Near-Data Processing: Challenges in the conception of architectures, their integration, and Programming Paradigms

João Vieira*, Nuno Roma*, Gabriel Falcao†, Pedro Tomás*

*INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal
†Instituto de Telecomunicações, University of Coimbra, Portugal

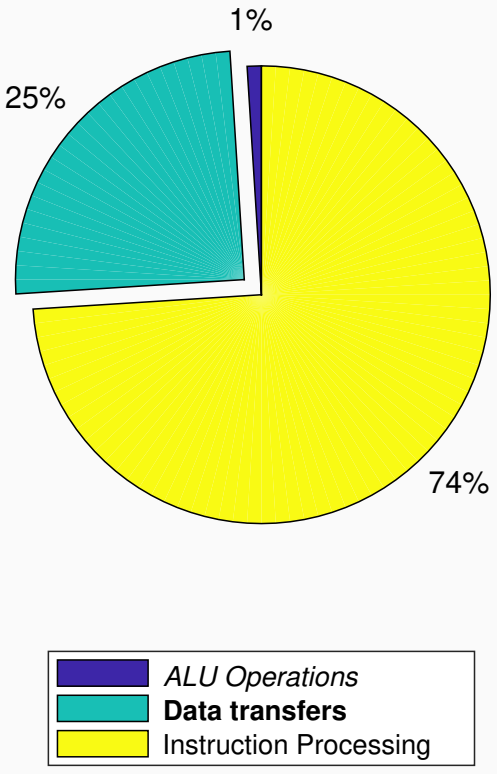
Motivation

- Memory systems **constrain performance**:
  - **Limited bandwidth**
  - **High latency** to the CPU.

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

- **25 %** is consumed **moving data across wires** *(on average)*.

CPU energy distribution



Near-Data Processing (NDP)

- **Move computation near where data is stored.**
- **Processing-in-Memory (PIM) repurposes memory technologies** to perform computation (e.g., DRAM, SRAM, RRAM).
- **Near-Memory Processing devices** are installed closer to memory devices.
  - **Higher bandwidth**
  - **Lower latency**

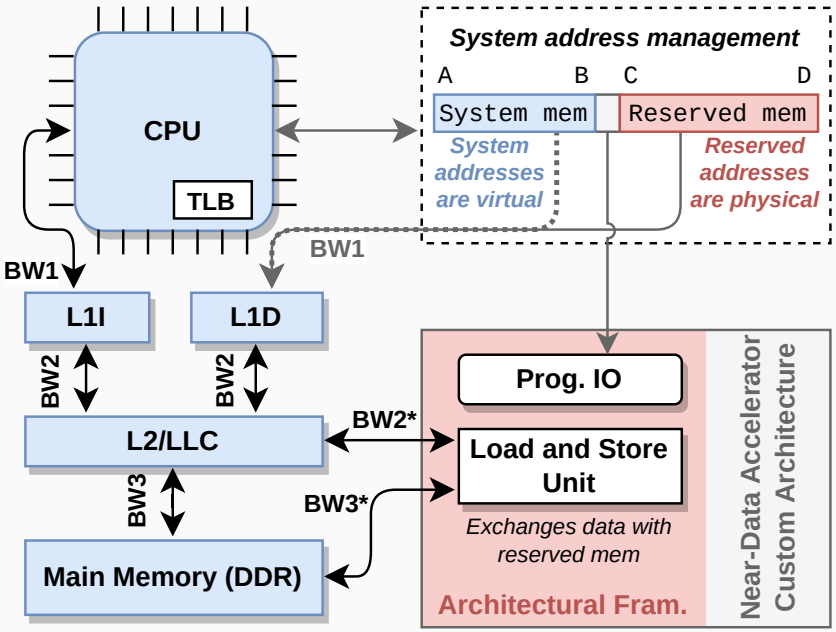
NDP Development Technologies (or scarcity of them...)

