Winter School on Operating Systems (WSOS 2023)

System-Level Performance Models for Disruptive Memory Technologies

Olaf Spinczyk, Daniel Friesel

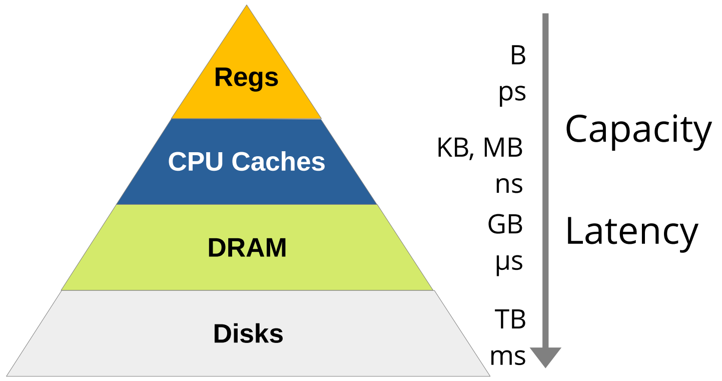
Embedded Software Systems Group

UNIVERSITÄT OSNABRÜCK

SPP 2377

Performance Models ...

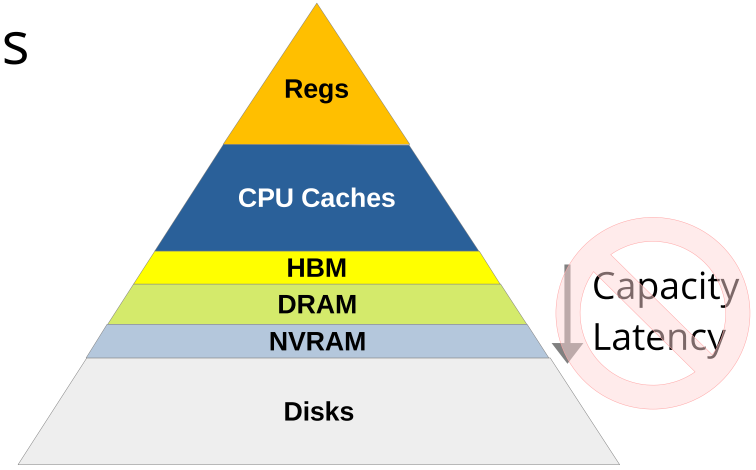
- Often implicit part of algorithms
  - Compute/data placement in systems
  - Goal: min latency, max throughput, ...
- E.g. data replacement policy in conventional memory hierarchy:
  - Goal: minimize data access latency on CPU
  - Performance model → Proxy: minimize #misses in CPU caches
  - Lower bound: OPT (Bélády / evict farthest-in-the-future)
  - Common strategy: Least Recently Used



… for Disruptive Memory Technologies

- New technologies challenge assumptions
  - High-Bandwidth Memory (**HBM**)
  - Non-Volatile RAM (**NVRAM**)
- Similar size/latency of layers
  - Cache bypass becomes feasible
  - Minimizing #misses no longer suitable [9]
  - OPT data replacement algorithm is no longer optimal [10]
  - LRU/FCFS may be insufficient → new policies needed

    →new performance models needed



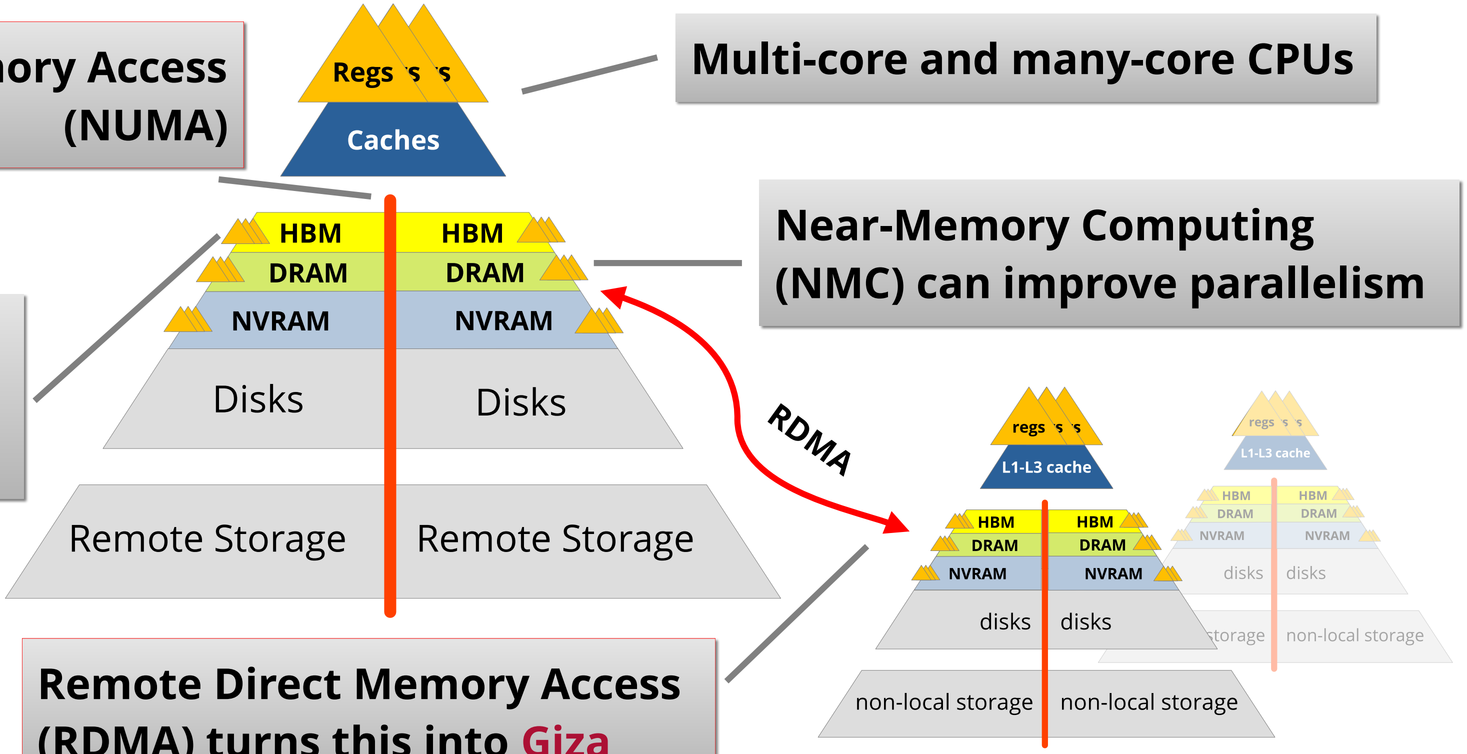
System-Level Models

Non-Uniform Memory Access (NUMA)

