in-flight, completed, pending, etc.)
- **An entry can be viewed as being in one of several states**
  - Status Bits updates correspond to state transitions
  - Entry can be modeled as small FSM
- **Entry behavior can be validated by exercising the FSM model**
  - Check all states can be entered
  - Check all states transitions can be performed
  - Check the FSM behavior exhaustively

© Roger Espasa, 2000

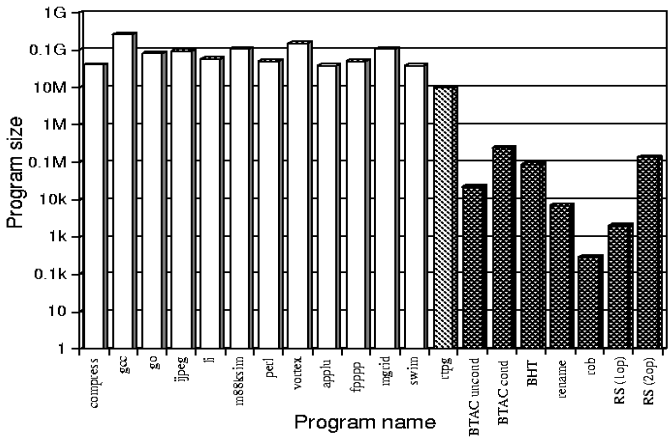
Example

- **From N. Utamaphethai, R.D. Blanton, J.P. Shen "A Buffer-Oriented Methodology for Microarchitecture Validation", In *Journal of Electronic Testing (JETTA),* Vol 16 n. 1-2, pp. 49-65, Feb 2000**

- **Software model of the PowerPC 604**
  - Modeled buffers:
    - BTAC: Branch Target Address Cache
    - BHT: Branch History Table
    - Rename Registers: GRB, FRB, CRB
    - Reservation Stations (1 operand and 2 operand): SFX0, SFX1, CFX, FPU

© Roger Espasa, 2000

Instructions Executed

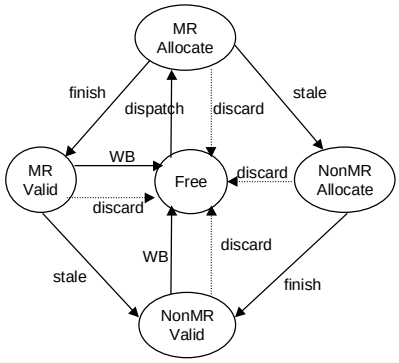


From N. Utamaphethai, R.D. Blanton, J.P. Shen "A Buffer-Oriented Methodology for Microarchitecture Validation", In *Journal of Electronic Testing (JETTA)*, Vol 16 n. 1-2, pp. 49-65, Feb 2000

© Roger Espasa, 2000

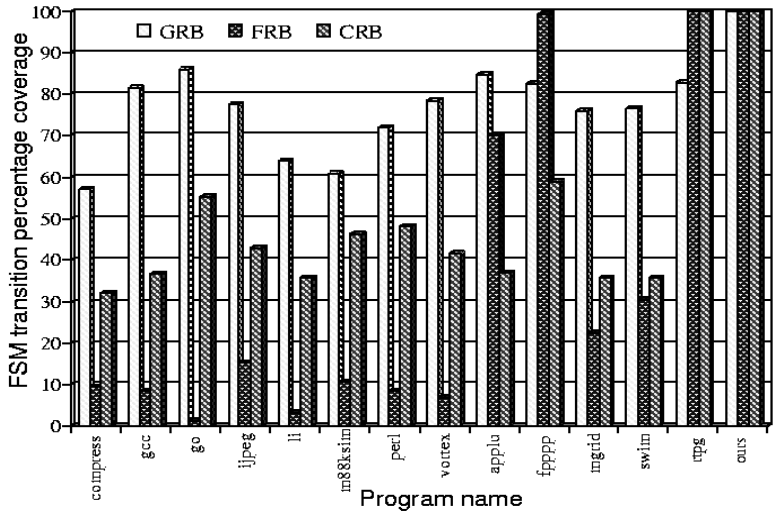
Rename Buffers FSM Model

- **PowerPC rename buffers**
  - MR = most recent value for a given logical register
  - Valid for GRB, FRB and CRB



