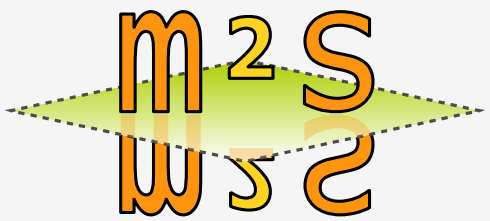
Programming and Simulating Fused Devices

Part 2
Multi2Sim



Rafael Ubal
Perhaad Mistry

Northeastern University
Boston, MA

Outline

1. Introduction
2. The x86 CPU Emulation
3. The Evergreen GPU Emulation
4. GPU Architectural Simulation
5. The Memory Hierarchy
6. Ongoing Work
7. Conclusions

1. Introduction

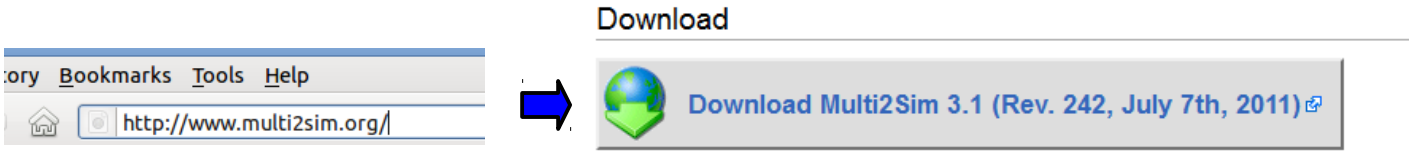
Multi2Sim Background

- **Multi2Sim 1.x version series, 2007 (MIPS-based)**
  - **Superscalar pipeline**
    Out-of-order execution, branch prediction, trace cache, etc.
  - **Multithreading**
    Fine-grain, coarse-grain and simultaneous (SMT).
- **Multi2Sim 2.x version series, 2008 (x86-based)**
  - **Multicore architecture.**
    Configurable memory hierarchy, cache coherence, interconnection networks.
  - **State-of-the-art benchmarks.**
    Tested support for common research benchmarks, available for download.
- **Multi2Sim 3.x version series, 2011 (x86+Evergreen)**
  - **GPU model**
    Support for OpenCL benchmarks.
    Emulation of Evergreen ISA.
    Architectural model of AMD Radeon 5870
  - **Visualization tools**
    CPU and GPU pipelines.
    Memory hierarchy.

1. Introduction

Quick Setup

- **User-friendly installation and test**




```
$ tar -xzf multi2sim-3.1.tar.gz

$ cd multi2sim-3.1
$  ./configure
$ make
$ sudo make install
```

- **Application-only simulator**

Original execution

```
$ ./test-args how are you
    arg[0] = 'how'
    arg[1] = 'are'
    arg[2] = 'you'
```

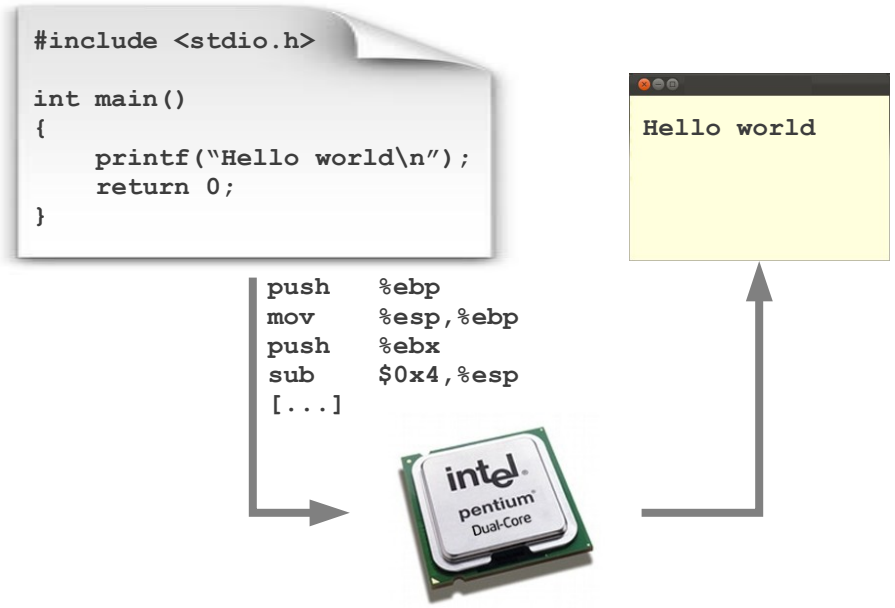
Simulated execution

```
$ m2s ./test-args how are you
  <... Simulator output ...>
    arg[0] = 'how'
    arg[1] = 'are'
    arg[2] = 'you'
  <... Simulator statistics ...>
```