Multi-core and many-core CPUs

Near-Memory Computing (NMC) can improve parallelism

Novel memory technologies have different properties



Applications contend for shared resources

Remote Direct Memory Access (RDMA) turns this into **Giza**

Outline

- **Benchmarks and models for individual components**
  CPU + DRAM + disruptive memory technology
- **System- and application-level performance models**
  interaction and contention between components and applications
- **Challenges and conclusion**

Caveat Emptor:
This is not a comprehensive survey

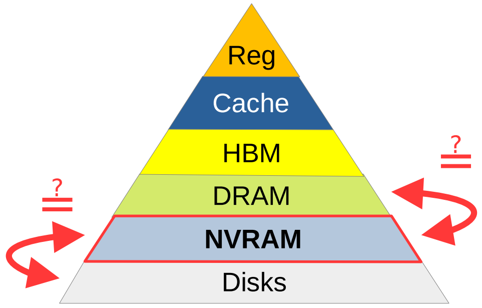
Individual Components

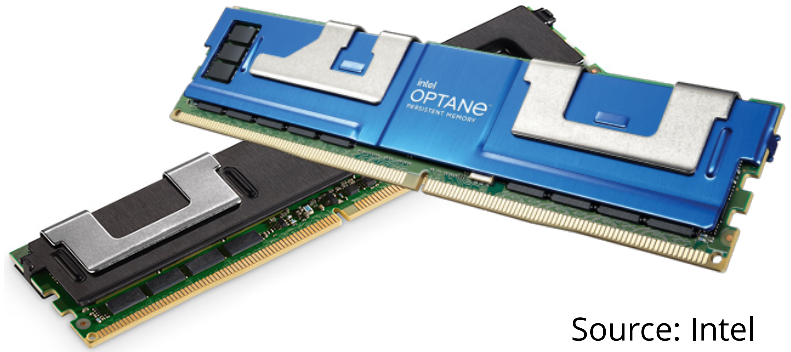
CPU and **DRAM** coupled with ...

- Non-Volatile Memory (**NVRAM**)
- High-Bandwidth Memory (**HBM**)
- Remote Direct Memory Access (**RDMA**)
- Near-Memory Computing (**NMC**)

Non-Volatile Memory (NVRAM)

- E.g. Intel Optane DC Persistent Memory (DCPMM)
  - DDR4 modules with persistent memory
  - Byte-adressable, larger and cheaper than DRAM
  - Discontinued, successor likely via PCIe
- Applications:
  - Main memory with DRAM cache
  - DRAM extension
  - Storage for databases and file systems
- Latency ≈ DRAM
- Bandwidth ≈ NVMe Disks



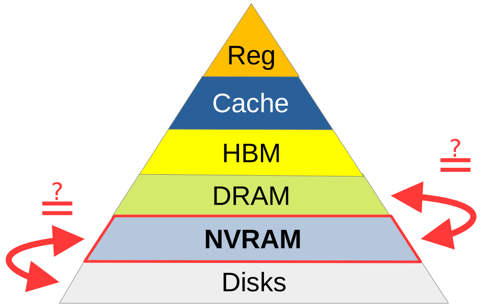


Source: Intel

Non-Volatile Memory (NVRAM)

- Breaks assumptions about memory hierarchy
  - Limited **write endurance**
  - **Asymmetric**: read latency ≠ write latency

⇒ Suitable data placement strategies?

