Architectural Simulation with gem5

Minicourse - SSCAD 2024

Iago Caran Aquino <i198921@dac.unicamp.br>

Sandro Rigo <srigo@unicamp.br>

Lucas Wanner <wanner@unicamp.br>

Laboratory of Computer Systems (LSC)
Institute of Computing, University of Campinas (IC-UNICAMP)

Minicourse• October 25th 2024, São Carlos-SP

ABOUT US

Sandro Rigo

Associate Professor

IC-UNICAMP

Iago Aquino

MSc Student

IC-UNICAMP

Lucas Wanner

Associate Professor

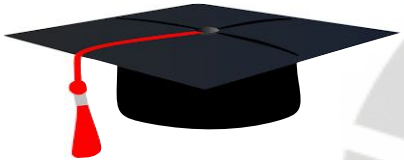
IC-UNICAMP

Campinas

Unicamp

1,900+ Faculty

8,000+ Staff



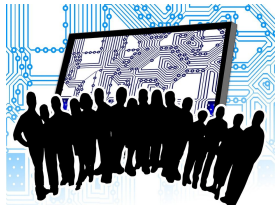
36,000+ Students
- **17,000+** Grad
- **19,000+** Undergrad

#1 University in Patent Filings
#1 pub/professor rate

Annual Budget
~0.9 Billion USD

Unicamp

IC-Unicamp



2 **Undergraduate Courses**
Computer Science &
Computer Engineering

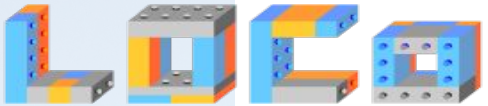
50-year old
Computer Science Course

2 **Graduate Courses**
Masters and Ph.D.
in Computer Science

Strong
International Collaboration

Research Labs

LSC COMPUTER SYSTEMS LABORATORY



Laboratory of Optimization and Combinatorics

Computer Systems Laboratory

{} LIDS

RECOD reasoning for complex data

LASCA LABORATÓRIO DE SEGURANÇA E CRIPTOGRAFIA

LASER

LSD laboratóriodesistemasdistribuídos UNICAMP

LMCAD

O Laboratório de Sistemas de Computação (LSC)

LSC COMPUTER SYSTEMS LABORATORY

https://www.linkedin.com/company/lsc-unicamp

https://www.instagram.com/lsc.unicamp/

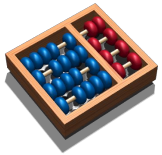
O Laboratório de Sistemas de Computação (LSC)

LSC COMPUTER SYSTEMS LABORATORY

https://www.linkedin.com/company/lsc-unicamp

https://www.instagram.com/lsc.unicamp/

Link para o material



https://tinyurl.com/gem5-sscad24

Outline

- Introduction
- gem5 Overview
- Environment Setup
- Building and running a model
- ISA extension

SECTION

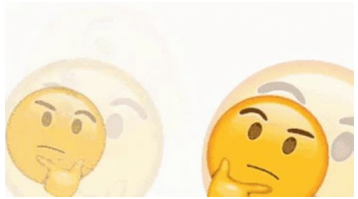
Introduction

What you are going to learn

The goal of this tutorial is to show how you can use gem5 to add and experiment with new matrix acceleration instructions to the RISC-V architecture

- Architectural Simulation
- gem5 overview
- How to setup a container for your experiments
- ISA modification
- How to run your code without having a compiler for your new instructions

