# Using the M5 Simulator

## ASPLOS XIII

Ali Saidi  Lisa Hsu  Kevin Lim  Steve Reinhardt[†]
Nathan Binkert[‡]  Stephen Hines[⋆]  Gabe Black
Gary Tyson[⋆]

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
[†]Also with Reservoir Labs, Inc.
[‡]HP Labs
[⋆]Florida State University

March 2nd, 2008



# Welcome!

- We're glad you're here!
- This tutorial is for **you**
  - Please ask questions! Don't save them for the break!
  - We intend the focus to be audience driven

# Outline







# Introduction & Overview

## Introduction & Overview

Ali Saidi

# 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

## View #1

- A framework for event-driven simulation
  - Events, objects, statistics, configuration

## View #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
  - At least three other “simulators” have been created using #1





# 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 with detailed I/O modeling
  - Multiple system simulation
- Almost entirely original code
  - Some SimpleScalar & SimOS used for bootstrapping
  - Now entirely replaced / segregated / deprecated
- No premeditated distribution plans
  - Just hacking together the system *we* wanted

# Key M5 Attributes

- Heavily object-oriented
  - Key to modularity, flexibility
- Necessarily complex
  - ∼91K lines of C++, ∼21K 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
- Let us know how we can help you contribute
  - Public wiki is up at http://www.m5sim.org
  - *Please* submit patches and additional features
  - Ability to add modules with `EXTRAS=`
  - The more active the community is, the more successful M5 will be!





# A Peek Inside

- *Very* brief overview of a few key concepts:
  - Objects
  - Events
  - Modes

# 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
    - Normally picoseconds in our examples
- Objects schedule their own events
  - Flexibility for detail vs. performance trade-offs
- E.g., a CPU typically schedules event at regular intervals
  - Every cycle or every *n* picoseconds
  - Won’t schedule self if stalled/idle

# A Peek Inside 3: Modes

- M5 has two fundamental modes
- Full system (FS)
  - For booting operating systems
  - Models bare hardware, including devices
  - Interrupts, exceptions, privileged instructions, fault handlers
- Syscall emulation (SE)
  - For running individual applications, or set of applications on MP/SMT
  - Models user-visible ISA plus common system calls
  - System calls emulated, typ. by calling host OS
  - Simplified address translation model, no scheduling
- Selected via compile-time option
  - Vast majority of code is unchanged, though





# Current Status

- Two+ CPU models
  - Simple functional
  - Execute-in-Execute OoO Model
- Memory system
  - Uniform connection model based on “ports”
  - Packet-based interface
    - Bus model: split transactions, snooping coherence
    - Enables point-to-point network models (future work)
- New Cache model
  - Large portions of the cache have been re-written
  - Caches now support multi-level coherence
  - Can create almost arbitrary levels of caches

# ISA Support

- Alpha
  - Syscall-emulation: Tru64 or Linux
  - Full-system: Compaq Tsunami, boots Linux, FreeBSD
  - Ethernet, IDE devices models; prebuilt benchmarks
- SPARC
  - Syscall-emulation: Linux
  - Full-system: Niagara T1: Solaris
  - Only uni-processor simulation in Solaris
    - MP code is done, but completely untested
    - No real device models, but documentation now available
- ARM, MIPS & X86
  - Syscall-emulation: Linux
  - Full-system: In progress





# Short-term To-do/Wish List

- Finish SPARC, ARM, MIPS & X86 full-system support
- Finishing touches on memory system
  - Get rid of some minor cruft
  - Support non-bus interconnects, directory coherence
- Continue Richer C++/Python interface
  - Much more flexibility for non-performance-critical parts
- More full-system benchmarks
- More community involvement: wiki, on-line repository

# Outline







# Compiling & Running M5

## Compiling & Running M5

Steve Reinhardt

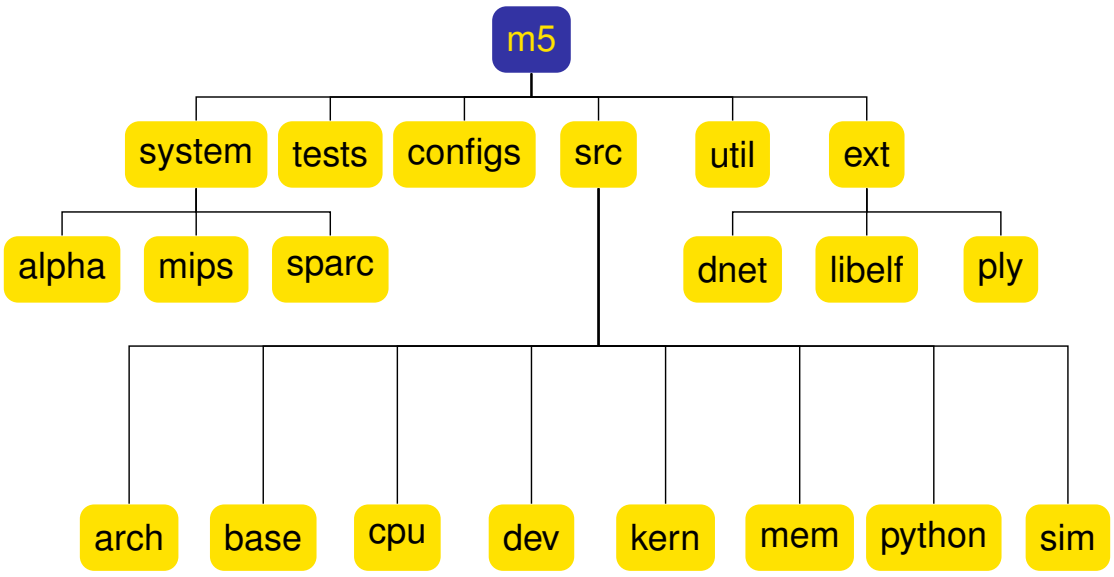
# Compiling & Running M5

- Source tree
- Building executables
- Running simulations
- Output files
- M5 Scripting
- Checkpointing
- Sampling & warm-up





# Source Tree Organization




# M5 Source Tree Organization

- configs: sample m5 scripts
- src/arch: architecture definition & ISA-specific components
- src/base: general data structures/facilities
- src/python: Python config code
- src/cpu, src/mem, src/dev: specific models
- src/sim: simulator base functionality
- system: platform specific code (palcode, firmware, bios, etc.) — packaged separately
- test: regression tests
- util: utility programs





# Building Executables

- Platforms
  - Linux, BSD, MacOS, Solaris, CYGWIN, etc
  - Little endian machines
    - Some architectures support big endian
  - 64-bit machines help a lot
- Tools
  - GCC/G++ 3.4+
    - Recently tested with 4.0-4.2
  - Python 2.4+
  - SCons 0.96.91+
    - Latest version (0.97.0d20071212) has problems
    - http://www.scons.org
  - SWIG 1.3.28+
    - http://www.swig.org