© Roger Espasa, 2000

Rename Buffers Transition Coverage

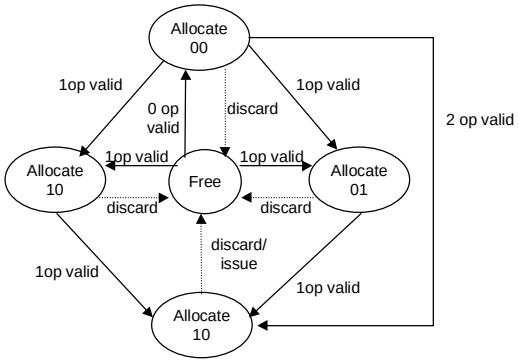


From N. Utamaphethai, R.D. Blanton, J.P. Shen "A Buffer-Oriented Methodology for Microarchitecture Validation", In *Journal of Electronic Testing (JETTA)*, Vol 16 n. 1-2, pp. 49-65, Feb 2000

© Roger Espasa, 2000

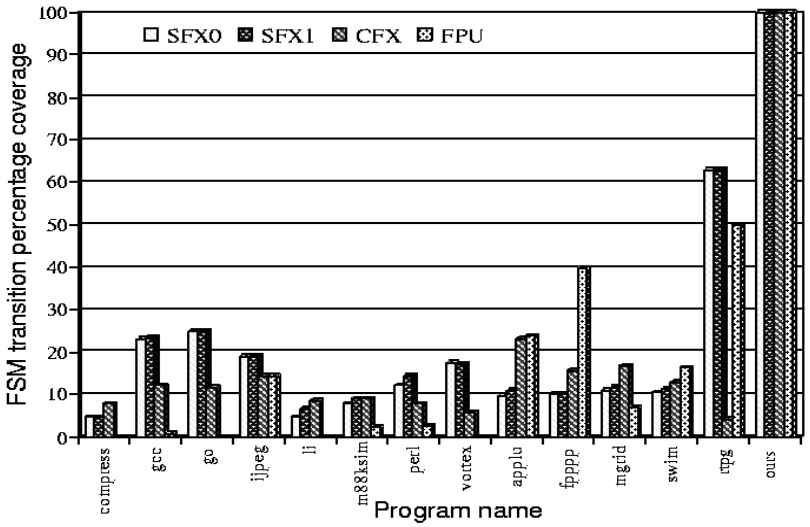
Reservation Station FSM Model

- **PowerPC rename buffers**
  - MR = most recent value for a given logical register
  - Valid for GRB, FRB and CRB



© Roger Espasa, 2000

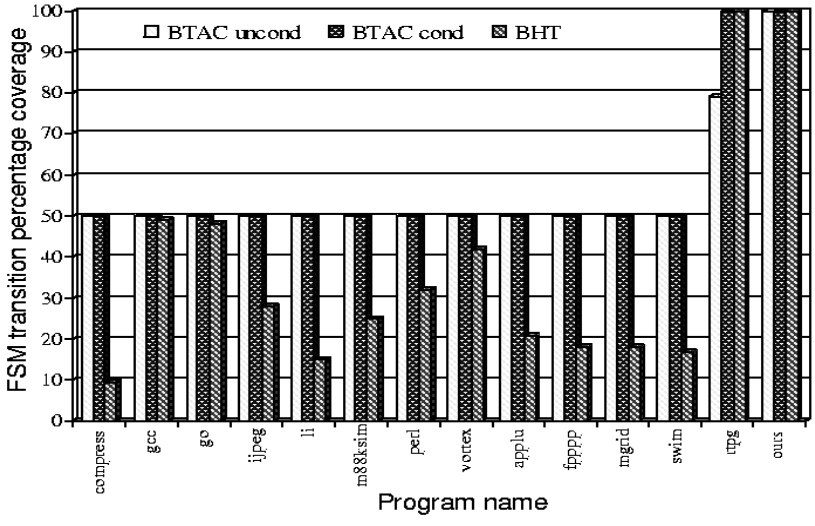
Reservation Station Transition Coverage



From N. Utamaphethai, R.D. Blanton, J.P. Shen "A Buffer-Oriented Methodology for Microarchitecture Validation", In *Journal of Electronic Testing (JETTA)*, Vol 16 n. 1-2, pp. 49-65, Feb 2000