But ... *why matrix acceleration???*

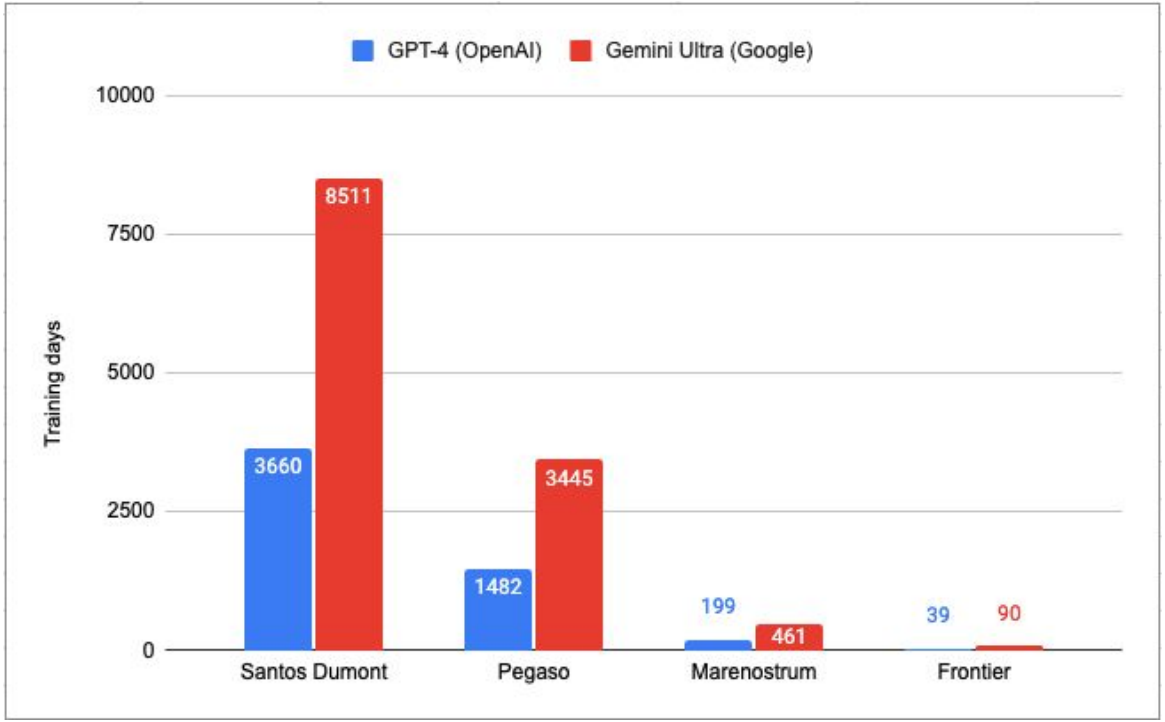


AI has become HPC

Year	AI Example	Computational Resources
2010	Deep Learning	Single GPUs or small clusters
2017	Transformers GPT-3 175 billion parameters	Hundreds of GPUs for weeks or even months
2023	GPT-4 170 trillion parameters	Thousands of devices (GPUs, TPUs, etc.) for months

HPC

How hard is to train a LLM model?



Matrix Multiplication is the most important operation in this task

* Estimations based on data published by the vendors

Matrix Multiplication

- BLAS and BLIS libraries optimize memory placement
- Computation is not the limiting factor for best performance
- Caching is very important