# Compile Targets

- build/<config>/<binary>
  - configs
    - ALPHA_SE (Alpha syscall emulation)
    - ALPHA_FS (Alpha full system)
    - MIPS_SE (MIPS syscall emulation)
    - SPARC_SE (SPARC syscall emulation)
    - SPARC_FS (SPARC full system)
    - X86_SE (X86 syscall emulation)
    - X86_FS (X86 full system)
    - ARM_SE (ARM syscall emulation)
    - can also define new configs with different compile option settings
  - binary
    - m5.debug – debug build
    - m5.opt – optimized build + debugging symbols + tracing
    - m5.fast – optimized build - no debugging





# Sample Compile

```
% scons build/ALPHA_FS/m5.debug

scons: Reading SConscript files ...
Checking for C header file fenv.h... yes
Building in /tmp/m5/build/ALPHA_FS
Options file /tmp/m5/build/options/ALPHA_FS not found,
  using defaults in build_opts/ALPHA_FS
Compiling in ALPHA_FS with MySQL support.
scons: done reading SConscript files.
scons: Building targets ...
g++ -o build/ALPHA_FS/base/circlebuf.do -c -pipe -fno-strict-aliasing
    -Wall -Wno-sign-compare -Werror -Wundef -g3 -gdwarf-2 -O0
    -DTHE_ISA=ALPHA_ISA -DDEBUG -Iext/dnet -I/usr/include/python2.4
    -Ibuild/libelf/include -I/usr/include/mysql -Ibuild/ALPHA_FS
    build/ALPHA_FS/base/circlebuf.cc
...
```

# Running Simulations

```
Usage:
m5.opt [m5 options] script.py [script options]
 --version            show program's version number and exit
 --help, -h           show this help message and exit
 ...
-d, --outdir=DIR     Set the output directory to DIR [Default: .]
 --interactive, -i    Invoke the interactive interpreter after running script
 --pdb                Invoke the python debugger before running the script
 -p, --path=<path>    Prepend PATH to the system path when invoking the script
 --quiet, -q          Reduce verbosity
 --verbose, -v        Increase verbosity

 --debug-break=TIME   Cycle to create a breakpoint

 --stats-file=FILE    Sets the output file for statistics [Default: m5stats.txt]

 --trace-help         Print help on trace flags
 --trace-flags=FLAG[,FLAG]
                      Sets the flags for tracing (-FLAG disables a flag)
 --trace-start=TIME   Start tracing at TIME (must be in ticks)
 --trace-file=FILE    Sets the output file for tracing [Default: cout]
 --trace-ignore=EXPR  Ignore EXPR sim objects

 <script file>        script file name which ends in .py. (Normally you can
                      run <script file> --help to get help on that script files'
                      parameters.
```





# Sample Run

```
% ./build/ALPHA_FS/m5.opt configs/example/fs.py
M5 Simulator System

Copyright (c) 2001-2008
The Regents of The University of Michigan
All Rights Reserved


M5 compiled Feb  2 2008 00:35:05
M5 started Mon Feb 11 12:26:06 2008
M5 executing on mbp.local
command line: ./build/ALPHA_FS/m5.opt configs/example/fs.py
Global frequency set at 1000000000000 ticks per second
      0: system.tsunami.io.rtc: Real-time clock set to Thu Jan  1 00:00:00 2009
Listening for system connection on port 3456
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
warn: Entering event queue @ 0.  Starting simulation...
```

# Output Files

- Output directory
  - config.ini
  - m5stats.txt
  - cpt.<number>/
  - console.<system>.sim_console (FS only)
- Database output
  - M5 can output stats to a MySQL database
  - Automatic graph generation (m5/utils/stats)





# m5term

- Connect to the simulated console interface (FS mode)

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

# Sample Terminal Output

```
%  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
#
```





# M5 Scripts

- Python
- Config objs mapped to simulator objs
- No need for scripts to generate multiple configuration files
  - All logic for running many simulations contained in a single configurable script!
- Variables with units are enforced
  - Latency must be ’2ns’, not simply 2
- Define and parse parameters in the script

```
import os, optparse, sys
import m5
from m5.objects import *

parser = optparse.OptionParser(option_list=m5.standardOptions)
parser.add_option("-t", "--timing", action="store_true")
(options, args) = parser.parse_args()
if args:
    sys.exit(’too many arguments’)
parser.parse_args()
```

# Sample Configuration

```
class System(LinuxSystem):
    if options.timing:
        cpu = DetailedCPU()
    else:
        cpu = SimpleCPU()
    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()
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])

m5.instantiate(root) # instantiate structure
code = m5.simulate() # simulate until termination

print 'Exiting @ cycle', m5.curTick(), 'because', code.getCause()
```





# Checkpointing

- Creating checkpoints
  - Script commands
  - M5 instruction
    - Insert special instruction into code to trigger a checkpoint to be dropped
    - Our benchmarks do this
- Running checkpoints
  - Same configuration as normal, just add a load checkpoint command to your script.
  - Must regenerate checkpoints 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

- Sampler object can switch CPU models on the fly
  - E.g., functional ⇒ detailed
  - Statistics dumped & reset at switch
  - Warm-up caches with a functional CPU, do measurements with a detailed CPU
- Can dump and/or reset statistics at other points too
  - E.g., measure statistics across disjoint intervals





# Simpoints & M5

- Preliminary support for easy use of Simpoints with M5
  - Some parameters/usage may change a little bit
  - Right now only support Alpha
- Allows you to generate a checkpoint at the beginning of a simpoint or run a single-simpoint easily from command line
- Simpoints taken from:

  `http://www-cse.ucsd.edu/čalder/simpoint/points/standard/spec2000-single-std-100M.html`
- Only single-phase Simpoints
  - Multi-phase Simpoints in the future

# Using Simpoints

**Create a checkpoint at instruction 5000000 / simpoint**

```
m5.fast  configs/example/se.py --bench=gcc_integrate --take-checkpoint=5000000 \
         --at-instruction
m5.fast  configs/example/se.py --bench=gcc_integrate --take-checkpoint=0 --simpoint
```

**Resume a checkpoint created at instruction 5000000/simpoint; run for 100000 instructions**

```
m5.fast configs/example/se.py --bench=gcc_integrate -d --caches --l2cache  \
        --checkpoint-restore=5000000 --at-instruction --max-inst=100000
m5.fast configs/example/se.py --bench=gcc_integrate -d --caches --l2cache \
        --checkpoint-restore=0 --simpoint  --max-inst=100000
```

**Fast forward in atomic mode for 5000000 and run detailed for 100000 instructions**

```
m5.fast configs/example/se.py --bench=gcc_integrate -d  --caches --l2cache
        --fast-forward=5000000 --max-inst=100000
```

**Fast forward for 5000000, warm-up caches for 1000000, and execute detailed for 100000 instructions**

```
m5.fast configs/example/se.py --bench=gcc_integrate --standard_switch --caches \
        --fast-forward=5000000  --max-inst=100000  --warmup-insts=1000000
```





# Outline



# Full System Workloads

## Full System Workloads

Lisa Hsu





# Up and Running Benchmarks

- All networking focused
- Surge
- SpecWEB99 (called SurgeSpecweb)
- iSCSI
  - Initiator
  - Target
- Netperf
  - stream – a transmit benchmark
  - maerts – a receive benchmark

# Two Key Components

- Disk images
  - Complete image (partition table and partition image in one)
  - M5 mounts this as its root filesystem.
  - Binaries to be run must be present on image
- rcS files (**`m5/configs/boot/*.rcS`**)
  - Exactly like normal boot scripts
  - Nice and flexible, read at runtime
  - These do NOT reside on the disk image
    - Specified in config file **`m5/configs/common/Benchmarks.py`**
  - Use them to start running a binary on the disk image, configure Ethernet interfaces, execute m5 instructions, etc.





