How Many Simulators Does it Take to Build a Chip?

Steve Keckler

Department of Computer Sciences
The University of Texas at Austin
Broader question: what tools and analysis are required to design a new processor?
- New ISA
- New microarchitectures (processor, memory system)
- New levels of design hierarchy

This is a “Design Experience” talk
- No new research results
- Insight into system design methodologies based on TRIPS
Outline

- **TRIPS System Design Overview**
  - ISA and microarchitecture
  - Prototype specifications

- **Simulators**
  - ISA and SW design
  - Microarchitecture design
  - System design

- **Hardware Validation Methodology**
  - Correctness and performance validation

- **Power Analysis**

- **TRIPS Software Tools**
  - Binary utilities, debugger, performance analysis

- **Conclusions**
TRIPS EDGE ISA

- **Explicit Data Graph Execution [IEEE Computer '04]**
  - Defined by two key features

- **Program graph is broken into sequences of blocks**
  - Basic blocks, hyperblocks (max 128 instruction in TRIPS)
  - Blocks commit atomically or not at all - a block never partially executes
  - Amortize overheads over many instructions
  - Compiler forms blocks via loop unrolling, predication, inlining, etc.

- **Within a block, ISA support for direct producer-to-consumer communication**
  - No shared named registers within a block (point-to-point dataflow edges only)
  - Instructions “fire” when their operands arrive
  - The block’s dataflow graph (DFG) is explicit in the architecture
TRIPS Processor Specifications

- An aggressive, general-purpose processor
  - Up to 16 instructions per cycle
  - Up to 4 loads and stores per cycle
  - Up to 64 outstanding L1 data cache misses
  - Up to 1024 dynamically executing instructions
  - Up to 4 simultaneous multithreading (SMT) threads
  - Inter- and intra-block speculation

- Memory system
  - 4 simultaneous L1 cache fills per processor
  - Up to 16 simultaneous L2 cache accesses
TRIPS Prototype Chip

- 2 TRIPS Processors
- NUCA L2 Cache
  - 1 MB, 16 banks
- On-Chip Network (OCN)
  - 2D mesh network
  - Replaces on-chip bus
- Controllers
  - 2 DDR SDRAM controllers
  - 2 DMA controllers
  - External bus controller
  - C2C network controller
## TRIPS Tile-level Microarchitecture

### TRIPS Tiles

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G:</td>
<td>Processor control - TLB w/ variable size pages, dispatch, next block predict, commit</td>
</tr>
<tr>
<td>R:</td>
<td>Register file - 32 registers x 4 threads, register forwarding</td>
</tr>
<tr>
<td>I:</td>
<td>Instruction cache - 16KB storage per tile</td>
</tr>
<tr>
<td>D:</td>
<td>Data cache - 8KB per tile, 256-entry load/store queue, TLB</td>
</tr>
<tr>
<td>E:</td>
<td>Execution unit - Int/FP ALUs, 64 reservation stations</td>
</tr>
<tr>
<td>M:</td>
<td>Memory - 64KB, configurable as L2 cache or scratchpad</td>
</tr>
<tr>
<td>N:</td>
<td>OCN network interface - router, translation tables</td>
</tr>
<tr>
<td>DMA:</td>
<td>Direct memory access controller</td>
</tr>
<tr>
<td>SDC:</td>
<td>DDR SDRAM controller</td>
</tr>
<tr>
<td>EBC:</td>
<td>External bus controller - interface to external PowerPC</td>
</tr>
<tr>
<td>C2C:</td>
<td>Chip-to-chip network controller - 4 links to XY neighbors</td>
</tr>
</tbody>
</table>
Grid Processor Tiles and Interfaces

- **GDN**: global dispatch network
- **OPN**: operand network
- **GSN**: global status network
- **GCN**: global control network
Non-Uniform L2 Cache (NUCA)

- **1MB L2 cache**
  - Sixteen tiled 64KB banks

- **On-chip network**
  - 4x10 2D mesh topology
  - 128-bit links, 366MHz (4.7GB/sec)
  - 4 virtual channels prevent deadlocks
  - Requests and replies are wormhole-routed across the network

