gem5: empowering the masses

Sascha Bischoff
Senior Software Engineer, Arm
Who am I, and why am I standing here?

Who am I?

• Senior software engineer in the Architecture group at Arm
  • Sub-team focuses largely on power
    – Standardizing
      ▪ Unite the ecosystem
    – Modelling
      ▪ Understand, design, implement
    – Implementation
      ▪ Deliver high-performance at low-power

Why am I standing here?

• We model past, present, and future systems in gem5
  • Understand the impacts of decisions on performance and power
  • Heavy focus on software power management
• We use the power modelling aspects of gem5 on a daily basis
  • Encourage others (the masses) to do the same!
• Working with gem5 since 2011

Why do we want to model power?

• Design decisions strongly impact power and performance
  • This includes both hardware and software
• We need to make sure that the hardware and software interoperate as efficiently as possible
• Software needs to understand the hardware in sufficient detail to make the correct decisions
  • E.g. Arm big.LITTLE & Energy Aware Scheduling

gem5 allows us to investigate both power and performance impact in a single framework
The two sides of power modelling

**Static (Leakage) Power**
- Power dissipation due to the non-switching activity
- Constant, unless:
  - Temperature changes
  - Voltage changes
  - Power state changes
- Not really *that* constant then

**Dynamic Power**
- Power dissipation due to the activity in the system
- Affected by
  - Frequency
  - Voltage
  - Power state
  - Activity
- Largest contributor to overall power
How do we model power dissipation in gem5, then?

• Roughly, with two equations!
  • One for dynamic power, and another for static power
  • Per sub-system
  • Per power state

• We need frequency, voltage, power states and switching activity
  • ClockDomain
  • VoltageDomain
  • PowerStates
  • gem5 statistics
Getting the voltage and frequency information

**ClockDomain**
- An ordered list of frequencies in *descending* order
- Wraps a set of components
  - ClockedObjects default to using Parent.clk_domain
- Requires an associated VoltageDomain

**VoltageDomain**
- An ordered list of voltages, one per frequency, in *descending* order
- Accessed via the ClockDomain

Both of the above are included in the stats output
An aside: controlling CPU voltage and frequency

**DVFSHandler**
- Responsible for handling the ClockDomains belonging to the CPUs in the system
  - Can include other components, e.g., caches, but must be paired with a CPU
- A centralized point to interface the domains with the EnergyCtrl
- Models the transition latency too
  - Changes take time!

**EnergyCtrl**
- Provides an interface to the OS
  - Reports available frequencies
  - Allows the OS to adjust the frequency
    - Linux CPUFREQ drivers are available [1]

**NOTE:** The gem5 CPU “socket_id” should match the ClockDomain’s “domain_id”. Alternatively, can convey this mapping in the Device Tree cpu-map node.

Power states

Each ClockedObject* has five power states:

1. UNDEFINED - Default state
2. ON
3. CLK_GATED
4. SRAM_RETENTION
5. OFF

Each power state will consume a different amount of power.

CPU models transition between these states based on their level of activity.
• Based on these, we can derive the corresponding states for their caches, etc.

* The DRAMCtrl is different, and tracks DRAM-specific states, with much higher accuracy.
ThermalDomains

- Static power is heavily affected by the silicon temperature
  - Need to also model this in gem5
- ThermalDomains allow the thermal properties of the device to be specified
  - Thermal RC network
    - Thermal resistance (K/W)
    - Thermal capacitance (Ws/K)
    - Initial temperature
- Feedback loops
  - Temperature feeds into power
  - Power feeds into temperature
gem5 statistics

- Voltages and clock periods are included in the gem5 stats output
  - Per ClockDomain and VoltageDomain
- Same applies to power state residency
  - I.e., how long has each object been in each power state?
- Naturally, switching activity is also included here
- These statistics can in turn be used to calculate the power dissipation
  - (Automatically) dump the statistics on each DVFS change
    - Avoid averaging
Calculating the power: PowerModel

• Essentially a list of pairs of equations
  • Static and dynamic
  • One pair per power state, ordered from most active to least active power state

• Exposes two methods:
  • getStaticPower();
  • getDynamicPower();

• Return power weighted by the time spent in each power state

• Used by the SubSystem and ThermalDomain

• Wraps the...
MathExprPowerModel

• Defines a power model as a pair of equations
  • Static & dynamic
• Uses the gem5 statistics infrastructure

# Equations for dynamic and static power in Watts
# Equations may use gem5 stats ie. "1.1*ipc + 2.3*12_cache.overall_misses"
# It is possible to use automatic variables such as "temp"
# You may also use stat names (relative path to the simobject)
dyn = Param.String("", "Expression for the dynamic power in Watts")
st = Param.String("", "Expression for the static power in Watts")
The big picture

- CPU frequency is controlled by the OS
  - Voltage is changed in line with frequency
- PowerModel(s)
  - Pick up voltage and frequency changes
  - Extract activity from the statistics
  - Obtain thermal from the ThermalDomain(s)
- ThermalDomain(s)
  - Recalculate thermal based on the power
- Results feed back into the statistics
Bringing it all together

• Time for an example!

• (Simplified) System with 4 little cores, 2 big cores
  • Run Dhrystone with 6 threads, followed by memcopy on a single little core

• Using Workload Automation (WA) to run the workloads
  • See Anouk’s presentation from earlier
  • “Interacting with gem5 using workload-automation & devlib”
  • Note: Successive runs are different as running with WA breaks determinism

• Sample the power at two different intervals
  • 1s and 1ms
Some example results

(Simplified) System with 4 little cores, 2 big cores
Comparing the results

- Much more detail for more frequent sampling
  - Much more data though!
    - 1.7MB vs 404MB (compressed!)
- Extra data points for DVFS changes
- Need to trade off output size for fidelity

![Graphs showing 1s and 1ms sampling intervals]
Thank You!
Danke!
Merci!
谢谢!
ありがとうございます!
Gracias!
Kiitos!
The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks