TÉCNICO LISBOA

Accelerating Memory-Bound Applications with Near-Data Processing: From Architectural Design to Full-System Simulation

PhD Dissertation of João Miguel Morgado Pereira Vieira
Advanced Studies Diploma in Electrical and Computer Engineering

PhD Supervisory Committee:
Advisor: *Doctor Pedro Filipe Zeferino Aidos Tomás, Associate Professor, IST*
Co-advisors:
- *Doctor Nuno Filipe Valentim Roma, Associate Professor, IST*
- *Doctor Gabriel Falcão Paiva Fernandes, Associate Professor, UC*

Elements of the jury:
President: *Doctor Helena Maria dos Santos Geirinhas Ramos, Full Professor, IST*
Committee members:
- *Doctor Leonel Augusto Pires Seabra de Sousa, Full Professor, IST*
- *Doctor João Paulo de Castro Canas Ferreira, Associate Professor, FEUP*
- *Doctor Hervé Miguel Cordeiro Paulino, Associate Processor, NOVA FCT*
- *Doctor Pedro Filipe Zeferino Aidos Tomás, Associate Professor, IST*
- *Doctor Luís Jorge Brás Monteiro Guerra e Silva, Assistant Professor, IST*

Outline

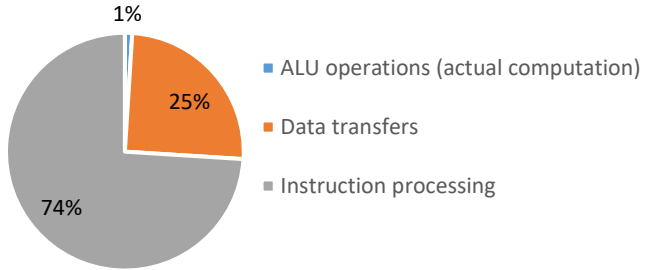
1. **Motivation and problem statement**
   - Near-Data Processing (NDP) and the challenges of Processing in Memory (PIM)
   - Processing Near Memory (PNM)
   - The scarcity of development technologies
2. **The Cache Compute System: An NDP solution for signal-processing**
   - Proposed architecture
   - Prototyping/Evaluation methodologies
3. **NDPmulator: Simulation platform for developing NDP solutions**
   - The architectural model
   - The simulation framework
   - Validation

Motivation and problem statement

- Memory systems **constrain performance**
  - **Limit bandwidth**
  - **High latency to the CPU**

Data-intensive algorithms exhaust memory resources before taking full advantage of processing hardware

CPU energy distribution



- On average, **25 % of energy is spent in data transfers**

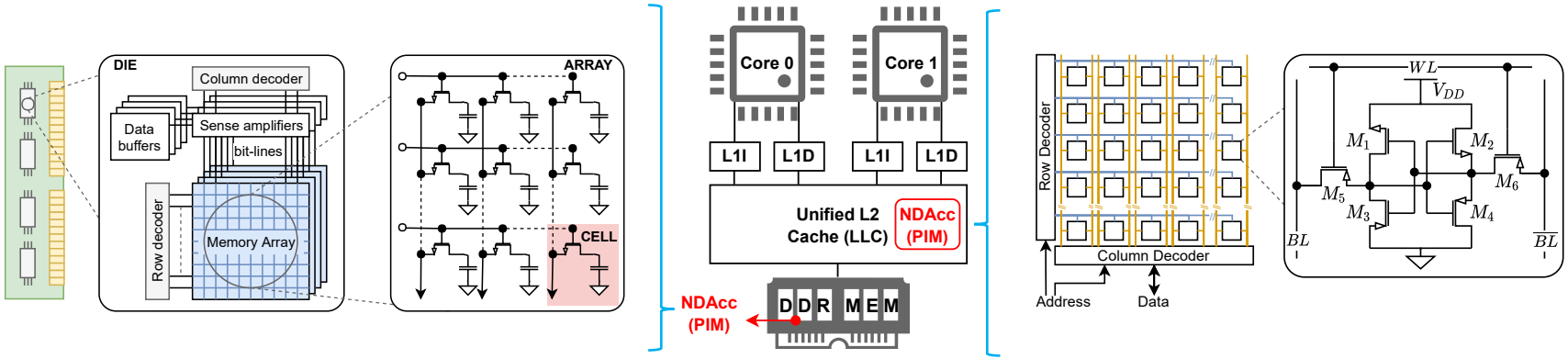
  Aga S. et al.: "Compute Caches". In HPCA, 2017.

- And that **fraction** can be **as high as 64 %**

  Boroumand, A. et al.: "Google workloads for consumer devices: Mitigating data movement bottlenecks." In ASPLOS, 2018.

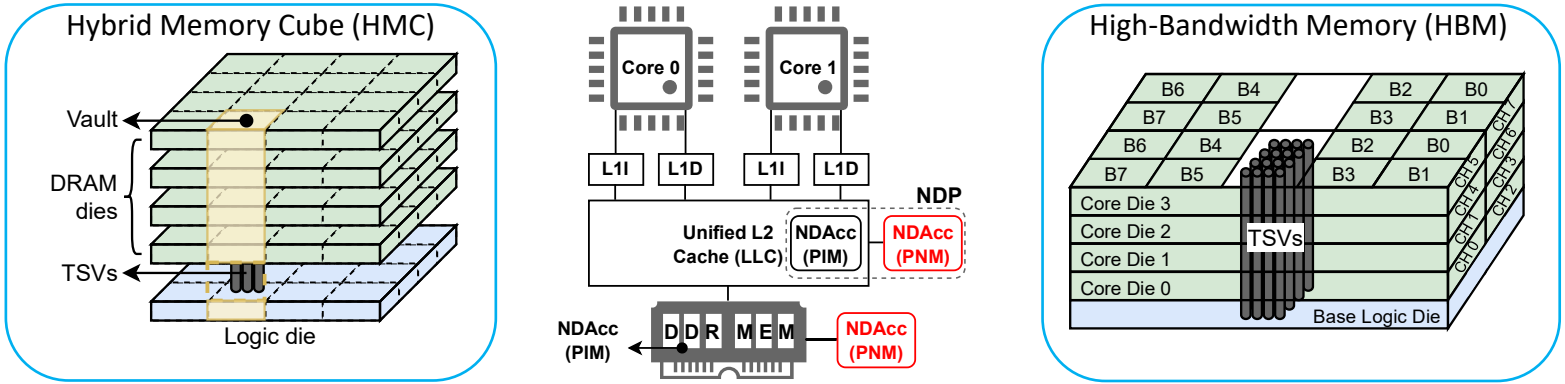
Solution: *Near-Data Processing (NDP) → Move computation near where data is stored*

Processing in Memory (PIM)



- **Processing in Memory (PIM) repurposes memory technologies** for computation
  - First solutions appeared in the 1980s
  - **Requires to reengineer memory chips**
  - **Hard to use** (each solution requires its own programming paradigm)
- **PIM uses bit-line computation**
  - Several **bit-lines are activated simultaneously**
  - Sensed result is interpreted as the **result of a simple logic operation**
  - **Only a few lines can be activated simultaneously**

Processing Near Memory (PNM)

