Using the M5 Simulator

Nathan Binkert, Ron Dreslinski,
Lisa Hsu, Kevin Lim, Ali Saidi
Prof. Steve Reinhardt

Welcome!

- This tutorial is for you
  - Feel free to ask questions
- We've got a lot to cover
  - Lots of cool stuff didn't even make the slides
  - Don't be offended if we have to move on
  - Come talk to us later
    - we're all here through Wednesday

Outline & Schedule

- Introduction & overview 2:00-2:20
- Running M5 2:20-2:45
- Full-System Workloads 2:45-3:00
- Current M5 object models
  - CPUs: simple, detailed 3:00-3:30
  - (break) 3:30-4:00
  - Memory System 4:00-4:35
  - I/O 4:35-4:55
- Extending M5 4:55-5:30

Introduction & Overview

Steve Reinhardt

Introduction & Overview

- What M5 is and is not
- A brief peek inside
- Current status & future developments

What is M5?

- A tool for simulating systems
  - Not just CPU cores: memory, I/O
  - Not just SPEC apps: full OS code
  - Not just single machines: client/server, etc.

Two Views of M5

1. A framework for event-driven simulation
   - Events, objects, statistics, configuration
2. A collection of predefined object models
   - CPUs, caches, busses, devices, etc.

- This tutorial focuses on #2
- You may find #1 useful even if #2 is not

Where Did M5 Come From?

- Born of frustration with existing tools
  - Did not do what we wanted
  - Did not scale with added complexity
- Desire to simulate TCP/IP performance
  - Full-system support
  - Multiple system simulation
- Almost entirely original code
  - Old CPU model based on SimpleScalar sim-outorder
  - Full-system support used SimOS as reference
- No premeditated distribution plans
  - Just hacking together the system *we* wanted

Key M5 Attributes

- Heavily object-oriented
  - Key to modularity, flexibility
- Necessarily complex
  - ~90K lines of C++, ~7K lines of Python
- Modular enough to hide the complexity
  - We hope!
- Free! All the code we wrote is open source
  - BSD-style license

What M5 is *Not*

- A hardware design language
  - Higher level for design space exploration, simulation speed
- A restrictive environment
  - Just C++/Python with an event queue and a bunch of APIs you can choose to ignore
- Finished!
  - Always room for improvement…

What We Would Like M5 to Be

- Something that spares you the pain we've been through
- A community resource
  - Modular enough to localize changes
  - Contribute back, and spare others some pain
- A path to reproducible/comparable results
  - A common platform for evaluating ideas

A Peek Inside: Objects

- Everything you care about is an object (C++/Python)
- Derived from SimObject base class
  - Common code for creation, configuration parameters, naming, checkpointing, etc.
- Uniform method-based APIs for object types
  - CPUs, caches, memory, etc.
  - Plug-compatibility across implementations
    - Functional vs. detailed CPU
    - Conventional vs. indirect-index cache
- Easy replication: MPs, multiple systems, …

A Peek Inside 2: Events

- Standard event queue timing model
  - Global logical time (in "ticks")
  - No fixed relation to real time
- Objects schedule their own events
  - Flexibility for detail vs. performance tradeoffs
- E.g., a CPU typ. schedules an event every cycle
  - Simple CPU won't schedule self if stalled/idle
  - Can also schedule every n^{th} cycle to model other clock rates
    - e.g., non-integer CPU/bus clock ratios

Current Model Status

- Three CPU models
  - One functional & two detailed OOO
    - old SimpleScalar-based & new in development
  - CPUs support Alpha ISA
    - Others "easily" added
- Two major cache models
  - Conventional & indirect-index
- Bus-based interconnect
  - Split transactions, snooping coherence

Current Status (cont'd)

- Syscall emulation mode
  - Alpha Tru64 or Linux application binaries
  - Host-based or SimpleScalar EIO traces
- Full-system mode
  - Models Compaq "Tsunami"-based system
    - Boots Linux 2.4 & 2.6 and L4; FreeBSD in progress
      - Ask us if you want Tru64
    - Extensions for >4 CPUs
  - Ethernet, IDE disk adapters
  - Handful of pre-built benchmarks available

Short-term Wish List

- Finish new detailed OOO CPU model
  - Add full system, SMT support
  - Obsolete SimpleScalar-based model
- Minor re-architecting of memory system
  - Support non-bus interconnects, directory coherence
- More ISAs
  - PowerPC, ARM likely candidates
  - Heterogeneous system support (?)
- More full-system benchmarks
- Better C++/Python integration

Outline & Schedule

- Introduction & overview 2:00-2:20
- Running M5 2:20-2:45
- Full-System Workloads 2:45-3:00
- Current M5 object models
  - CPUs: simple, detailed 3:00-3:30
  - (break) 3:30-4:00
  - Memory System 4:00-4:35
  - I/O 4:35-4:55
- Extending M5 4:55-5:30

Running M5

Nathan Binkert

Running M5

- Source tree
- Building executables
- Running simulations
- Specifying configurations
- Output files
- Checkpointing
- Sampling & warm-up

Source Tree Organization

- m5: actual simulator source
- m5-test: regression tests
- ext: 3rd-party packages
  - dnet, libelf, ply
- linux-dist: source for disk images

M5 Source Tree Organization

- base: general data structures/facilities
- sim: simulation core
- python: Python config code
- arch: ISA-specific components
- cpu, mem, dev: specific models

compiled in

- build: build directories
- test: component tests
- util: utility programs
- config: sample configurations