# 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
```

# Adding Your Own Benchmarks

- Full system simulation currently supports Alpha
  - Need to compile Alpha binaries (www.kegel.com/crosstool)
  - Or, we are providing a crosstool on the CD
  - If you have an Alpha, use that
- Add the benchmark binaries to disk image
- Create .rcS file that executes the binary
- Please share them with others!





# Mounting Disk Images

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

**As root:**

```
mount -oloop,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



# I/O Models

## I/O Models

Lisa Hsu





# Overview

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

# Device Basics `src/dev/*`

- All based on **`MemObject`**
  - **PioDevice** - Acts only as a target for requests with **`pioPort`**
  - **DMADevice** - Can also act as a master with a **`dmaPort`**
  - **PCIDev** - PCI Configuration space, PCI interrupt handling
- Each device responds to one or more address ranges
- A device implements a **`read()`** and **`write()`** for basic PIO reads and writes
- **`DMADevice::dmaWrite(), dmaRead()`** 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 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
  - Set **`InterruptLine`** parameter of PciConfigData to change





# Disk Interface

`dev/ide_*.{hh,cc}`

- Modeled as an IDE controller and separate IDE Disk(s)
- Copy-on-write layer in between
- Simple timing support
- Emulates an 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

## In FSConfig.py

```
self.disk0 = CowIdeDisk(driveID='master')
self.disk2 = CowIdeDisk(driveID='master')
self.disk0.childImage(mdesc.disk())
self.disk2.childImage(disk('linux-bigswap2.img'))
```

## In Benchmarks.py, to specify default image:

```
def disk(self):
        if self.diskname:
            return disk(self.diskname)
        else:
            return env.get('LINUX_IMAGE', disk('linux-latest.img'))
```

## In Benchmarks.py, to specify an image for a specific benchmark:

```
'PovrayBench':  [SysConfig('povray-bench.rcS', '512MB', 'povray.img')],
```

# NIC Device Model `src/dev/ns_gige.{hh,cc}`

- National Semiconductor DP83820
  - Unmodified Linux driver will run on it
  - Actual hardware only allows DMA on 8-byte boundary
    - Our model/Linux patch allows arbitrary DMA
  - Added interrupt coalescing, configurable FIFO sizes
  - If Linux driver didn't use it, model doesn't support it
- Modeled Device Features/Components:
  - PCI bus interface
  - Device registers
  - Tx/Rx FIFOs
  - Buffer Management Scheme
  - Receive Packet Filtering Logic
  - Checksum Offloading





# NIC Device Model `src/dev/i8254xGBe.{hh,cc}`

- Intel 82574GI
  - Linux driver will run with TSO turned off
  - Hope to implement TSO in model soon
  - If Linux driver didn't use it, probably not modeled correctly
- Model doesn't have any statistics yet
  - Hope to add them soon

# Etherlink

`m5/dev/etherlink.{hh,cc}`

- Configurable
  - Link delay
  - Bandwidth
- Point-to-point
  - Connect any two NICs
- Packet dump
  - Dumps a pcap formatted Ethernet trace
    - Read with tcpdump, Wireshark





# CPU Models

## CPU Models

Kevin Lim

# CPU Section Overview

- Models:
  - Simple CPU
  - O3 CPU
- Key classes:
  - StaticInst – Decoded instruction
  - DynInst – Stores dynamic information
- Architected state:
  - Interface:
    - ThreadContext – External interface
    - ExecContext – ISA Interface
  - Implementation:
    - SimpleThread





# Simple CPU Model `src/cpu/simple/base.{hh,cc}`

- Simulates a single-threaded in-order 1 CPI machine
- Uses of the SimpleCPU:
  - Warming up caches
  - Driving systems that do not require detailed modeling
- Ideal starting point to learn how CPU models work in M5
  - Simple overview of fetching, executing, and retiring instructions
  - Handles all the calls to support full system mode

# Simple CPU Subclasses

- AtomicSimpleCPU
  - Memory accesses are atomic
  - Latency of cache accesses are an estimate
  - Fastest functional simulation
  - Can roughly estimate superscalar CPU by ticking multiple times
- TimingSimpleCPU
  - Memory accesses use timing path
  - CPU waits until memory access returns
  - Fast, provides some level of timing





# O3 CPU Model

`src/cpu/o3/*`

- Detailed out-of-order CPU
  - IQ, ROB, LSQ
  - Renaming with a physical register file
  - Functional units with varying latencies
  - Branch prediction
  - Memory dependence prediction
- Syscall emulation support
  - Alpha, MIPS, SPARC ISAs
  - Simultaneous multithreading (SMT), UP and MP
- Full-system support
  - Alpha ISA
  - Supports UP and MP booting into Linux