- Up to 10 memory requests per cycle
- Up to 128 bytes per cycle returned to the processors
- Individual banks reconfigurable as scratchpad
### TRIPS Chip Implementation

<table>
<thead>
<tr>
<th>Process Technology</th>
<th>130nm ASIC with 7 metal layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die Size</td>
<td>18.3mm x 18.37mm (336 mm²)</td>
</tr>
<tr>
<td>Package</td>
<td>47mm x 47mm BGA</td>
</tr>
<tr>
<td>Pin Count</td>
<td>626 signals, 352 Vdd, 348 GND</td>
</tr>
<tr>
<td># of placed cells</td>
<td>6.1 million</td>
</tr>
<tr>
<td>Transistor count (est.)</td>
<td>170 million</td>
</tr>
<tr>
<td># of routed nets</td>
<td>6.5 million</td>
</tr>
<tr>
<td>Total wire length</td>
<td>1.06 km</td>
</tr>
<tr>
<td>Power (measured)</td>
<td>36W at 366MHz, 1.5V (chip has no power mgt.)</td>
</tr>
<tr>
<td>Clock period</td>
<td>2.7ns (actual) 4.5ns (worse case sim)</td>
</tr>
</tbody>
</table>

Experiments show that chip achieves 400MHz at 1.6V
Chip Area Breakdown

Overall Chip Area:
- 29% - Processor 0
- 29% - Processor 1
- 21% - Level 2 Cache
- 14% - On-Chip Network
- 7% - Other

Processor Area:
- 30% - Functional Units (ALUs)
- 4% - Register Files & Queues
- 10% - Level 1 Caches
- 13% - Instruction Queues
- 13% - Load & Store Queues
- 12% - Operand Network
- 2% - Branch Predictor
- 16% - Other
TRIPS Motherboard

- 1 motherboard includes:
  - 4 daughter-boards
  - 4 TRIPS chips
  - 8 GBytes DRAM
  - PowerPC 440GP control processor
  - I/O: ethernet, serial, C2C links
  - FPGA I/O interface

- Peak performance
  - 48 GFlops at 366 MHz
  - 180 Watts
TRIPS System I

- 8 TRIPS boards
- 374 Gflops/Gops peak
- 5 boards currently deployed
TRIPS System Software Stack

- TRIPS Resource Manager (TRM)
- File system
- Runtime services
- Login/debug/etc.

Local Resource Manager (LRM) listens to HostPC
- Runs embedded Linux
- PPC EBI device driver to control TRIPS chips
- PPC EBI ↔ TRIPS EBC

- Runs TRIPS apps
- Interrupts PPC if necessary
- System calls, exceptions
Outline

- **TRIPS System Design Overview**
  - ISA and microarchitecture
  - Prototype specifications

- **Simulators**
  - ISA and SW design
  - Microarchitecture design
  - System design

- **Hardware Validation Methodology**
  - Correctness and performance validation

- **Power Analysis**

- **TRIPS Software Tools**
  - Binary utilities, debugger, performance analysis

- **Conclusions**
## TRIPS Simulator Overview

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Purpose</th>
<th>Speed</th>
<th>LoC</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsim_arch</td>
<td>ISA emulator&lt;br&gt;ISA and SW design</td>
<td>1M instr/sec</td>
<td>5.4K</td>
<td>None</td>
</tr>
<tr>
<td>tsim_proc</td>
<td>uarch simulator (1 proc.)&lt;br&gt;perf. analysis, HW validation</td>
<td>1-2K instr/sec</td>
<td>37.2K</td>
<td>5%</td>
</tr>
<tr>
<td>tsim_cyc</td>
<td>uarch cycle estimator&lt;br&gt;SW perf. analysis</td>
<td>500K instr/sec</td>
<td>7.7K</td>
<td>20-30%</td>
</tr>
<tr>
<td>tsim_sys</td>
<td>multiprocessor and system parallel apps, system software</td>
<td>tsim_cyc/ procs</td>
<td>5.2K</td>
<td>~30%</td>
</tr>
<tr>
<td>tsim_ocn</td>
<td>interconnect and NUCA cache&lt;br&gt;uarch design, perf. analysis</td>
<td>200K cyc/sec</td>
<td>7.8K</td>
<td>10%</td>
</tr>
<tr>
<td>tsim_nuca</td>
<td>flexible NUCA simulator&lt;br&gt;architecture tradeoffs</td>
<td>400K cyc/sec</td>
<td>5.2K</td>
<td>20%</td>
</tr>
<tr>
<td>tmax</td>
<td>flexible uarch simulator&lt;br&gt;TRIPS extension studies</td>
<td>100K instr/sec</td>
<td>33K</td>
<td>~15%</td>
</tr>
</tbody>
</table>