not compiled in

Building Executables

- Platforms
  - Linux, BSD, CYGWIN (most UNIX like systems?)
    - Linux is primary, others may take a tiny bit of work
  - Little endian machines!
  - 64-bit machines help a lot
- Tools
  - GCC/G++ 3.0+
    - Recently tested with 3.3-3.5
  - Python 2.4
  - SCons (we use 0.95 or 0.96.1)
    - http://www.scons.org

```
% cd m5
% cd build
% scons ALPHA_SE/m5.opt ALPHA_FS/m5.debug
scons: Reading SConscript files ...
Configuring options for directory 'ALPHA_FS'.
Compiling with MySQL support!
scons: done reading SConscript files.
scons: Building targets ...
/z/binkertn/research/m5/build/head/m5/arch/isa_parser.py
   m5/arch/alpha/isa_desc ALPHA_FS/arch/alpha arch/alpha
Generating ALPHA_FS/arch/alpha/decoder.hh
Generating ALPHA_FS/arch/alpha/decoder.cc
Generating ALPHA_FS/arch/alpha/inorder_cpu_exec.cc
Generating ALPHA_FS/arch/alpha/simple_cpu_exec.cc
Generating ALPHA_FS/arch/alpha/fast_cpu_exec.cc
Generating ALPHA_FS/arch/alpha/full_cpu_exec.cc
Generating ALPHA_FS/arch/alpha/alpha_full_cpu_exec.cc
echo '#include "arch/alpha/isa_traits.hh"' >
   ALPHA_FS/targetarch/isa_traits.hh
echo '#include "arch/alpha/alpha_memory.hh"' >
   ALPHA_FS/targetarch/alpha_memory.hh
echo '#include "arch/alpha/byte_swap.hh"' >
   ALPHA_FS/targetarch/byte_swap.hh
python m5/base/traceflags.py ALPHA_FS/base/traceflags
g++ -pipe -fno-strict-aliasing -Wall -Wno-sign-compare -Werror -Wundef -g -
   gstabs+ -O0 -DFULL_SYSTEM -DUSE_MYSQL -DSTATS_BINNING -DDEBUG -Iext/dnet
   -I/usr/local/include/mysql -I/usr/include/mysql -IALPHA_FS -Im5 -c -o
   ALPHA_FS/arch/alpha/decoder.do ALPHA_FS/arch/alpha/decoder.cc
...
```

Running Simulations

```
Usage:
m5.debug [-d <dir>] [-E <var>[=<val>]] [-I <dir>] [-P <python>]
        [--<var>=<val>] <config file>

   -d             set the output directory to <dir>
   -E             set the environment variable <var> to <val> (or
   'True')
   -I             add the directory <dir> to python's path
   -P             execute <python> directly in the configuration
   --var=val      set the python variable <var> to '<val>'
   <configfile>   config file name (ends in .py)
```

```
% ALPHA_FS/m5.debug -d output -ETEST=SPECWEB
   configs/fullsys/run.py
% ALPHA_FS/m5.debug -d output -EDUMPFILE=ethertrace
   -ETEST=NETPERF_STREAM configs/fullsys/run.py
   --Trace.flags="EthernetAll"
% ALPHA_SE/m5.opt -d output m5-test/test1/run.py
```

```
% ~/m5/build/ALPHA_FS/m5.debug -d output ~/m5/configs/fullsys/run.py
M5 Simulator System
Copyright (c) 2001-2005
The Regents of The University of Michigan
All Rights Reserved

This code is part of the M5 simulator, developed by Nathan Binkert,
Erik Hallnor, Steve Raasch, and Steve Reinhardt, with contributions
from Ron Dreslinski, Dave Greene, Lisa Hsu, Ali Saidi, and Andrew
Schultz.

M5 compiled on May 31 2005 11:43:24
M5 executing on ziff.eecs.umich.edu
M5 simulation started Tue May 31 11:45:02 2005
Listening for console connection on port 3456
      0: system.tsunami.io: Real-time clock set to Sun Jan  1 00:00:00 2006
command line: /n/ziff/z/binkertn/build/head/ALPHA_FS/m5.debug -d output
   /n/ziff/z/binkertn/research/m5/head/configs/fullsys/run.py

Listening for remote gdb connection on port 7000
warn: Entering event queue.  Starting simulation...
```


m5term

- allows user to connect to the simulated console interface

```
% cd m5
% cd util/term
% make
gcc -o m5term term.c
% make install
sudo install -o root -m 555 m5term /usr/local/bin
```


```
% m5term localhost 3456
==== m5 slave console: Console 0 ====
M5 console
Got Configuration 127
memsize 8000000 pages 4000
First free page after ROM 0xFFFFFC0000018000
HWRPB 0xFFFFFC0000018000 l1pt 0xFFFFFC0000040000 l2pt 0xFFFFFC0000042000
   l3pt_rpb 0xFFFFFC0000044000 l3pt_kernel 0xFFFFFC0000048000 l2reserv
   0xFFFFFC0000046000
CPU Clock at 2000 MHz IntrClockFrequency=1024
Booting with 1 processor(s)
...
...
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 480k freed
init started:  BusyBox v1.00-rc2 (2004.11.18-16:22+0000) multi-call binary

PTXdist-0.7.0 (2004-11-18T11:23:40-0500)

mounting filesystems...
EXT2-fs warning: checktime reached, running e2fsck is recommended
loading script...
Script from M5 readfile is empty, starting bash shell...
# ls
benchmarks  etc         lib         mnt         sbin        usr
bin         floppy      lost+found  modules     sys         var
dev         home        man         proc        tmp         z
#
```


Configuration Files

- Python
- Config objs mapped to simulator objs
- No need for scripts to generate configs
  - All logic for running many simulations contained in a single set of configurable config files!
- Pass parameters via environment vars
  - -E<var>[=<val>]
- Variables with units are enforced
  - Latency must be ‘2ns’, not simply 2

```
class DCache(BaseCache):
    latency = 3 * Parent.clock.period
    size = '32kB'
    mshrs = 32

class CPU(SimpleCPU):
    dcache = DCache(in_bus=NULL, out_bus=Parent.membus)
    icache = ICache(in_bus=NULL, out_bus=Parent.membus)

class System(LinuxSystem):
    cpu = CPU()
    membus = Bus(width=16, clock='400MHz')
    ram = BaseMemory(in_bus=Parent.membus, latency='40ns',
                     addr_range=[ Parent.physmem.range ])
    physmem = PhysicalMemory(range=AddrRange('128MB'))
    tsunami = Tsunami()
    simple_disk = SimpleDisk(disk=Parent.tsunami.disk0.image)
    sim_console = SimConsole(listener=ConsoleListener(port=3456))
    kernel = '/dist/m5/system/binaries/vmlinux-latest'
    pal = '/dist/m5/system/binaries/ts_osfpal'
    console = '/dist/m5/system/binaries/console_ts'
    boot_osflags = 'root=/dev/hda1 console=ttyS0'

root = Root(clock='2GHz')
root.client = System(readfile='/dist/m5/system/boot/netperf-stream-client.rcS')
root.server = System(readfile='/dist/m5/system/boot/netperf-server.rcS')
root.etherlink = EtherLink(int1 = Parent.server.tsunami.etherint[0],
                           int2 = Parent.client.tsunami.etherint[0])
```


Using Configurations

- Language is semi-declarative
- Order on the command line may matter

```
% m5 -d output configs/tutorial/fullsys.py --DCache.size='64kB'
--EtherLink.speed='10Gbps' --Root.clock='4GHz'
```