Sequential Multiply

$$\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

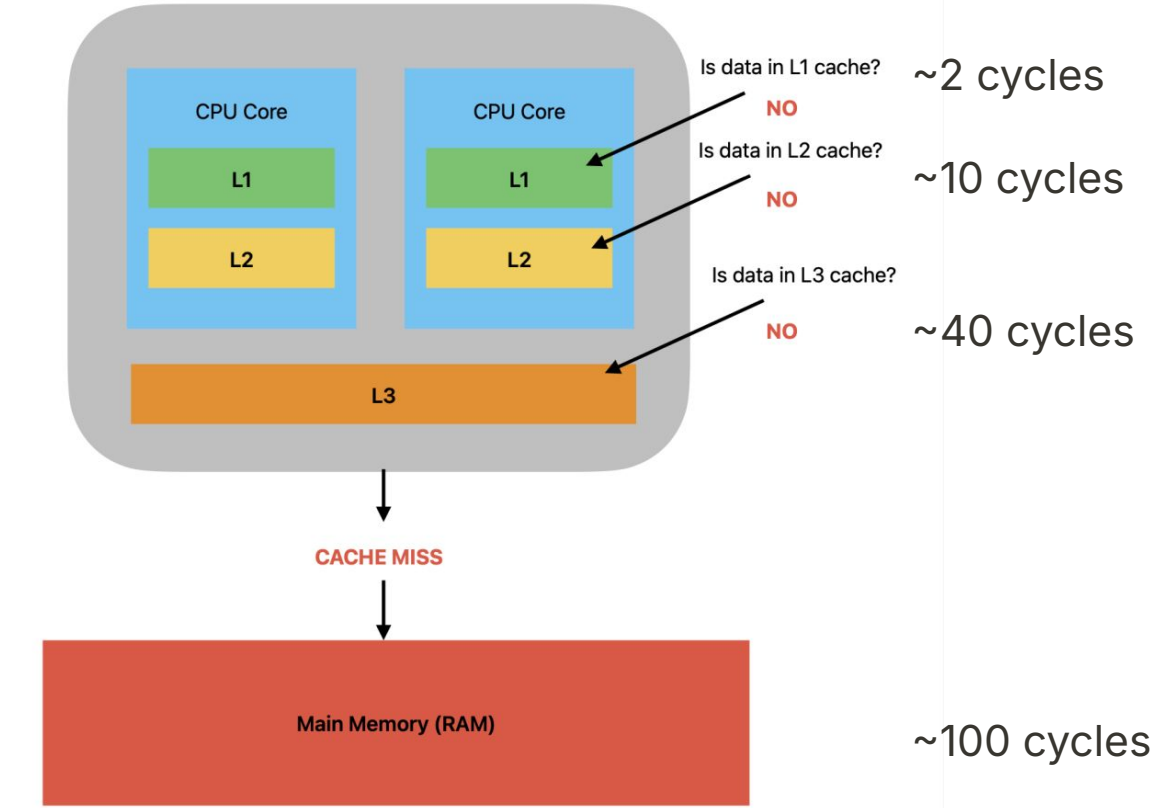
Vector Multiply

$$\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

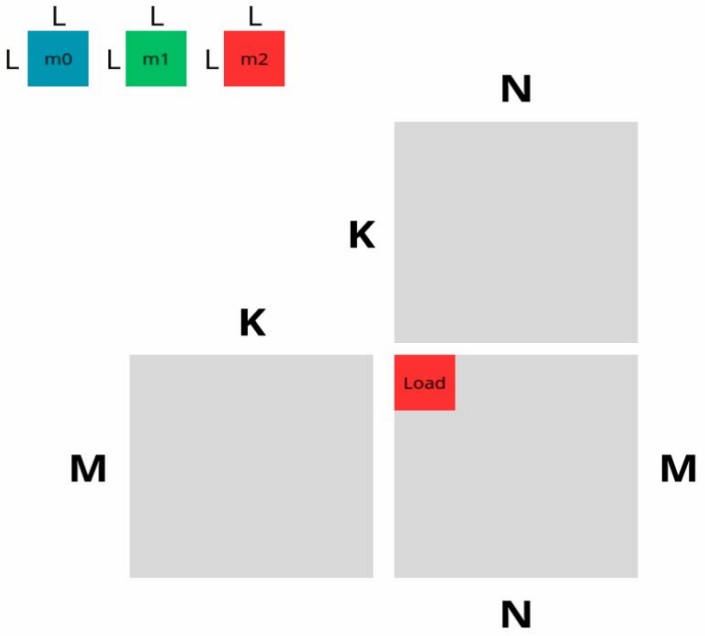
Matrix Multiply

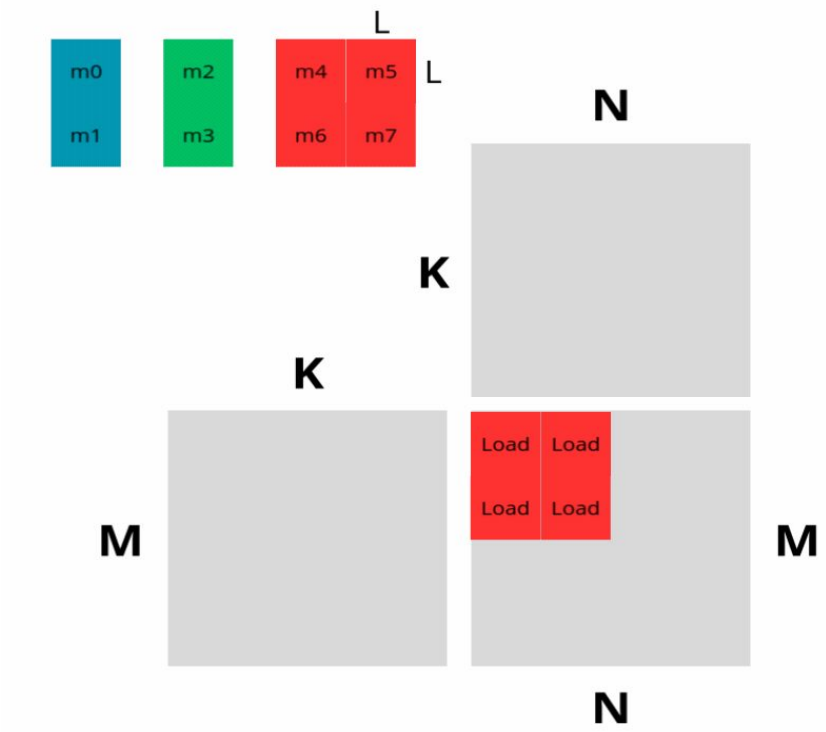
$$\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

Cache Hierarchy



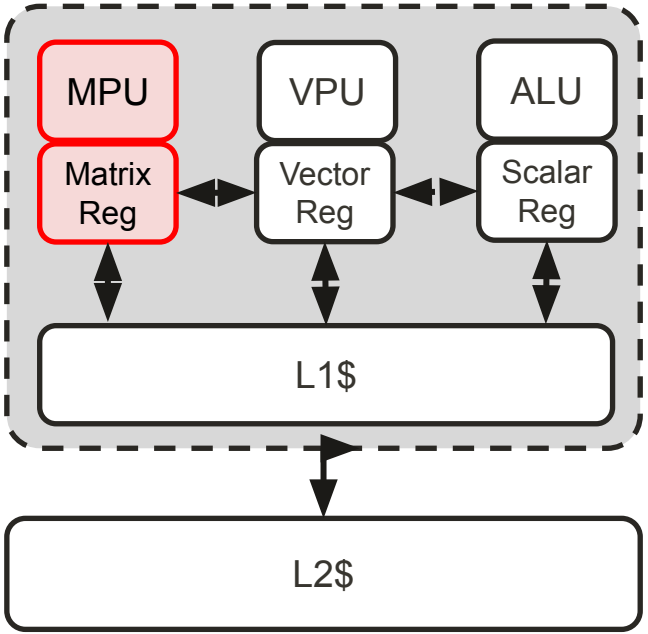
Matrix Multiplication





[1] Kazushige Goto and Robert A. van de Geijn. 2008. *Anatomy of high-performance matrix multiplication.* ACM Trans. Math. Softw. 34, 3, Article 12 (May 2008), 25 pages. https://doi.org/10.1145/1356052.1356053

Attached In-core Matrix Accelerator



Attached In-Core

- Has its own Register File
- Does not directly access the Vector Registers
- Core communication using a dedicated interface
- Connected to the L1 cache

Architectural Simulation

Designing hardware involves significant trade-offs between performance, power consumption, and cost.

- Design Space Exploration
- Performance Prediction
- Verification and Validation
- Cost Reduction
  - Tapeout cost (estimates grabbed from the web)
    - Old Tech : 180nm; $20,000 - $100,000
    - Advanced: 7nm - 15nm; tens of millions
    - Cutting Edge: 3nm; 1 Billion USD (Apple's M3, M3 Pro and M3 Max)

SECTION

gem5 Overview

Why gem5?

- Modular simulator platform
- System-level architecture and processor microarchitecture
- More than **4500** citations
- Used by: ARM Research, AMD Research, Google, Micron, HP, Samsung, and others