- **tsim** processor simulators share common infrastructure (5.2K LoC)
- Total simulator code: 126K LoC
- TRIPS RTL design - 229K LoC
  - Processor: 169K LoC
  - NUCA + peripherals: 60K LoC
Design Phases

2000-2002
- Early architecture development (Grid Processor and NUCA)
- High-level simulation, experiments
- Chip and system specification
- Construction of cycle-simulator
- Tile-level RTL and verification

2003
- Trimaran-based simulator
- First ISA simulator
- tsim_nuca
- tsim_services
- tsim_arch
- tsim/RTL validation

2004
- tmax
- Manufacturing

2005
- tsim_proc
- tsim_ocn
- tsim_sys
- tsim_cyc
TRIPS ISA Design

First TRIPS exploration (Micro '01)
- Trimaran VLIW compiler (block formation)
- Instruction rescheduler for ALU array
- Custom high-level simulator
- Useful - but a long way from our final implementation

TRIPS ISA #1
- Specification, assembler, simulator
- Flawed in a number of ways
  - Predication model was broken
  - Instruction encodings were complicated
  - Didn’t have all of the byte operations

TRIPS ISA #2
- Implemented in tsim_arch (C++)
  - Executes 1 block at a time, follows data dependences
  - Statistics: instruction counts, dataflow depth
- Experiments proved out ISA, added features
  - Store null operations, constant generation
TRIPS Microarchitecture Design

- Tile-level specifications and interfaces

- Cycle-precise C++ performance models
  - tsim_proc - all processor uarch features
    - Fully pipelined design of processor
    - Performance analysis of processor protocols (fetch, bypass, commit, etc.)
    - Common infrastructure for pipeline (wire/register models)
  - tsim_ocn - same for NUCA + interconnect

- Uses
  - Performance analysis: accurate but slow
  - Reference model for RTL design (all latencies)
  - Functional and performance validation
Performance Tools

- Tool to show control flow visualization of each block in a function
  - Allows programmer to optimize block formation

- Tool to show dataflow between instructions within a block
  - Allows programmer to evaluate instruction level optimizations
Performance Tools

- **Cycle by cycle view of all chip resources, shows interactions between 8 blocks in flight**
  - Allows programmer to understand how resources are used by multiple blocks in flight

<table>
<thead>
<tr>
<th>CYCLE</th>
<th>RLEMWSB</th>
<th>RLEMWSB</th>
<th>RLEMWSB</th>
<th>RLEMWSB</th>
<th>RLEMWSB</th>
<th>RLEMWSB</th>
<th>RLEMWSB</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2608</td>
<td>__<strong>a</strong></td>
<td>..3....</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2609</td>
<td>fetch</td>
<td>..21...</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2610</td>
<td>.......</td>
<td>..21...</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2611</td>
<td>.......</td>
<td>..12...</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2612</td>
<td>.......</td>
<td>..2....</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2613</td>
<td>.......</td>
<td>..3....</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2614</td>
<td>c....</td>
<td>..21...</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2615</td>
<td>.......</td>
<td>..12...</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2616</td>
<td>.......</td>
<td>........</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2617</td>
<td>.......</td>
<td>..11...</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2618</td>
<td>..1....</td>
<td>..11...</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2619</td>
<td>..2....</td>
<td>........</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2620</td>
<td>..2....</td>
<td>..1....</td>
<td>........</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2621</td>
<td>..2....</td>
<td>..1.1..</td>
<td>........</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2622</td>
<td>..2....</td>
<td>..1...</td>
<td>1......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>2623</td>
<td>..1....</td>
<td>...............</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
<td>.......</td>
</tr>
</tbody>
</table>
System-level Simulation