2. The x86 CPU Emulation

Emulation vs. Architectural Simulation

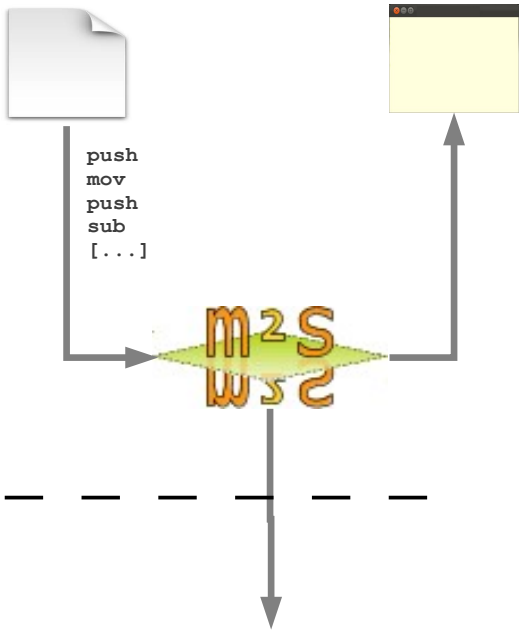
- **Native execution**
  - x86 ISA runs on a processor.
  - Result is observed in output devices.



2. The x86 CPU Emulation

Emulation vs. Architectural Simulation

- **Emulation (or functional simulation)**
  - Multi2Sim mimics behavior of processor.
  - 2-Step process
    - Program loading
    - Simulation loop

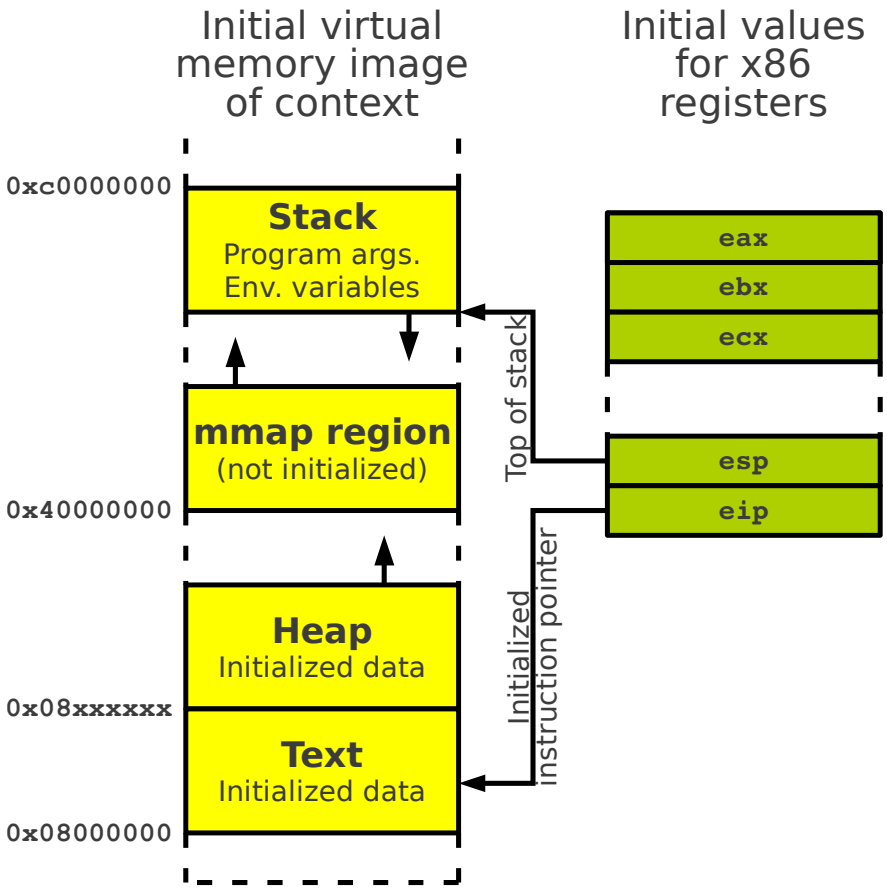


- **Architectural (or timing) simulation**
  - Trace of instructions consumed from emulator.
  - Timing model of hardware structures.

2. The x86 CPU Emulation

Program Loading

- **Initialization of a process state**



1) Parse ELF executable

- ELF sections.
- Initialized code and data.