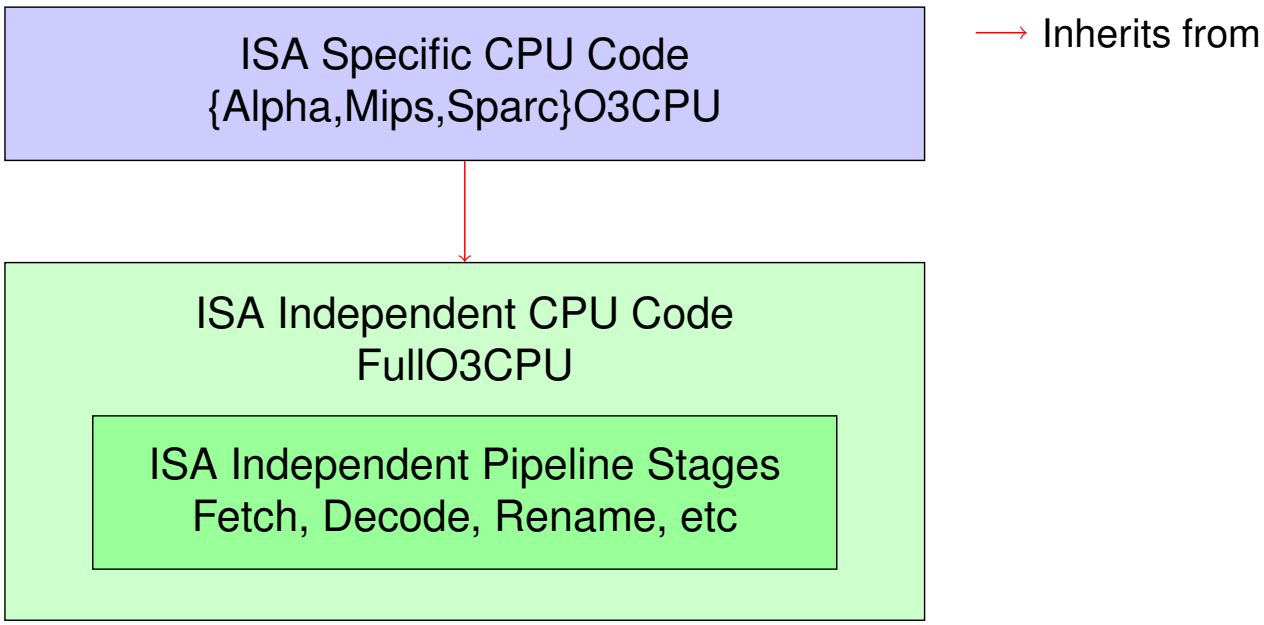
# Timing accuracy

- Many models execute instructions at the beginning or end of a pipeline
  - "Functional-first" or "timing-first"
  - Trades-off timing interaction for other details
- O3 CPU executes at execute, modeling the timing for each pipeline stage
  - Important for coherence, I/O
  - UP and MP system studies
- Forces both timing and execution to be accurate



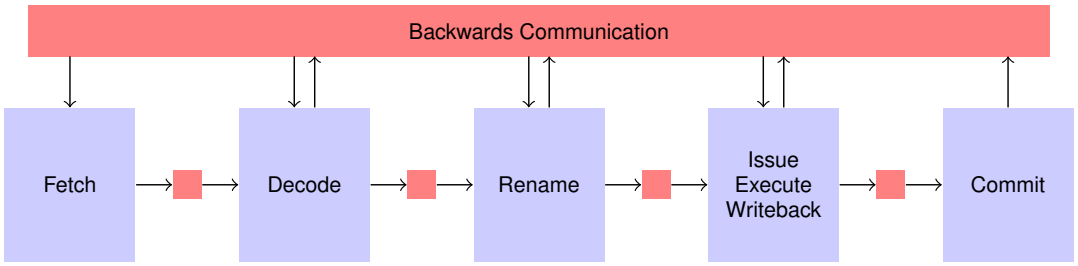


# Code architecture




# Pipeline layout

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








# Time Buffers

`src/base/timebuf.hh`

- Similar to queues
  - Are **`advance()`**'d each CPU cycle
- Each pipeline stage places information into time buffer
  - Next stage reads from time buffer by indexing into appropriate cycle
- Used for both forwards and backwards communication
  - Avoids unrealistic interaction between pipeline stages
- Time buffer class is templated
  - Its template parameter is the communication struct between stages

# Template Policies in the O3 CPU

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





# Template Policy Example

- Template policy code example:

```
template<class Impl>
class  DefaultFetch  {
    typedef typename Impl::BP  BranchPred;
    ...
};
```

- DefaultFetch is able to obtain full type of the Branch Predictor
  - Can extend to CPU stages, CPU pointer, instruction types, etc.

# O3 CPU Features on the Horizon

- Full system support for SPARC, MIPS
  - Currently Alpha is the only one supported
  - Gabe Black, Korey Sewell have done lots of work for ISA support
- SMT in full system mode





# StaticInst Class `src/cpu/static_inst.{hh,cc}`

- Represents a decoded instruction
  - Has classifications of the inst
  - Corresponds to the binary machine inst
  - Only has static information
- Has all the methods needed to execute an instruction
  - Tells which regs are source and dest
  - Contains the **`execute()`** function
  - ISA parser generates **`execute()`** for all insts

# DynInst Class `src/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
- Provides ExecContext interface





# Architected state

- Two main interfaces for architected state
  - ThreadContext
  - ExecContext
- Implementation of state and interface:
  - SimpleThread

# ThreadContext

`src/cpu/thread_context.hh`

- Interface for accessing total architectural state of a single thread
  - PC, register values, etc.
- Used to obtain pointers to key classes
  - CPU, process, system, ITB, DTB, etc.
- Abstract base class
  - Each CPU model must implement its own derived ThreadContext





# ExecContext

`src/cpu/exec_context.hh`

- Implicit interface used by ISA code to access CPU state
- ExecContext is not derived from
  - File is for documentation only
- Implementations:
  - SimpleCPU class
  - DynInst class

# SimpleThread

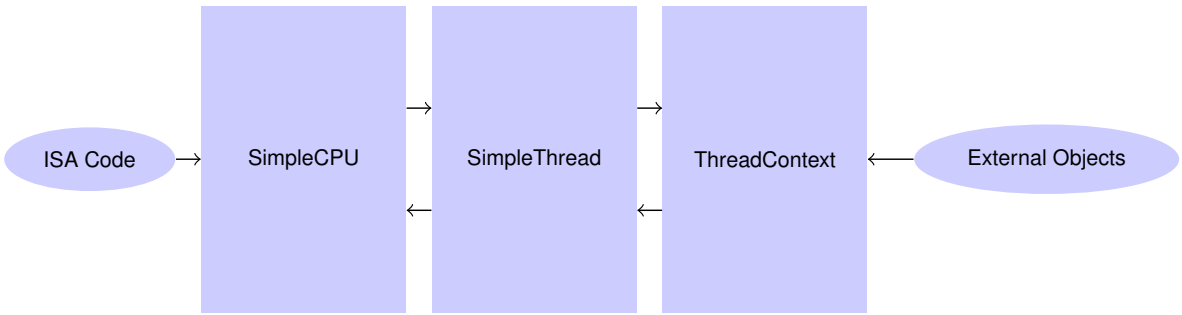
`src/cpu/simple_thread.hh`

- Provides all state necessary for execution
  - Holds all architectural state
  - Sufficient for simple CPU models
- Provides same functions as ThreadContext interface
  - Allows for a ThreadContext proxy to call functions on SimpleThread
- Used to be ExecContext in M5 v1.x





# Interface Example




# Memory System

## Memory System

Steve Reinhardt





# Outline

- Memory System Overview
- Ports
- Requests and Packets
- Access Modes
- Interconnects
- Caches & Coherence
- Prefetching
- DRAM Models

# Memory System Overview

- Integrated timing and functional operation
  - Needed for accurate modeling of time-dependent accesses
    - When coupled with “execute-in-execute” CPU model
  - Prevents components from cheating
- Uniform interfaces for easy composability & extensibility
  - **`MemObject`** is common base class
  - **`Port`** provides uniform means for connecting objects
  - **`Packet`** provides uniform communication among objects
- Multiple modes for trading off accuracy vs. performance
  - Functional, Atomic, Timing
- See Memory System wiki page for more info