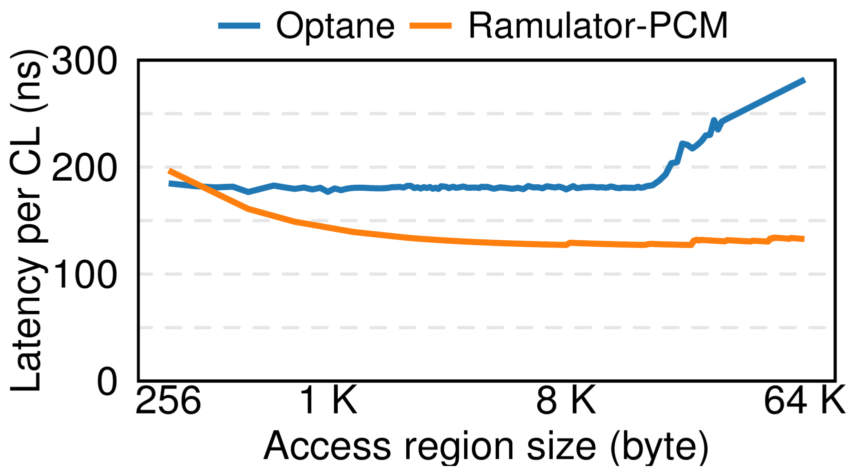


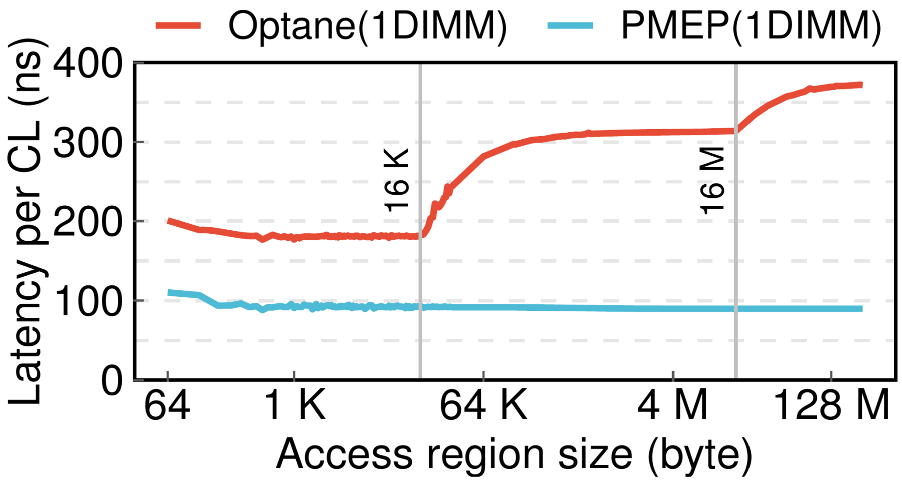
	DIMM Capacity	R/W Latency	R/W Bandwidth	Endurance
DRAM	8 .. 64 GB	70 ns	75 GB/s	∞
NVRAM	128 .. 512 GB	~200 / ~600 ns	8 / 3 GB/s	~1M writes

Source: [10]

NVRAM: More than just slower RAM [11]

- Simulators (e.g. *PMEP*, *Ramulator*) do not reflect reality
- Latency and bandwidth are not constant





NVRAM: Caches and Queues [11]

- Microbenchmarks → queues and buffers between CPU and NVRAM
- R/W Pending Queue
- Load-Store Queue
  (reordering → write combining)
- Read-Modify-Write Buf.
  (Media uses 256B blocks)
- Addr. Indirection Trans.
  (wear-leveling)

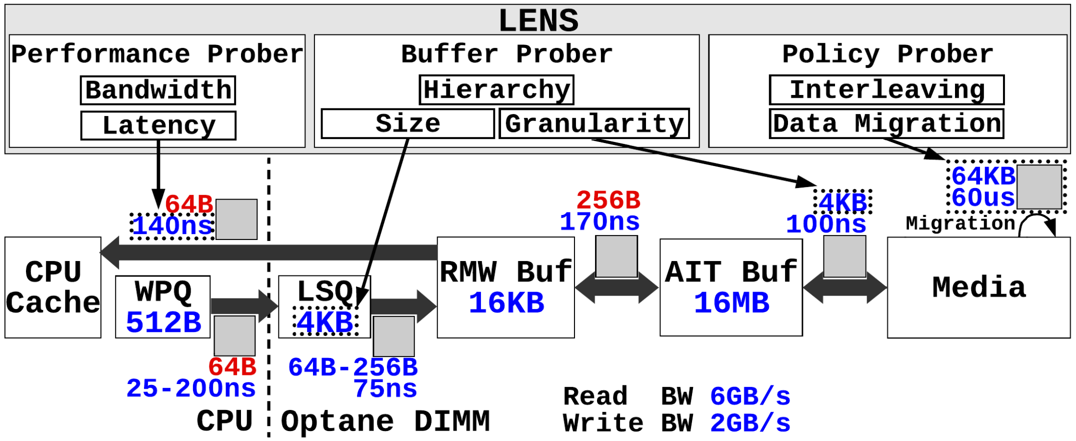
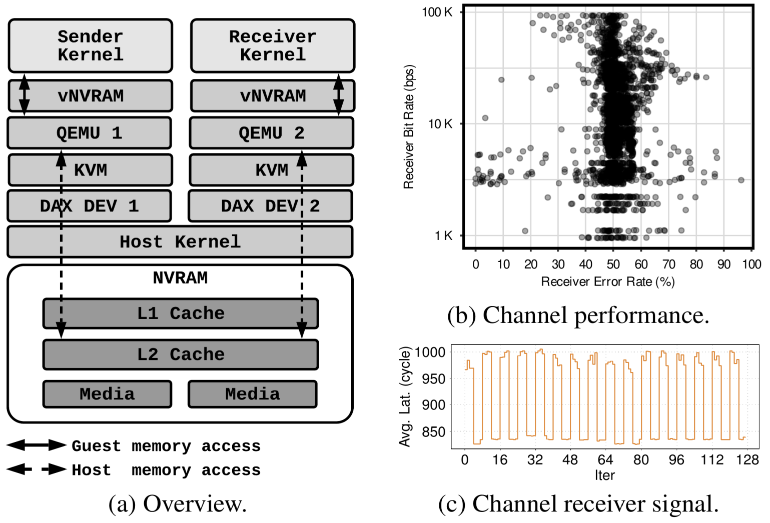


Fig. 4. LENS probers and Optane DIMM parameters. Red numbers are obtained from Intel documents; blue numbers are characterized by LENS.

NVRAM Models: Security [12]

- NVRAM is a shared resource
- Contention between applications
  → Side-channel attacks

Should performance models incorporate security information?

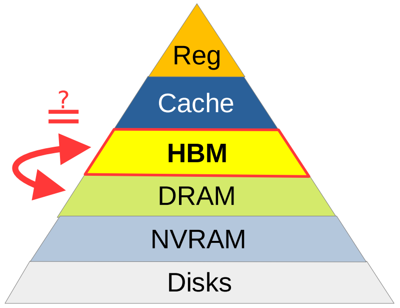


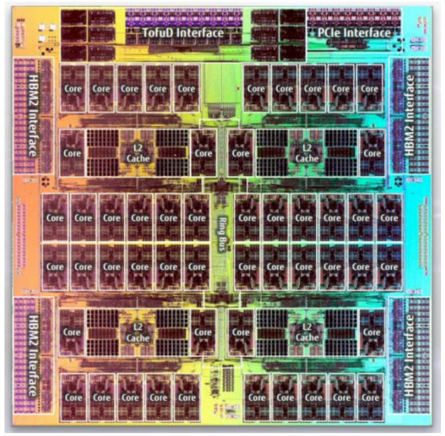
(a) Overview. (b) Channel performance. (c) Channel receiver signal.

Figure 9: Cross-VM covert channel. (c) is a receiver signal from (b) that achieves 11 kbps bandwidth with 3.5% error rate. This channel uses 14 blocks, 1 MiB stride size and 16 repeat rounds. Y-axis shows the average per-block latency.