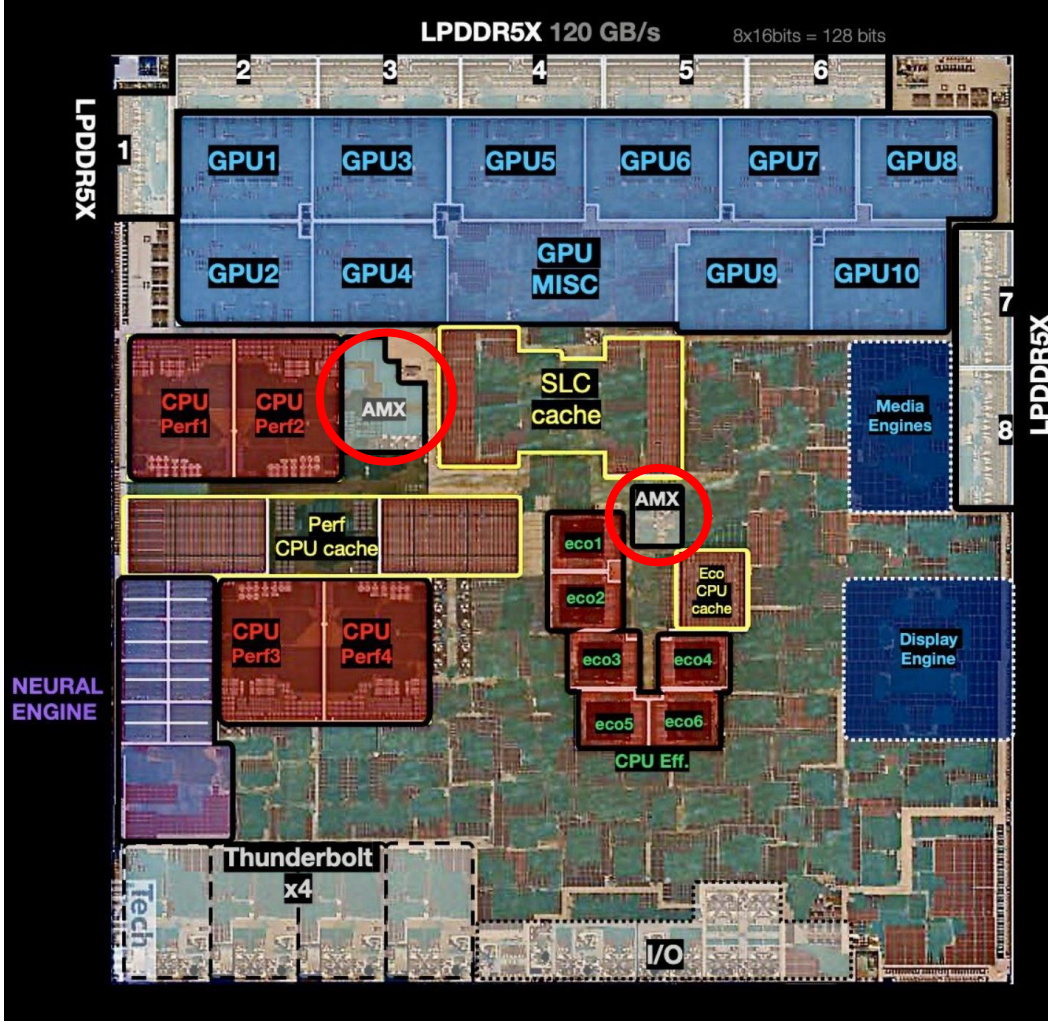


- **Processing Near Memory (PNM) devices** are installed **close** to memory devices
  - Not as efficient as PIM, but **allow more complex operations**
  - Still have **higher bandwidth** and **lower latency** than the CPU
- PNM devices can be implemented using hybrid memory technologies (e.g., HMC, HBM)
  - **3D-stacked memory arrays in the same chip as logic dies**
  - Dies communicate through **high-performance Through-Silicon Vias (TSVs)**
  - *In HBM2, bandwidth between logic and memory dies can be as high as 256 GB/s*

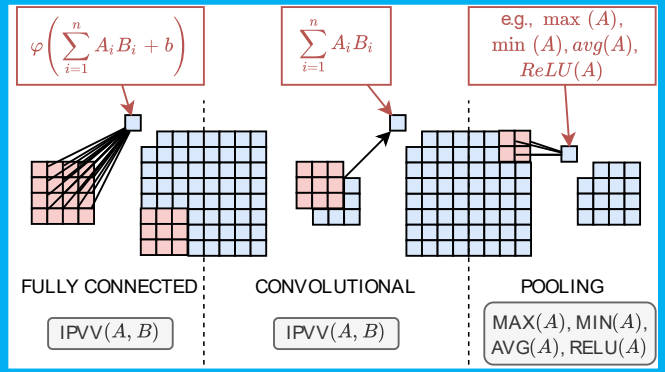
Example: Apple Silicon AMX on M4 SoC

- Motivated by the need of executing **data-intensive workloads such as Neural Networks (NNs)**
- Microarchitecture implementation of ARM SME
- **Direct access to Last-Level Cache (LLC):**
  - High-performance computing near data
  - Processing Near Memory

Note: This image is simply illustrative of the example and bears no value for further analysis



TÉCNICO LISBOA



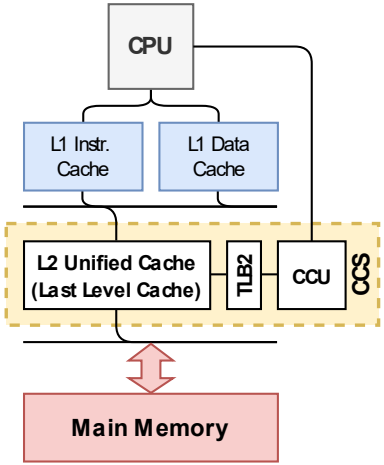
The Compute Cache System (CCS): Signal Processing, AI/Machine Learning

Convolutional Neural Networks (CNNs) are characterized by massive datasets

Depending on the CNN model, thousands of elementary operations are performed

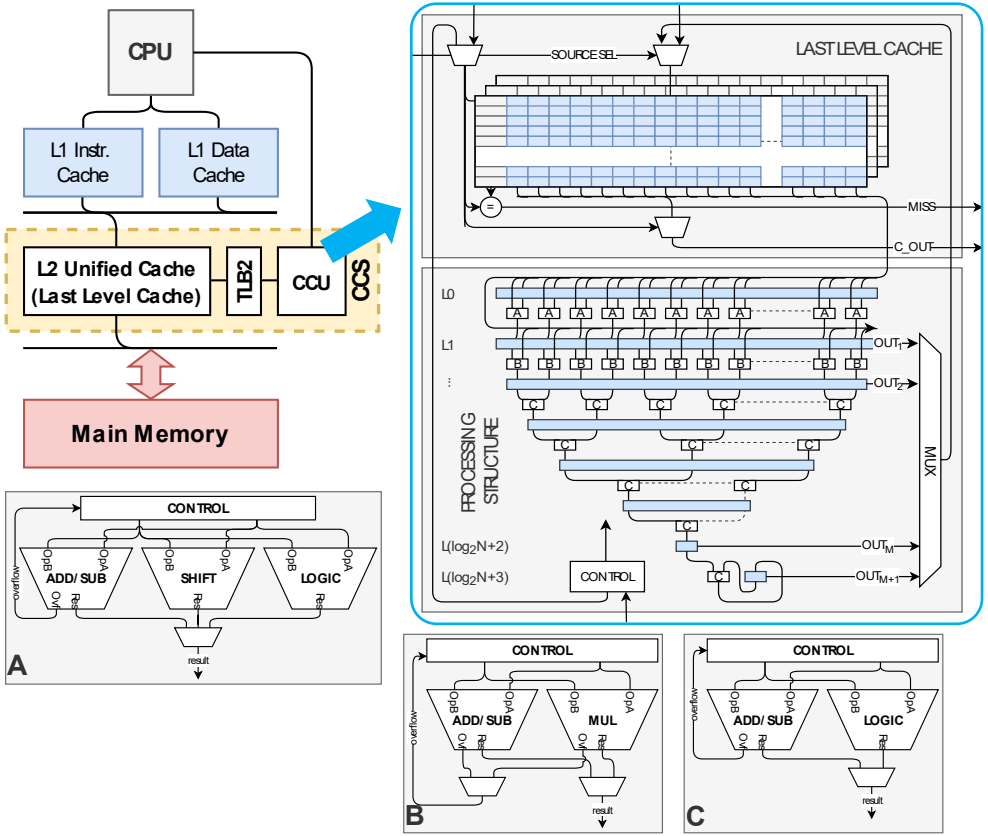
Usually, over **90 % of the CNN's workload** is due to convolutional and pooling layers