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

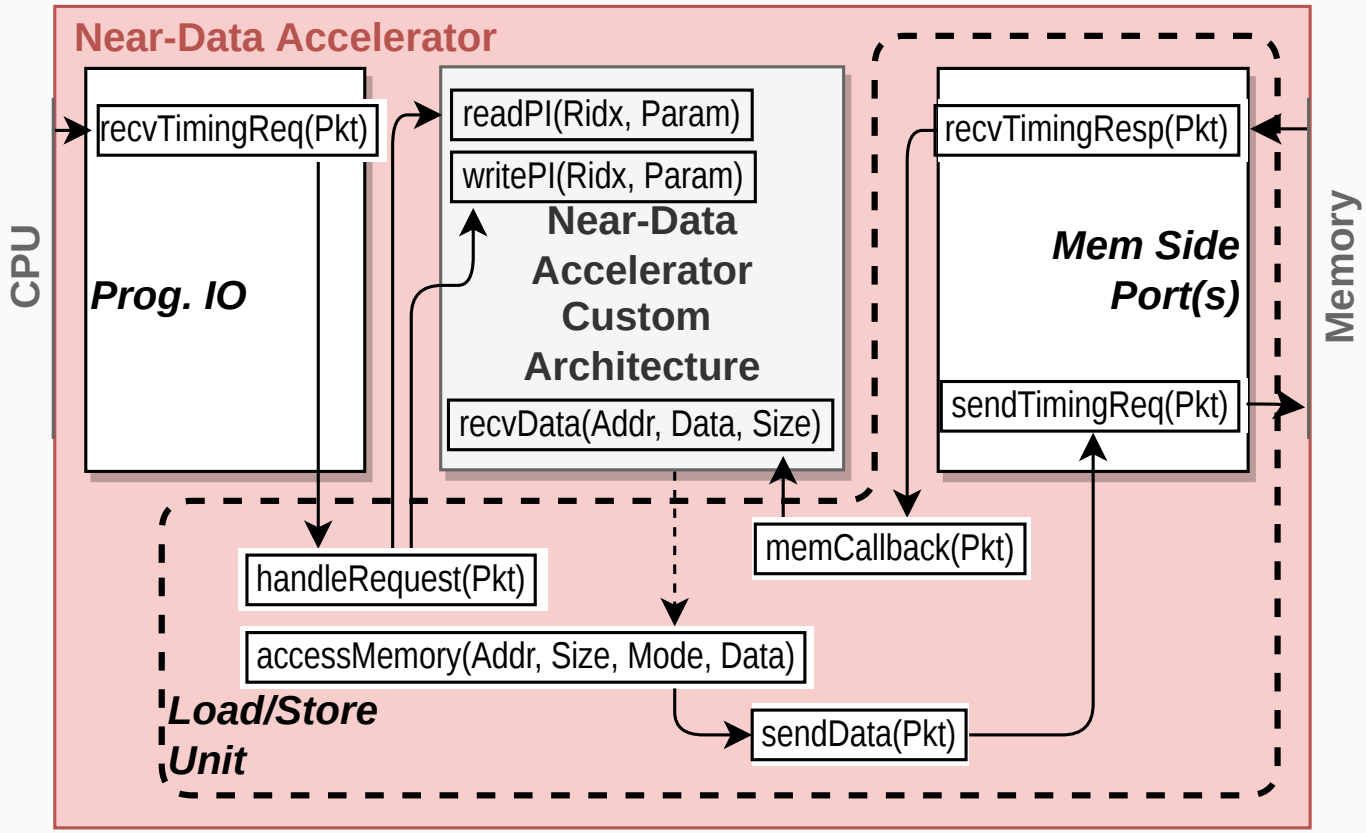
NDPmulator Overview



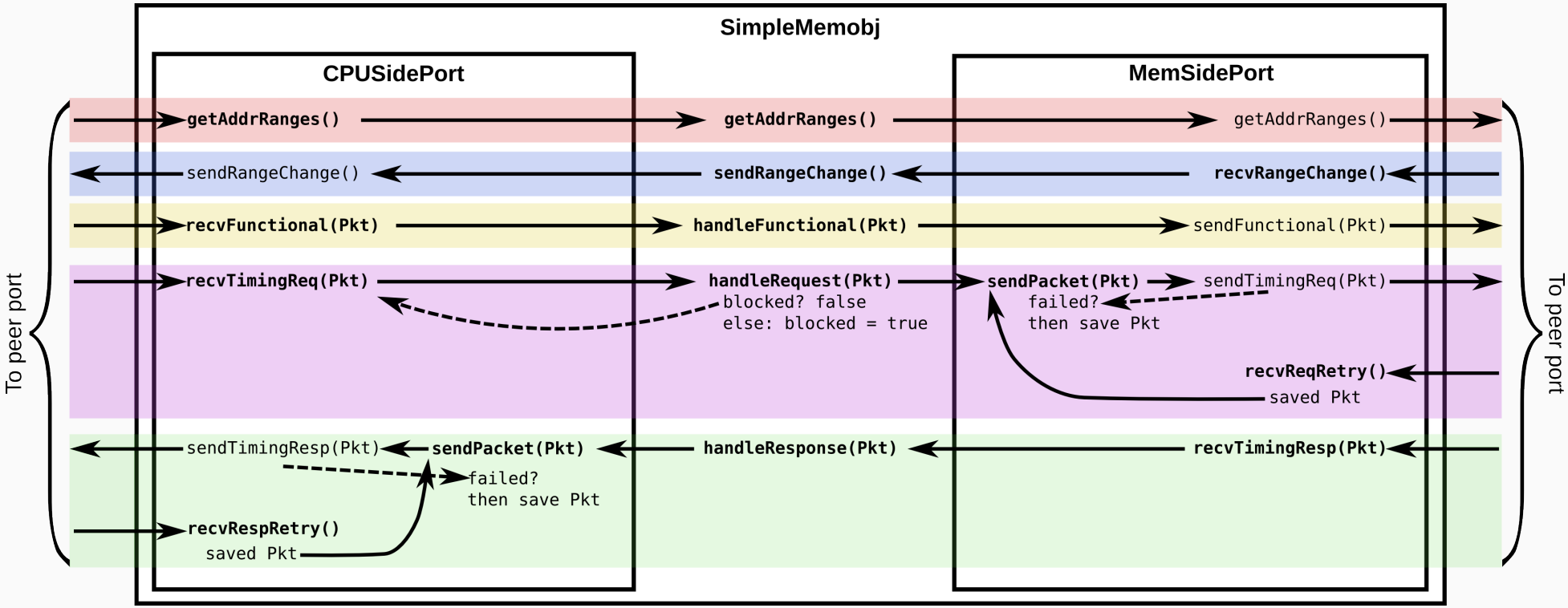
Typical NDP-enabled system

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

# NDAcc
system.ndacc = NDAcc(
    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
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(
    0x40000000,  # Host address space
    0x40000000,  # NDA address space
    0x40000000   # Address range)
```

NDPmulator Architecture



Opposed to...


SimpleMemobj
CPUSidePort
MemSidePort
To peer port
To peer port
getAddrRanges()
getAddrRanges()