High-Bandwidth Memory (HBM)

- 3D-stacked DRAM close to CPUs
  - E.g. Fujitsu A64FX, FPGAs, Intel Knights Landing (KNL)
  - Apple M1/M2: on-chip DDR5
- Applications:
  - HBM as DRAM cache
  - HBM as separate memory
- Latency ≈ DRAM
- Bandwidth > DRAM





Source: Fujitsu (A64FX)

HBM as Last-Level Cache [9]

- No hardware cache control
- Two sources of contention
  - Data placement in HBM
    → Replacement algorithm
  - Data transfers to/from DRAM
    → Scheduling algorithm
- **Channel model**

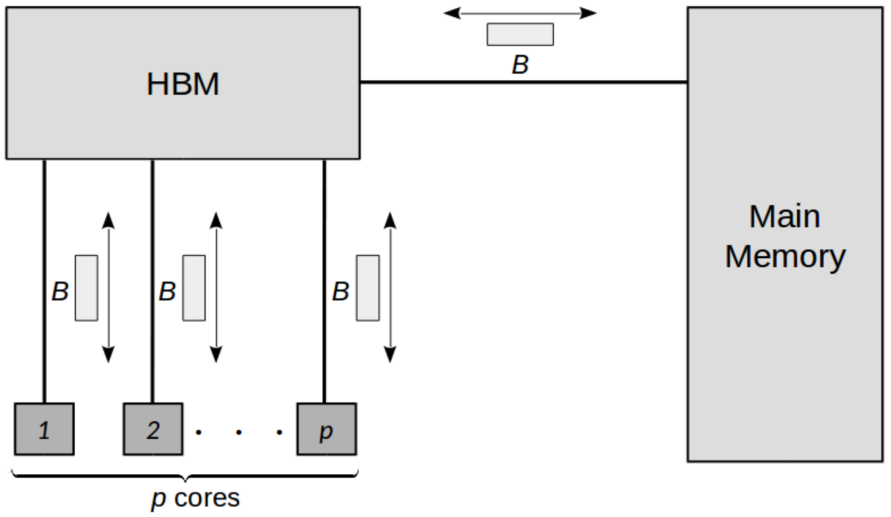


Figure 1: The HBM model with p cores and two levels of memory.

- KNL: LRU placement still useful; FCFS DRAM access provably bad

HBM Benchmarks [13]

- KNL benchmarks: max. sustained bandwidth
- Same architecture, different results
- ≠ Channel model (single-threaded view, no contention / parallelism)

Should performance models take a single- or a multi-threaded view? What about contention?

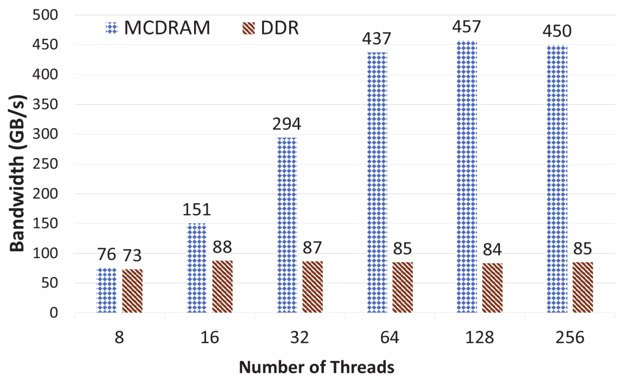


Fig. 1: Stream Triad Benchmark results for MCDRAM and DDR with $KMP\_AFFINITY = Scatter$.

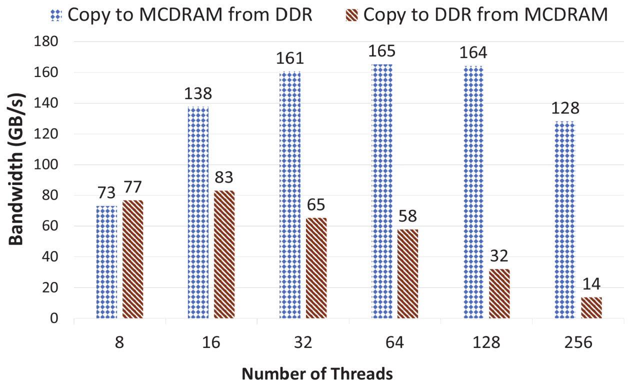
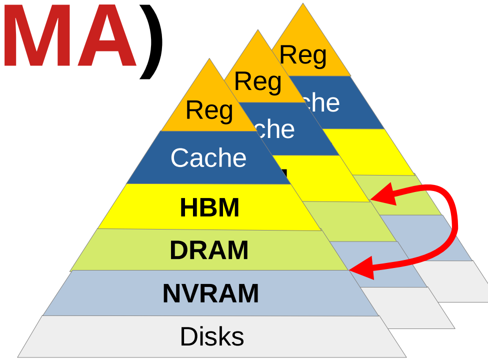
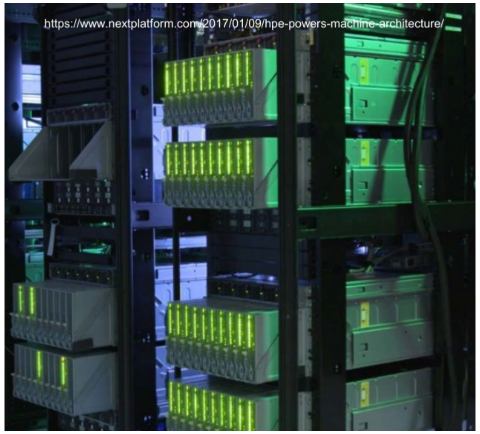


Fig. 2: Copy operation to and from the two memory types.