An NDP solution for signal-processing



- The proposed **Cache Compute System (CCS)** consists of **dedicated hardware to perform computation near the cache,** taking advantage of:
  - Operands locality
  - Long cache lines
  - Lower latency to the main memory
  - Existing cache coherence protocols
- **The CCS is integrated with the processing system:**
  - As a slave of the processing core
  - Communicating with the LLC to get operands and store results

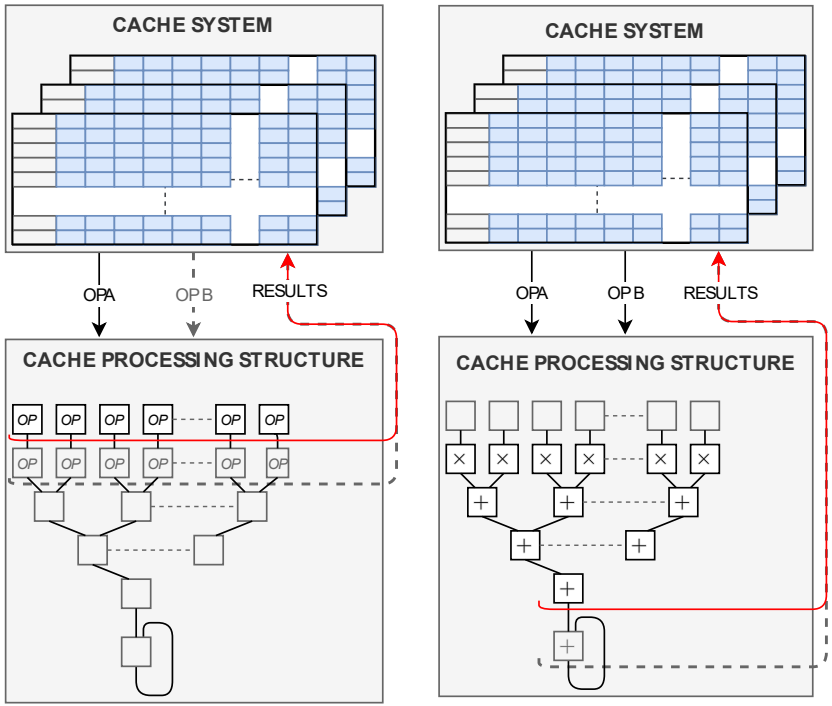
An NDP solution for signal-processing



- The micro-architecture of the CCS consists of an **inverted binary tree of Processing Elements (PEs)** optimized for:
  - **Map Operations**
  - **Reduce Operations**
- Different levels have **different PEs** to accommodate the functionality of the **different commands**

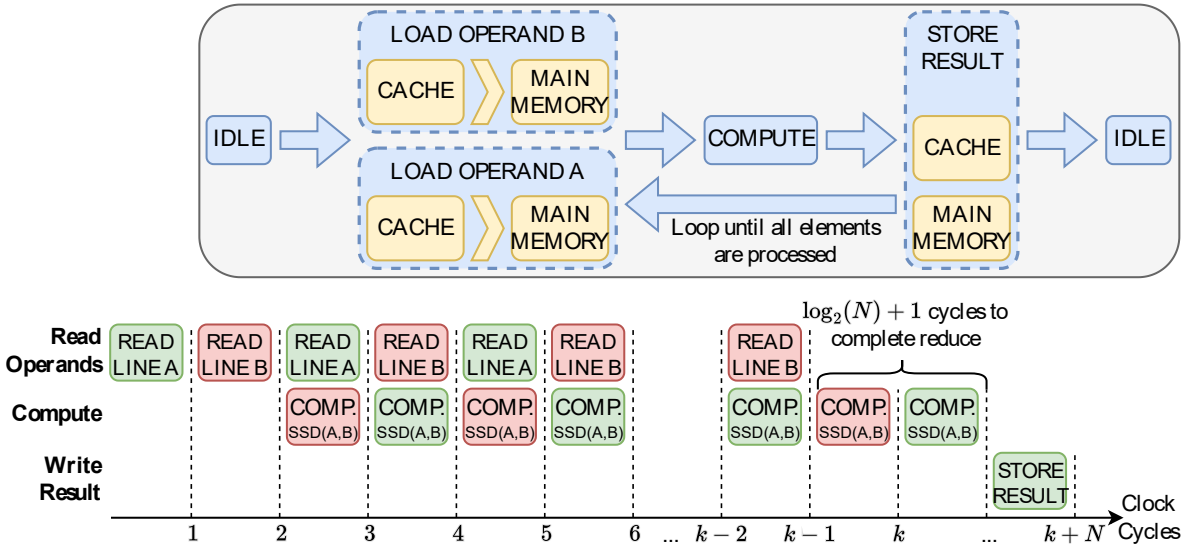
47 different commands

The CCS is optimized for Map and Reduce Operations



- **Map operations:** Same (simple) transformation in all elements of the input dataset
  - E.g.: Increment all elements of a vector; vector addition
- **Reduce operation:** Process all elements of the input dataset into a smaller output
  - E.g.: Sum of all elements of a vector; multiply and accumulate two vectors

The CCS is optimized for Map and Reduce Operations



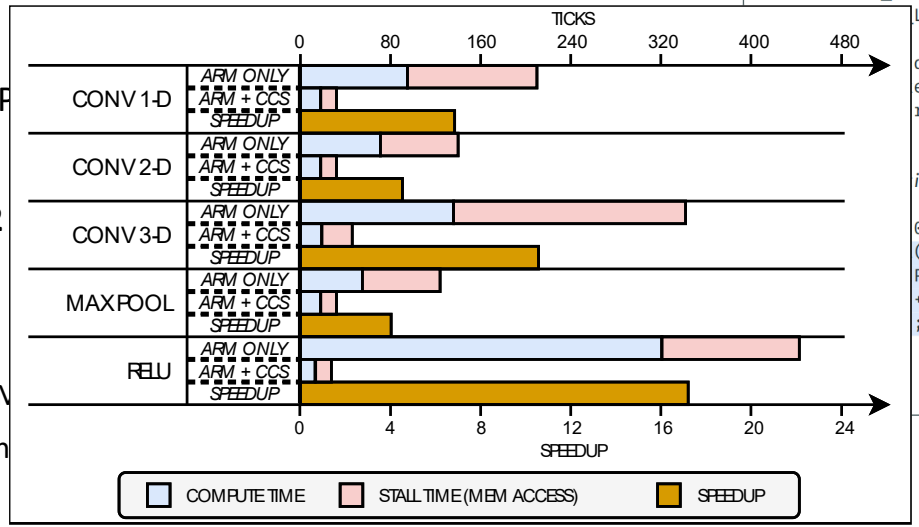
Evaluation methodology and preliminary results