© Roger Espasa, 2000

BTAC Transition Coverage



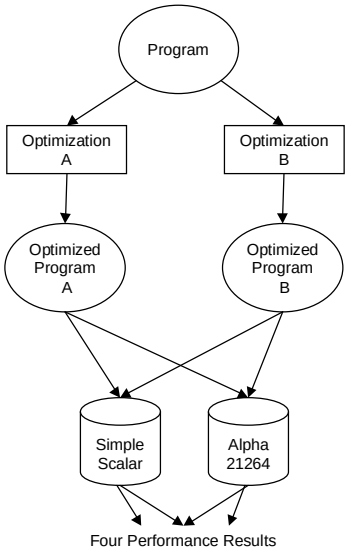
From N. Utamaphethai, R.D. Blanton, J.P. Shen "A Buffer-Oriented Methodology for Microarchitecture Validation", In *Journal of Electronic Testing (JETTA)*, Vol 16 n. 1-2, pp. 49-65, Feb 2000

© Roger Espasa, 2000

Validating Timing of a Performance Model

© Roger Espasa, 2000

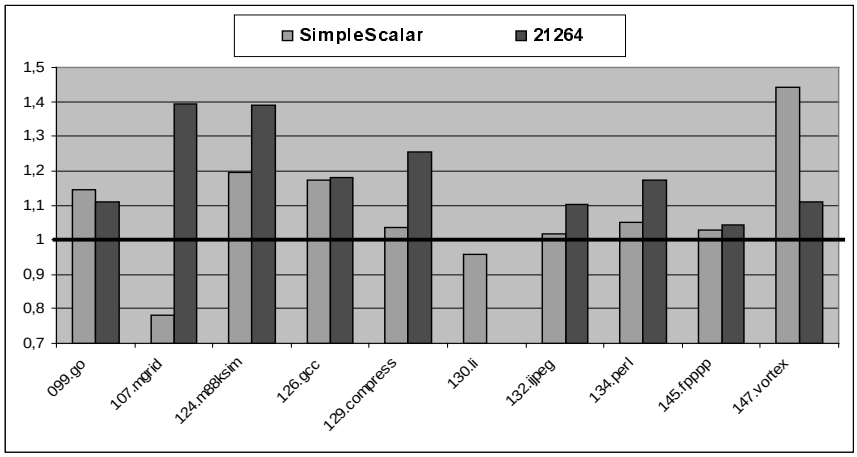
A Simple Experiment…



- **Compare the speedups obtained by optimizations A and B**
  - On SimpleScalar simulator
  - On real Alpha 21264
- **Comparing direct cycle counts**
  - Is difficult
  - Might be unfair
- **Expectation is that**
  - Performance trends will be the same
- **Data from M. Fernández, UPC**