gem5's Components

gem5/src/python/gem5/components

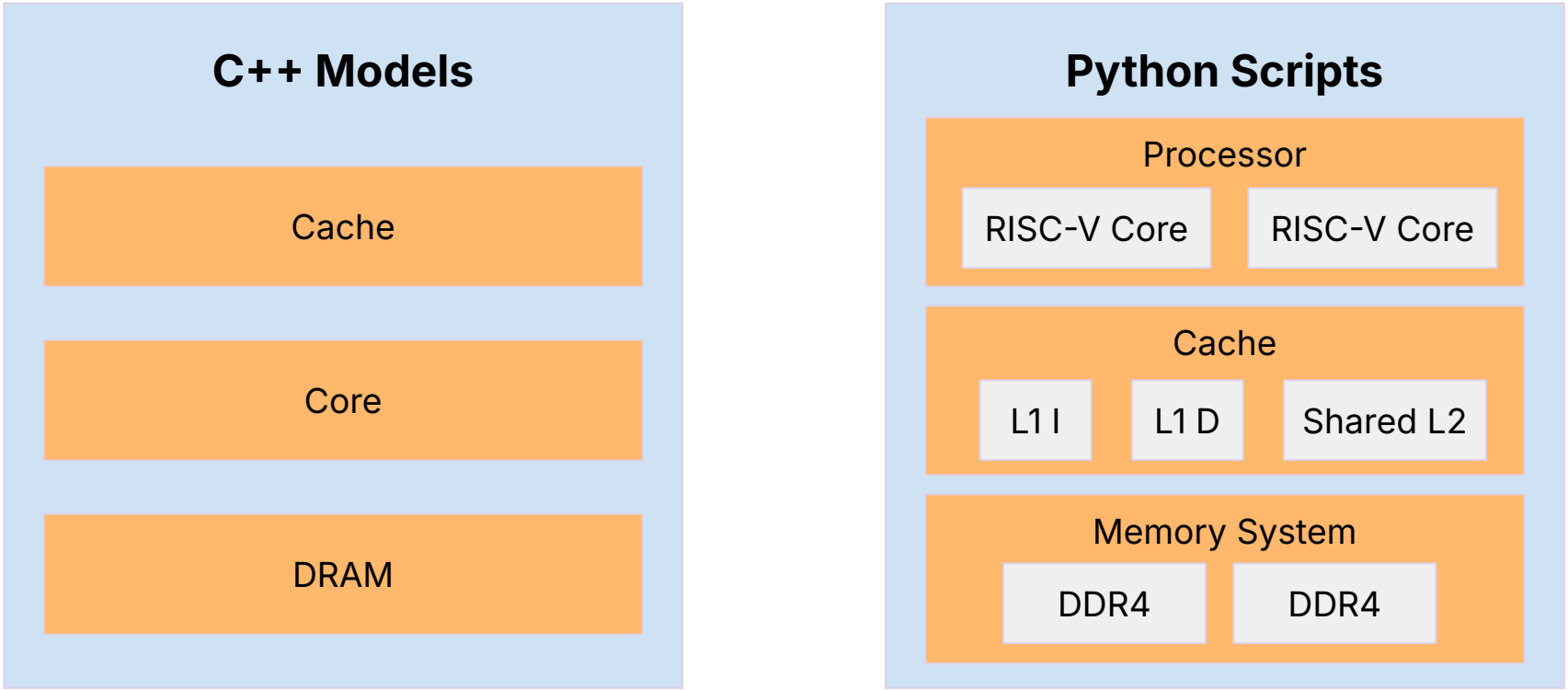
```
----/boards
----/cachehierarchies
----/memory
----/processors
```

gem5/src/python/gem5/prebuilt

```
----/demo/x86_demo_board
----/riscvmatched
```

- gem5 stdlib in src/python/gem5
- Two types
  - Prebuilt: full systems with set parameters
  - Components: Components to build systems
- Prebuilt
  - riscvmatched: Models SiFive Unmatched