- Simulated using **gem5 architectural simulator**
- **ARM Cortex-A53**
- CCS connected to the CPU through **memory-mapped interface and coupled to LLC**
- CCS can process **up to 2 16-element vectors of 32-bit fixed-point operands per cycle**
- **Programming the CCS:**
  - **Library** (written in ARMv8 Assembly) to allow low-level control over the CCS
  - **Framework** (written in C) to offload convolutional and pooling layers of CNNs to the CCS

```
#define L_SIZE          // size of cache line
#define DATA_WIDTH      // width of data matrix
#define DATA_HEIGHT     // height of data matrix
#define KERNEL_LENGTH   // size of the kernel

external int **data;    // data addr
external int *kernel;   // kernel addr
external int **result;  // result addr
int a[L_SIZE];          // data buffer

// 2-D convolution
for (int i = 0; i < DATA_WIDTH; i++) {
  for (int j = 0; j < DATA_HEIGHT; j++) {
    gather_data(a, i, j);
    ccs_setup(IPVV, KERNEL_LENGTH, 0, a, kernel,
        result + i * DATA_WIDTH + j, 1);
    ccs_start();
  }
}
ccs_wait();
```

Evaluation methodology and preliminary results

- Simulated using **gem5 architectural simulator**
- **ARM Cortex-A53**
- CCS connected to the CP
  coupled to LLC
- CCS can process **up to 2**
  operands per cycle
- **Programming the CCS:**
  - **Library** (written in ARM
  - **Framework** (written in
    CNNs to the CCS




```
#define L_SIZE          // size of cache line
#define DATA_WIDTH      // width of data matrix
#define DATA_HEIGHT     // height of data matrix
LENGTH  // size of the kernel

data;   // data addr
ernel;  // kernel addr
result; // result addr
        // data buffer

ion
 i < DATA_WIDTH; i++) {
0; j < DATA_HEIGHT; j++) {
(a, i, j);
PVV, KERNEL_LENGTH, 0, a, kernel,
+ i * DATA_WIDTH + j, 1);
;
```

Learnings from the CCS

Developing Near-Data Accelerators (NDAccs) can be complex (they are closely coupled with the existing memory hierarchy)

- **Fabricating chips is expensive**
- **FPGA-based prototyping is slow,** and the available resources may not suffice to emulate the entire processing system
- **Existing simulators have limited support for NDAccs**
  E.g., gem5, gem5-X, gem5-Aladdin, gem5-SALAM, ZSim, Ramulator.
- There is a need for a simulation platform dedicated to the development of NDAccs that allows to simulate entire NDP-enabled systems

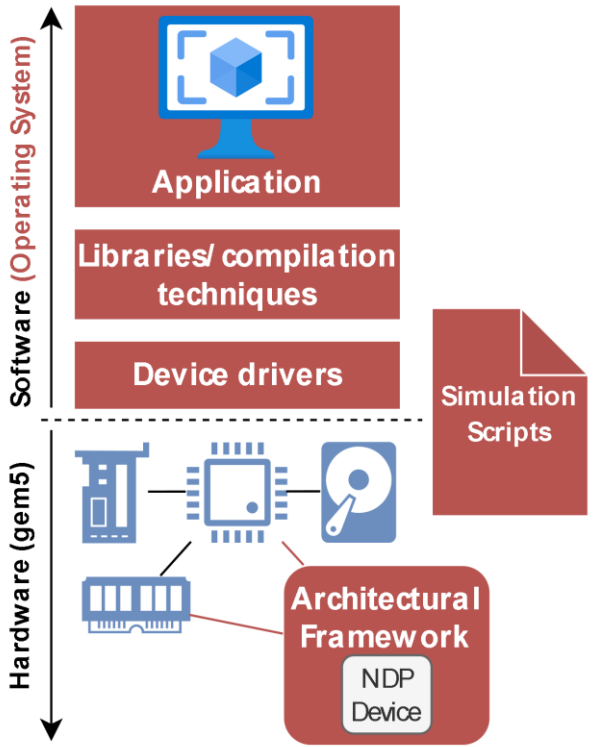
There is a need for a simulation platform dedicated to the development of NDAccs that allows to simulate entire NDP-enabled systems

TÉCNICO LISBOA

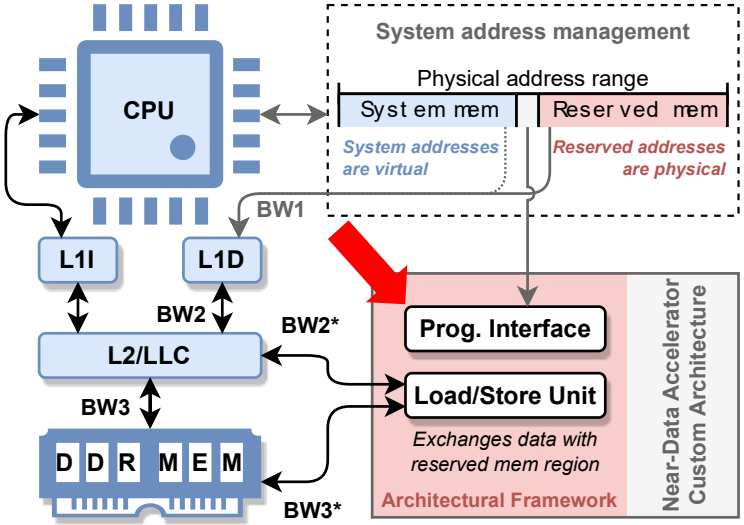
A Full System Solution for the Development of NDAccs

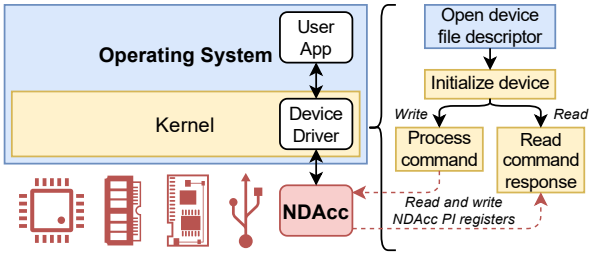
NDPmulator Overview

- From tool **structure perspective – three components:**
  - Architectural model
  - Simulation scripts
  - Application development paradigm
- From an **architectural perspective – three features:**
  - CPU-NDAcc programming interface
  - Load/store unit
  - Virtual memory translation and management
- **Two modes:**
  - Bare metal evaluation – **SE mode**
  - With Operating System (OS) – **FS mode**



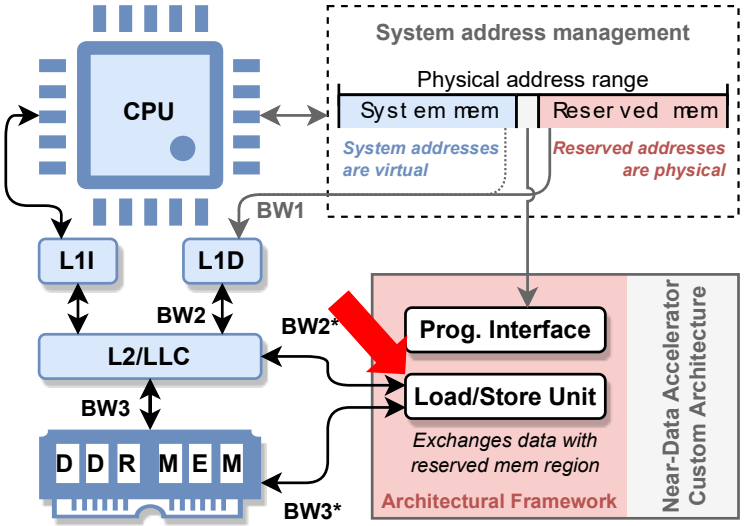
Architectural components – Programming Interface