© Roger Espasa, 2000

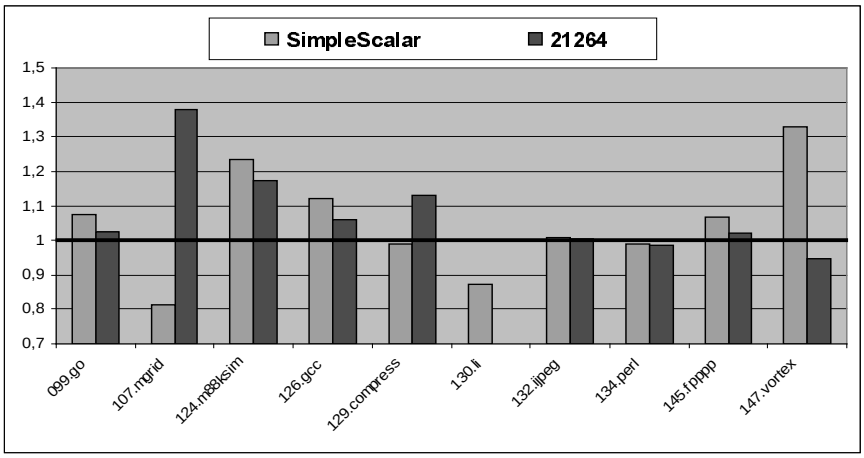
SimpleScalar vs. 21264



Speedup due to Optimization A

© Roger Espasa, 2000

SimpleScalar vs. 21264



Speedup due to Optimization B

© Roger Espasa, 2000

We have a problem!

- **The speedup predictions do not match at all**
- **SimpleScalar both predicts higher and lower speedups**
  - If it always were lower or higher it might be okay…

- **We need tools and tests to fix SimpleScalar**
  - (alternative: quit using SimpleScalar!)

© Roger Espasa, 2000

Validation Levels

- **Timing validation from "easy" to "hard"**
  - Run One instruction through model
  - Run Pairs of instructions through model
  - Run Single Basic Blocks through model
  - Run Small Loops through model
  - Run Multi-Basic-Block test cases through model
  - Run Full Benchmarks through model

- **Does timing match expectations?**

- **Problem: define "expectation"**
- **From P. Bose, "*Performance Test Case Generation for Microprocessors*", IBM Research Report 21052**

© Roger Espasa, 2000

Gold Signature

- **Gold Signature = the reference timing behavior of an instruction or group of instructions when run through the pipeline**

- **Sometimes, computing the Gold Signature**
  - Is as difficult as establishing the correct timing of the model
  - Depends on the validation level
    - Single instruction: “easy” to compute manually
    - Pair of instructions: “medium”
    - Loops: “difficult”
  - Depends on the model maturity
    - Early-stage validation vs. Late-stage validation

© Roger Espasa, 2000

Gold Signature: One Instruction

- **Manually computed by architects**
  - Tabulated and stored in Database
  - Updated when design changes

- **Signature captures basic “transit time”of instruction under different conditions**
  - Perfect Cache vs. Real Cache, etc..

- **Example: fadd**

C:	1	2	3	4	5	6	7	8	9
A:	IF	DE; RR	IS	E1	E2	E3	WR	RE	WB

© Roger Espasa, 2000

Gold Signature: Pairs of Instructions

- **Compute and store in Database**
- **Assume different conditions**
  - Same/next cycle fetch
  - With/without dependences
  - Always hit in L1 and/or TLB
  - Always hit in L2 and/or TLB
  - etc…

C:	N	N+1	N+2	N+3	N+4	N+5	N+6	N+7	N+8	N+9	N+10
fld	IF	DE; RR	EA; DL; TL	CA; AL; WR	RE	WB					
fadd	IF	DE; RR	-	-	IS	E1	E2	E3	WR	RE	WB

© Roger Espasa, 2000

Gold Signature: Single Basic Block

- **Three major steps**
  - Compute gating bandwidth
  - Compute Idealized Bound (I-Bound)
    - Assuming same instruction in BB
    - Assuming different instructions in BB
  - Compute Realistic Bound (R-Bound)
    - Add bubble effects due to dependences
    - Add cache miss effects
    - Add other effects

© Roger Espasa, 2000

Gating Bandwidth

- **A sequence of N instructions consisting of**
  - N_L = number of load instructions
  - N_S = number of store instructions
  - N_F = number of floating point instructions
  - N_{MA} = number of multiply-add instructions
  - $N = N_L + N_S + N_F + N_{MA}$
- **A set of architectural parameters**
  - Fetch_bw
  - Num_load_ports
  - Num_store_ports
  - Dispatch_bandwidth
  - address_gen_bw
  - lsu_issue_bw
  - fpu_issue_bw