2) Initialize stack

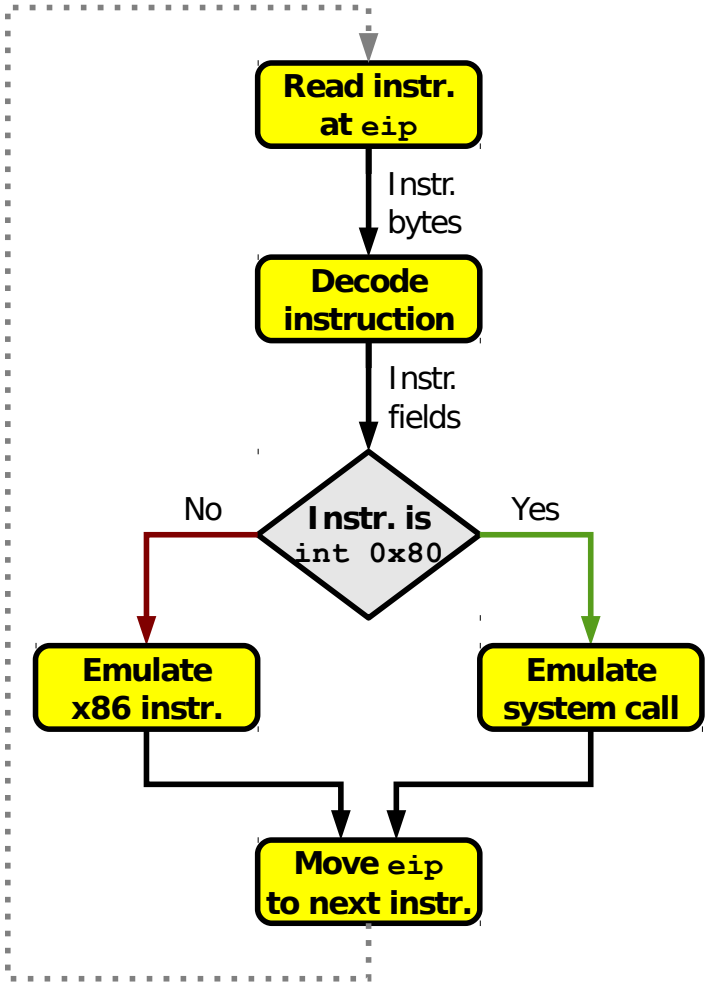
- Program headers.
- Arguments.
- Environment variables.

3) Initialize registers

- Program entry point → `eip`
- Stack pointer → `esp`

2. The x86 CPU Emulation

Simulation Loop



- **Emulation of x86 instructions**
  - Update memory map (if needed).
  - Update x86 registers.
  - Example: `add [bp+16], 0x5`
- **Emulation of Linux system calls**
  - Analyze system call code and args.
  - Update memory map.
  - Update eax with return value.
  - Example: `read(fd, buf, count);`

3. The Evergreen GPU Emulation

OpenCL program binaries

OpenCL Host Program

vector_add.c

```
int main()
{
        [ ... ]

    clCreateProgramWithSource(...,
        "vector_add.cl", ...);
    clCreateKernel(..., "vector_add",
        ...);

    buf1 = clCreateBuffer(..., CL_MEM_READ,
        size, ...);
    buf2 = clCreateBuffer(..., CL_MEM_READ,
        size, ...);
    buf3 = clCreateBuffer(..., CL_MEM_WRITE,
        size, ...);

    clSetKernelArg(..., 0, buf1, ...);
    clSetKernelArg(..., 1, buf2, ...);
    clSetKernelArg(..., 2, buf3, ...);

    clEnqueueNDRangeKernel(...);

        [ ... ]
}
```

OpenCL Device Kernel

vector_add.cl

```
__kernel void vector_add(
    __read_only __global int *buf1,
    __read_only __global int *buf2,
    __write_only __global int *buf3)
{
    int id = get_global_id(0);

    buf3[id] = buf1[id] + buf2[id];
}
```