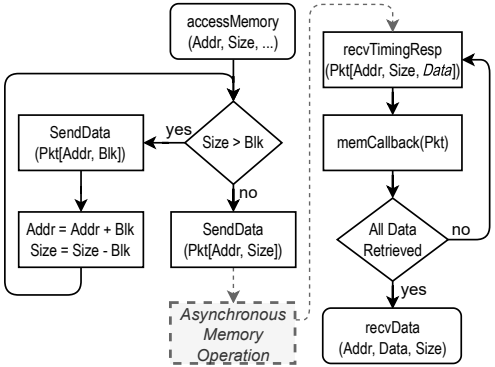




- Communication through programming interface implemented using **Memory-Mapped IO (MMIO)**
- On top of an OS → **device driver**

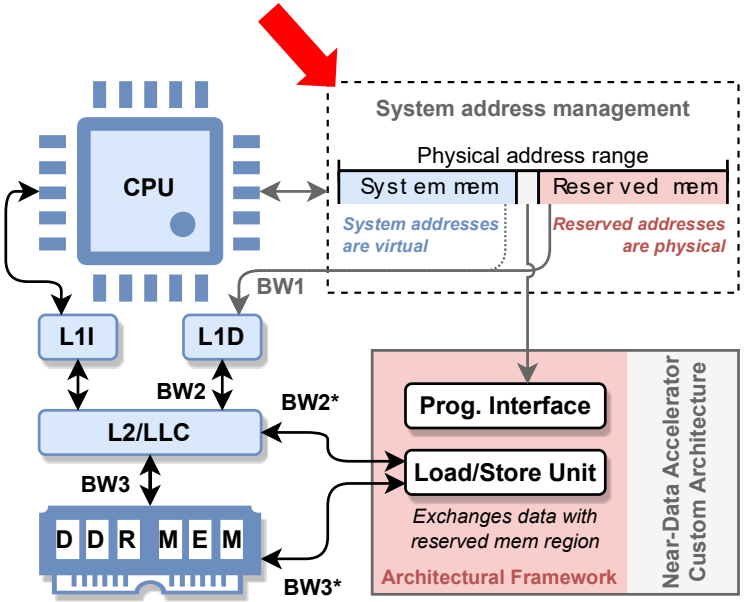
Architectural components – Load/Store Unit





- Memory accesses are **split into smaller requests**
- **Gather/store data to FIFOs**
- Coupled to **any memory**

Architectural components – Virtual Memory Management

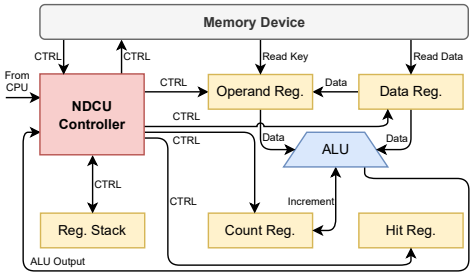


NDP-enabled memory = general-purpose memory

Direct address mapping → **no explicit translation**

Three case studies

Das P. and Kapoor H.
Towards near-data processing of compare operations in 3d-stacked memory

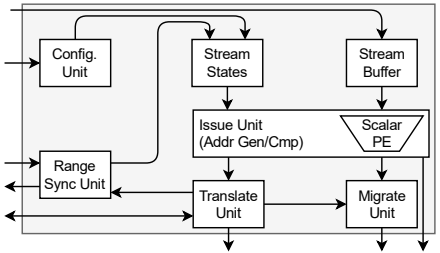


System Configuration	
CPU	Single-core, x86-64, 2 GHz
L1 i-cache	32 kB, 4-way, 64 B line
L1 d-cache	32 kB, 8-way, 64 B line
L2 cache	256 kB, 16-way, 64 B line
Main memory	DDR3 1600 MHz
Executed Benchmarks	

`hit best`
`hit avg`
`hit worst`
	Searches a DB attribute column for a specific value and returns as soon as it is found terminating the search

`count`	Counts the number of occurrences of an attribute
`max`	Determines the maximum value of an attribute

Wang Z. *et al.*
Near-stream computing: General and transparent near-cache acceleration

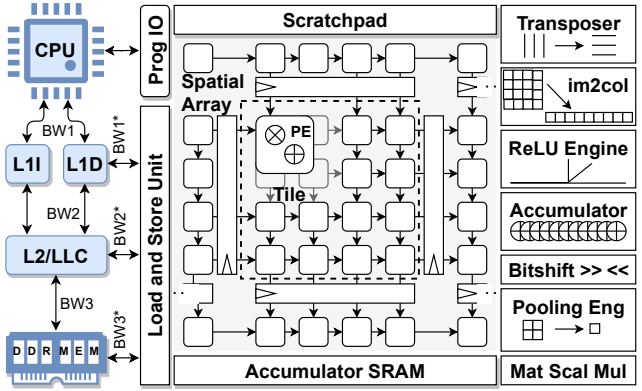


System Configuration	
CPU	Single-core, x86-64, 2 GHz
L1 i-/d-cache	32 kB, 8-way, 64 B line
L2 cache	256 kB, 16-way, 64 B line
L3 cache	256 MB, 16-way, 64 B line
Main memory	DDR4 3200 MHz
Executed Benchmarks	

`stencil1d`
`stencil2d`
`stencil3d`
	1D, 2D, and 3D stencil algorithm—used to approximate the solution of partial differential equations based on a discrete set of values

`dwt2d`	2D Discrete Wavelet Transform
`gauss elim`	Gaussian elimination algorithm
`mm inner`	Matrix Multiplication (inner product)
`mm outer`	Matrix Multiplication (outer product)
`conv2d` `conv3d`	2D and 3D convolution kernels—the most used operation in CNNs

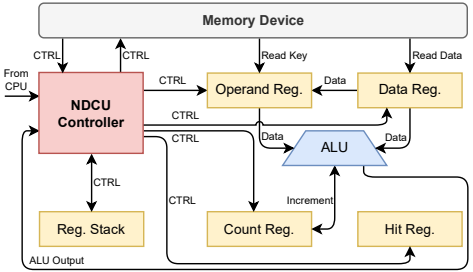
Genc H. *et al.*
Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration

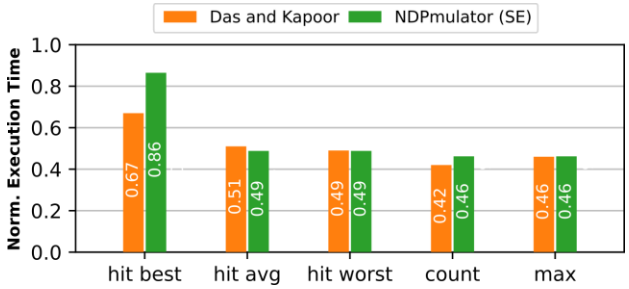


System Configuration	
CPU	Single-core, x86-64, 2 GHz
L1 i-/d-cache	32 kB, 8-way, 64 B line
L2 cache	256 kB, 8-way, 64 B line
Main memory	DDR3 1600 MHz
Executed Benchmarks	
`conv2d` `conv3d`	2D and 3D convolution kernels—the most used operation in CNNs
`maxpool`	Max pooling kernel
`relu`	Rectified Linear Unit kernel
`mm`	Matrix Multiplication
Full CNN benchmarks	12 darknet CNN models and 3 hand-tunned CNN models

Results (Das P. and Kapoor H.)