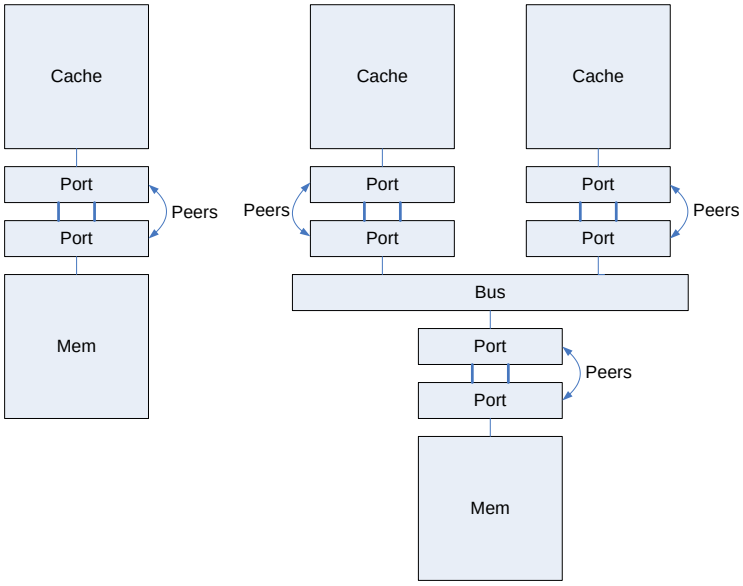


# Ports

**`src/mem/port.{hh,cc}`**

- Method for connecting **`MemObject`**s together
- Each **`MemObject`** subclass has its own **`Port`** subclass(es)
  - Specialized to forward packets to appropriate methods of **`MemObject`** subclass
- Each pair of **`MemObject`**s is connected via a pair of **`Port`**s (“peers”)
- Function pairs pass packets across ports
  - **`sendTiming()`** on one port calls **`recvTiming()`** on peer
- Result: class-specific handling with arbitrary connections and only a single virtual function call

# Ports Example Diagram








# Ports Example Code

- **`Cache`** has two **`Port`** subclasses
  - **`CpuSidePort`**
  - **`MemSidePort`**
- Packets are handled differently depending on which side they come in

```
Tick
Cache<TagStore>::CpuSidePort::recvAtomic(PacketPtr pkt)
{
    return myCache()->atomicAccess(pkt);
}

Tick
Cache<TagStore>::MemSidePort::recvAtomic(PacketPtr pkt)
{
    return myCache()->snoopAtomic(pkt);
}
```

# Connecting Ports

- Port connections are done in Python
  - Define Port parameter in SimObject's Python class
  - Connect two objects by assigning ports
    - Symmetric, unlike normal assignment
- In C++ , ends up calling:
  - **`getPort(name)`** on SimObjects to get Port objects
  - **`setPeer()`** on both Port objects to tie together

```
class BaseCache(MemObject):
    cpu_side = Port("Port on side closer to CPU")
    mem_side = Port("Port on side closer to MEM")
```

```
class Bus(MemObject):
    port = VectorPort("vector port for connecting devices")
```

```
if options.l2cache:
    system.l2 = L2Cache(size='2MB')
    system.tol2bus = Bus()
    system.l2.cpu_side = system.tol2bus.port
    system.l2.mem_side = system.membus.port
```





# Requests and Packets `src/mem/(request|packet).*`

Requests

- Contain information persistent throughout the transaction
  - Virtual/physical addresses, size
  - Stats/debug info: PC, CPU, and thread ID of initiator

Packets

- For point-to-point transfer between memory objects
  - Command (ReadReq, WriteReq, ReadResp, etc.) (**`MemCmd`**)
  - Address/size (may differ from request, e.g., cache miss)
  - Pointer to Request
  - Pointer to data (if any)
  - Source & dest addresses (relative to interconnect)
    - Dest may be "broadcast" for bus
  - SenderState opaque pointer

# Access Modes

- Three access modes: Functional, Atomic, Timing
- Selected by choosing function on initial Port:
  - **`sendFunctional()`**, **`sendAtomic()`**, **`sendTiming()`**
- Functional mode:
  - Just “make it happen”
  - Used for loading binaries, debugging, etc.
  - Accesses happen instantaneously updating data everywhere in the hierarchy
  - If devices contain queues of packets they must be scanned and updated as well





# Access Modes (cont’d)

- Atomic mode:
  - Requests complete before **`sendAtomic()`** returns
  - Models state changes (cache fills, coherence, etc.)
  - Returns approx. latency w/o contention or queuing delay
  - Used for fast simulation, fast forwarding, or warming caches
- Timing mode:
  - Models all timing/queuing in the memory system
  - Split transaction
    - **`sendTiming()`** just initiates send of request to target
    - Target later calls **`sendTiming()`** to send response packet
- Atomic and Timing accesses can not coexist in system

# Interconnects

`src/mem/(bus|bridge).{hh,cc}`

- Port interface enables various interconnects
- Currently only a simple bus model and bus bridge
- Buses
  - Configurable width and clock speed
  - Broadcast capable
- Bridges
  - Simple object to connect two buses
  - Queues requests coming from either side and forwards them out the other side
  - Configurable amount of queuing space





# Caches

`src/mem/cache/*`

- Recently (2.0b4) refactored for clarity and code reuse
- Single cache model with several components:
  - Cache: request processing, miss handling, coherence
  - Tags: data storage and replacement (LRU, IIC, etc.)
  - MSHR & MSHRQueue: track pending/outstanding requests
    - Also used for write buffer
- Parameters: size, hit latency, block size, associativity, number of MSHRs (max outstanding requests)

# Coherence Protocol

- Recently (2.0b4) rewritten for flexibility
  - Support nearly arbitrary multi-level hierarchies
  - At the expense of some realism
- MOESI bus-based snooping protocol
- Does *not* enforce inclusion
- Magic “express snoops” propagate upward atomically
  - Avoid complex race conditions when snoops get delayed
  - Timing is similar to some real-world configurations:
    - L2 keeps copies of all L1 tags
    - L2 and L1s snooped in parallel
- See Coherence Protocol wiki page for more info
- Someday we’d like to add a directory protocol with point-to-point network support...





# Prefetching

`src/mem/cache/prefetcher/*`

- Cache has generic interface for hardware prefetch modules
- Priority given to demand misses, separate prefetch queue
- Several prefetchers supported
  - N-Block Ahead
  - Tagged Prefetching
  - Stride Prefetching
  - Some variants of Global History Buffer prefetching

# DRAM Models

`src/mem/{physical|dram}.*`

- Two included models
  - Simple constant latency (**`physical.*`**)
  - A more detailed DRAM model from ETH (**`dram.*`**)
    - We don't vouch for its accuracy though
- Others have written additional models, we just haven't incorporated them into our tree yet
  - Prof. Brian Davis's students at Michigan Tech
  - ??