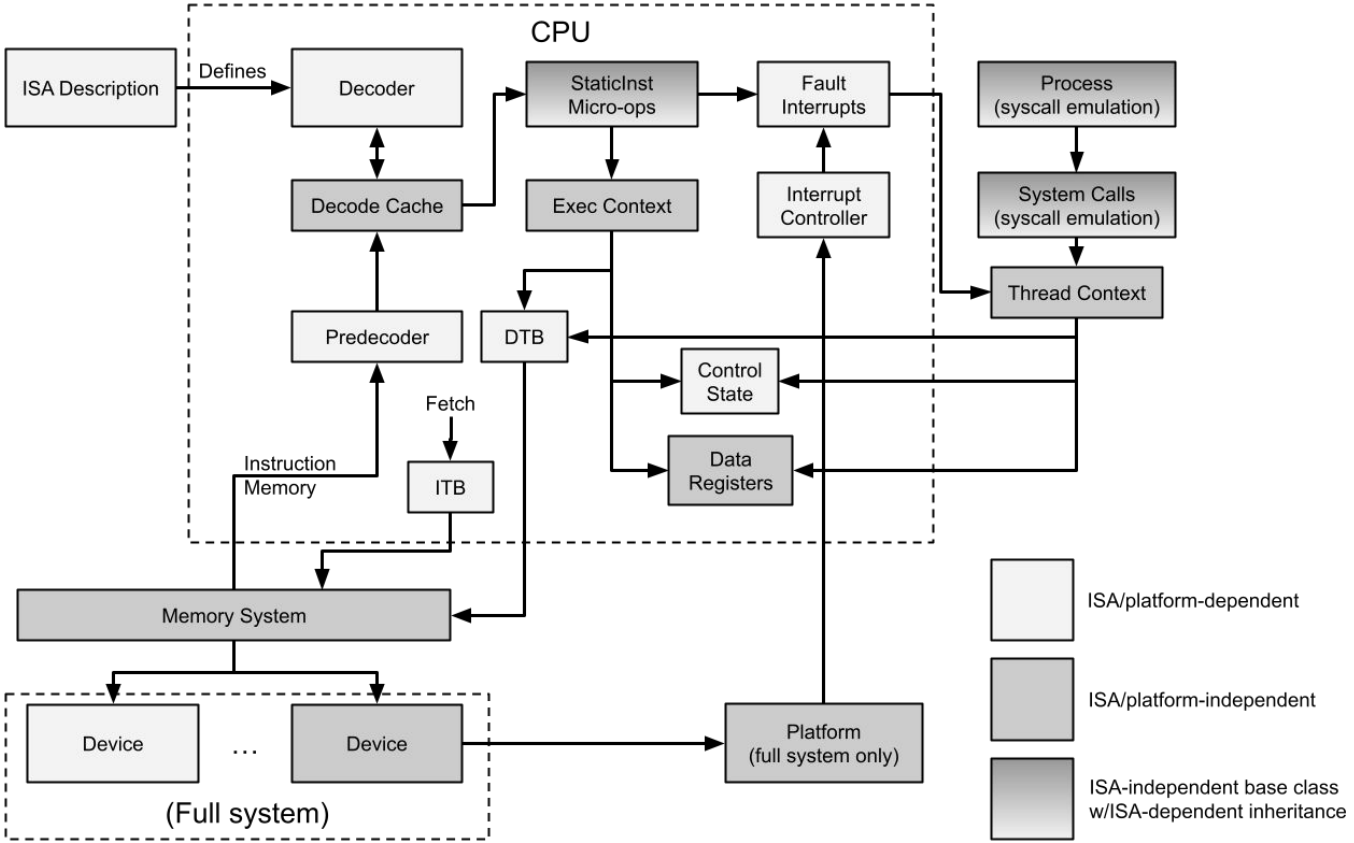
gem5's Architecture



Components: Processors

- SimpleCPU
  - TimingSimpleCPU: Every instruction takes 0 cycles plus fetch time.
  - AtomicSimpleCPU: Does not run memory system timing.
- MinorCPU
  - In-order 4 stage pipeline with Branch Prediction, etc.
- O3CPU
  - Out-of-order with all the details, Reorder Buffer, Branch Prediction, Physical Register File, etc.
- KVMCPU

CPU Model and ISA Separation



Syscall Emulation vs Full System Simulation

SE-Mode

- Emulates only the CPU and memory hierarchy
- System Calls are relayed to the host OS
- No boot time
- Only supports Linux

FS-Mode

- Emulates the complete computer system (CPU, memory, I/O devices, OS, etc)
- Detailed analysis of hardware-software interactions
- Configuration takes more time and resources

SECTION

Environment Setup

Installing RISC-V Toolchain


```
sudo apt install -y autoconf automake autotools-dev curl python3 python3-pip libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev ninja-build git cmake libglib2.0-dev libslirp-dev

export RISCV=/opt/riscv

export PATH=$PATH:$RISCV/bin

git clone https://github.com/riscv/riscv-gnu-toolchain -b 2024.04.12

cd riscv-gnu-toolchain

./configure --prefix=$RISCV --enable-multilib

make -j`nproc`

cd ..

rm -rf riscv-gnu-toolchain
```

Getting gem5

sudo apt install -y build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python-dev python

git clone https://github.com/gem5/gem5.git

cd gem5

git checkout tags/v24.0.0.1

python -m venv .venv

source .venv/bin/activate

pip install -r requirements.txt

pre-commit install

pre-commit install -t prepare-commit-msg

scons build/RISCV/gem5.opt -j`nproc`

Using the Dockerfile

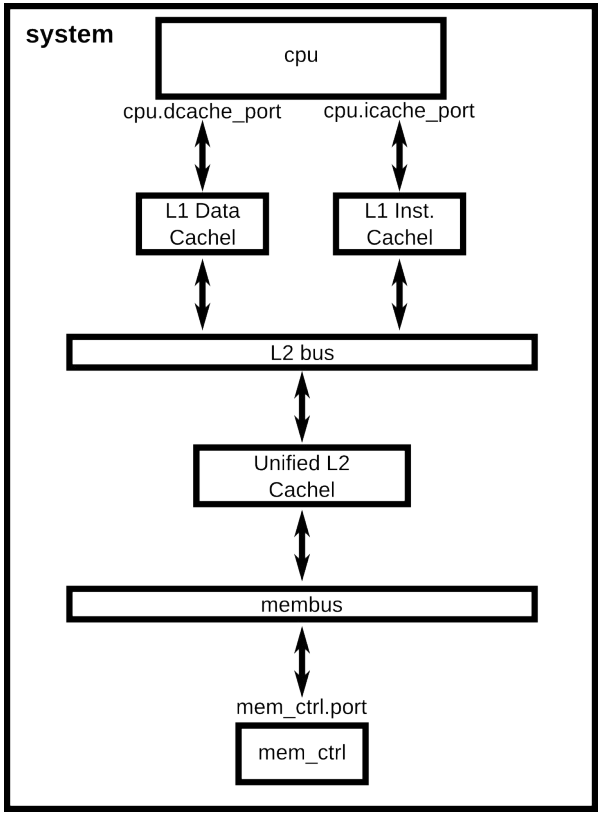

```
mkdir sscad
cd sscad
git clone https://github.com/LSC-Unicamp/gem5-sscad-2024.git .
./setup-env.sh
```

```
docker run -i -t -v ./gem5:/home/gem5 riscv-tools
```

SECTION

Building a gem5 script

Simulation Script




```
system = System()

system.cpu = RiscvAtomicSimpleCPU()

system.mem_mode = "atomic"
system.mem_ranges = [AddrRange("512MB")]
system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR4_2400_8x8()
system.mem_ctrl.dram.range = system.mem_ranges[0]

system.cpu.icache = L1ICache()
system.cpu.dcache = L1DCache()

system.l2bus = L2XBar()

system.l2cache = L2Cache()
system.l2cache.connectCPUSideBus(system.l2bus)
system.membus = SystemXBar()
system.l2cache.connectMemSideBus(system.membus)

system.workload = SEWorkload.init_compatible(binary)
```

