Introduction to gem5

Sotiris Totomis

CS425 – Computer Systems Architecture

Topics for today

- Basic information
- Building/running instructions
- Fundamental way of operation
- Demo of examples

What is it?

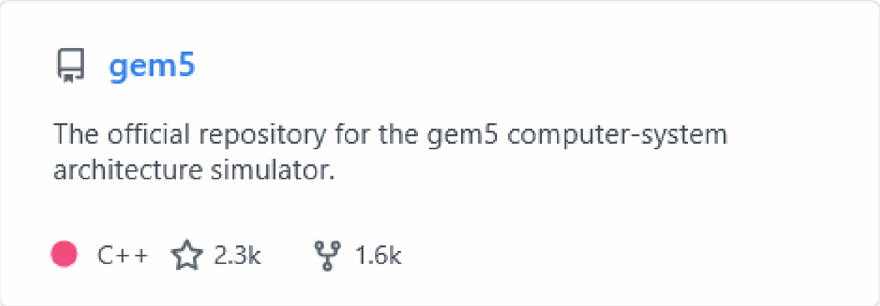
- Open source simulator for computer architecture
    - Models the behavior of the <u>target architecture</u> to evaluate computer systems
- Research tool to simulate the performance of complex designs
    - CPUs
    - Memory systems
    - Interconnects & protocols
- *The goal:* Analysis of different architectural choices that depend either on hardware or software/OS design without having to build or own the real systems

Who uses it?

- Education (Our case!)
  - We use gem5 to implement or measure concepts learned in this course
- Academic research
  - *Survey:* “70% of all computer architecture research uses simulations for novel design proposals and gem5 is by-far the most popular”
- Industry research
  - Known contributors: Google, ARM, AMD
  - Many more, but through internal forks (not open sourced)

How can I use it?

- A must visit: https://www.gem5.org/
  - Documentation and detailed instructions of use
  - Dependencies installation instructions
- gem5 is on GitHub, so…
  - `git clone https://github.com/gem5/gem5`



- For the course assignments we will provide standalone forks for each assignment (to be discussed)
  - The main goal: Study, measure and understand architectural tradeoffs of different system components

Building it

- After dependency installation

```
scons build/target_arch/gem5.opt -j #NCPUS
```

- Many target architectures in gem5
  - RISCV
  - ARM
  - X86
- If host has poor memory, be cautious with #NCPUS
- .opt stands for both simulator optimizations and asserts/DPRINTFs
- Initial compilation takes time, get it done early on the assignments!

Running it

- After a graceful compilation: Just an example of cmd

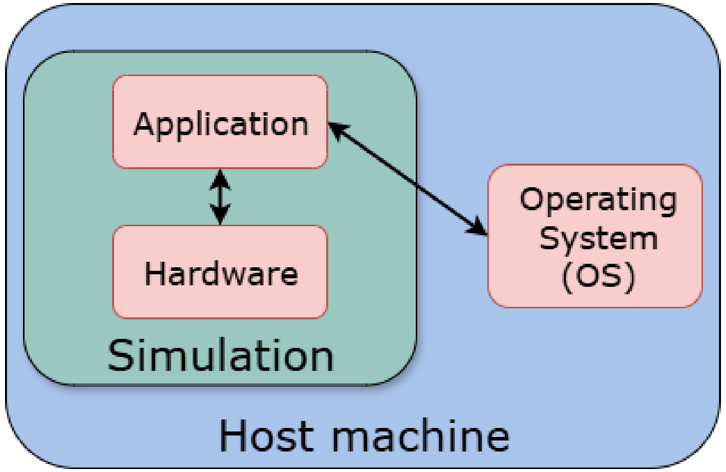
```
gem5/build/RISCV/gem5.opt \
gem5/configs/test.py \
-c hello_world -cpu-type=SimpleCPU l1d_size=‘64kB’
-l1d_assoc=2 -cacheline_size=64 -bp_type=‘SimpleBP’
```

- The gem5’s binary *
- The frontend script (our blueprint) (explained later) *
- The system parameters we want to simulate (fall back to defaults if not given)