- **Primary needs**
  - Software development and performance analysis of parallel codes/libraries
  - Develop/validate system SW and external interfaces

- **Simulators**
  - `tsim_cyc` - performance estimator
    - Analyzes blocks, dependence graphs, resources, placement
    - Accounts for block speculation/overlap and caching
    - Computes block latency
    - Substantial tuning drove accuracy from 50% to 30%
    - Includes some empirically derived constants
  - `tsim_sys`
    - Models chip interface to PowerPC control chip
    - Plugs into trips host monitor (same interface as hardware)
    - Serves as gasket for RTL to plug into host monitor
Outline

- TRIPS System Design Overview
  - ISA and microarchitecture
  - Prototype specifications

- Simulators
  - ISA and SW design
  - Microarchitecture design
  - System design

- Hardware Validation Methodology
  - Correctness and performance validation

- Power Analysis

- TRIPS Software Tools
  - Binary utilities, debugger, performance analysis

- Conclusions
RTL Verification

- Ensure functional and performance correctness

- Multiple levels of abstraction
  - Tile-level verification
  - Partition-level verification
    - Processor: 30 tiles
    - OCN: 40 tiles
  - Chip-level verification
    - All 106 tiles

- Substantial time commitment
  - 60-70% of overall effort
  - Considerable design of test strategy and infrastructure
  - Fully automated
    - Could perform a full test verification, synthesis, timing analysis, etc. in under two days
    - Parallelized widely on cluster and desktop machines (Condor)
Design and Verification Strategy

- **Organization**
  - Tile-level design/verification teams (typically 2 students/staff)
  - Additional verification teams: processor, OCN, full-chip, performance
  - Separate design and verification reviews

- **Tile-level verification**
  - Custom testbench per unit
  - **Goal**: complete testing of functionality and interfaces
  - Random test transactions, modes + auto checking, Verilog, C++, PLI
  - **Coverage analysis**: manual and automatic

**ET test bench**

- **Random Test Generator (C++)**
  - **Seed**
  - **Test parameters**
  - **Block dispatch**

- **ET RTL Model**
  - **ET inputs**

- **Equivalence Checker (C++)**
  - **Block commit/flush**

- **ET C++ model (tsim_proc)**
GP Partition Verification
(Processor - 30 tiles)
GP Verification Details

- **Test suites**
  - hand generated TASL (from tsim verification)
  - C-language tests (short)
  - microbench (shortened from tsim)
  - Random TASL tests
  - dietlibc and fdlibm tests
  - Performance tests

- **Randomization**
  - Random programs
  - Random external latencies
  - Random mode/mode changes

- **Coverage assessment**
  - Assertions in code and state machines
  - Analysis of code line coverage (automated)
Performance Verification of RTL

- **Goal:** ensure that RTL implements performance specification
  - Hidden issues: extra pipeline stages, arbitration policy mismatches, speculation/event ordering

- **General approach**
  - Hand analysis of RTL (waveforms) to check protocol latencies
  - Cycle-to-cycle comparison of RTL to tsim_proc
    - Selected suite of small programs intended to exercise system

- **Results**
  - Initially, RTL slower by 8%
  - Finally, RTL slower by 4%
  - Why not closer?
    - Highly speculative architecture with predictors
    - Exact event ordering is prohibitively expensive to produce
Principal Performance Bugs

- Extra pipeline bubbles during I-cache refill
- Extra bubble during instruction decode
- Extra states in state machine for block deallocation
- Extra states in state machine for register commit
- Underestimate of block flush latency
- Incorrect arbitration priority of selection in branch predictor
- Serialization of local and remote operand delivery

- Dependence predictor gave performance instability
  - Pathological cases in small programs sometimes caused large swing in number of block flushes
Chip-level Verification