AMD Evergreen kernel binary

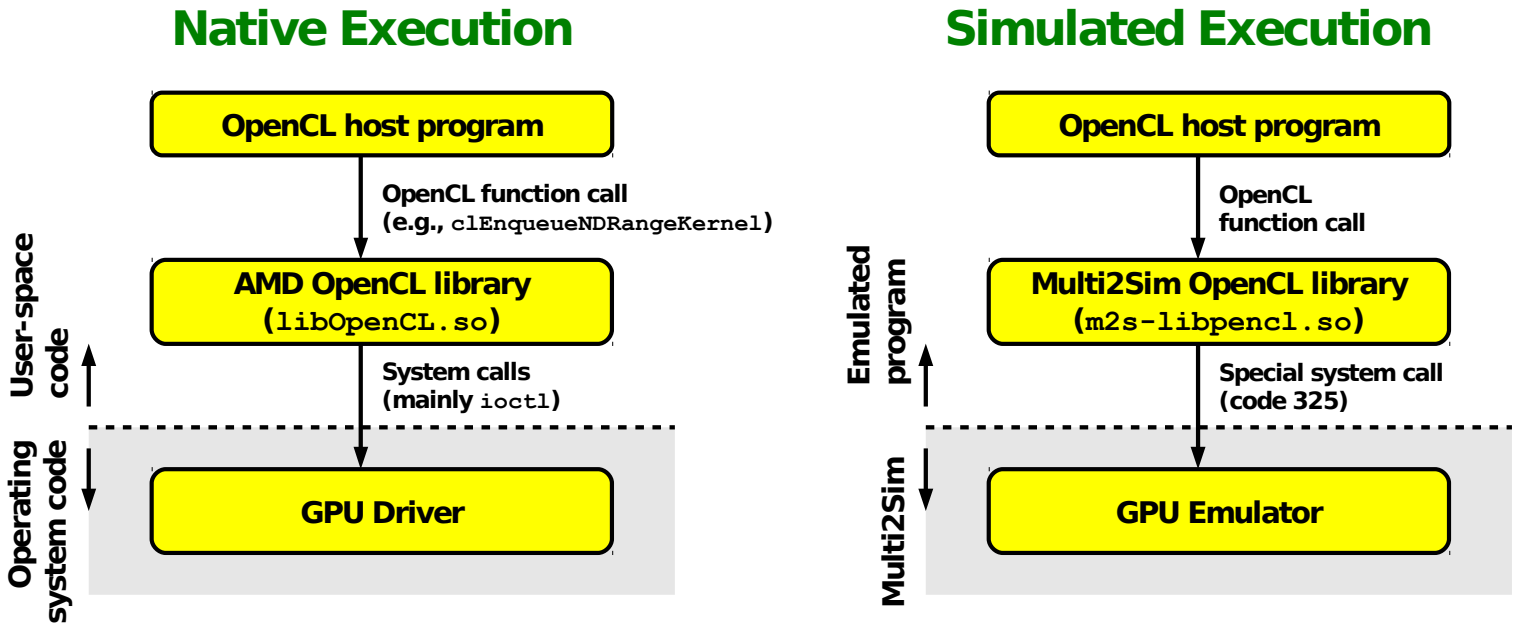
vector_add.bin

x86 executable binary

vector_add

3. The Evergreen GPU Emulation

The OpenCL Call Stack

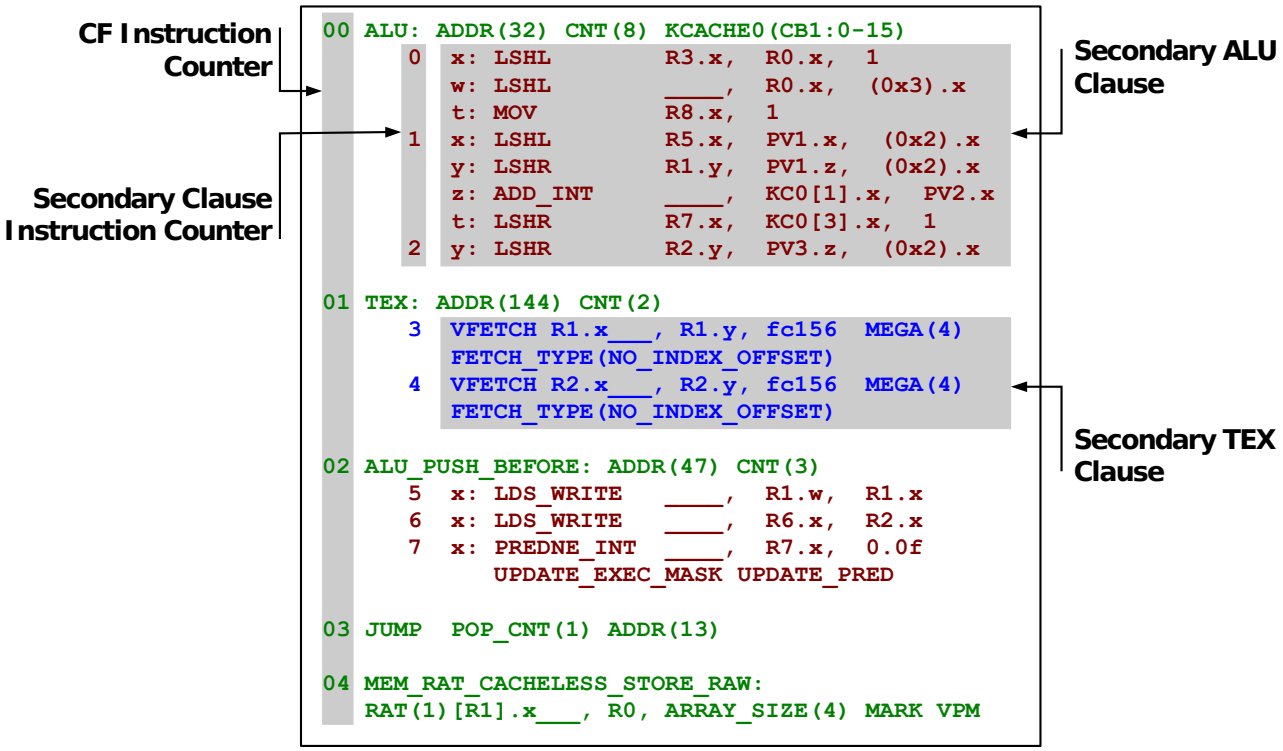


- **Comparison**
  - OpenCL function calls are forwarded to `m2s-libopencl.so`.
  - Each function is implemented as a system call 325.
  - Multi2Sim emulates GPU after `clEnqueueNDRangeKernel`.

3. The Evergreen GPU Emulation

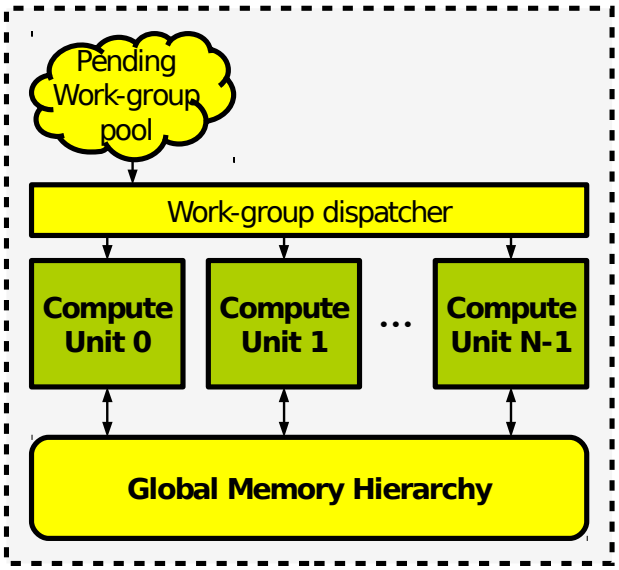