getAddrRanges()
sendRangeChange()
sendRangeChange()
recvRangeChange()
recvFunctional(Pkt)
handleFunctional(Pkt)
sendFunctional(Pkt)
recvTimingReq(Pkt)
handleRequest(Pkt)
blocked? false
else: blocked = true
sendPacket(Pkt)
failed?
then save Pkt
sendTimingReq(Pkt)
recvReqRetry()
saved Pkt
sendTimingResp(Pkt)
sendPacket(Pkt)
failed?
then save Pkt
handleResponse(Pkt)
recvTimingResp(Pkt)
recvRespRetry()
saved Pkt

NDPmulator does even more

System Emulation (SE) is the simplest mode supported by NDPmulator:

- No Operating System (OS) is present;
- Almost-bare-metal environment;
- No multi-tasking or OS-specific features.

NDPmulator does even more

System Emulation (SE) is the simplest mode supported by NDPmulator:

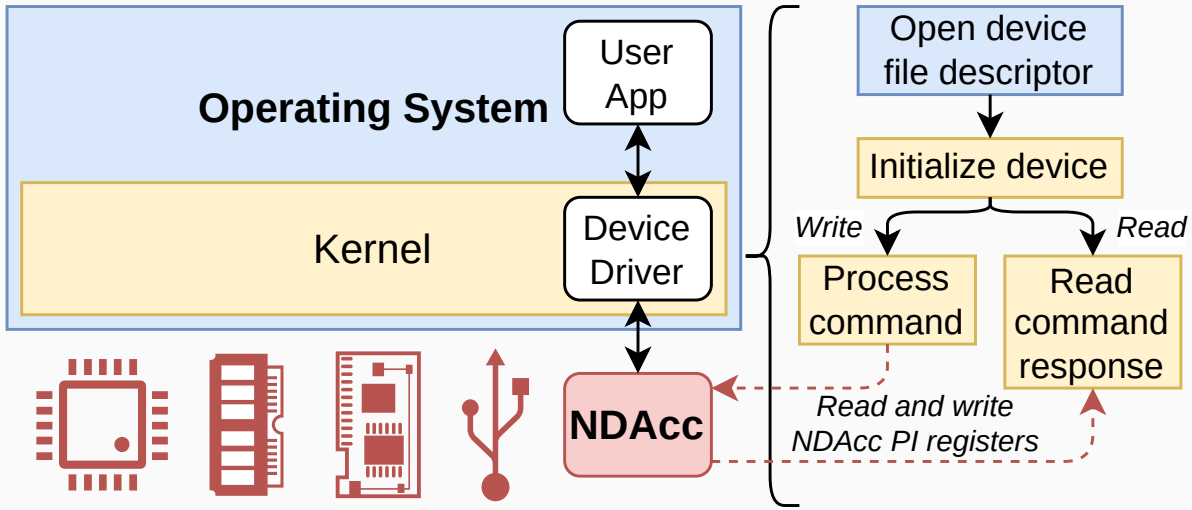
- No Operating System (OS) is present;
- Almost-bare-metal environment;
- No multi-tasking or OS-specific features.

But...

Full System Support

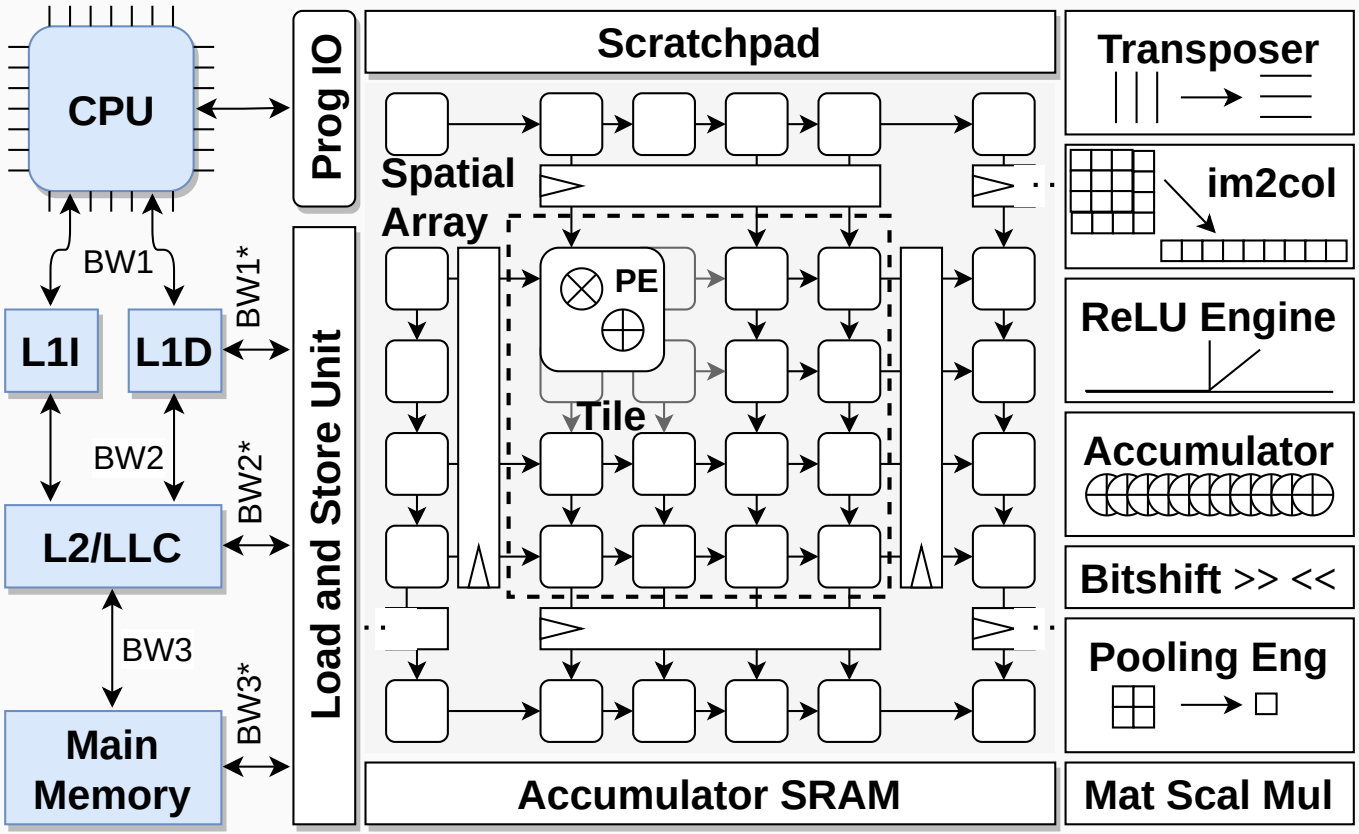
NDPmulator provides full support to **Full System (FS) simulation:**

- A conventional OS is present (Linux);
- Simulation under a realistic multi-tasking environment;
- Process isolation and management of resources is guaranteed by the OS.



Let us see some use cases

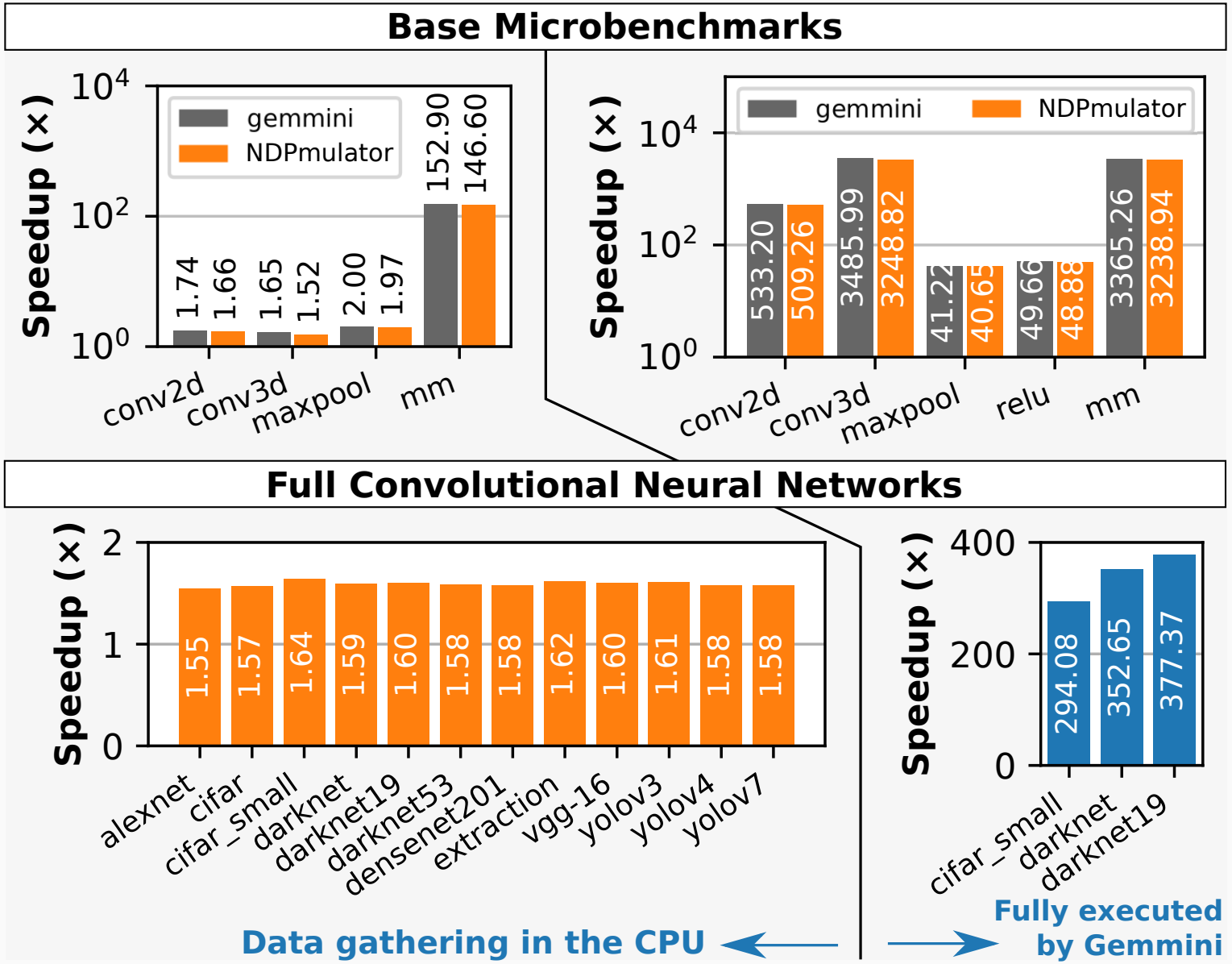