# Outline



# Extending M5

## Extending M5

Nate Binkert, Stephen Hines & Gabe Black





# Extending M5

- Overview of M5 internals — Nate
- Defining new objects — Nate
- Statistics — Nate
- ISA Description Language — Stephen
- M5–ARM — Stephen
- Future Development — Gary
- M5–X86 — Gabe

# Execution Process

**src/sim/main.cc, src/python/m5/main.py**

- M5 embeds Python interpreter, simply invokes your script
- **m5** Python module exposes M5 functions
  - Native Python via embedded zip archive
  - C++ via SWIG wrappers
- Up to script to:
  - Process command-line args
    - Easy to include standard M5 args (using **optparse**)
    - Add your own customized for your simulation
  - Build configuration
  - Unserialize from checkpoint, if any
  - Start processing events from event queue
- On exiting Python, M5 will dump statistics





# Building Configurations

- Configuration is a tree of objects
- Build it in Python
  - Instantiate Python SimObject classes
    - "Instantiates" children too... tree copy
  - Can subclass Python SimObjects
    - Override default parameter values
    - Attach child objects
  - Can also "clone" SimObject instances
  - Parameter values are inherited from base class and/or clone ancestor unless overridden
    - Even if base class value is changed later
    - Exception: can't modify child structure after subclass/clone
- Instantiate C++ copy via **m5.instantiate(root)**

# A Configuration Script In Depth

- Configuration scripts are semi declarative.
- Even though the setting of `myparam` on `BaseCPU` occurs after the declaration of `NewCPU`, all `NewCPU` parameters get the update

```
class BaseCPU(SimObject):
    myparam = Param.string(...)
class NewCPU(BaseCPU): pass
BaseCPU.myparam = "value"
```

- This particular instantiation of `NewCPU` overrides the default

```
cpu = NewCPU(myparam="new value")
```





# A Configuration Script In Depth 2

- All instantiations of `FooCPU` will receive the overridden value of `myparam`

```
class FooCPU(NewCPU):
    myparam = "correct value"
cpu1 = FooCPU()
cpu2 = FooCPU()
```

- `cpu1` and `cpu2` will inherit this new value, but `cpu` will not.

```
FooCPU.param = "this is what I really want"
```

# Adding a SimObject Parameter

- Add a variance to physical memory latency
  - Uniformly in range range [*lat*, *lat* + *var*]
- Need to:
  - Add the parameter
  - Write code to change the latency
- Need to change:
  - Parameters file: `PhysicalMemory.py`
  - Object header file: `physical.hh`
  - Object C++ file: `physical.cc`





# Parameters file changes `src/mem/PhysicalMemory.py`

- Add the parameter to the Python file
  - Default doesn't effect simulation
  - Scons/Python will automatically create a header file with parameter added

```
class PhysicalMemory(MemObject):
    type = 'PhysicalMemory'
    port = VectorPort("the access port")
    range = Param.AddrRange(AddrRange('128MB'), "Device Address")
    file = Param.String("", "memory mapped file")
    latency = Param.Latency('1t', "latency of an access")
+   latency_var = Param.Latency('0ns', "access variablity")
    zero = Param.Bool(False, "zero initialize memory")
```

# Header file changes `src/mem/physical.hh`

- Add a variable to the `PhysicalMemory` object
  - Not strictly necessary
    - Could just always get the parameter from the Params Structure
  - It's faster to cache the value in the Object however

```
    uint8_t *pmemAddr;
    int pagePtr;
    Tick lat;
+   Tick lat_var;
    std::vector<MemoryPort*> ports;
    typedef std::vector<MemoryPort*>::iterator PortIterator;
```





# C++ file changes `src/mem/physical.cc`

- Save the value of the latency variance parameter in the object
  - The parameter name is exactly the name in the Python file
- Change `PhysicalMemory::calculateLatency()` to use the new parameter
  - If `lat_var` is not zero, get a uniform value from 0, `lat_var` and add it to the latency.

```
PhysicalMemory::PhysicalMemory(const Params *p)
-   :  MemObject(p), pmemAddr(NULL), lat(p->latency)
+   :  MemObject(p), pmemAddr(NULL), lat(p->latency),
+   lat_var(p->latency_var)
```

```
-   return lat;
+   Tick latency = lat;
+   if (lat_var != 0)
+   latency += random_mt.random<Tick>(0, lat_var);
+   return latency;
```

# Creating New SimObjects

- Derive C++ class from C++ SimObject
  - Defines simulation behavior
  - See **`src/sim/sim_object.{cc,hh}`**
  - Add C++ filename to **`src/SConscript`**
- Derive Python class from Python SimObject
  - Defines parameters, ports for configuration
  - Add file in **`C++`** directory
- C++ needs parameter/creation boilerplate
  - Normally just a typedef defining Params and a params() function
- That's it! Recompile and use!





# Creating a simple I/O Device

- Create a simple device that just prints a warning when it's accessed
- Need to:
  - Create SimObject
  - Create Python SimObject Parameters
  - Add both files to SConscript
  - Add device to configuration
- Create:
  - Object header file: `src/dev/hellodevice.hh`
  - Object C++ file: `src/dev/hellodevice.cc`
  - Parameters file: `src/dev/HelloDevice.py`
- Modify:
  - SConscript: `src/dev/SConscript`
  - Configuration: `configs/common/FSConfig.py`

# HelloDevice SimObject src/dev/hellodevice.hh

- Base classes provide almost all functionality required for a standard I/O Device

```
#include "dev/io_device.hh"
#include "params/HelloDevice.hh"
class HelloDevice : public BasicPioDevice
{
  private:
    std::string devname;

  public:
    typedef HelloDeviceParams Params;

  protected:
    const Params *
    params() const
    {
        return dynamic_cast<const Params *>(_params);
    }

  public:
    HelloDevice(Params *p);

    virtual Tick read(PacketPtr pkt);
    virtual Tick write(PacketPtr pkt);
};
```





# HelloDevice SimObject src/dev/hellodevice.cc

```
#include "dev/hellodevice.hh"
HelloDevice::HelloDevice(Params *p)
    : BasicPioDevice(p), devname(p->devicename)
{
    pioSize = 0x10;
}

Tick
HelloDevice::read(PacketPtr pkt)
{
    Addr daddr = pkt->getAddr() - pioAddr;
    assert(pkt->getSize() == 8);
    pkt->allocate();
    pkt->set((uint64_t)0);
    pkt->makeAtomicResponse();
    warn("Device %s: Read request at offset %#X", devname, daddr);
    return pioDelay;
}
```

# HelloDevice SimObject src/dev/hellodevice.cc

```
Tick
HelloDevice::write(PacketPtr pkt)
{
    Addr daddr = pkt->getAddr() - pioAddr;
    assert(pkt->getSize() == 8);
    pkt->makeAtomicResponse();
    warn("Device %s: Write request at offset %#X", devname, daddr);
    return pioDelay;
}

HelloDevice *
HelloDeviceParams::create()
{
    return new HelloDevice(this);
}
```





# HelloDevice SimObject src/dev/HelloDevice.py

- Inheritance works with SimObject parameters
- HelloDevice has 5 other parameters defined in parent classes
- `devicename` is only only one specific to HelloDevice

```
from m5.params import *
from Device import BasicPioDevice

class HelloDevice(BasicPioDevice):
    type = 'HelloDevice'
    devicename = Param.String("String to print on access")
```

# HelloDevice SimObject `src/dev/SConscript`

- Add the Python file and the C++ file to the SConscript

```
SimObject('BadDevice.py')
+SimObject('HelloDevice.py')
SimObject('Device.py')
SimObject('DiskImage.py')
SimObject('Ethernet.py')
Source('baddev.cc')
+Source('hellodevice.cc')
Source('disk_image.cc')
Source('etherbus.cc')
Source('etherdump.cc')
```