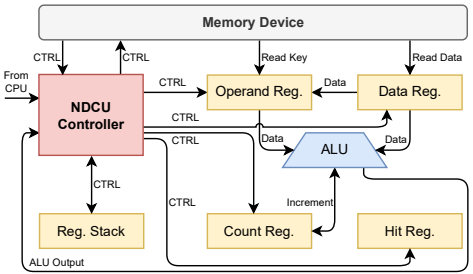
Das P. and Kapoor H.
Towards near-data processing of compare operations in 3d-stacked memory

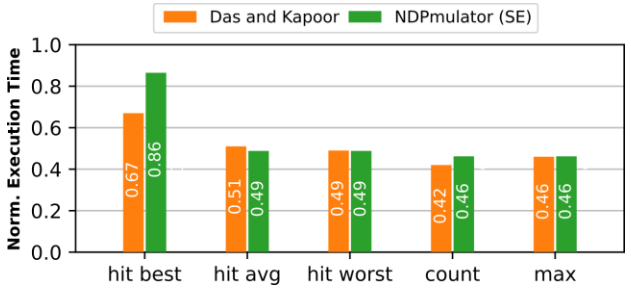


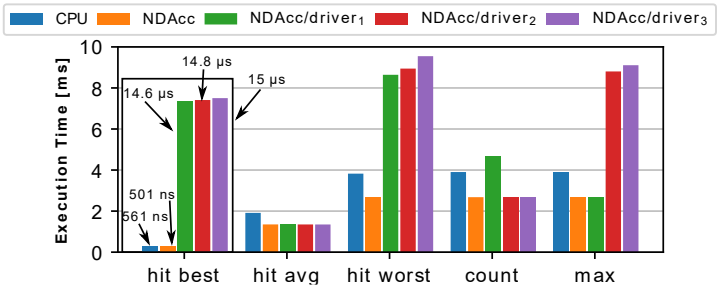


Results (Das P. and Kapoor H.)

Das P. and Kapoor H.
Towards near-data processing of compare operations in 3d-stacked memory





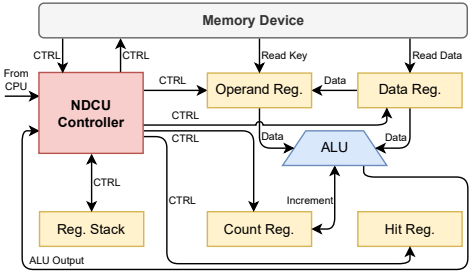


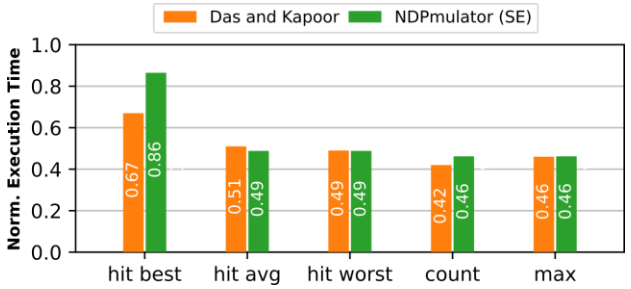
System Emulation versus Full System

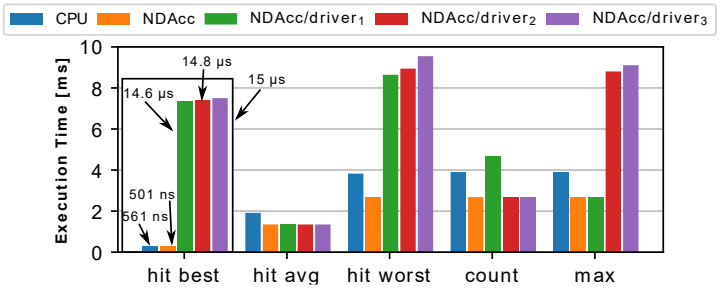
- **Overhead of up to 7 ms**
- **Dynamic overhead – OS dependent**

Results (Das P. and Kapoor H.)

Das P. and Kapoor H.
Towards near-data processing of compare operations in 3d-stacked memory

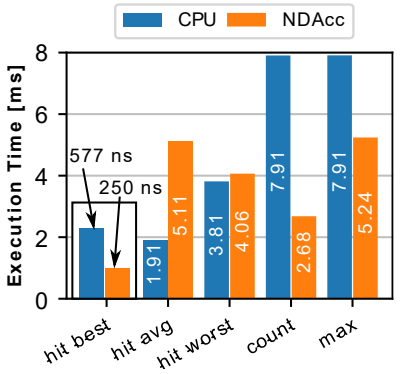






System Emulation versus Full System

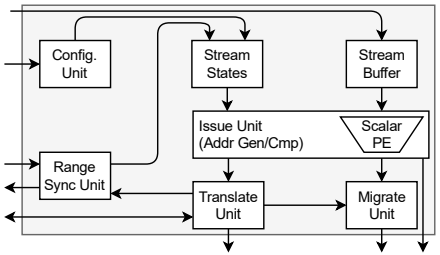
- **Overhead of up to 7 ms**
- **Dynamic overhead – OS dependent**

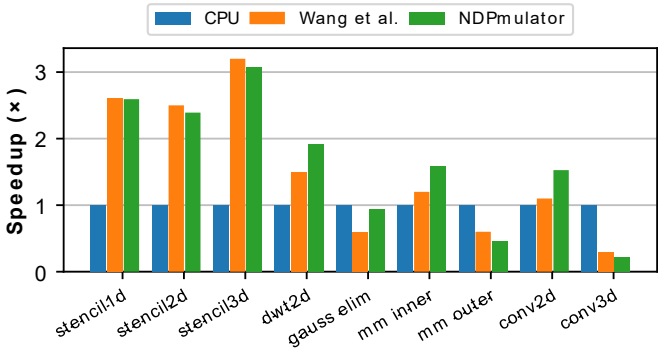


Concurrent execution with memory-intensive workload

Results (Wang Z. *et al.*)

Wang Z. *et al.*
Near-stream computing: General and transparent near-cache acceleration



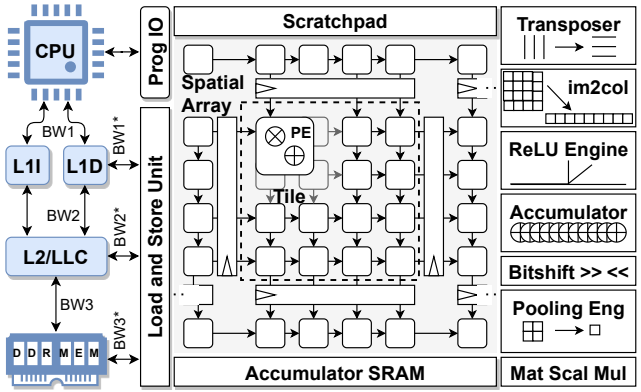


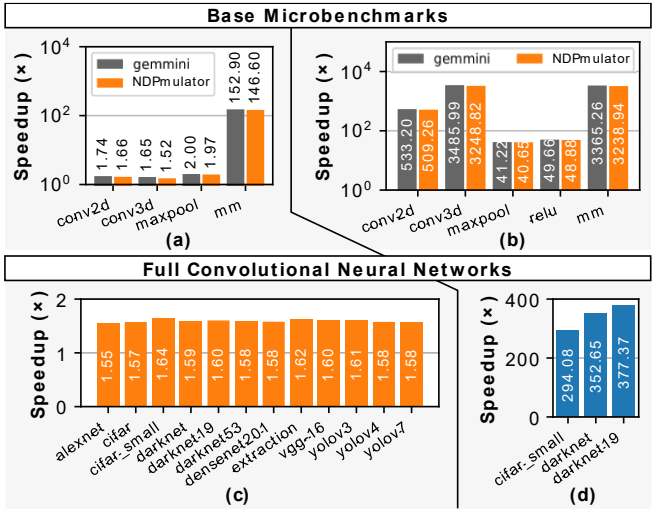
- **Error below 5 %**
- **Independent of data type:**
  - Das P. and Kapoor H. → `uint64`
  - Wang Z. et al. → `float32`

Results (Gen H. *et al.*)