Evergreen Assembly Code

- **Structure**
  - Main Control Flow (CF) clause.
  - Secondary Arithmetic-Logic (ALU) and Texture (TEX) clauses.
  - ALU instructions are VLIW.



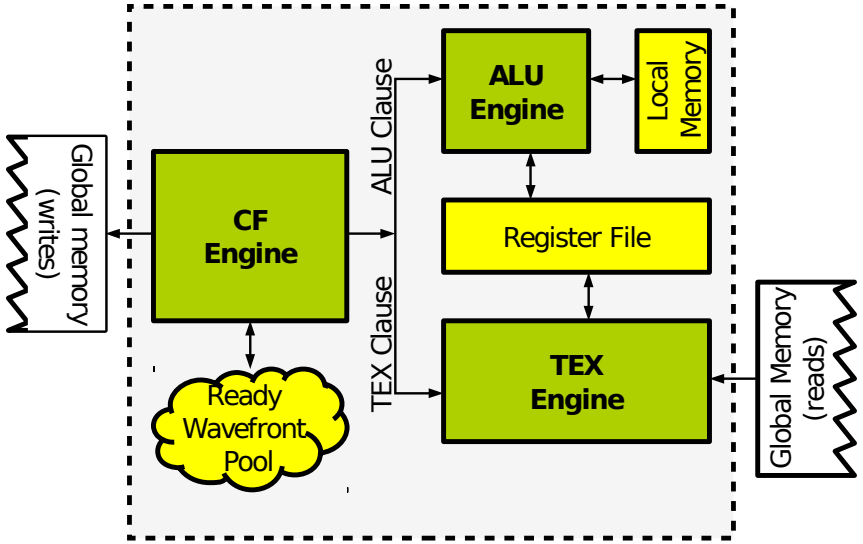
4. The GPU Architectural Simulation

AMD Evergreen GPU Architecture



- **The GPU Compute Device**
  - Pool of pending work-groups (Wgs).
  - Set of compute units (Cus).
  - Dispatcher – maps WGs to CUs.
  - Global memory hierarchy.