*mandatory arguments

SE vs. FS

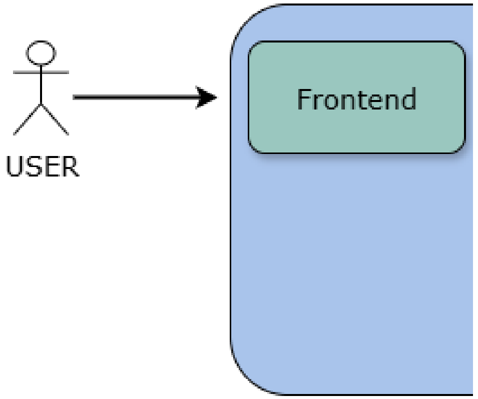
- Systemcall (or syscall) emulation (SE)
  - We simulate
    - The application (compiled with the toolchain of target architecture)
    - The hardware of the target architecture
  - We do not simulate
    - The Operating System calls and behavior
  - SE relays system calls to the host OS for processing
  - Faster simulations
  - No capturing of OS behavior
  - Our case! We do not care about OS for now!



- Full-System simulation (FS)
  - We simulate everything, even the OS
  - Capture OS behavior and interaction with application/hardware
  - (Much) Slower simulations

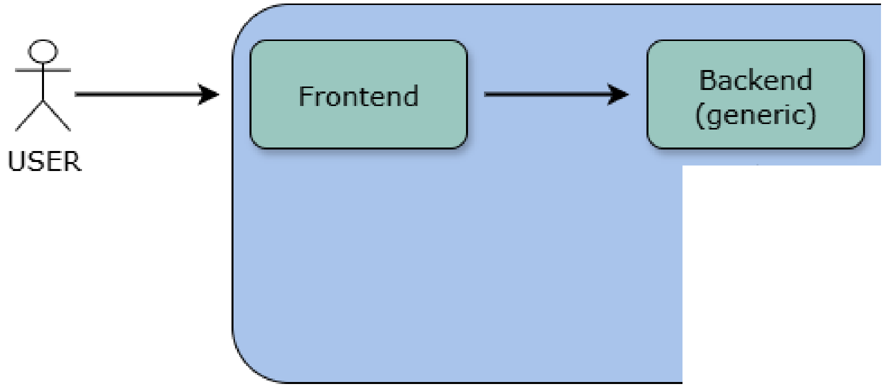
How to interact with it? Frontend

- The user invokes the frontend of gem5 through command line to configure the system parameters for simulation
  - e.g. CPU type, clock frequency, cache parameters, memory type etc.
  - Just like a recipe
- Always free to modify/extend the already given (or new) frontend scripts
  - Build and configure different systems
  - Written in Python
  - Demo example incoming



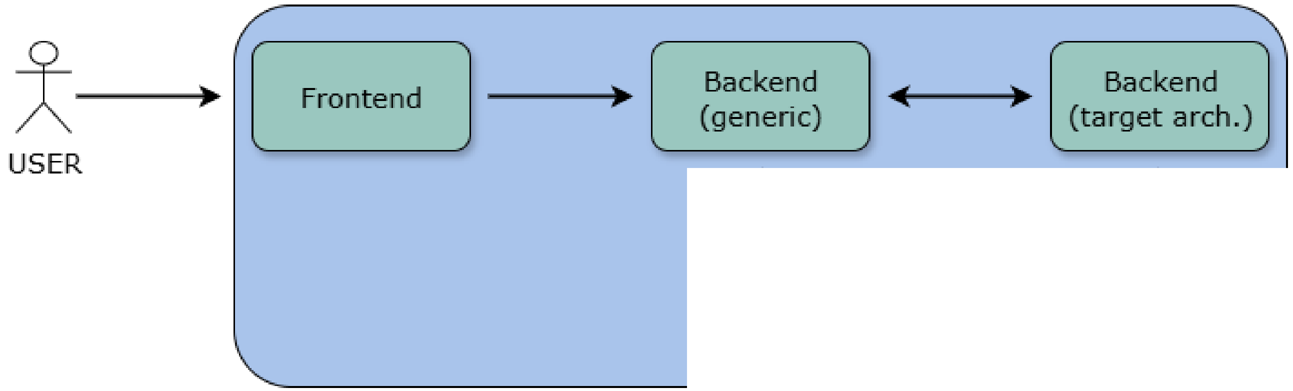
Backend (generic)

- The frontend is responsible to instantiate the generic backend of gem5 that corresponds to the described frontend configuration
- The backend is the detailed description (in terms of functionality and design) of the architectural components of the system
  - Written in C++
  - It describes the models of CPU, cache etc. as C++ classes
  - Extra functionalities/components (architecture agnostic) are implemented here!



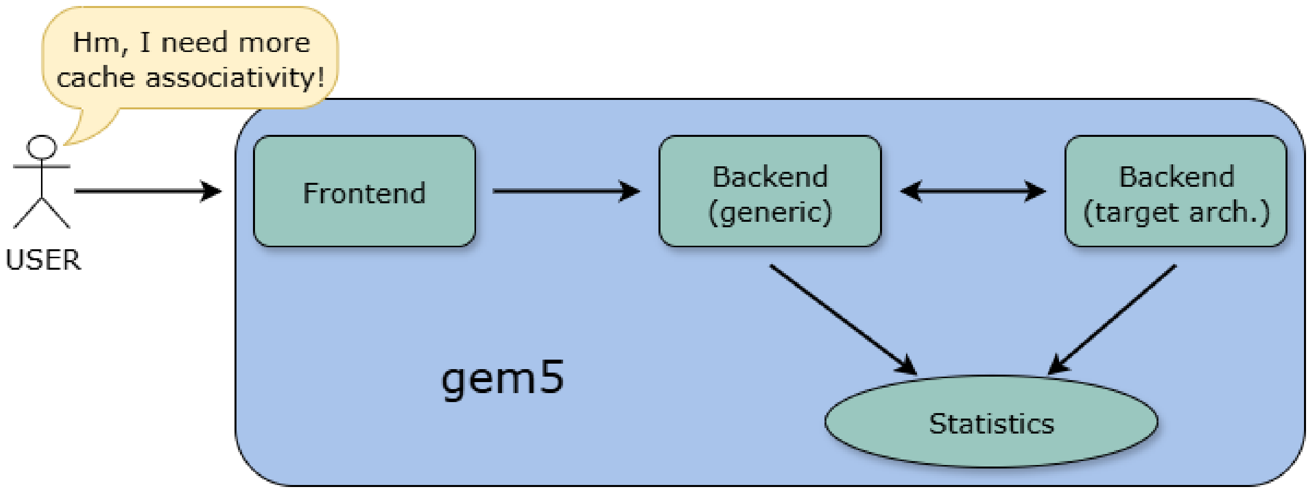
Backend (architecture specific)

- The generic portion of the backend has interfaces to interact with the architecture specific backend
- The architecture backend is responsible to provide detailed implementations of components that are architecture-dependent
  - e.g. Translation Lookaside Buffer (TLB)
  - But, why TLBs?