Remote Direct Memory Access (RDMA)



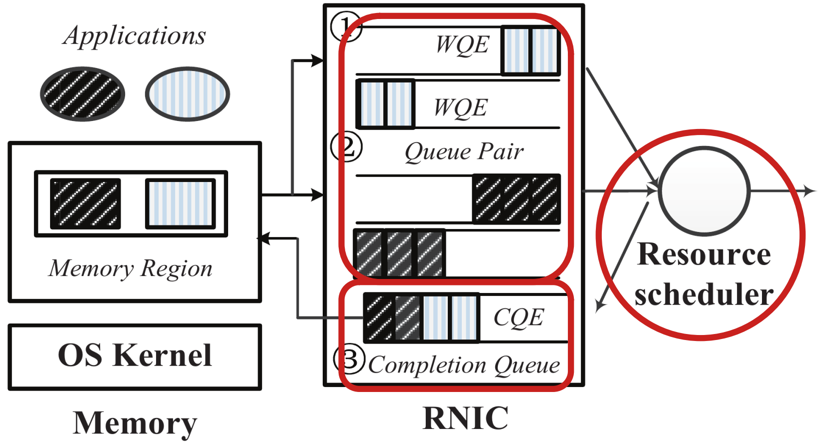
- High-Bandwidth Interconnect (≥100 Gbit/s)
  - e.g. InfiniBand
  - Connected via PCIe, DMA capable
  - Supports DRAM and NVRAM, at least
- High-capacity, high-latency
- Interconnect ≈ CPU/Socket fabric
- E.g. HPE: 160 TB of fabric-attached shared memory


https://www.nextplatform.com/2017/01/09/hpe-powers-machine-architecture/

Source: Kimberley Keeton, NVMW'19 Keynote

RDMA Network Interface [14]

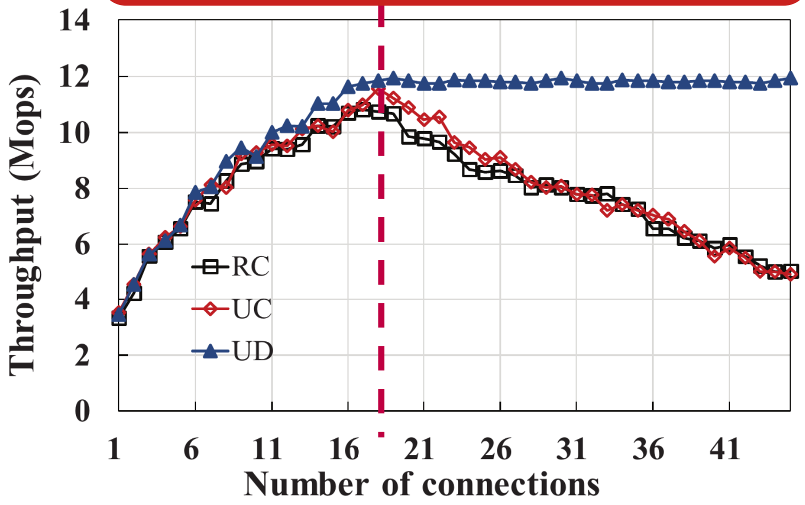
- RDMA NIC handles transfers itself
  - Bypasses host CPU / OS kernel
  - Writes → L3 Cache (DDIO) or RAM
- One R/W queue pair per application
  - Work Queue Entries (WQEs) ≙ data transfers
- One completion queue for entire NIC
- Built-in scheduler manages queues and transfers



RDMA: Connection Cache [15]

- RDMA NIC associates transfers (queues) with network connections
  - Built-in metadata cache
  - Too many connections (RC, UC) → cache thrashing
- ≠ Conventional network approach: Resource Utilization Maximization
- Proposal: Connection-aware workload placement

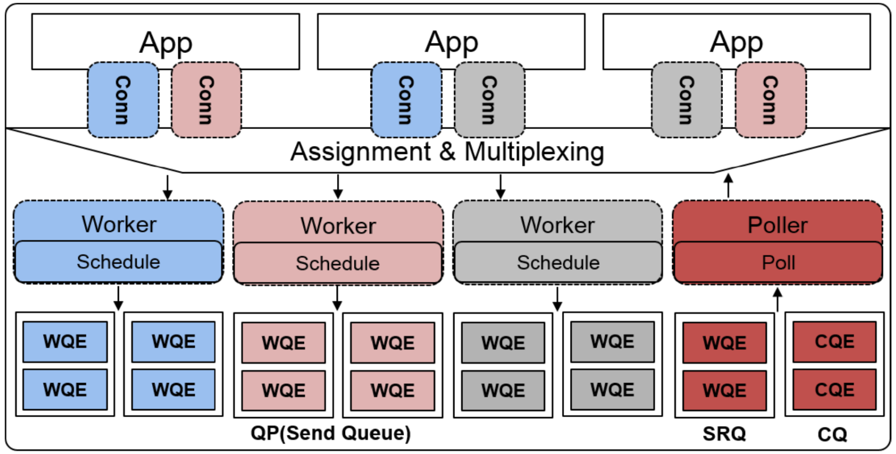
Performance model: NIC with max #connections



(c) Average throughput of RC and UC connections drop drastically when the number of concurrent connections grows larger than 20.

RDMA: Scheduler [15]

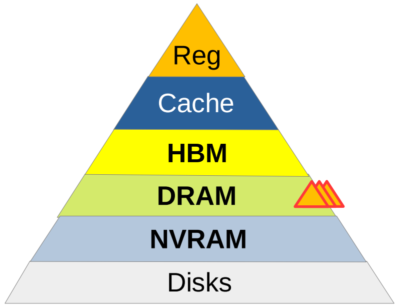
- RDMA transfers run to completion (no preemption)
- Large transfers can cause delays
- Proposed solution: Virtual RDMA

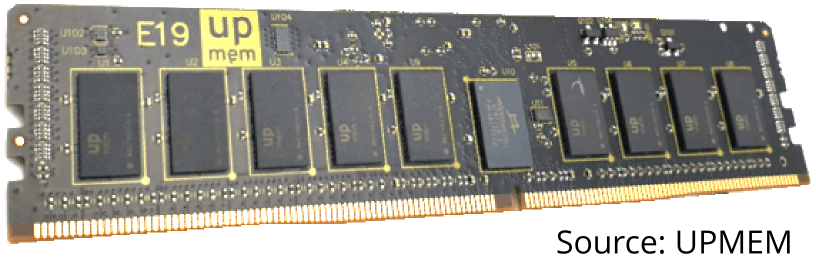


How to incorporate performance models of RDMA targets?