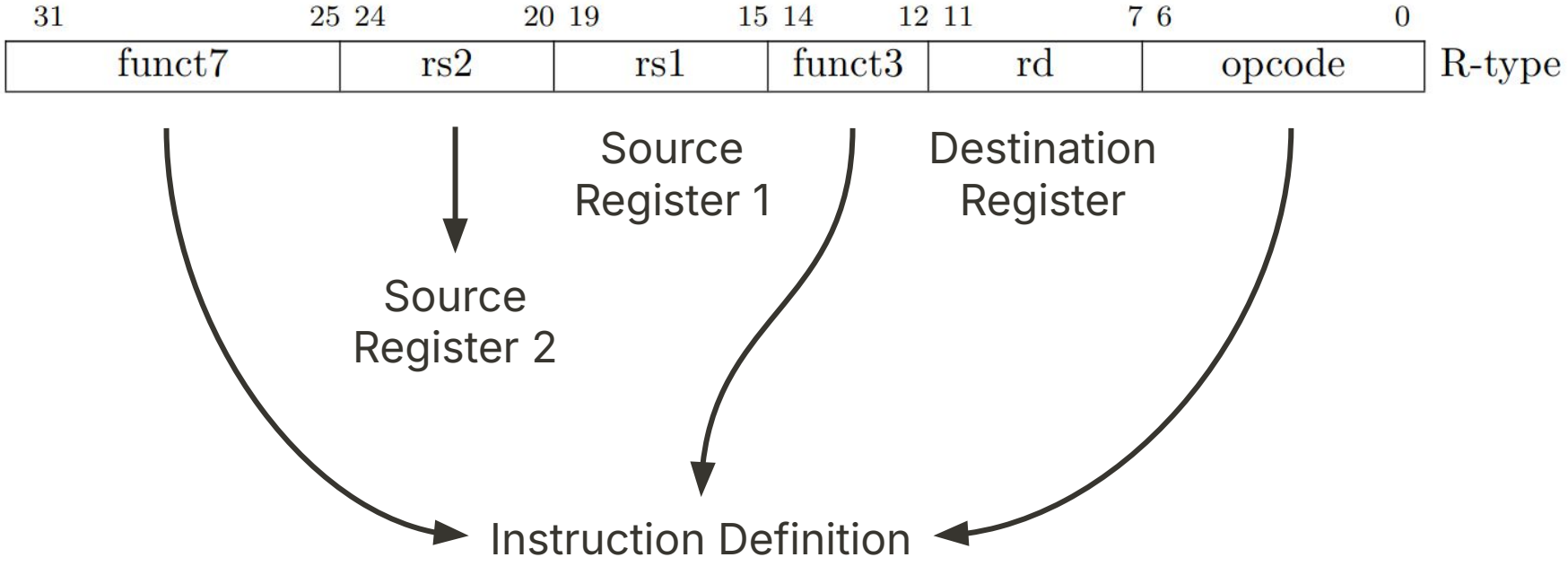
SECTION

Demo: Running gem5

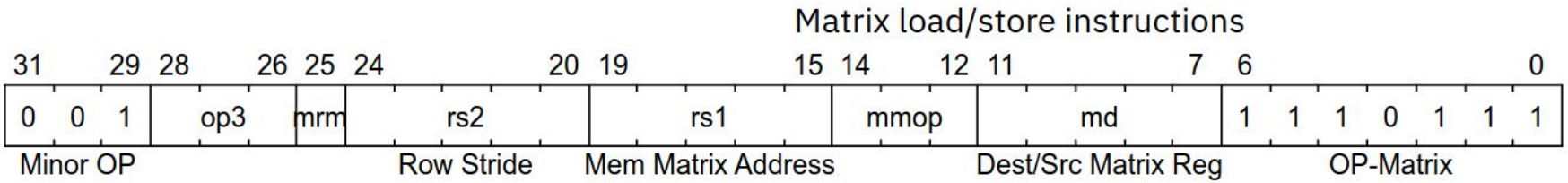
SECTION

Matrix Instructions

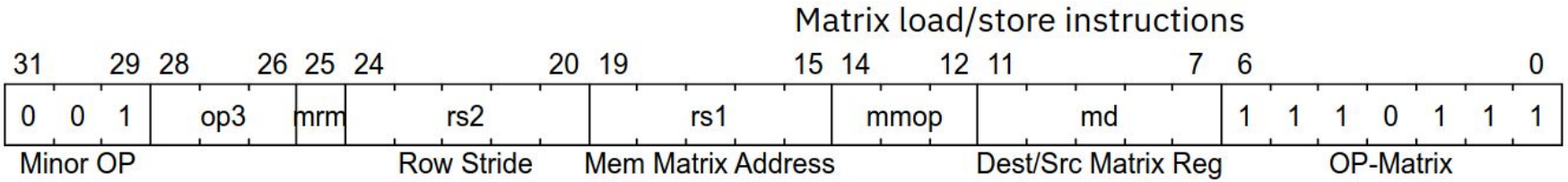
Instruction Encoding



Instruction Encoding



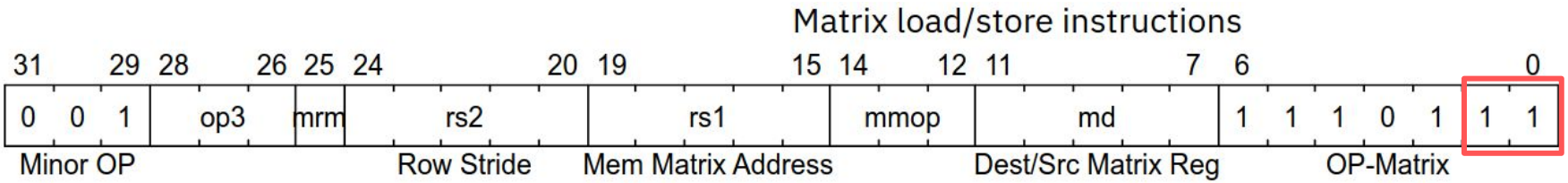
Instruction Encoding



Matrix Load: `0b001 000 1 rs2 rs1 000 md 1110111`

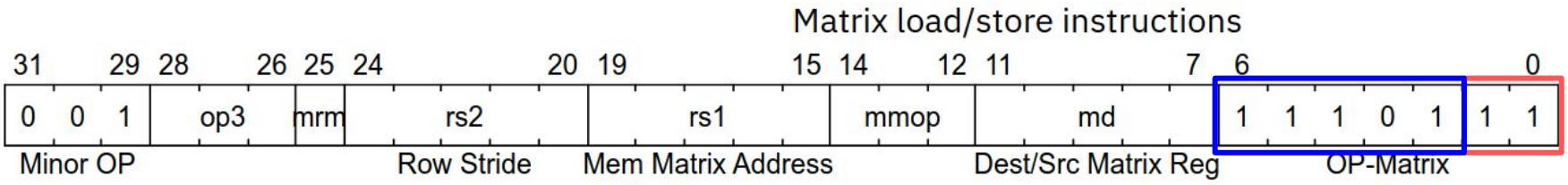
- op3, mrm - not being used for decode in this example
- mmop - indicates the type of memory operation (load/store)

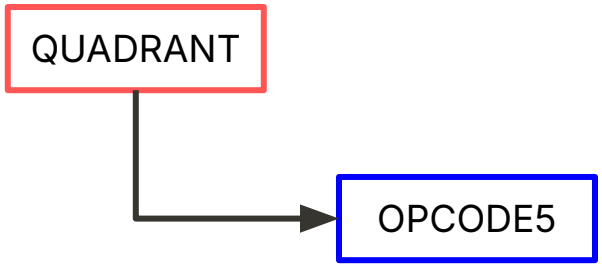
Instruction Decoding



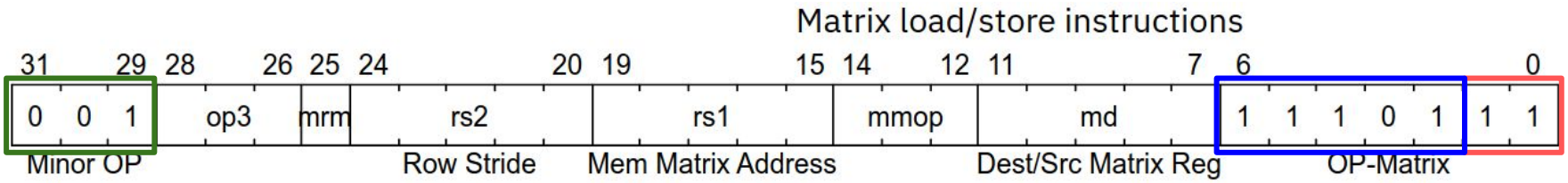
QUADRANT

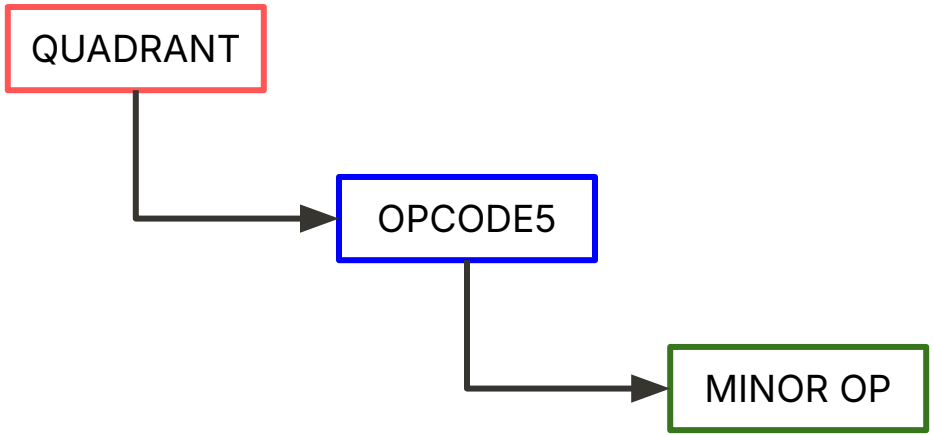
Instruction Decoding





Instruction Decoding





Using the Instructions


```
mls m0, t0, t1 - 0b 001 0001 00101 00110 100 00000 1110111

asm(".word 0b00100010010100110100000001110111")
```

Using the Instructions


```
mls m0, t0, t1 - 0b 001 0001 00101 00110 100 00000 1110111

asm(".word 0b0010001001010011010000001110111")

#define MLS(md, rs1, rs2)   ".word 0b0010001" rs2  rs1 "100" md "1110111"

asm(MLS(00000, 00101, 00110))
```

Using the Instructions


```
mls m0, t0, t1 - 0b 001 0001 00101 00110 100 00000 1110111

asm(MLS(00000, 00101, 00110))

#define MLS(md, rs1, rs2)   ".word 0b0010001" rs2  rs1 "100" md "1110111"

#define M0                          "00000"
#define M1                          "00001"
#define M2                          "00010"

#define T0                          "00101"
#define T1                          "00110"
#define T2                          "00111"

asm(MLS(M0, T0, T1))
```