Genc H. *et al.*

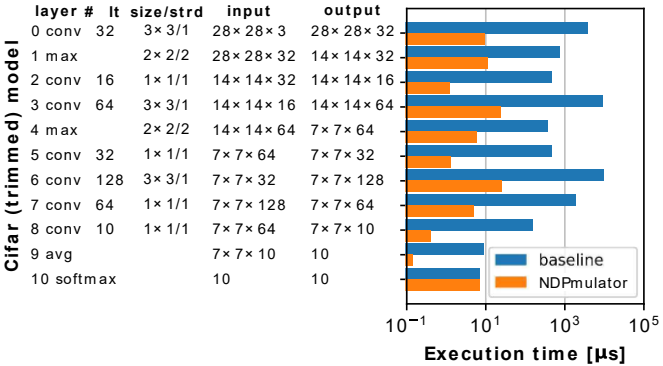
Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration





(a) and (c): data is gathered by the CPU

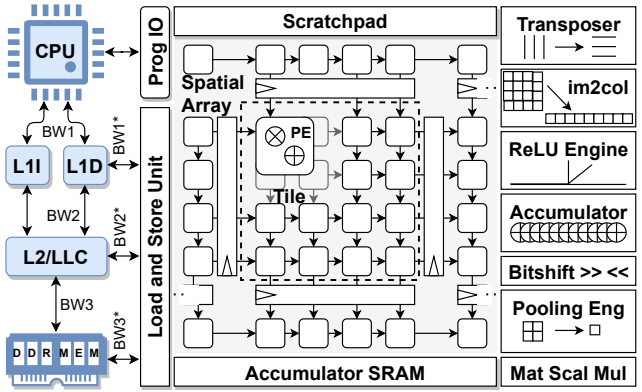
(b) and (d): data is gathered by NDAcc

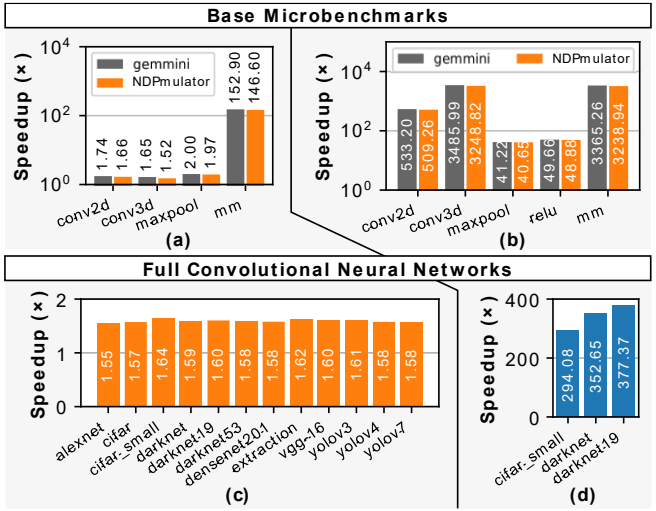


Results (Gen H. *et al.*)

Genc H. *et al.*

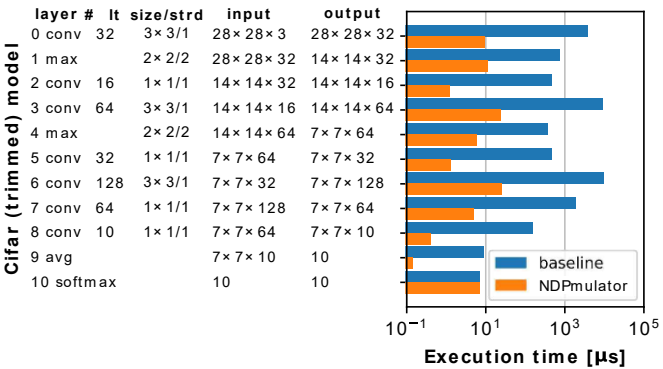
Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration

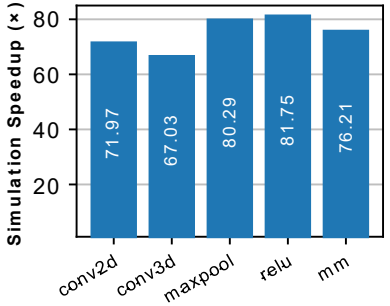




(a) and (c): data is gathered by the CPU

(b) and (d): data is gathered by NDAcc





Simulation performance improvements compared with Gemmini's Verilator-based simulation tool

Comparison with alternative solutions

	gem5-Aladdin	gem5-SALAM	NDPmulator
Development tier (analogy)	HLS	HLS	RTL description
Simulation flow	Multi-step, analytical analysis (requires the use of multiple tools and analytical models)	Multi-step	Single-step
Performance simulation	Trace-based	Clock-cycle/ interval-based	Clock-cycle/ interval-based
Energy estimation	Custom model	McPAT/Cacti support	Possible using McPAT/Cacti gem5 integration
Area estimation	Not supported	McPAT/Cacti support	Possible using McPAT/Cacti gem5 integration
Requires rebuilding gem5	No (architecture of accelerators unknown to gem5)	No (architecture of accelerators unknown to gem5)	Partial rebuild (new devices are gem5 components)
Address translation	Custom	Not supported	Standard (with particular address mapping)
Simulation mode	SE only	FS bare-metal/ limited support to FS with SO	SE/FS

TÉCNICO LISBOA

In summary, the contributions of this PhD thesis…

Conclusions

- **Cache Compute System (CCS):**
  - It was developed for data-intensive tasks like CNNs and clustering, minimizing data movement and improving efficiency.
  - **Performance Gains:** CCS achieved up to 40.6× speedup over an ARM Cortex-A53 CPU in benchmarks, benefiting from cache locality and parallelism.
  - **Architectural Flexibility:** Unlike traditional PIM, CCS supports standardized functional units and easier programming integration.
- **NDPmulator:**
  - **Toolchain Development:** A gem5-based pre-RTL toolchain was created for NDAcc design, offering full-stack simulation for realistic system evaluation.
  - **First-of-Its-Kind:** The proposed toolchain is the first cycle-accurate simulation framework specifically for NDAccs, supporting multiple ISAs and OS-level evaluation.
  - **Impact:** The study advances NDP adoption and provides essential tools for designing and validating new memory-bound accelerators.

Future research directions

- **Minimizing Offloading Overhead:** Develop lightweight protocols to reduce latency and synchronization delays when transferring instructions or computational kernels to Near-Data Accelerators (NDAccs).
- **Unified Programming Model:** Create standardized frameworks that abstract hardware complexities, facilitating seamless integration and utilization of NDAccs across various applications.
- **Extending NDPmulator for Streaming:** Enhance simulation tools like NDPmulator to support streamed memory accesses, enabling accurate modeling of streaming NDP architectures.
- **Compiler Integration:** Develop tools to translate compiler-generated code into configurations for streamed NDAccs, ensuring efficient execution of streaming operations.