Output Files

- Current Directory or -d <dir>
  - config.py, config.ini, config.out
  - console.<system>.sim_console
  - output
  - stats.txt
  - cpt.<number>/
- Database Output
  - M5 can output to a MYSQL database

Checkpointing

- --Serialize.cycle=<start cycle>
- --Serialize.period=<repeat interval>
- --Serialize.count=<# of checkpoints>

- M5 instruction
  - Insert special instruction into code to trigger a checkpoint to be dropped
  - Our benchmarks do this

Starting From a Checkpoint

- Same configuration as normal except you add:

--Root.checkpoint=<path>/cpt.<number>

- Checkpoints must be regenerated with some config changes
  - Most config changes that are architecturally visible (because the kernel may have behaved differently)
  - Physical memory size, new kernels

Sampling & Warm-up

- M5 can
  - dump statistics many times
  - aggregate statistics based on some event
    (keep stats according to kernel mode or user mode)
- Switch between CPU configurations

  Functional CPU ➔ Detailed CPU
  - Warm-up caches in a functional CPU, do measurements in a detailed CPU

Outline & Schedule

- Introduction & overview 2:00-2:20
- Running M5 2:20-2:45
- Full-System Workloads 2:45-3:00
- Current M5 object models
  - CPUs: simple, detailed 3:00-3:30
  - (break) 3:30-4:00
  - Memory System 4:00-4:35
  - I/O 4:35-4:55
- Extending M5 4:55-5:30

Full System Workloads

Lisa Hsu

Basic Operation

- Disk images
  - Raw copies of Linux disk image
  - Binaries to be run must be present on image
- rcS files (m5/configs/boot/*.rcS)
  - Exactly like normal boot scripts
  - Use them to start running a binary on the disk image, configure ethernet interfaces, etc.
  - Can also execute m5 instructions
  - Nice and flexible, since not compiled in
    - Specified in configuration by readfile='path/to/script.rcS'

See for yourself!

- Going into / of disk image and typing ls will show:

```
benchmarks  etc     lib         mnt      sbin  usr
bin         floppy  lost+found  modules  sys   var
dev         home    man         proc     tmp   z
```

- Snippet of .rcS file:

```
echo -n "setting up network..."
/sbin/ifconfig eth0 192.168.0.10 txqueuelen 1000
/sbin/ifconfig lo 127.0.0.1
echo -n "running surge client..."
/bin/bash -c "cd /benchmarks/surge && ./Surge 2 100 1 192.168.0.1 5"
echo -n "halting machine"
m5 exit
```

Up and Running Benchmarks

- All networking focused
- SpecWEB99
- Netperf
  - stream – a transmit benchmark
  - maerts – a receive benchmark
- In progress:
  - NFS (server works; client tuning needed)
  - iSCSI
  - video streaming

Adding Your Own Benchmarks

- Highly encouraged! ☺
  - Please share them with others!
- Since M5 is Alpha targeted, need to compile Alpha binaries
  - Cross-compiler can be downloaded from www.kegel.com/crosstool
  - Or, if you have an Alpha, use that
  - Add the benchmark binaries to disk image
  - Create .rcS file that executes the binary

Mounting Disk Images

- If you need to mount a disk image to change something (like add a benchmark binary)
- As root:

```
mount -o loop,offset=32256 myimage.img /mnt/point
```

- You can then manipulate the file system directly and copy in binaries
- Don't forget to unmount!

Example

- New benchmark: mybench
- Compile and put it in disk image:

```
cp mybench /mnt/point/benchmarks/mybench
```

- Create .rcS files:

```
#! /bin/sh
ifconfig eth0 192.168.0.1
echo "executing mybench"
eval /benchmarks/mybench
```

Outline & Schedule

- Introduction & overview 2:00-2:20
- Running M5 2:20-2:45
- Full-System Workloads 2:45-3:00
- Current M5 object models
  - CPUs: simple, detailed 3:00-3:30
  - (break) 3:30-4:00
  - Memory System 4:00-4:35
  - I/O 4:35-4:55
- Extending M5 4:55-5:30

CPU Models

Kevin Lim

CPU Models

- Models:
  - Simple CPU
  - Detailed CPU
- Key classes:
  - StaticInst – Decoded instruction
  - ExecContext – Execution context

Simple CPU Model

m5/cpu/simple/simple_cpu.{hh,cc}

- Uses of the SimpleCPU:
  - Warming up caches
  - Driving systems that do not require detailed modeling
- Ideal starting point to learn how CPU models work within M5
  - Simple overview of fetching, executing, and retiring instructions
  - Handles all the calls to support full system mode

Simple CPU Details

- Simulates an in-order 1 CPI machine
  - Stalls on I or D cache misses
  - Single threaded
  - Can roughly model a superscalar machine by ticking multiple times
- Can be extended to simple pipeline
  - Add in functional units with latencies

Functionality

- Execution driven
  - Tied closely with ExecContext class
- Works in full system and syscall emulation modes
  - Handles interrupts, traps, etc.

Detailed CPU Model

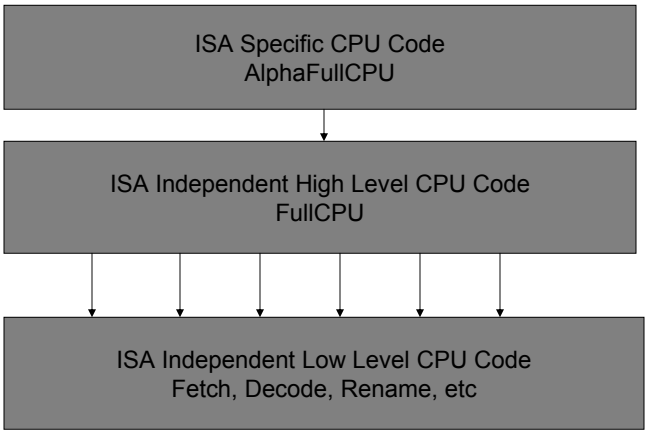
m5/cpu/o3/*, m5/encumbered/cpu/full

- Currently two detailed models within M5
  - Old model, distantly based on sim-ooo
    - Used for SMT and full system mode, but will eventually be phased out
  - New model
    - Will be the focus of this section
    - More closely couples execution and timing
    - Uses template policies