Using the Instructions


```
#define M0                                  "00000"
#define M1                                  "00001"
#define M2                                  "00010"
...

#define T0                                  "00101"
#define T1                                  "00110"
#define T2                                  "00111"
...

#define MLS(md, rs1, rs2)   ".word 0b0010001" rs2  rs1 "100" md "1110111"
#define MSS(md, rs1, rs2)   ".word 0b0010001" rs2  rs1 "101" md "1110111"
...

#define MMACF(md, ms1, ms2) ".word 0b0110101" ms2  ms1 "000" md "1110111"
...
```

SECTION

Adapting the Multiplication Kernel

Naive Multiplication


```
void naive_multiply(int M, int N, int K, int ldc, int ldb, int lda, float *C, float
*A, float *B) {
  for (int i = 0; i < M; i++) {
     for (int j = 0; j < N; j++) {
      float sum = 0.0f;
      for (int k = 0; k < K; k++) {
        sum = (double) A[i * lda + k] * (double) B[k * ldb + j] + (double) sum;
       }
       C[i * ldc + j] = sum;
     }
  }
}
```

- Matrix sizes
- Leading dimensions
- Pointers to matrix data

Exploiting Data Parallelism

- Each instruction will operate over a 4×4 panel
- The remaining borders need to be treated separately
- Matrix data needs to be moved in and out of the matrix registers

Sequential Multiply

$$\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

Matrix Multiply

$$\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

Naive Multiplication


```
void naive_multiply(int M, int N, int K, int ldc, int ldb, int lda, float *C, float
*A, float *B) {
  for (int i = 0; i < M; i++) {
     for (int j = 0; j < N; j++) {
       float sum = 0.0f;
       for (int k = 0; k < K; k++) {
         sum = (double) A[i * lda + k] * (double) B[k * ldb + j] + (double) sum;
        }
        C[i * ldc + j] = sum;
     }
  }
}
```

Naive Multiplication


```
void naive_multiply(int M, int N, int K, int ldc, int ldb, int lda, float *C, float
*A, float *B) {
  int panel_size = 4;
  for (int i = 0; i < M; i += panel_size) {
     for (int j = 0; j < N; j =+ panel_size) {
      float sum = 0.0f;
      for (int k = 0; k < K; k += panel_size) {
        sum = (double) A[i * lda + k] * (double) B[k * ldb + j] + (double) sum;
       }
       C[i * ldc + j] = sum;
     }
  }
}
```

Naive Multiplication


```
void naive_multiply(int M, int N, int K, int ldc, int ldb, int lda, float *C, float
*A, float *B) {
  int panel_size = 4;
  for (int i = 0; i < M; i += panel_size) {
     for (int j = 0; j < N; j =+ panel_size) {
      float sum = 0.0f;
      for (int k = 0; k < K; k += panel_size) {
        sum = (double) A[i * lda + k] * (double) B[k * ldb + j] + (double) sum;
       }
       C[i * ldc + j] = sum;
     }
  }
}
```

Naive Multiplication


```
void naive_multiply(int M, int N, int K, int ldc, int ldb, int lda, float *C, float
*A, float *B) {
  int panel_size = 4;
  for (int i = 0; i < M; i += panel_size) {
     for (int j = 0; j < N; j =+ panel_size) {
        MZERO(M2);
        for (int k = 0; k < K; k += panel_size) {
           MLS(M0, A, lda);
           MLS(M1, B, ldb);
           MMACF(M2, M0, M1);
        }
        MSS(M2, C, ldc);
     }
  }
}
```

Naive Multiplication


```
void naive_multiply(int M, int N, int K, int ldc, int ldb, int lda, float *C, float
*A, float *B) {
  int panel_size = 4;
  for (int i = 0; i < M; i += panel_size) {
     for (int j = 0; j < N; j =+ panel_size) {
       MZERO(M2);
       for (int k = 0; k < K; k += panel_size) {
         MLS(M0, A, lda);
         MLS(M1, B, ldb);
         MMACF(M2, M0, M1);
       }
       MSS(M2, C, ldc);
     }
  }
}
```

Naive Multiplication


```
void naive_multiply(int M, int N, int K, int ldc, int ldb, int lda, float *C, float
*A, float *B) {
  int panel_size = 4;
  for (int i = 0; i < M; i += panel_size) {
     for (int j = 0; j < N; j =+ panel_size) {
       float *C_ptr = C + j + i * ldc;
       MZERO(M2);
       for (int k = 0; k < K; k += panel_size) {
         float *A_ptr = A + k + i * lda;
         float *B_ptr = B + k + j * ldb;
         MLS(M0, A_ptr, lda);
         MLS(M1, B_ptr, ldb);
         MMACF(M2, M0, M1);
       }
       MSS(M2, C_ptr, ldc);
     }
  }
}
```

SECTION

Adding new instructions

Parts of an instruction

- ISA Description is generated by ISAParser step, using the instruction class definitions and the decoder
  - Instruction format: aggregates different templates to generate a class template composed of
    - header_output: Class declaration
    - decoder_output: Class constructor
    - decode_block: Instruction instantiation
    - exec_output: Execute implementation
  - Decoder specification
    - Specializes the class template into each instruction

Step-by-Step

1. Add required bitfields and operands
2. Create base class
3. Create instruction templates
4. Create instruction format
5. Create decoder rule

Thank You!

Email i198921@dac.unicamp.br

srigo@unicamp.br

wanner@unicamp.br

GitHub https://github.com/LSC-Unicamp/gem5-sscad-2024