Gemmini Neural Network NDAcc



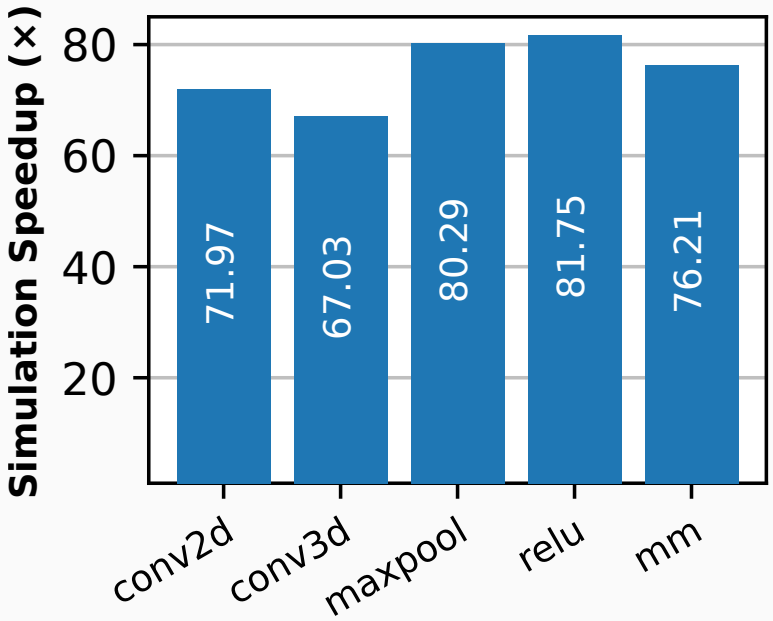
Gemmini – Experimental Setup

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

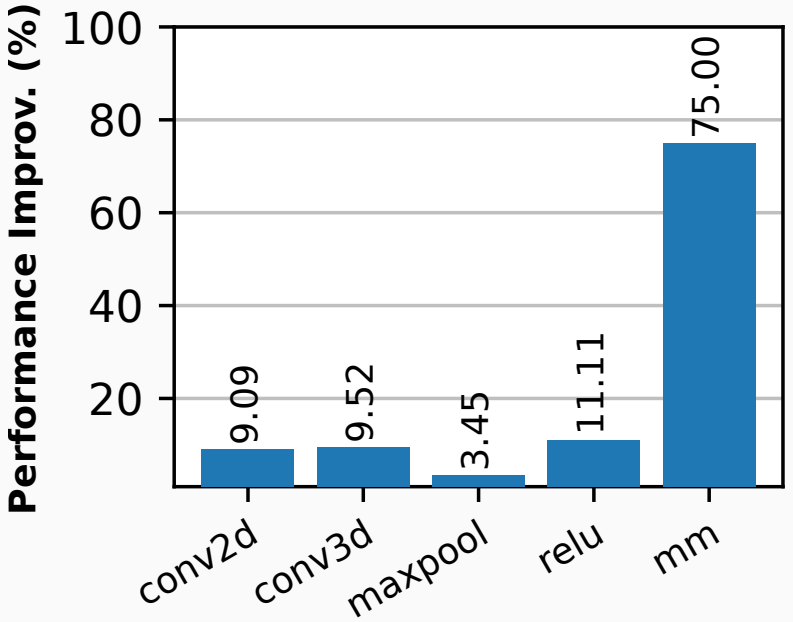
Gemmini – Experimental Results



Gemmini – Experimental Results (2)



Simulation performance benefits



Benefits of 32 B over 16 B bus

Conclusions

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

- **NDPmulator:** Full-stack framework allowing fast development and validation of NDAccs.

  The developer only needs to implement the custom NDAcc

- **Full support for NDP-enabled systems emulation**, allowing for accurate performance estimation.
- **Full System support** and compatibility across multiple ISAs.

NDPmulator is open-source

github.com/hpc-ulisboa/NDPmulator

J. Vieira, N. Roma, G. Falcao, and P. Tomás. **"gem5-ndp: Near-Data Processing Architecture Simulation From Low-Level Caches to DRAM"**. In SBAC-PAD. IEEE, 2022.

J. Vieira, N. Roma, G. Falcao, and P. Tomás. **"gem5-accel: A Pre-RTL Simulation Toolchain for Accelerator Architecture Validation"**. In CAL. IEEE, 2023.

J. Vieira, N. Roma, G. Falcao, and P. Tomás. **"NDPmulator: Enabling Full-System Simulation for Near-Data Accelerators From Caches to DRAM"**. In IEEE Access, 2024.

Q&A