Timing accuracy

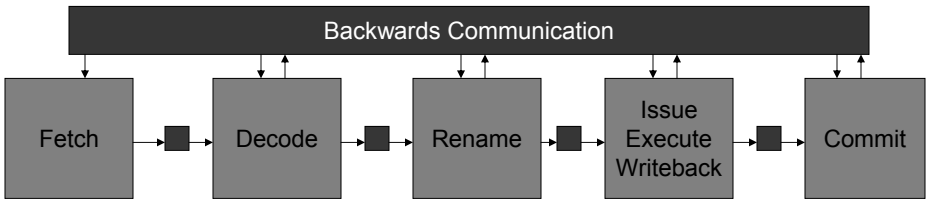
- Previous model executes instructions at fetch
  - Feeds instruction to timing backend
- New model executes at execute, modeling the timing for each pipeline stage
  - Important for coherence
  - UP and MP system studies
- Forces both timing and execution to be accurate

Code Hierarchy



Pipeline layout

- OoO pipeline, loosely based on Alpha 21264
- Low level structure:
  - Red is a time buffer



Time Buffers

- Similar to queues
  - Are `tick()`'d each CPU cycle
- Each pipeline stage places information into time buffer
  - Next stage reads info out of time buffer at appropriate cycle
- Used for both forwards and backwards communication

Time Buffer Use

- Time buffer class is templated
  - Its template parameter is the communication struct between stages
- Stages must communicate to each other via the time buffer
  - Avoids unrealistic interaction between pipeline stages

Template Policies

- Template policy classes used to define CPU policies
  - Gives full type information
  - Avoids virtual functions
- "Impl" class is passed in as template parameter to all classes
  - Impl defines all the important types, classes, pipeline stages, etc

Impl Template Policy

- Impl's are used to define specific CPU instances, down to the ISA
  - To create different types of CPUs, create a new Impl and define all of the specific types
- *_impl.hh files due to using templates in our infrastructure
  - *.cc files have instantiations

Future Directions

- New model will eventually add full system support, SMT support
- Will include a checker/verifier at commit
- Also will abstract away ISA specific functions
  - Lower levels of code can be shared across platforms

StaticInst Class

m5/cpu/static_inst.{hh,cc}

- Represents a decoded instruction
  - Has classifications of the inst
  - Corresponds to the binary machine inst
  - Decoded once
  - Only has static information
- Has all the methods needed to execute an instruction
  - Tells which regs are source and dest

Individual Use of StaticInst

- Contains the execute() function
  - Specific instruction classes override this
  - Python generates execute() for all insts
- Templated on the ISA
  - Different ISAs can have specific instantiations of StaticInst

DynInst Class

m5/cpu/base_dyn_inst.{hh, cc}

- Dynamic version of StaticInst
  - Used for detailed CPU model
  - Holds PC, results, renamed regs, etc
- Templated on CPU policy

ExecContext Class

m5/cpu/exec_context.{hh,cc}

- Represents total architectural state of a single thread in the system
  - PC, register values, memory, etc.
- Contains pointers to key classes
  - Everything needed to functionally execute an instruction

ISA Details

- Currently has a few ISA specific details
- Future direction has state being removed from ExecContext, and ExecContext serving mainly as an interface

Break

3:30 – 4:00

Feel free to ask us questions!

Outline & Schedule

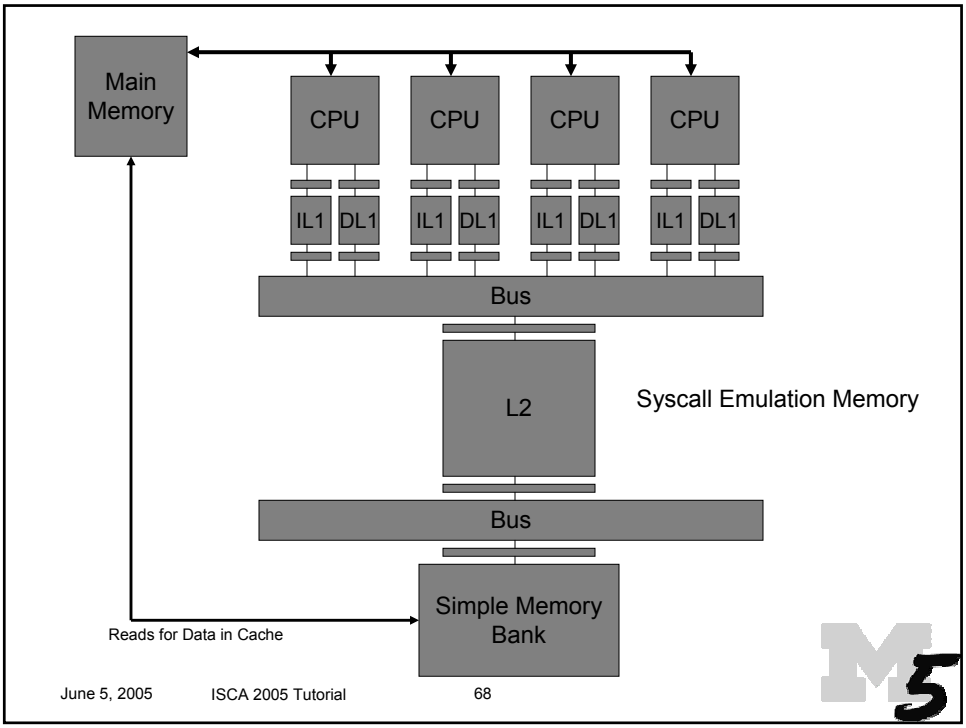
- ❑Introduction & overview 2:00-2:20
- ❑Running M5 2:20-2:45
- ❑Full-System Workloads 2:45-3:00
- ❑Current M5 object models
  - CPUs: simple, detailed 3:00-3:30
  - (break) 3:30-4:00
  - Memory System 4:00-4:35
  - I/O 4:35-4:55
- ❑Extending M5 4:55-5:30