Near-Memory Computing (NMC)

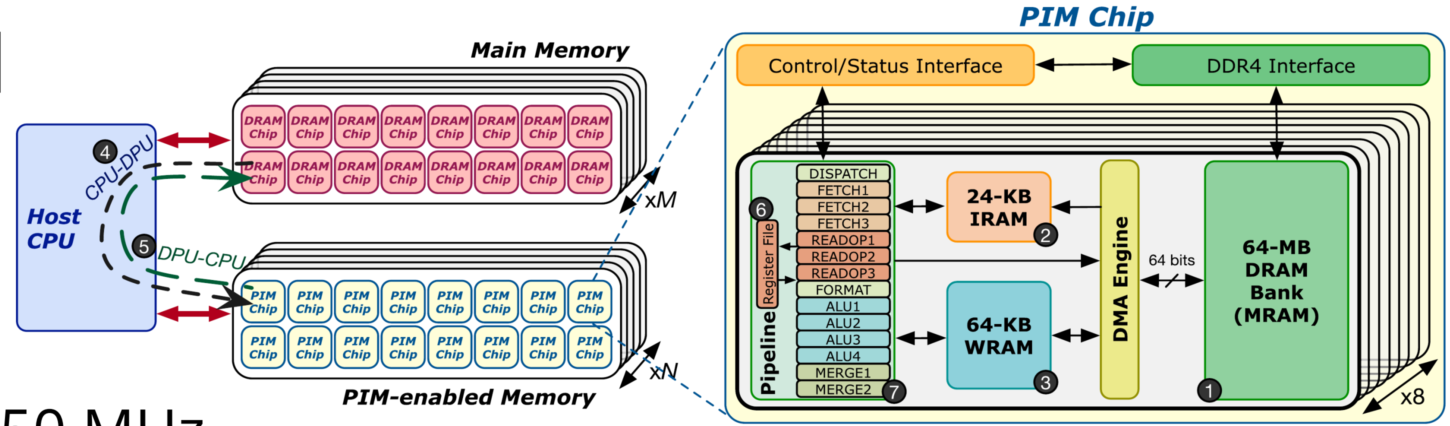
- Memory-bound workloads limited by RAM latency / bandwidth → add CPUs to memory
- Past: experiments with FPGAs and logic circuits
- Recently: UPMEM **PIM** (“Processing in Memory”)
  - DDR4 memory with on-chip 32-bit CPUs
  - Limited hardware support (needs BIOS patches)
  - Limited power supply → low clock rate
  - OS API: Regular DRAM ≠ PIM DRAM





Source: UPMEM

UPMEM PIM



Source: UPMEM

- 8 GB DDR4
- Custom API
- 128× CPU @ ~350 MHz
  - a.ka. DPU / „Data Processing Unit"
  - Up to 24 Threads (*Tasklets*) each
  - Manual cache control: IRAM ② / WRAM ③
  - 11 sequential pipeline levels ⑦
  - Threads can communicate / synchronize
- No shared memory / IPC between DPUs

How to find suitable workloads?

PrIM Microbenchmarks [16]

- Comprehensive performance evaluation
  - Arithmetic throughput, op. intensity
  - WRAM („Cache“) and DRAM transfers
  - Stride and operand sizes
- Models: Linear regression + prose
  - Hints for developers, e.g.
  - ≥11 Tasklets for arithmetic saturation
  - Best for integer addition/subtraction
  - Suitable for memory-bound tasks

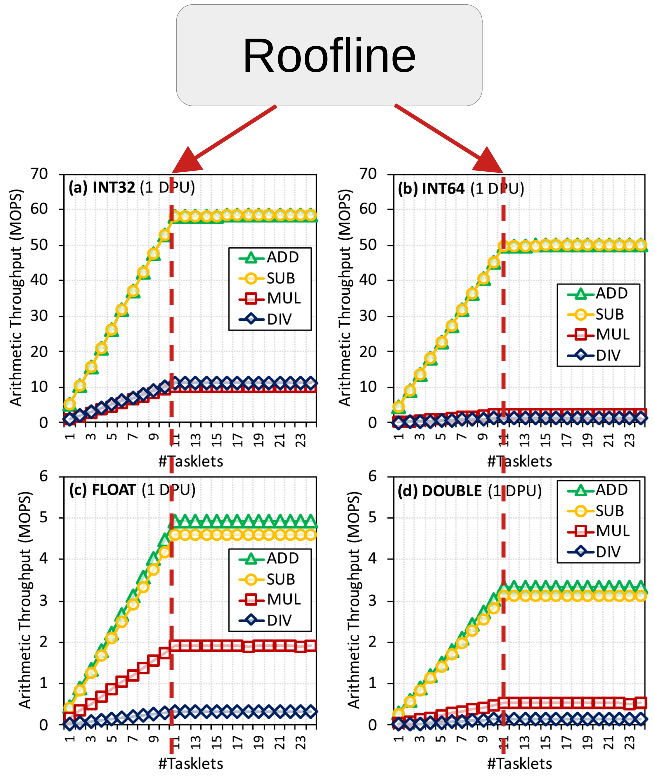


FIGURE 4. Throughput of arithmetic operations (ADD, SUB, MUL, DIV) on one DPU for four different data types: (a) INT32, (b) INT64, (c) FLOAT, (d) DOUBLE.

NMC: Predicting offloading suitability [17]

- Given an application: will it benefit from NMC?