© Roger Espasa, 2000

Gating Bandwidth

- **Sustained cycles-per-instruction will be limited by one of**
  - $cpi_{fetch_bound} = \lceil N / fetch_bw \rceil$
  - $cpi_{load_port_bound} = \lceil N_L / Num_load_ports \rceil$
  - $cpi_{store_port_bound} = \lceil N_S / Num_store_ports \rceil$
  - $cpi_{dispatch_bound} = \lceil N / dispatch_bw \rceil$
  - $cpi_{compl_bound} = \lceil N / completion_bw \rceil$
  - $cpi_{agen_bound} = \lceil (N_L + N_S) / address_gen_bw \rceil$
  - $cpi_{lsu_issue_bound} = \lceil (N_L + N_S) / lsu_issue_bw \rceil$
  - $cpi_{fpu_issue_bound} = \lceil (N_F + N_{MA}) / fpu_issue_bw \rceil$
- **Gating bandwidth =**
  - $MAX(cpi_{fetch_bound}, cpi_{load_port_bound,}, cpi_{store_port_bound}, cpi_{dispatch_port_bound}, cpi_{compl_bound}, cpi_{agen_bound}, cpi_{lsu_issue_bound}, cpi_{fpu_issue_bound})$

© Roger Espasa, 2000

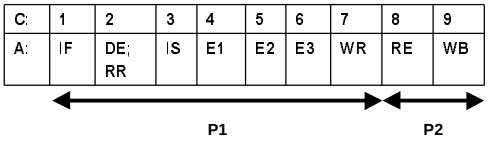
Idealized Bound

- **Assumptions (from [Bose et al.]):**
  - Only FP instructions
  - All units fully pipelined
  - Perfect Branch Overlap (no dispatch across branch)
  - Effectively infinite queues and buffers
  - Perfect Caches
  - No dependences
- **I-Bound for N identical instructions**
- **I-Bound for N (possibly) different instructions**

© Roger Espasa, 2000

I-Bound for N identical instructions

- **p1 = cycles from IF to "Instruction Finish"**
- **p2 = cycles for Instruction Completion and WB**



- **g = gating bandwidth**
- **I-Bound = (p1 – 1) + ⌈N / g⌉ + p2**

© Roger Espasa, 2000

I-Bound for N possibly different instructions

- **I, J are two different instructions**
  - Data-independent
  - Can be issued on same cycle (resource & bandwidth permitting)
  - Gating issue bandwidth is 2
  - P1(J) >= P1(I)
- **Pair Cycle Count = max(p1(I) + p2(J), p1(J) + p2(J))**
- **N repetitions of Pair = p1(J) – 1 + ⌈N / 2⌉ + p2(J)**
- **Can be generalized to most cases**
  - Although expressions can become quite complex

© Roger Espasa, 2000

Realistic Bound

- **Add bubble effects due to dependences**
- **Add cache miss effects**
- **Add other effects**

© Roger Espasa, 2000

R-Bound under data dependences

- **I, J are two different instructions**
  - Can be data dependent (bubble of length b)
  - Can be issued on same cycle (resource & bandwidth permitting)
  - Gating issue bandwidth is 2
  - P1(B) >= P1(A)
- **Pair Cycle Count = max(p1(I) + p2(J), p1(J) + p2(J)) + b**
- **N repetitions of Pair = p1(J) – 1 + b + ⌈N / 2⌉ + p2(J)**
- **Can be generalized to most cases**
  - Although expressions can become quite complex

© Roger Espasa, 2000

Gold Signature: Loops

- **Use techniques just describe for loops**
  - Repeat L times instruction sequence
  - Apply I-Bound and R-Bound computation
  - Might consider adding cost for loop-closing branch

© Roger Espasa, 2000

State Transition Signatures

- **Instead of encoding the transit of each instruction through pipeline…**
- **Summarize the information in a "simple" state vector**
  - (IB,LSQ, LSU0, LSU1, Cache0, Cache1, Complete, WB)
  - (6,0,0,0,0,0,0,0) indicates
    - 6 instructions in instruction buffer (IB)
    - No other instruction in pipeline
  - (8,2,1,1,0,0,0,0)
    - 8 instructions in IB
    - 2 instructions in LSQ
    - 1 instruction in LSU0 and 1 instruction in LSU1
  - Question: is this transition legal according to bandwidth and latency parameters?