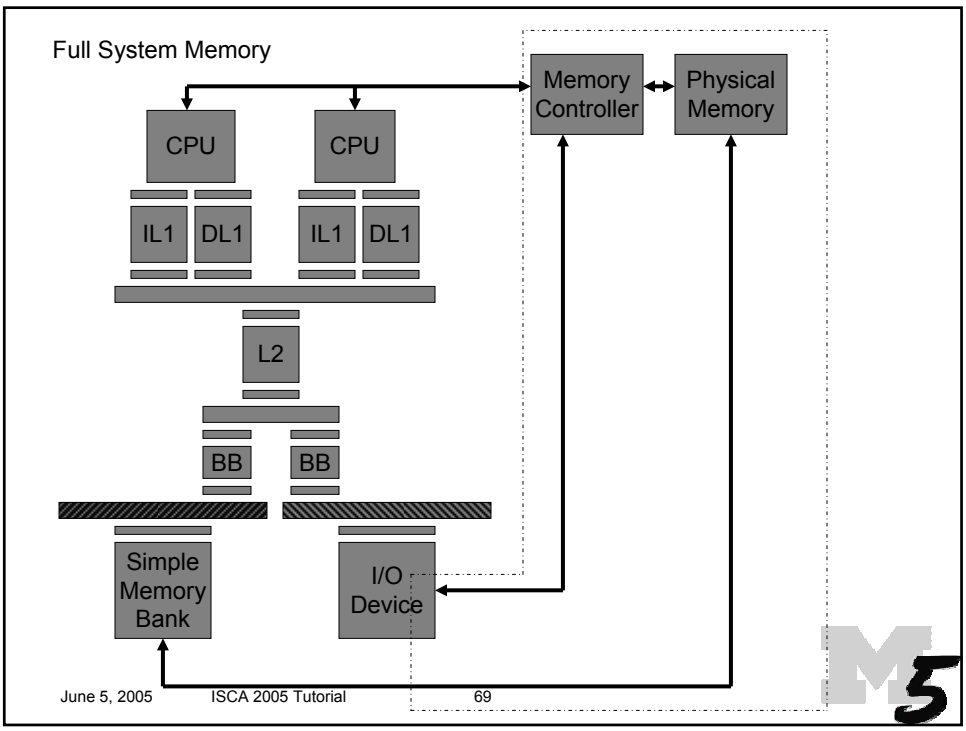
Memory System

Ron Dreslinski

Outline

- Overview
- Caches
- Coherence Support
- Buses
- Bus Interfaces
- Bus Bridges
- Walkthrough of Memory Call Graph


Main Memory
CPU
CPU
CPU
CPU
IL1
DL1
IL1
DL1
IL1
DL1
IL1
DL1
Bus
L2
Syscall Emulation Memory
Bus
Simple Memory Bank
Reads for Data in Cache
June 5, 2005
ISCA 2005 Tutorial
68


Full System Memory
Memory Controller
Physical Memory
CPU
CPU
IL1
DL1
IL1
DL1
L2
BB
BB
Simple Memory Bank
I/O Device
June 5, 2005
ISCA 2005 Tutorial
69

Overview

- Two versions of the memory system
  - Separate timing / functional models
    - Capability to have data in the timing caches
    - Memory tester object
  - New unified timing/functional access model
    - Works with simple CPU and exec-in-exec CPU
    - Doesn't support older full CPU model (execute at fetch)

Overview Cont.

- Supports atomic and event driven accesses
  - Atomic model
    - Each memory transaction is independent
    - Each transaction is followed completely through the memory system
    - Speeds simulation, useful when timing not important (miss stream)
  - Event driven model
    - Events generated for each transaction as it traverses the hierarchy
    - Multiple transactions interact in the memory system
    - Useful when timing information is important, or MP system interaction is desired

Functional Memory

m5/mem/functional/*

- There are several classes derived from functional memory including:
  - Full system
    - Physical memory
    - Memory controller
  - Syscall emulation
    - Main memory

Timing Memory

m5/mem/timing/*

- Simple memory bank
  - Configurable parameters:
    - Address range
    - Latencies
    - Snarf updates
    - Do writes (used for memory tester)

`MemReq` Obejcts

m5/mem/mem_req.{cc,hh}

- All memory interactions are described in terms of a memory request (**`MemReq`**) object
- Encapsulates all relevant information
  - Virtual and physical address
  - Request size
  - Requesting device
  - Etc.
- Makes memory system independent

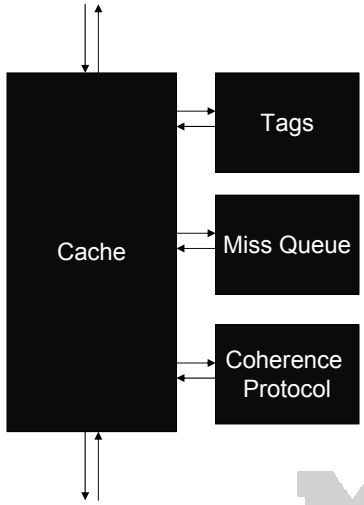
`MemReq` Flow

- CPU or device generates a **`MemReq`**
- **`MemReq`** is passed to the proper bus/cache
- Each cache creates a new **`MemReq`** to pass through the hierarchy, holding on to the request it needs to respond to
- Eventually the data requested is reached and the responses are processed

Caches

m5/mem/cache/*

- Templated on:
  - Cache tags (LRU, etc.)
  - Buffer type (blocking, MSHR)
  - Coherence protocol (uni, MSI, etc.)



Main Cache Parameters

- Size
- Associativity
- Block Size
- Latency
- Trace address

Cache Tags

m5/mem/cache/tags/*

- Holds blocks and block state
- Contains the replacement policy
  - Optimal
  - LRU
  - Generational (IIC)

Miss Queue

m5/mem/cache/miss/*

- Blocking buffer
  - Used to simulate a blocking cache
  - Speeds simulation in atomic bus model
- MSHR
  - Blocks when miss or writeback queue is full
  - Configurable parameters
    - Size of miss and writeback queues
    - Number of targets per MSHR

Coherence Support

m5/mem/cache/coherence/*

- Uni coherence model
  - Single Processor
  - Handles DMA invalidate forwarding up the cache hierarchy
    - CSHR's
    - Additional cache functions to parallel getMemReq, etc. but in the opposite direction
- MOESI based Model
  - Also has CSHR support invalidates

MOESI Bus Coherence

- Configurable parameters:
  - Protocol type (MSI,MOSI,MESI,MOESI)
  - Upgrades
- Support for memory to snarf updates
  - Memory object must be the slave device on the coherent bus

Coherence Limitations

- Currently only a single level coherence model exists
  - Coherent bus must be bus closest to CPU
  - Only one coherent bus allowed in a system
  - Only L1's allowed to be above coherence bus (doesn't forward snoops up the hierarchy)
  - Other levels of cache should be uni-coherent
- Coherence works in event driven mode if:
  - Ownership protocol is used or next level snarfs updates