- **Compute Unit**
  - Pool of pending wavefronts (Wfs)
  - Three execution engines.
  - Local memory.
  - Register file.



4. The GPU Architectural Simulation

Novel Simulation Capabilities

- **Evergreen ISA Emulation**
  - Emulation of an ISA (vs. intermediate languages).
  - Support for AMD SDK 2.5.
  - Validated with self-test option in benchmarks.

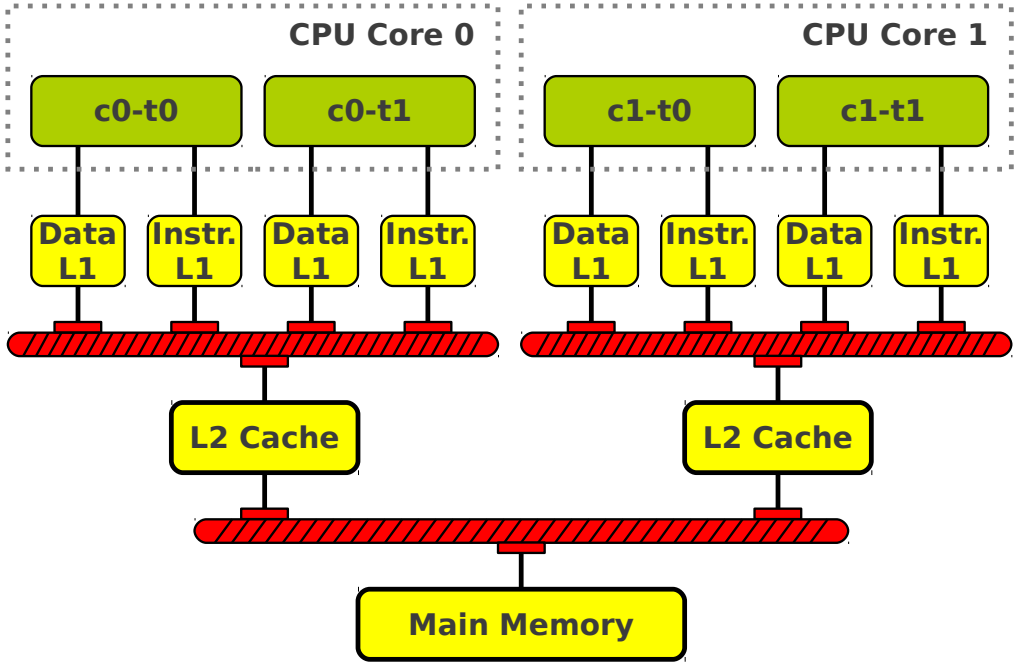
- **AMD GPU Architecture Simulation**
  - Instruction pipelines for CF/ALU/TEX engines.
  - Configurable hardware structures.
  - Configurable global memory hierarchy.
  - Validated architectural model of ATI Radeon 5870.

Demo

5. The Memory Hierarchy

CPU Memory Hierarchy

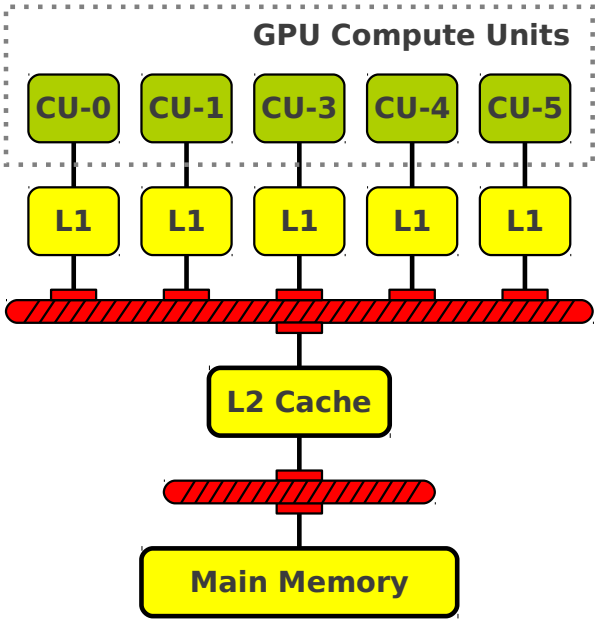
- **Example**
  - CPU with 2 cores and 2 threads per core.
  - Cache coherence: MOESI protocol.



5. The Memory Hierarchy

GPU Memory Hierarchy

- **Example**
  - GPU with 5 compute units.
  - No cache coherence (enforced by programming model)
  - False sharing solved with write bit masks and merging.