© Roger Espasa, 2000

Rule-Based Checker

- **"Legal" transitions can be captured by a set of rules**
  - if (fetch_bw == 6 && no_instruction_completed) then sum of state I+1 = sum of state I + fetch_bw
- **Latency information can also be checked by verifying that "objects" arrive to the appropriate buffer in the specified number of cycles**
- **A program can be written to mechanically check every transition for "legality"**
  - If a rule is not obeyed, flag an error

© Roger Espasa, 2000

Part 4:
Performance Counters

© Roger Espasa, 2000

References

© Roger Espasa, 2000

[ASH86] A. Agarwal, R. L. Sites, M. Horowitz. “ATUM: A new Technique for Capturing Address Traces Using Microcode”. In *Proceedings of the 13th Annual Symposium on Computer Architecture*, pp. 280-290, June 1986.

[BAB96] D.C. Burger, T.M. Austin, S. Bennet. “Evaluating Future Microprocessors: The SimpleScalar Tool Set”. *TR R1308, University of Wisconsin-Madison*, July 1996

[Bed90] R. Bedicheck. “Some Efficient Architecture Simulation Techniques”. In *Winter’90 USENIX Conference*, June 1990

[CK94] R. F. Cmelik, D. Keppel. “Shade: A Fast Instruction-set Simulator for Execution Profiling”. In *Proc. Of the ’94 SIGMETRICS Conference*, pp. 128—137, May 1994.

[Cla83] D.W. Clark. “Cache Performance in the VAX-11/780”. In *ACM Transactions on Computer Systems*, 3:24-37, 1983

[Hol91] M. Holliday. “Techniques for Cache and Memory Simulation Using Address Reference Traces”. In *Intl. Journal of Computer Simulation*, 1:129-151, 1991.

[RHW96] M. Rosenblum, S. Herrod, E. Witchell, and A. Gupta: “Complete Computer System Simulation: the SimOS Approach”. In *IEEE Parallel and Distributed Technology*, 1996.

© Roger Espasa, 2000

[SE94] A. Srivasta, E. Eustace. “ATOM: A System for Building Customized Program Analysis Tools”. In *Proc. Of the SIGPLAN’94 Conference on Programming Language Design and Implementation*, pp 196—205, June 1994.

[Smi91] M.D. Smith. “Tracing with Pixie”. Technical Report, Stanford University, 1991.

[VF94] J. Veenstra and R. Fowler. “MINT: A Front End for Efficient Simulation of Shared Memory Multiprocessors”. In *Proc. Of the Second Intl. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pp. 201-207, 1994

© Roger Espasa, 2000

Course Projects

© Roger Espasa, 2000

P1- Emulator interface for Dixie

- **Dixie currently has no interface to talk to a modeling subsystem**
- **The project consists of adding to the Dixie VIRTUAL MACHINE the following set of calls:**
  - dixid = dixie_fetch(UINT64 pc, INSN_T *instruction)
  - dixie_issue(dixid)
  - dixie_read(dixid)
  - dixie_write(dixid)
  - dixie_commit(dixid)
  - dixie_fastfwd(UINT64 num_instructions)
- **Implications:**
  - DVM must do renaming
  - DVM must buffer speculative stores

© Roger Espasa, 2000

P2- Value Checking in SS/Jinks

- **Add value propagation in Simplescalar or Jinks**
- **Goal: validation of simulator**
- **Values in the simulator can help uncover bugs**
  - In dependency logic
  - In cache hierarchy

© Roger Espasa, 2000

P3- FSM Validation of SS

- **Apply the "Buffer Oriented Methodology" to SimpleScalar**
- **Model the RUU and similar buffers as FSM**
- **Code a partial tour generator for FSMs**
- **Generate assembly programs from FSM**
- **Test transition coverage**