- Focus primarily on
  - Interfaces among controllers and partitions
  - EBI, SDRAM, C2C, and other external interfaces
  - Internal scan circuits (post synthesis/scan insertion)

- Full-chip simulation speed
  - RTL: 25 processor cycles/second
  - Gate-level netlist: 0.6 processor cycles/second

- Other top-level verification issues
  - Bugs found:
    - Unconnected and inverted test clock
    - Inconsistent implementation of error conditions
  - Internal scan was a real pain (not well integrated into ASIC flow)
  - Found bugs in synthesis for certain (legal) coding styles
  - Came across usual problems with 3-state logic simulation
    - X-optimism in RTL
    - X-pessimism in gates
    - Result: added more reset logic in RTL
  - Total of 268 bugs found and fixed
    - Most at tile-level
    - Relatively few at processor, OCN, full-chip
Benefits of Tiled Design

- **Modularity**
  - Designed and verified only 11 tiles, 106 total instantiations

- **Verification**
  - Modularity was a big help

- **Timing:**
  - trivial at top level, no global timing paths

- **P&R:**
  - all wires to nearest neighbors, no global wires
  - But - each tile’s physical design was a little different

- **Hardware bringup**
  - 1 day for simple program
  - 21 days for first SPEC benchmark
  - No chip bugs found to date
  - 15 board bugs, most fixable via “blue-wire”
Outline

- TRIPS System Design Overview
  - ISA and microarchitecture
  - Prototype specifications

- Simulators
  - ISA and SW design
  - Microarchitecture design
  - System design

- Hardware Validation Methodology
  - Correctness and performance validation

- Power Analysis

- TRIPS Software Tools
  - Binary utilities, debugger, performance analysis

- Conclusions
TRIPS Power Modeling and Analysis

- **Goal** - power model for TRIPS simulator
  - Validated to the hardware
  - Provide insight into TRIPS power efficiency/usage

- **Challenges**
  - Validation is tricky because hardware looks like a black box (limited probing capability)
  - Large different in level of abstraction
  - Initial error of 3x relative to HW

- **Our approach** - leverage RTL to bridge gap
  - Still requires some contortions....
Architecture Power Models

- Approach similar to Wattch
  - Count power consuming events
  - Apply per-event power model
    - CACTI for array structures
    - Vendor DIMM model
    - Wattch models for ALUs
    - Specification driven latch and clock tree counts/models

---

1. Benchmark Binary
2. tsim_proc
3. tsim_ocn
4. Core Power Model
5. Core Power Estimate
6. Memory Power Estimate
7. Total Power Estimate
8. Micron DIMM Power Model
9. OCN/NUCA Power Model
TRIPS Power Measurement

- **Power Measurement**
  - Agilent current probe outputs voltage proportional to current drawn
  - NI DAQ unit samples voltages
  - Voltage samples processed to measure power

- **Experiments to isolate power**
  - Motherboard
  - Heatsink
  - DIMMs
  - Clock Tree

- **Lots of useful data for validation**
RTL Power Validation

- Based on gate-level activity factors
  - Within 6% of hardware measurements
- Provides fine-grained power breakdown
  - Missing in measured hardware power
  - Useful for validating architectural power models
- Validated architectural power models
  - Validated architecture simulator to within 15-20% of HW
Improving the Architecture Models

- Average shown is for 25 benchmarks (only a subset shown here)
- Legend
  - Base : Base architectural power models
  - Base + C: More accurate latch counts
  - Base + C + T: accurate Technology Models
  - Base + C + T + P: Actual latch counts
  - Base + C + T + P + G: Actual gate counts for control logic
Outline

- TRIPS System Design Overview
  - ISA and microarchitecture
  - Prototype specifications

- Simulators
  - ISA and SW design
  - Microarchitecture design
  - System design