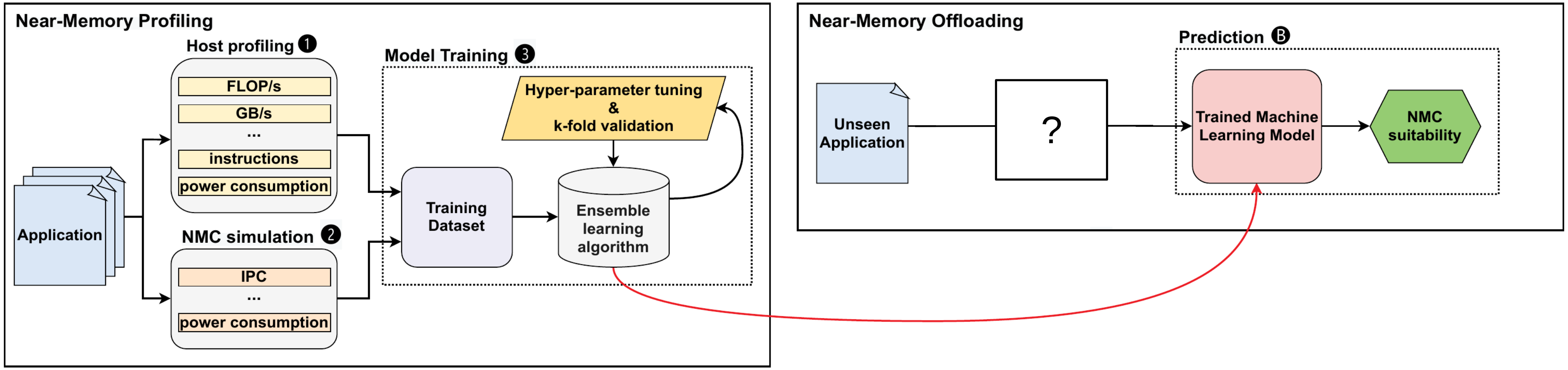


Fig. 2: *Near-Memory Computing Profiling and Offloading (NMPO) overview.*

Outline

- **Benchmarks and models for individual components**
  CPU + DRAM + disruptive memory technology
- **System- and application-level performance models**
  interaction and contention between components and applications
- **Challenges and conclusion**

Motivation: Systems with Combined DMTs

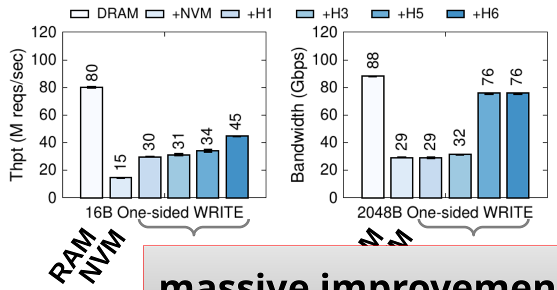
- The future will bring the new technologies all at once

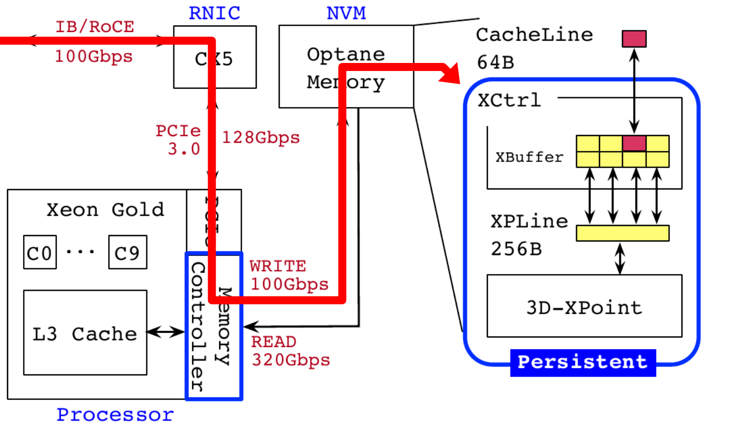
  … and system software must handle them *efficiently*!

- Example: RDMA + NVM interactions [5]

RDMA WRITE to NVM

Can we reach a full 100Gbps? No





ware components of a node with NVM in an RDMA-

H1. Avoid cross-socket NVM accesses
H3. Disable DDIO
H5. Use XPLine granularity for writes
H6. Use PCIe DW granularity (64B) for small writes

massive improvements by taking the architecture into account

Wanted: A Full-System and Application Model

same level of abstraction

- Features of **full-system model**
  - Description of **hardware components**
    - Detailed sub-models, especially for **DMTs**
    - Memory: capacity, alignment requirements, ...
    - Compute units: processing power, cache sizes / strategy, ...
  - Topology of **interconnects**
    - Bandwidth limits, latencies, arbitration mechanisms, ...
- Features of **application model**
  - Structure, thread interaction, resource demands, ...
- Purpose
  - **Offline** planning/optimization → compiler, simulator, design space exploration
  - **Online** placement/scheduling → operating system, database management, language runtime

Offline Models: Literature Review [6]

- **Feature rich,**
  but *no DMT* support
- Used for detailed simulation
- Low accuracy
- Extreme simulation runtimes

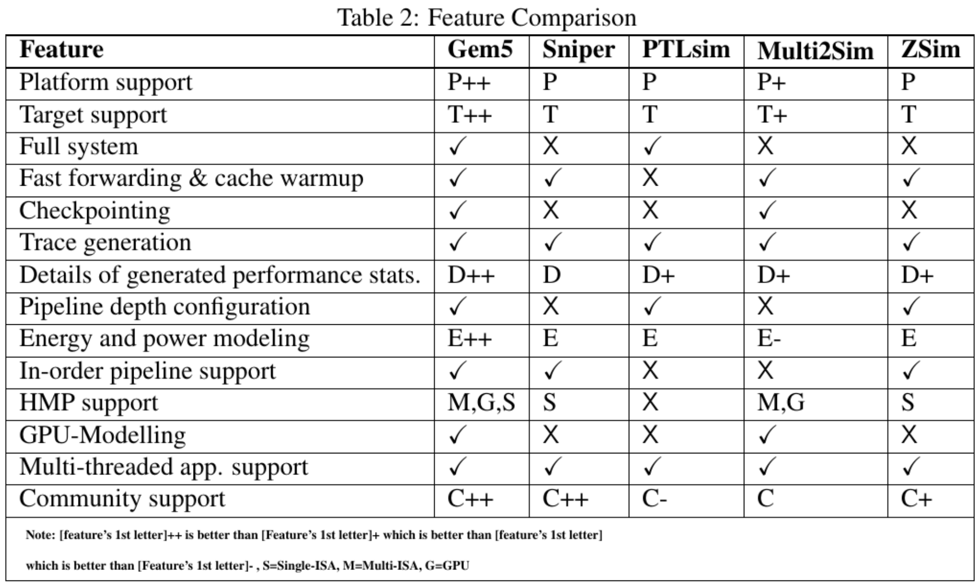


Table 2: Feature Comparison

Feature	Gem5	Sniper	PTLsim	Multi2Sim	ZSim
Platform support	P++	P	P	P+	P
Target support	T++	T	T	T+	T
Full system	✓	X	✓	X	X
Fast forwarding & cache warmup	✓	✓	X	✓	✓
Checkpointing	✓	X	X	✓	X
Trace generation	✓	✓	✓	✓	✓
Details of generated performance stats.	D++	D	D+	D+	D+
Pipeline depth configuration	✓	X	✓	X	✓
Energy and power modeling	E++	E	E	E-	E
In-order pipeline support	✓	✓	X	X	✓
HMP support	M,G,S	S	X	M,G	S
GPU-Modelling	✓	X	X	✓	X
Multi-threaded app. support	✓	✓	✓	✓	✓
Community support	C++	C++	C-	C	C+