© Roger Espasa, 2000

P4 – Apply ‘link’ model to SS/Jinks

- **Create a ‘wire’ library**
- **Break the OOO simulator in SS/Jinks into boxes**
- **Add wires between boxes**
- **Target: 5% deviation from Alpha 21264**

© Roger Espasa, 2000

Backup

© Roger Espasa, 2000

Validating PM Timing

- **Performance Signature of single instructions**
  - Manually set by designers
  - Checked against simulator
- **Performance Signature for pairs of instructions**
  - Table driven?
  - Pairs of instructions can be repeated in loop mode
- **Estimation of Performance Bounds**
  - I-bound
  - R-bound
- **Automatic Test Case Generation**
  - Random or Pseudo-random
    - With/without bias

© Roger Espasa, 2000

Modeling Challenges

- **Two types of bugs**
  - "functional" : core dumps and major mistakes
  - "performance" : can hide inside your simulator for years
- **Verification at all levels is necessary**

© Roger Espasa, 2000

Mètodes per a l'Avaluació de Sistemes Informàtics

Roger Espasa, Jorge Garcia
Dept. Arquitectura de Computadors
Universitat Politècnica de Catalunya
Abril-Juny 2000

© Roger Espasa, 2000

Course Outline

- **Fundamentals of Microprocessor Simulators**
  - Tracing programs and systems
  - Feeding a microprocessor simulator
- **Developing a Simulator**
  - Software Engineering Challenges
  - Cycle-by-cycle vs. Event-driven
  - Common techniques
  - Backward signals in a pipeline
  - Simulator-Feeder Interaction
- **Validating a Performance Simulator**
  - Micro kernels
  - Automatic Test Generation
- **Performance/Event Counters**

© Roger Espasa, 2000

Lab sessions

- **L1-L2: Verificacio de Rendiment**
  - Escriure loops senzills per verificar parts de l"arquitecura
    - Hit sempre
    - Miss L1, Hit L2
    - Replay-Trap
  - Executar en kemet i medir temps
  - Executar en SimpleScalar
  - Presentar diferencies
- **Dissenyar un processador**

© Roger Espasa, 2000

Jorge

1	Simulacion
2	Modelado Analtico. Monitorizacion Unix NT
3	Caracterizacion carga trabajo (SPEC). Prediccion carga trabajo.
4	
5	
6	Disenyo Experimentos
7	
8	Presentacion Trabajos
9	
10	

© Roger Espasa, 2000

Roger

1	Instrumentacion + Contadores
2	Estructuras Simulacion
3	
4	Generacion Tests
5	
6	?
7	
8	Presengtacion Trabajos
9	
10	

© Roger Espasa, 2000

Modeling the Pipeline Stages

- **Pipeline: forward & backward**
  - Real processor signals should have a software equivalent
  - Send signals between stages both forward and backward

© Roger Espasa, 2000

Backward Signals in a Pipeline

© Roger Espasa, 2000

Simulator-Feeder Interaction

© Roger Espasa, 2000

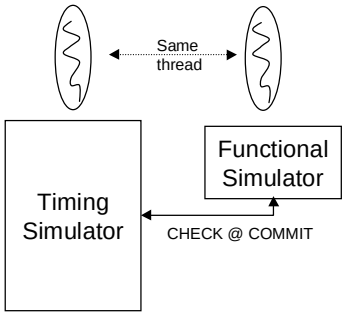
Fast Simulation Techniques

- **Techniques for Fast Simulation [Moudgill]**

© Roger Espasa, 2000

Correctness Verification

- **Instruction Verifier**
  - Timing versus functional
  - Uncovered many obvious bugs we had not seen!
  - Uncovered many subtle cache access bugs



DIFFICULT

© Roger Espasa, 2000

Performance Verification

- **Make sure simulator matches machine being built**
- **Ideally, validate simulator against RTL**
  - Still, must allow for some % of deviation since model is never perfect (otherwise, it would be RTL ☺)
- **How did we do it**
  - Eye inspection of micro-kernels
  - Comparison against a second simulator
    - We were lucky to be in a software transition between two simulators
  - RTL checking in near future
- **Research in this topic would be welcome!**

© Roger Espasa, 2000