Publications

International Journals

- João Vieira, Nuno Roma, Gabriel Falcão, and Pedro Tomás. **A Compute Cache System for Signal Processing Applications.** Journal of Signal Processing Systems (JSPS), 93(10):1173–1186, 2021
- João Vieira, Nuno Roma, Gabriel Falcão, and Pedro Tomás. **gem5-accel: A Pre-RTL Simulation Toolchain for Accelerator Architecture Validation.** IEEE Computer Architecture Letters (CAL), 23(1):1–4, 2024
- João Vieira, Nuno Roma, Gabriel Falcão, and Pedro Tomás. **NDPmulator: Enabling full-system simulation for near-data accelerators from caches to DRAM.** IEEE Access, 12:10349–10365, 2024

International Conferences

- João Vieira, Nuno Roma, Gabriel Falcão, and Pedro Tomás. **Processing Convolutional Neural Networks on Cache.** In International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pages 1658–1662. IEEE, 2020
- João Vieira, Nuno Roma, Gabriel Falcão, and Pedro Tomás. **gem5-ndp: Near-Data Processing Architecture Simulation From Low Level Caches to DRAM.** In International Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD), pages 197–200. IEEE, 2022

Other publications and achievements

International Journals

- João Vieira, Rui Policarpo Duarte, Horácio C. Neto: **kNN-STUFF. kNN STreaming Unit for Fpgas.** IEEE Access 7: 170864-170877 (2019)

International Conferences

- João Vieira, Edouard Giacomin, Yasir Mahmood Qureshi, Marina Zapater, Xifan Tang, Shahar Kvatinsky, David Atienza, Pierre-Emmanuel Gaillardon. **A Product Engine for Energy-Efficient Execution of Binary Neural Networks Using Resistive Memories.** VLSI-SoC 2019: 160-165
- João Vieira, Nuno Roma, Pedro Tomás, Paolo Ienne, Gabriel Falcão Paiva Fernandes. **Exploiting Compute Caches for Memory Bound Vector Operations.** SBAC-PAD 2018: 197-200

National Conferences

- João Vieira. **Playing BlokusDuo in a ZYNQ Device: A Quest for an Efficient Algorithm.** XVI Jornadas sobre Sistemas Reconfiguráveis (REC'2020), February 2020, Lisbon Portugal.

Book chapters

- João Vieira, Edouard Giacomin, Yasir Mahmood Qureshi, Marina Zapater, Xifan Tang, Shahar Kvatinsky, David Atienza, Pierre-Emmanuel Gaillardon: **Accelerating Inference on Binary Neural Networks with Digital RRAM Processing.** VLSI-SoC (Selected Papers) 2019: 257-278

Patents

- Pierre-Emmanuel Gaillardon, Edouard Giacomin, João Vieira. **Digital RRAM-Based Convolutional Block.** United States patent US11450385B2. September 20, 2022

Master thesis

- João Vieira. **Exploiting Processing Near Cache for Memory Bound Vector Operations.** Instituto Superior Técnico, 2018

Internships

- **[Jan'2018—Sep'2018]** Processor Architecture Laboratory, Swiss Institute of Technology Lausanne, Switzerland
- **[Jan'2019—Aug'2019]** Laboratory for NanoIntegrated Systems, University of Utah, United States of America

Current position

- **CPU Post-Silicon Power and Performance Senior Engineer at Qualcomm Technologies, Ireland**

Thank you!

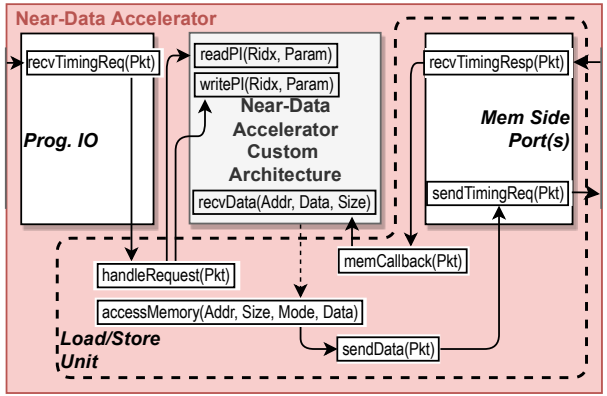

TÉCNICO
LISBOA


inesc id
lisboa


instituto de
telecomunicações

NDPmulator Structure

Architectural model



Simulation scripts

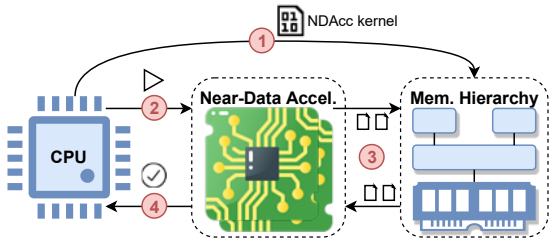
```
# Standard processing system
system.cpu = TimingSimpleCPU()
system.cpu.icache = L1Cache()                    (Part 1)
system.cpu.dcache = L1Cache()
system.l2cache = L2Cache()
system.mem_ctrl = DDR3_1600_8x8()

# NDAcc
system.ndacc = NDAcc(                            (Part 2)
    ndacc_rnge=('0x40000000', '0x40001000'))

class CustomL2XBar(L2XBar):
    def __init__(self):
        super(CustomL2XBar, self).__init__()
        width = 32  # 256-bit bus width (BW2*)

# Connect NDAcc to CPU                           (Part 3)
system.ndacc.cpu_port = system.cpu.dcache_port
system.cpu.dcache.cpu_side = system.ndacc.mem_side

# Connect NDAcc to L2
system.l2bus = CustomL2XBar()
system.ndacc.dma_port = system.l2b.slave
system.l2cache.cpu_side = system.l2bus.master

# Let NDAcc operate on upper 1GB of a 2GB RAM
system.cpu.workload[0].map(                      (Part 4)
    0x40000000,  # Host address space
    0x40000000,  # NDAcc address space
    0x40000000   # Address range)

# Start simulation
exit_event = m5.simulate()
```

SE

```
ndacc = NDAcc(ndacc_rnge = (                     (Part 1)
        '0x40000000',   # Lower PI address
        '0x40001000'))  # Upper PI address

processor = SimpleProcessor(
    cpu_type=CPUTypes.TIMING, isa=ISA.X86,
    num_cores=1)

cache_hierarchy = NDAccCompatibleCacheHierarchy(
    l1d_size = "32kB", l1i_size = "32kB",
    l2_size = "256kB", ndacc = ndacc)            (Part 2)

memory = SingleChannelDDR3_1600(size="2GB")

board = X86Board(
    clk_freq = "2GHz", processor = processor,
    cache_hierarchy = cache_hierarchy,
    memory = memory)

board.set_kernel_disk_workload(
    kernel = CustomResource("vmlinux"),
    kernel_args=[
        "earlyprintk=ttyS0", "console=ttyS0",
        "lpj=7999923", "root=/dev/sda1",         (Part 3)
        # Reserve 1GB shared memory with NDAcc
        "mem=1G", "memmap=1G@0", "memmap=1G$1G"],
    disk_image =
        CustomDiskImageResource("root_fs.img"))

simulator = Simulator(board=board)
simulator.run()
```

FS

Programming paradigm




```
int main() {
    // Assign NDAcc memory addresses
    volatile void *ndacc_control = NDACC_CRTL_ADDR;
    volatile DATA_TYPE *dataset = NDACC_DATA_ADDR;
    volatile void *kernel = NDACC_KRNL_ADDR;

    // Dataset and kernel are prepared
    initData(dataset);
    initKernel(kernel);

    // CPU launches kernel on NDAcc
    __ndaccLaunchKernel(ndacc_control);

    ... // CPU processes tasks in background

    // CPU waits for NDAcc to finish
    while (!__ndaccReady(ndacc_control));

    ... // CPU post-processes the results
```

SE

```
int main() {
    // Allocate memory in CPU/NDAcc shared region
    DATA_TYPE *dataset = ndaccAlloc(...);
    void *kernel = ndaccAlloc(...);

    // Open file descriptor (device driver)
    int *fd_ndacc =
        open("/dev/ndacc", O_RDWR | O_SYNC);

    // Dataset and kernel are prepared
    initData(dataset); initKernel(kernel);

    // CPU launches kernel on NDAcc
    __ndaccLaunchKernel(fd_ndacc, dataset, kernel);

    ... // CPU processes tasks in background

    // CPU waits for NDAcc to finish
    while (!__ndaccReady(fd_ndacc));

    ... // CPU post-processes the results
```

FS

A demonstration of easy reconfigurability