# Add HelloDevice SimObject to configuration

- Instantiate a copy of the object
- Connect it's pio port to the iobus

```
self.tsunami.ethernet.pio = self.iobus.port
self.simple_disk = SimpleDisk(disk=RawDiskImage(image_file = mdesc.disk(),
                      read_only = True))
+self.hello = HelloDevice(pio_addr=0x801fcabcde0, devicename="HelloDevice1")
+self.hello.pio = self.iobus.port
self.intrctrl = IntrControl()
self.mem_mode = mem_mode

self.sim_console = SimConsole()
```

# Running with HelloDevice

- Modified a copy of the Alpha Console to access 0x801fcabcde0 at boot

```
M5 Simulator System

Copyright (c) 2001-2008
The Regents of The University of Michigan
All Rights Reserved


M5 compiled Feb 15 2008 16:54:22
M5 started Fri Feb 15 16:54:28 2008
M5 executing on zeep
command line: ./build/ALPHA_FS/m5.opt configs/example/fs.py
Global frequency set at 1000000000000 ticks per second
warn: kernel located at: /dist/m5/system/binaries/vmlinux
      0: system.tsunami.io.rtc: Real-time clock set to Thu Jan  1 00:00:00 2009
Listening for system connection on port 3456
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
warn: Entering event queue @ 0.  Starting simulation...
warn: Device HelloDevice1: Read request at offset 0
```





# Serialization `src/sim/serialize.{hh,cc}`

- 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)
- Common error: adding field to object and not updating **`serialize()`**/**`unserialize()`**

# Events `src/sim/eventq.{hh,cc}`

- 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





# The M5 Statistics Package

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

# Statistics Example – header file

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

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





# Statistics Example – cc file

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

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

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%
```





# ISA Description Language

**src/arch/isa_parser.py, src/arch/*/isa/***

- Custom domain-specific language
- Defines decoding & behavior of ISA
- Generates C++ code
  - Scads of **StaticInst** subclasses
  - **decodeInst ()** function
    - Maps machine instruction to **StaticInst** instance
  - Multiple scads of **execute()** methods
    - Cross-product 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': ('IntReg', 'uq', 'PALMODE ? AlphaISA::reg_redir[RA] : RA',
           'IsInteger', 1),
    'Rb': ('IntReg', 'uq', 'PALMODE ? AlphaISA::reg_redir[RB] : RB',
           'IsInteger', 2),
    'Rc': ('IntReg', 'uq', 'PALMODE ? AlphaISA::reg_redir[RC] : RC',
           'IsInteger', 3),
    'Fa': ('FloatReg', 'df', 'FA', 'IsFloating', 1),
    'Fb': ('FloatReg', 'df', 'FB', 'IsFloating', 2),
    'Fc': ('FloatReg', 'df', 'FC', 'IsFloating', 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
  - Alpha: 3437 lines of isa description → 39K lines of C++
    - ∼15K generic decode, ∼12K for each of 2 CPU models
  - Characteristics auto-extracted from C
    - source, dest regs; func unit class; etc.
  - `execute()` code customized for CPU models
- Thoroughly documented (for us, anyway)
  - See wiki pages





# M5–ARM

- Embedded systems platforms (cell phone, GPS, . . . )
  - Increasing functionality and complexity
  - Cost/benefit can have huge impact
  - Need for better analysis tools
- ARM architecture v6 target – popular
- SE for now, FS in progress
- With the move to multicore, M5 will be a great environment for exploring whole system designs

# ARM ISA Challenges

- RISC vs. CISC ($\mu$-ops)
  - Load/Store multiple
  - Double-word loads
- Predication
  - Supported by nearly every instruction
  - Fortunately most are Always
- Lessons Learned





# RISC vs. CISC ($\mu$-ops)

- **LDM/STM** (Load/Store Multiple)
  - Decoded into several $\mu$-ops
  - `stmdb sp!, {r4, r5, r6, fp, ip, lr, pc}`
    1. Decrement (`db`) the stack pointer (`sp!`) using **subi_uop**
    2. Execute 7 **str_uop** with varying offsets and register sources
  - Too many combinations of registers/offsets to statically specify $\rightarrow$ dynamically generate $\mu$-op list during decode
- **LDFD/STFD** and **LDFE/STFE** (double- and quad-word loads/stores)
  - Similar decomposition into **ldr_uop**, **str_uop**, **addi_uop**, **subi_uop**, **addi_rd_uop**, and/or **subi_rd_uop**

# Dynamic $\mu$-ops .../arm/isa/formats/macromem.isa

```
def template MacroStoreConstructor {{
    inline %(class_name)s::%(class_name)s(MachInst machInst)
        :  %(base_class)s("%(mnemonic)s", machInst, %(op_class)s)
    {
        %(constructor)s;
        uint32_t regs_to_handle = reglist;
        uint32_t j = 0,
               start_addr = 0,
               end_addr = 0;
        // etc.
        uint32_t newMachInst = 0;
        newMachInst = machInst & 0xffff0000;
        microOps[0] = new Addi_uop(newMachInst);

        for (int i = 1; i < ones+1; i++)
        {
            // Get next available bit for transfer
            while (!  ( regs_to_handle & (1<<j)))
                j++;
            regs_to_handle &=  (1<<j);

            microOps[i] = gen_ldrstr_uop(machInst, loadop, j, start_addr);

            if (up)
                start_addr += 4;
            else
                start_addr -= 4;
        }
```





# Predication src/arch/arm/isa/decoder.isa

- 16 predicates based on Negative, Zero, Carry, Overflow flags
- Need to examine Current Program Status Register (CPSR) and condition code specified by instruction (top 4 bits)
- Keep ISA description clean in decoder (primarily 1-liners)

```
// etc.
    0x4: add_rs({{ Rd = Rn + Rm_Rs; }});
    0x5: adc_rs({{ Rd = Rn + Rm_Rs + Cpsr<29:>; }});
// etc.
```

# Predicated Execution `src/arch/arm/isa/pred.isa`

```
def template PredOpExecute {{
    Fault %(class_name)s::execute(%(CPU_exec_context)s *xc, Trace::InstRecord *traceData)
const
    {
        Fault fault = NoFault;

        %(fp_enable_check)s;
        %(op_decl)s;
        %(op_rd)s;
        %(code)s;

        if (arm_predicate(xc->readMiscReg(ArmISA::CPSR), condCode))
        {
            if (fault == NoFault)
            {
                %(op_wb)s;
            }
        }
        else
            return NoFault;
            // Predicated false instructions should not return faults

        return fault;
    }
}};
```





# Lessons Learned (so far)

- Adding a new ISA is not an impossible task
- Most ISAs have enough similarities that existing models provide a good starting point
- Only ∼50 lines changed in files outside src/arch/ to support a whole new architecture!
  - 3553 lines of isa description → 56K lines of C++
  - ∼18K generic decode, ∼19K for each of 2 ARM CPU models
- Plenty of good wiki documentation on ISA descriptions `http://www.m5sim.org`

# Obtaining M5–ARM

- Check out **http://www.cs.fsu.edu/~hines/m5-arm/**
- Grab **arm_extras.tar.gz**

```
...  Assuming your m5 is /home/hines/m5
% cd /home/hines
% tar xvjf arm_extras.tar.gz
% cd /home/hines/m5
% scons build/ARM_SE/m5.opt EXTRAS=/home/hines/arm_extras
```

- Cross-compiler toolchain on M5 CD or build one from scratch based on online M5–ARM documentation
- Use just like any other SE-mode simulator in M5





# Future Development for M5–ARM

openmoko

**http://www.openmoko.org/**

android

**http://code.google.com/android/**

# M5–X86

- Work is supported by HP
- We hope to release as open source
- One year of development so far





# What works

- SPEC CPU2000
  - Integer benchmarks
  - Compared cycle by cycle to real machine
- Partialy booting Linux
  - Loads kernel image and starts execution
  - Up to initializing the local APIC

# Specifically...

- Instructions
  - Variable length fetch
  - Most prefixes
- Decode
  - Complex address decoding
  - Decomposition into microops
    - Microop ISA
    - Microcode assembler
- Memory
  - Unaligned addresses
  - Segmentation
  - Paging





# What doesn't

- Interrupts/Exceptions
- Most of XMM and MMX
- Loading segments
- Non-TLB miss faults in SE
- Priviledge levels
- Debug register support
- Performance monitoring
- Secure virtual machine
- System management mode
- Virtualization extensions
- IORR, MTRR, PAT
- Serialization
- Most devices

# Next steps

- Virtualization extension based CPU model
- Bring up other OSes/hypervisors
  - BSDs
  - Windows
  - Xen
- Statistics
- Instrument linux kernel
- M5 pseudo instructions





# Outline



# Debugging M5

## Debugging M5

Nate Binkert





# Debugging Facilities

- Tracing
  - Instruction Tracing
  - Diffing Traces
- Using the C++ Debugger
  - Debugging Coherence State
  - Remote Debugging
- Python Debugging

# Tracing

**src/base/trace.***

- printf is a nice debugging tool
- Keep good printfs for tracing
- Lots of debug output is a very good thing
- Add new flags to SConscripts
  - Individual flags: `TraceFlag('NewFlag')`
  - Groups of flags: `CompoundFlag('NewFlags', ['NewFlag1, 'NewFlag2'])`
- Fetch, Decode, Ethernet, IPI, TLB, DMA, Bus, Cache, Loader, AlphaConsole, etc.

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

Command line flags:
m5.opt --trace-flags="Space Separated List" --trace-start=<tick number>

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





# Instruction Tracing

**src/sim/insttracer.hh**

- Separate from the general trace facility
- Per-instruction records populated as instruction executes
  - Start with PC and mnemonic
  - Add argument and result values as they become known
- Printed to trace when instruction completes
- Flags for printing cycle, symbolic addresses, etc.

# Diffing Traces `util/{rundiff,tracediff}`

- Often useful to compare traces from two simulations
  - Find where known good and modified simulators diverge
- Standard *diff* works only on files (not pipes)
  - ...but you really don't want to run to completion
- **`util/rundiff`**
  - Perl script for diffing two pipes on the fly
- **`util/tracediff`**
  - Handy wrapper for using **`rundiff`** to compare M5 outputs
  - **`tracediff "a/m5.opt|b/m5.opt" --trace-flags=Exec`** compares instruction traces from two builds of M5
  - See comments for details





# Using GDB with M5

```
% gdb ~/build/newmem/build/ALPHA_FS/m5.debug
GNU gdb 6.1-20040303 (Apple version gdb-437) (Fri Jan 13 18:45:48 GMT 2006)
... more gdb header ...

(gdb) b main
Breakpoint 1 at 0x39027: file /Users/nate/build/newmem/build/ALPHA_FS/sim/main.c, line 175.

(gdb) run configs/test/single_fs.py
Starting program: /Volumes/work/build/newmem/build/ALPHA_FS/m5.debug configs/test/single_fs.py

Breakpoint 1, main (argc=34608072, argv=0x21013e0) at /Users/nate/build/newmem/build/ALPHA_FS/
sim/main.cc:175
175         myProgName = argv[0];
(gdb) call schedBreakCycle(3000)

(gdb) cont
Continuing.

... M5 Output ...

Program received signal SIGTRAP, Trace/breakpoint trap.
0x9003d9ec in kill ()
(gdb) p curTick
$1 = 3000

(gdb)
```

# Using GDB with M5

- Several M5 functions designed to be called from GDB:
  - **`schedBreakCycle()`**
  - **`setTraceFlag()`**/**`clearTraceFlag()`**
  - **`dumpTraceStatus()`**
  - **`eventqDump()`**
  - **`SimObject::find()`**





# Debugging Coherence State

- New in 2.0b5: special **`PrintReq`** functional memory request traverses hierarchy, printing relevant state
- Needs to be injected from CPU
  - (Some) CPU objects have **`printAddr()`** method
- Framework is in place but not fully fleshed out
  - Memory objects could print more info
  - Verbosity flag currently unused
  - Some CPUs (e.g., O3) don't have **`printAddr()`** yet
- See example on next page

# Debugging Coherence State Example

- Example below uses **`MemTest`** pseudo-CPU
- Should work with SimpleCPU variants too
- Note use of **`SimObject::find()`** to get C++ pointer

```
(gdb) set print object
(gdb) call SimObject::find("system.physmem.cache0.cache0.cpu")
$4 = (MemTest *) 0xf1ac60
(gdb) p (MemTest*)$4
$5 = (MemTest *) 0xf1ac60
(gdb) call $5->printAddr(0x107f40)

system.physmem.cache0.cache0
  MSHRs
    [107f40:107f7f] Fill   state:
      Targets:
        cpu: [107f40:107f40] ReadReq
system.physmem.cache1.cache1
  blk VEM
system.physmem
  0xd0
(gdb)
```





# Remote Debugging

```
% ~/m5/build/ALPHA_FS/m5.debug ~/m5/configs/test/single_fs.py
M5 Simulator System
Copyright (c) 2001-2006
The Regents of The University of Michigan
All Rights Reserved

M5 compiled Jun 17 2006 09:58:47
M5 executing on ziff.eecs.umich.edu
M5 started Sat Jun 17 15:15:58 2006
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/test/single_fs.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)
```





# Python Debugging

- It is possible to drop into the python interpreter (`-i` flag)
  - This currently happens after the script file is run
  - If you want to do this before objects are instantiated, remove them from script
- It is possible to drop into the python debugger (`--pdb` flag)
  - Occurs just before your script is invoked
  - Lets you use the debugger to debug your script code
- Code that enables this stuff is in src/python/m5/main.py
  - At the bottom of the main function
  - Can copy the mechanism directly into your scripts, if in the wrong place for you needs

# Outline







# Wrap-Up

## Wrap-Up

Ali Saidi

# 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 ASPLOS





# Keep In Touch

- Come talk to us at ASPLOS
  - We're (almost) all here through Wednesday
- Check http://www.m5sim.org
  - Use the wiki to learn more
  - Contribute things you learn on your own
- Use, subscribe to our mailing lists:
  - m5-users@m5sim.org
  - m5-announce@m5sim.org