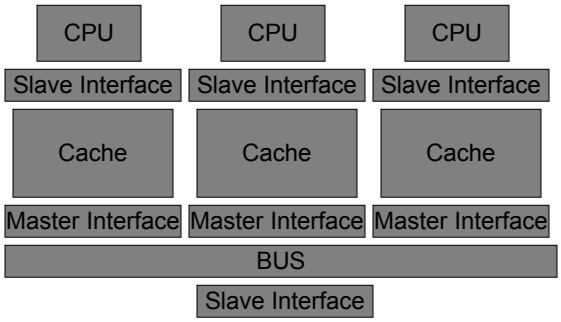
Buses

m5/mem/bus/*

- Support atomic or split-transaction
  - All timing in event driven mode done in bus
- Have separate address and data busses and an arbiter to determine events
- Configurable parameters:
  - Width
  - Clock rate

Bus Interfaces

- Generic interface between bus and memory objects
- Caches have master and slave interfaces



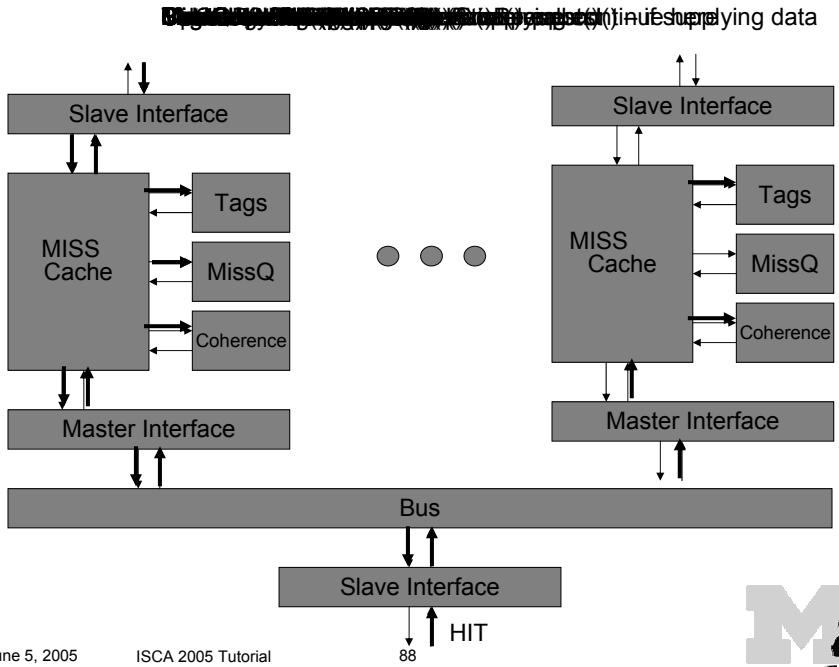
Bus Interfaces Cont.

- Master interface
  - Connected closer to memory
  - Initiates bus transactions on cache misses
  - Has snoop path to forward address bus requests to the cache
  - Responds with the data when snoops need to supply
- Slave interface
  - Connected closer to the CPU
  - Initiates bus transactions on coherence (`CSHR`) requests
  - Responds with the data, if a snoop didn't supply it

Bus Bridges

- Simple objects to connect two different speed busses
- Queues requests coming from either side and forwards them out the other when the arbiter grants the bus

Memory Flow



Outline & Schedule

- ❑Introduction & overview 2:00-2:20
- ❑Running M5 2:20-2:45
- ❑Full-System Workloads 2:45-3:00
- ❑Current M5 object models
  - CPUs: simple, detailed 3:00-3:30
  - (break) 3:30-4:00
  - Memory System 4:00-4:35
  - I/O 4:35-4:55
- ❑Extending M5 4:55-5:30

I/O Models

Ali Saidi and Lisa Hsu

Overview

- Device Basics
- Miscellaneous Devices
- Disk Model
- Network Model

Device Basics

dev/*

- All based on **`FunctionalMemory`**
  - **`PioDevice`** - Contains a pointer to the platform and pioInterface
  - **`DMADevice`** - Additionally contains a pointer to dmaInterface
  - **`PCIDev`** - PCI Configuration space, PCI interrupt handling
- Each device is sensitive to one or more address ranges (base + size)
  - Functional access with **`MemoryController::add_child()`**
  - Timing access with **`PioInterface::addAddrRange()`**
- A device implements a Read() and Write() for basic PIO reads and writes
- **`DMAInterface::doDMA()`** for DMA reads/writes

Miscellaneous Devices

- **`AlphaConsole`** - Back door into simulator for console code
- **`BadDev`** - Device panics on access
- **`CowDiskImage`** - Copy-on-write disk image
- **`PciConfigAll`** - PCI configuration space object
- **`PciDev`** - PCI device base class
- **`PacketFifo`** - FIFO for network packets
- **`SimConsole`** - Device that provides the console
- **`Tsunami`** - The tsunami platform that links together all it's devices
- **`TsunamiCChip`** - Tsunami interrupt controller
- **`TsunamiPChip`** - Tsunami PCI interface
- **`TsunamiIO`** - Legacy I/O devices (RTC, PIT, etc)
- **`Uart`** - Serial UART

PCI & Interrupts

- Tsunami platform
- Four physical PCI slots
- Only implement one PCI bus
  - Possible to implement second, just not done
- Devices need their own interrupt and device ID
  - No interrupt sharing allowed
  - Simulator will panic if detected
- PciConfigData
  - InterruptLine - needs to be different for each device (0x1c-0x1e)

Disk Interface

dev/ide_*

- Modeled as a IDE controller and separate IDE Disk(s)
- Copy-on-write layer in between
- Simple timing support
- Emulates a Intel IDE 2 channel controller
  - Can connect 4 IDE devices

Disk Images

- Can be created with mkblankimage.sh
- Disk images are contiguous blocks created with dd
- Unlike a normal image it needs to contain a partition table
- Can be mounted with the loopback device

Specifying Disk Image

```
class FilesetDisk(IdeDisk):
  raw_image = RawDiskImage(image_file =
            disk('fileset.img'), read_only=True)
  image = CowDiskImage(child = parent.raw_image,
                       read_only=False)

self.disk2 = LinuxSwapDisk(driveID='master')
self.ide = IdeController(disks=[parent.disk0,
  parent.disk1, parent.disk2],…
```


NIC Device Model

m5/dev/ns_gige*

- National Semiconductor DP83820
  - Gigabit Ethernet Full Duplex PCI controller
  - Their spec is actually public
- Modeled Device Features/Components:
  - PCI bus interface
  - Device registers
  - Tx/Rx FIFOs
  - Buffer Management Scheme
  - Receive Packet Filtering Logic
  - Checksum Offloading

NIC Device Model cont.