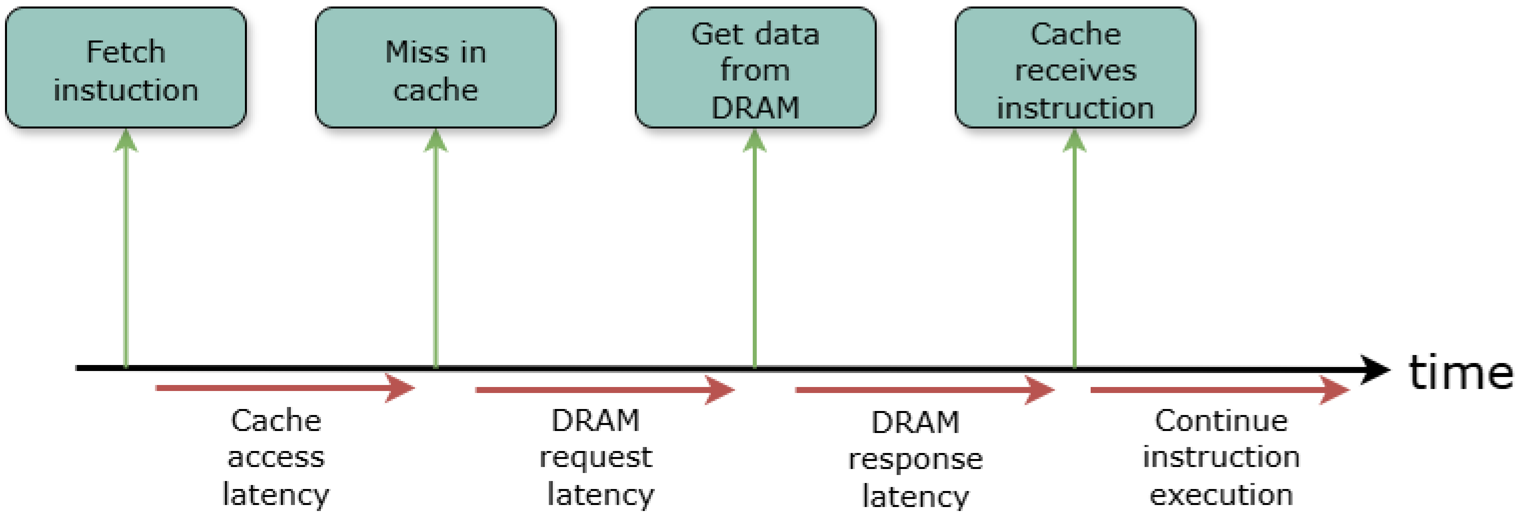
Statistics

- During the simulation of the benchmark, both generic and architecture backends measure latency of components/functionalities, number of events etc.
  - All types of metrics are reported to the statistics file for the user to access
  - e.g. CPI, IPC, cache misses, number of translations, latency of translations etc.



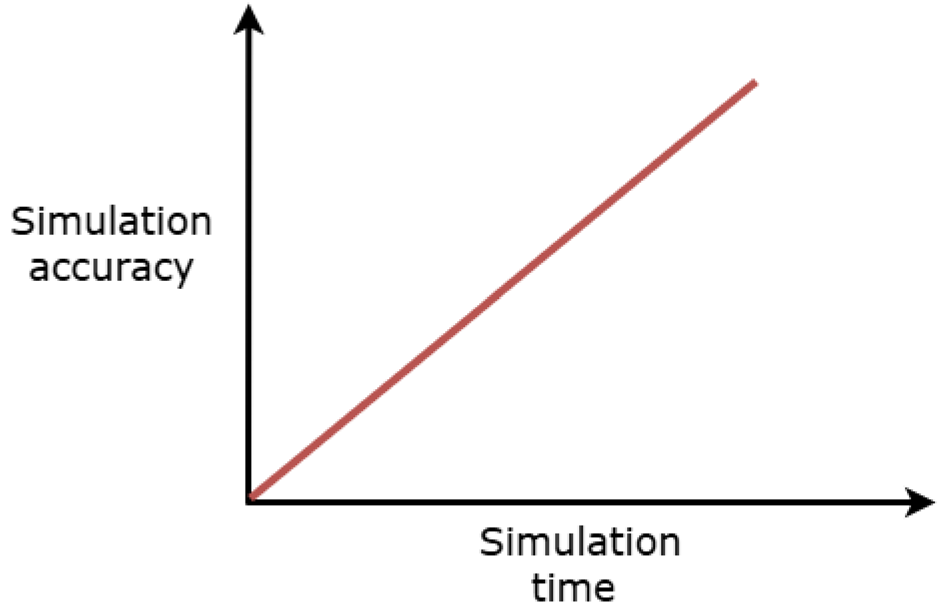
Ticks & discrete event simulation

- gem5 uses a time unit called “tick”
  - But how can we interpret a tick?
    - Global simulation tick rate = 1 picosecond per tick (or 10^{12} ticks per second)
- gem5 is a discrete event simulator (adds implementation complexity)
  - The time progresses when an event is executed (simulation performance)



Slower vs. faster simulations

- gem5 is slow
  - The nature of simulation
  - Approximately, 1 second of simulation is 100k (+) seconds on the host!
- But we do not need to simulate everything in detail or at all!
  - e.g. a very detailed cache/protocol model
  - OS interaction
- It is all about tradeoffs…



Step-by-step demo

- Building a binary with RISCV toolchain (**musl - no glibc**)
  - **Need static compilation**
  - Disassemble/objdump utility
- Frontend script configuration in gem5
  - Let’s explore a configuration
- Run a simple example with the default parameters & MinorCPU
- Explore the m5out directory for statistics and detailed configuration
  - Focus on a single metric (e.g. CPI)
  - Make changes that reflect to this metric
  - Re-run and explore again the statistics!