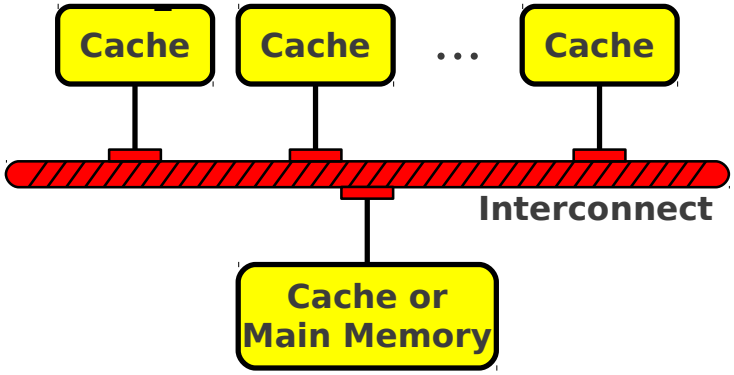


Demo

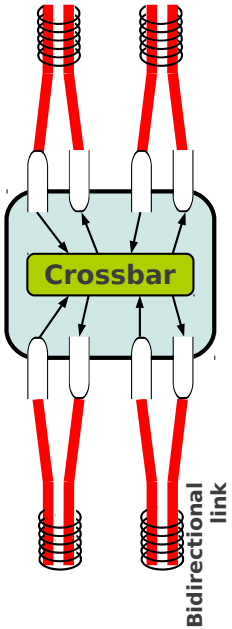
5. The Memory Hierarchy

Interconnection Networks

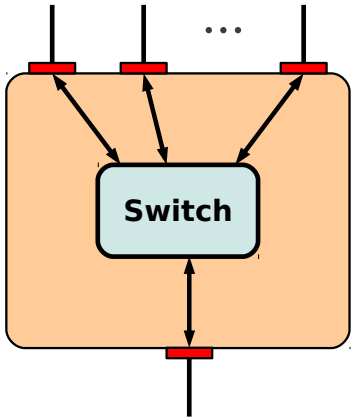
Two levels of cache



Switch



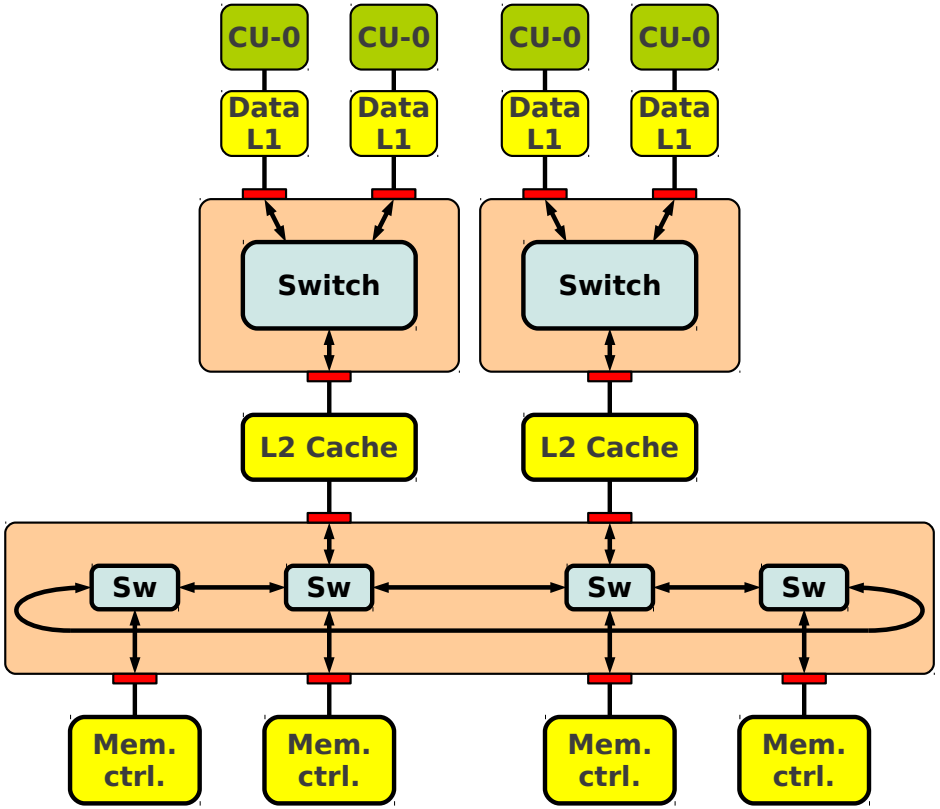
Default interconnect



5. The Memory Hierarchy

Interconnection Networks

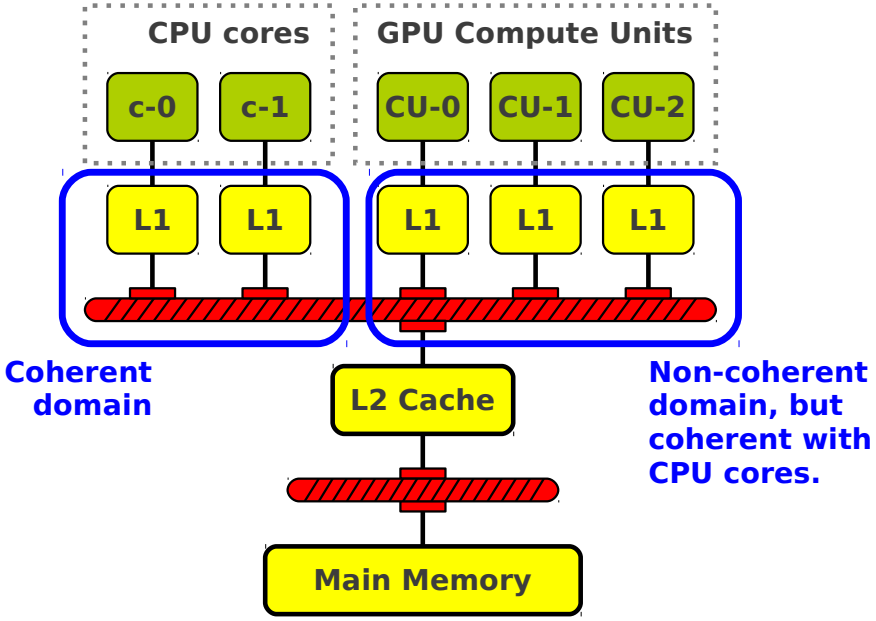
Custom interconnect example



6. Ongoing Work

Architectural Simulation

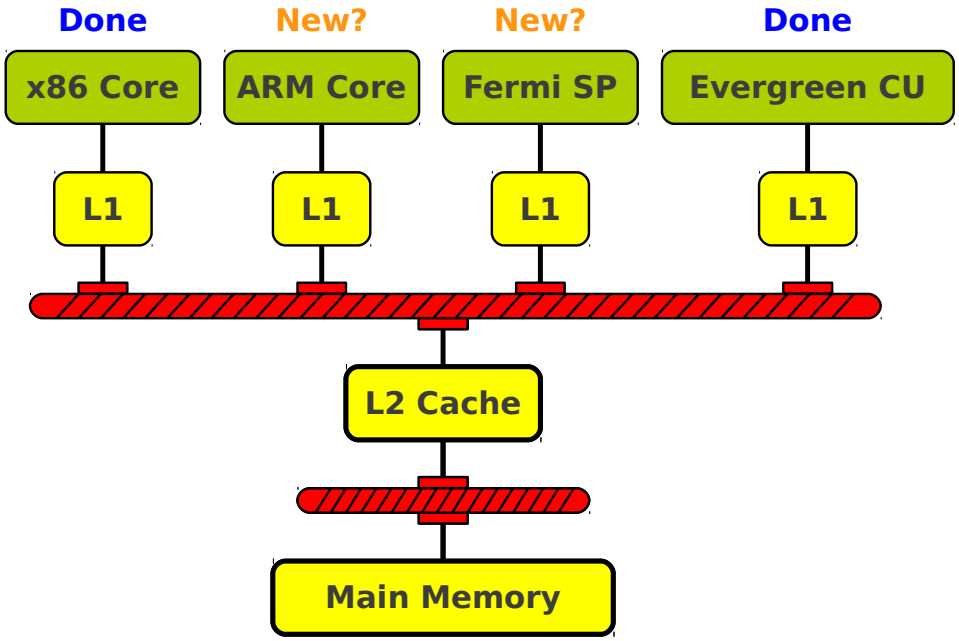
- **Fusion system with cache coherence**
  - Common memory hierarchy and protocol for CPU/GPU.
  - Extension of MOESI protocol to allow for shared writes.



6. Ongoing Work

Functional Simulation

- **Extension to new architectures**
  - ARM CPU
  - NVIDIA Fermi GPU



6. Conclusions

- **Other resources available online**
  - Mailing list for announcements of new versions.
  - Multi2Sim forum.
  - Simulator guide with execution examples.
  - Tools to manage configuration & statistic text files.

- **Multi2Sim's present and future**
  - Currently used by many research groups.
  - Currently being extended to latest architecture trends.
  - Growing team of developers.