- Unmodified Linux driver will run on our model (linux/drivers/net/ns83820.c)
- If Linux doesn't use it, we don't model it
  - Except for Packet Filtering – it's modeled
  - Multiple Rx/Tx priority queues
  - Power Management schemes
  - FIFO drain thresholds
- But these are easy to add if desired

NIC Added Features

- Interrupt coalescing
  - Reduce interrupt load in a high bandwidth environment
    - collects interrupts for **`intr_delay`** time before poking CPU
- Rx/Tx FIFO size
  - Parameters to define Rx/Tx FIFO sizes
    - **`rx_fifo_size, tx_fifo_size`**
- Set in Python config object file
  - m5/python/m5/objects/Ethernet.py
  - NSGigE object description

Linux Mods for NIC

- Actual device has bug that does not allow unaligned copies
  - Fixed that in the device, and removed the associated workaround in the Linux driver
- Checksum offloading didn't work in every case in Linux, so we patched it.

Etherlink

m5/dev/etherlink.{cc/hh} m5/python/objects/Ethernet.py

- Configurable link
  - Link delay
  - Bandwidth
- Attached NICs
  - Connect any two NICs
- Packet dump
  - Dumps a pcap formatted Ethernet trace
    - Read with tcpdump, Ethereal

Outline & Schedule

Introduction & overview	2:00-2:20
Running M5	2:20-2:45
Full-System Workloads	2:45-3:00
Current M5 object models	
CPUs: simple, detailed	3:00-3:30
(break)	3:30-4:00
Memory System	4:00-4:35
I/O	4:35-4:55
Extending M5	4:55-5:30

Extending M5

Steve Reinhardt
Nate Binkert

Extending M5

❑Overview of M5 internals	Steve
❑Defining new objects	Steve
❑ISA Description Language	Steve
❑Debugging	Nate
❑Statistics	Nate

Execution Process

m5/sim/main.cc

- Process command-line args
- Build configuration
- Unserialize from checkpoint, if any
- Set up statistics
- Start processing events from event queue
- On processing SimExitEvent:
  - Dump statistics
  - Exit

Configuration Processing

- Configuration script is a Python program
- Python “m5” module in m5/python/m5
  - Object descriptions in m5/python/m5/objects
- C++ forks Python interpreter (m5/python/pyconfig.cc)
  - Send it Python scripts/code from cmd line
  - Python ends by printing final config
    - ini-file format for historical reasons (in config.ini)
    - Hope to get rid of this step soon

SimObject Construction

m5/sim/builder.*, m5/sim/configfile.*

- C++ builds **`ConfigHierarchy`** from .ini file
- Linking in a .o file automatically registers object in global table
  - Via **`REGISTER_SIM_OBJECT()`** macro
  - Magic of C++ global object constructors
- Two-pass initialization
  - First pass: walk tree, call builder "create" method
    - Defined by **`CREATE_SIM_OBJECT()`** macro
    - All parameter values available
    - Calls C++ constructor
  - Second pass: call **`init()`** methods
    - Guaranteed that all other objects are constructed

Memory State Initialization

- Side effect of object creation
  - Process object for syscall emulation
    - **`LiveProcess`** for direct emulation (m5/sim/process.*)
    - **`EIOProcess`** for SimpleScalar EIO trace playback (m5/non-free/eio/*)
    - Auto-detects Tru64 vs Linux binaries
  - System object for full-system mode
    - **`LinuxSystem`**, **`Tru64System`** (m5/kern/*)
    - Console/PAL code
    - Kernel image
- m5/base/loader/* has aout, ecoff, elf loader code, plus symbol table support

Serialization

m5/sim/serialize.{cc,hh}

- Create/restore state checkpoints
- Serializable is base of Event & SimObject
  - defines `serialize()`, `unserialize()` methods
  - override to save/restore object state
  - .ini-format text file
- If checkpoint is specified, M5 will call `unserialize()` on all objects after creation
  - State identified by object name (system0.cpu0)
  - OK if no checkpointed state (e.g., added cache)
  - Detailed CPU model can unserialize from SimpleCPU serialized state
- Common error: adding field to object and not updating `serialize()`/`unserialize()`

Events

m5/sim/eventq.*

- Event object is abstract superclass
- Derive new subclass for specific event
  - Add fields for event-specific data
  - Override `process()` method for action
- `schedule(Tick t)` puts on event queue
- Events may be statically or dynamically allocated
  - Setting AutoDelete flag will call delete after processing

Creating New SimObjects

- Derive C++ class from SimObject
  - m5/sim/sim_object.{cc,hh}
- Define parameters in a .py file
  - in m5/python/m5/objects directory
- C++ needs parameter/creation boilerplate
  - Ugly macros in m5/sim/builder.hh
    - {BEGIN,END}_{DECLARE,INIT}_SIM_OBJECT_PARAMS
    - {CREATE,REGISTER}_SIM_OBJECT
  - Plan to be cleaning this up before long…

ISA Description Language

arch/isa_parser.py, arch/alpha/isa_desc

- Custom domain-specific language
- Defines decoding & behavior of ISA
- Generates C++ code
  - Scads of **`StaticInst`** subclasses
  - **`decodeInst()`** function
    - Maps machine inst. to **`StaticInst`** instance
  - Multiple scads of **`execute()`** methods
    - Cross-prod. of CPU models and **`StaticInst`** subclasses

Definitions etc.

```
def bitfield OPCODE   <31:26>;
def bitfield RA       <25:21>;
def bitfield RB       <20:16>;
def bitfield INTFUNC  <11: 5>; // function code
def bitfield RC       < 4: 0>; // dest reg

def operands {{
    'Ra': IntRegOperandTraits('uq', 'RA', 'IsInteger', 1),
    'Rb': IntRegOperandTraits('uq', 'RB', 'IsInteger', 2),
    'Rc': IntRegOperandTraits('uq', 'RC', 'IsInteger', 3),
}}

def format LoadAddress(code) {{
   // Python code here...
}}

def format IntegerOperate(code) {{
   // Python code here...
}}
```

Instruction Decode & Semantics

```
decode OPCODE {
   format LoadAddress {
        0x08: lda({{ Ra = Rb + disp; }});
        0x09: ldah({{ Ra = Rb + (disp << 16); }});
   }
   format IntegerOperate {
        0x10: decode INTFUNC {
            0x00: addl({{ Rc.sl = Ra.sl + Rb_or_imm.sl; }});
            0x20: addq({{ Rc = Ra + Rb_or_imm; }});
            0x22: s4addq({{ Rc = (Ra << 2) + Rb_or_imm; }});
            0x32: s8addq({{ Rc = (Ra << 3) + Rb_or_imm; }});
            // etc.
        }
   }
  // etc.
}
```