- Hardware Validation Methodology
  - Correctness and performance validation

- Power Analysis

- TRIPS Software Tools
  - Binary utilities, debugger, performance analysis

- Conclusions
TRIPS Compiler Toolchain

- Compiles both C and Fortran programs
- Successfully compiles SPEC2K suite
- Support for using profile information to tune compiler heuristics
- Binary utilities based on Gnu binutils (reuse!)
- Lines of code
  - Scale ~300k
  - TRIPS backend ~30k
  - TRIPS scheduler ~21k
**TRIPS Debugger (tdb)**

- **tdb is a port of gdb**
  - Use standard client server approach
  - Use TRM’s standard hardware read/write commands as conduit

- **Challenges for debugging block oriented architecture**
  - Atomic block execution - can’t break at instruction level
  - Correlation with source code line numbers

- **Experience:**
  - Breaking at block boundaries has worked reasonably well
  - Partial solution: debug at lower optimization levels (pre hyperblock formation)
  - Block instruction-by-instruction is a possible extension
Performance Tools

- Program visualization tools described earlier
  - Program structure analysis
  - Simulator support to aid application optimization
- Rich set of HW performance counters
- Port of PAPI API to TRIPS
  - Used to measure performance of regions of code

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Convolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Cycles</td>
<td>211,603,224</td>
</tr>
<tr>
<td>Blocks Fetched</td>
<td>11,225,376</td>
</tr>
<tr>
<td>Blocks Committed</td>
<td>11,214,982</td>
</tr>
<tr>
<td>I-cache misses</td>
<td>12</td>
</tr>
<tr>
<td>Branch Predictions</td>
<td>11,214,982</td>
</tr>
<tr>
<td>Branch Mispredictions</td>
<td>8,285</td>
</tr>
<tr>
<td>Load Flushes</td>
<td>0</td>
</tr>
<tr>
<td>DT Load Accesses</td>
<td>201,398,453</td>
</tr>
<tr>
<td>DT Store Accesses</td>
<td>67,108,869</td>
</tr>
<tr>
<td>DT Load Misses</td>
<td>68,650,202</td>
</tr>
<tr>
<td>DT Store Misses</td>
<td>477</td>
</tr>
<tr>
<td>MT Read Accesses</td>
<td>8,421,638</td>
</tr>
<tr>
<td>MT Write Accesses</td>
<td>4,201,473</td>
</tr>
<tr>
<td>MT Cache Misses</td>
<td>67,792</td>
</tr>
<tr>
<td>MT Cache Fills</td>
<td>67,792</td>
</tr>
<tr>
<td>MT Cache Spills</td>
<td>41,769</td>
</tr>
<tr>
<td>Branch Prediction Rate</td>
<td>0.999261256</td>
</tr>
<tr>
<td>DT0 access %</td>
<td>0.250083959</td>
</tr>
<tr>
<td>DT1 access %</td>
<td>0.249961809</td>
</tr>
<tr>
<td>DT2 access %</td>
<td>0.249992512</td>
</tr>
<tr>
<td>DT3 access %</td>
<td>0.24996172</td>
</tr>
</tbody>
</table>
Conclusions

- Built more tools than we originally expected
  - Each had their place and was widely used
  - Still using simulators as observability is better than HW
    - Instruction distribution
    - Interactions that are difficult to capture in counters
- Common infrastructure across tools worked well
- Simulator validation is hard (ISCA '01)
  - Even when you have intimate knowledge of the design
  - Difficult to reproduce speculative events
  - Power validation is even worse
- Simulators for HW design quickly became too rigid for research
  - Had to develop newer flexible simulator
  - Tremendous benefit to validate new simulator against HW; gives credence to experiments on new architectures
Challenges

- **Systems complexity is continuing to increase**
  - Still driven by Moore’s law
  - Multicores are a double-edged sword
    - More elements to simulate
    - But don’t need to simulate all at same level of detail

- **What about FPGAs**
  - Standard emulation could have accelerated RTL
    - But - learning curve is steep
  - Approaches like FAST [Micro ’07] could accelerate low-level simulation
    - But - have to implement in an HDL (Bluespec can help)
    - Still no substitute for flexibility and observability of simulators

- **Parallel simulation**
  - We primarily needed simulation throughput - Condor was fine
  - Still open questions on parallel simulation of parallel systems