Key Features

- Very compact representation
  - Most instructions take 1 line of C code
  - 2639 lines of isa_desc → 39K lines of C++
    - 18K generic, 11K for each of 2 CPU models
  - Characteristics auto-extracted from C
    - source, dest regs; func unit class; etc.
  - `execute()` code customized for CPU models
- Amazingly well documented (for us, anyway)
  - See doxygen docs

Debugging M5

- DPRINTF and Tracing
- Instruction Tracing
  - rundiff
- Using the Debugger
- Remote GDB

Tracing

m5/base/trace.{cc,hh} m5/base/traceflags.py

- printf is a nice debugging tool
- Keep good printfs for tracing
- Lots of debug output is a very good thing
- Add new flags to traceflags.py
  - Individual flags in the baseFlags array
  - Groups of flags in the compoundFlagMap dict
- Fetch, Decode, Ethernet, IPI, TLB, DMA, Bus, Cache, Loader, AlphaConsole, etc…

Tracing

m5/base/trace.{cc,hh} m5/base/traceflags.py

```
DPRINTF(Flag, "normal printf %s\n", "arguments");

Command line flags:
m5.opt --Trace.flags="Space Separated List"

From gdb:
(gdb) call setTraceFlag("Flag")
(gdb) call clearTraceFlag("Flag")
```

Instruction Tracing

- --Trace.flags="InstExec"
- --ExecutionTrace.speculative=True
  - capture speculative instructions
- --ExecutionTrace.print_cycle=True
  - print cycle number

Using GDB with M5

```
% gdb m5/build/ALPHA_SE/m5.debug
(gdb) run m5-test/test1/run.ini --debug:break_cycles="1000 2000"
Starting program: /z/stever/bk/m5/build/ALPHA_SE/m5.debug m5-
   test/test1/run.ini --debug:break_cycles="1000 2000"
Starting simulation...
Program received signal SIGTRAP, Trace/breakpoint trap.
0xfffffe002 in ?? ()
(gdb) p curTick
$1 = 1000
(gdb) c
Program received signal SIGTRAP, Trace/breakpoint trap.
0xfffffe002 in ?? ()
(gdb) p curTick
$2 = 2000
(gdb) call sched_break_cycle(3000)
(gdb) c
Program received signal SIGTRAP, Trace/breakpoint trap.
0xfffffe002 in ?? ()
(gdb) p curTick
$3 = 3000
(gdb)
```


Remote Debugging

```
% ~/m5/build/ALPHA_FS/m5.debug ~/m5/configs/fullsys/run.py
M5 Simulator System
Copyright (c) 2001-2005
The Regents of The University of Michigan
All Rights Reserved

This code is part of the M5 simulator, developed by Nathan Binkert,
Erik Hallnor, Steve Raasch, and Steve Reinhardt, with contributions
from Ron Dreslinski, Dave Greene, Lisa Hsu, Ali Saidi, and Andrew
Schultz.

M5 compiled on May 31 2005 11:43:24
M5 executing on ziff.eecs.umich.edu
M5 simulation started Tue May 31 11:45:02 2005
Listening for console connection on port 3456
      0: system.tsunami.io: Real-time clock set to Sun Jan  1 00:00:00 2006
command line: /n/ziff/z/binkertn/build/head/ALPHA_FS/m5.debug
   /n/ziff/z/binkertn/research/m5/head/configs/fullsys/run.py

Listening for remote gdb connection on port 7000
warn: Entering event queue.  Starting simulation...
```

Remote Debugging

```
% gdb-linux-alpha arch/alpha/boot/vmlinux
... gdb banner ...
This GDB was configured as "--host=i686-pc-linux-gnu --
   target=alpha-linux"... (no debugging symbols found)...
(gdb) set remote Z-packet on [ This can be put in .gdbinit ]
(gdb) target remote ziff:7000
Remote debugging using ziff:7000
0xfffffc0000496844 in strcasecmp (a=0xfffffc0000b13a80 "", b=0x0)
   at arch/alpha/lib/strcasecmp.c:23
23              } while (ca == cb && ca != '\0');
(gdb)
```

The M5 Statistics Package

- Statistics types
  - Scalar<>
  - Vector<>
  - Formula
  - Distribution<>
  - VectorDist<>
- M5 has phases, once it moves to the running phase, no new stats

Statistics Example

.hh file

```
class MySimObject : public SimObject
{
  private:
    Stats::Scalar<> txBytes;
    Stats::Formula txBandwidth;
    Stats::Vector<> syscall;

  public:
    void regStats();
};
```

.cc file (regStats)

```
txBytes
    .name(name() + ".txBytes")
    .desc("Bytes Transmitted")
    .prereq(txBytes)
    ;

txBandwidth
    .name(name() + ".txBandwidth")
    .desc("Transmit Bandwidth (bits/s)")
    .precision(0)
    .prereq(txBytes)
    ;

txBandwidth = txBytes *
    Stats::constant(8) / simSeconds;

syscall
    .init(SystemCalls<Linux>::Number)
    .name(name() + ".syscall")
    .desc("number of syscalls executed")
    .flags(total | pdf | nozero | nonan)
    ;
```

Statistics Output

client.tsunami.etherdev.txBandwidth	4302720		
client.tsunami.etherdev.txBytes	13446		
server.tsunami.etherdev.txBandwidth	4684921600		
server.tsunami.etherdev.txBytes	14640380		
sim_seconds	0.025000		
server.cpu.kern.syscall	492		
server.cpu.kern.syscall_1	189	38.41%	38.41%
server.cpu.kern.syscall_2	249	50.61%	89.02%
server.cpu.kern.syscall_3	54	10.98%	100.00%

Wrap-Up

Steve Reinhardt

Thank You!

- We hope you found this tutorial useful
- We hope you find M5 useful too
- We'd love to work with you to make M5 even more useful to the community
- We value your feedback
  - Please fill out a questionnaire
  - Hand it to one of us
    - On your way out, or during ISCA

Keep In Touch

- Come talk to us at ISCA
- Check http://m5.eecs.umich.edu for updates
- Use, subscribe to our mailing lists:
  - m5sim-users@lists.sourceforge.net
  - m5sim-announce@lists.sourceforge.net