Note: [feature's 1st letter]++ is better than [Feature's 1st letter]+ which is better than [feature's 1st letter] which is better than [Feature's 1st letter]- , S=Single-ISA, M=Multi-ISA, G=GPU

application „model“: x86 machine code

Offline Models: Literature Review [6]

- Feature rich,
  but *no DMT* support
- **Used for detailed simulation** $\longrightarrow$
- Low accuracy
- Extreme simulation runtimes

application „model“: x86 machine code

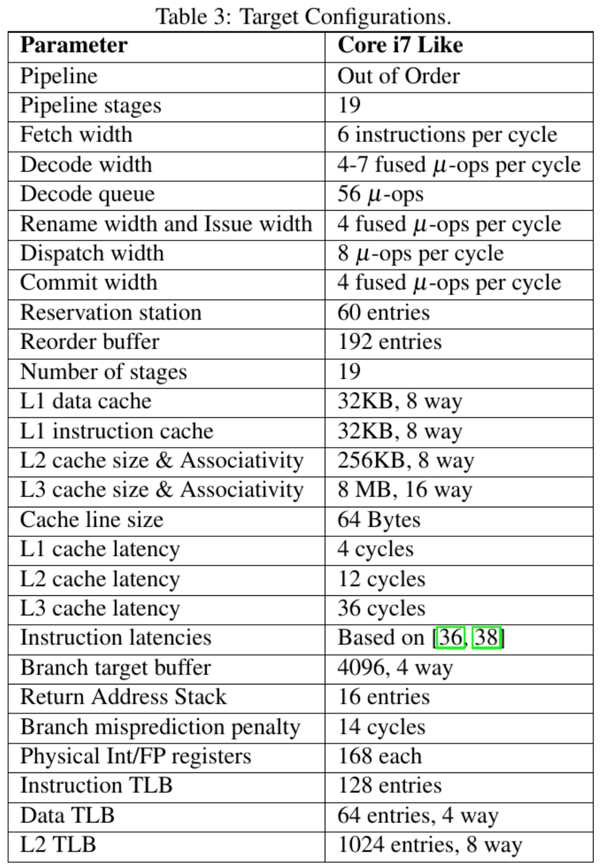


Table 3: Target Configurations.

Parameter	Core i7 Like
Pipeline	Out of Order
Pipeline stages	19
Fetch width	6 instructions per cycle
Decode width	4-7 fused μ-ops per cycle
Decode queue	56 μ-ops
Rename width and Issue width	4 fused μ-ops per cycle
Dispatch width	8 μ-ops per cycle
Commit width	4 fused μ-ops per cycle
Reservation station	60 entries
Reorder buffer	192 entries
Number of stages	19
L1 data cache	32KB, 8 way
L1 instruction cache	32KB, 8 way
L2 cache size & Associativity	256KB, 8 way
L3 cache size & Associativity	8 MB, 16 way
Cache line size	64 Bytes
L1 cache latency	4 cycles
L2 cache latency	12 cycles
L3 cache latency	36 cycles
Instruction latencies	Based on [36, 38]
Branch target buffer	4096, 4 way
Return Address Stack	16 entries
Branch misprediction penalty	14 cycles
Physical Int/FP registers	168 each
Instruction TLB	128 entries
Data TLB	64 entries, 4 way
L2 TLB	1024 entries, 8 way

Offline Models: Literature Review [6]

- Feature rich,
  but *no DMT* support
- Used for detailed simulation
- **Low accuracy**
- Extreme simulation runtimes

application „model“: x86 machine code

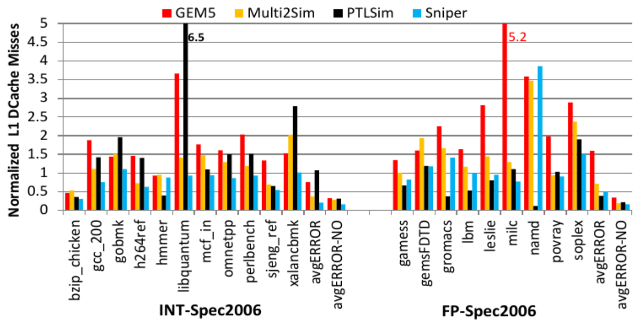


Figure 2: Normalized L1 DCache Misses

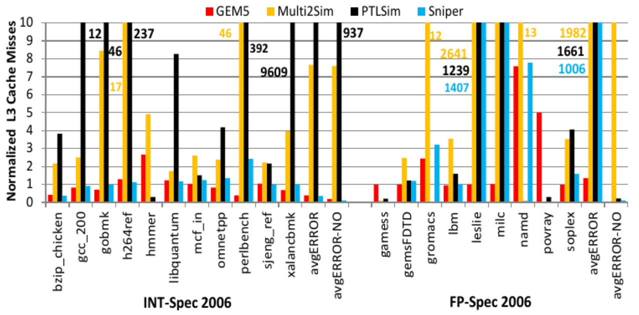


Figure 3: Normalized L3 Cache Misses

Offline Models: Literature Review [6]

- Feature rich,
  but *no DMT* support
- Used for detailed simulation
- Low accuracy
- **Extreme simulation runtimes**

- Hardly affordable during the development cycle
- Change granularity? Use measurements?

application „model“: x86 machine code

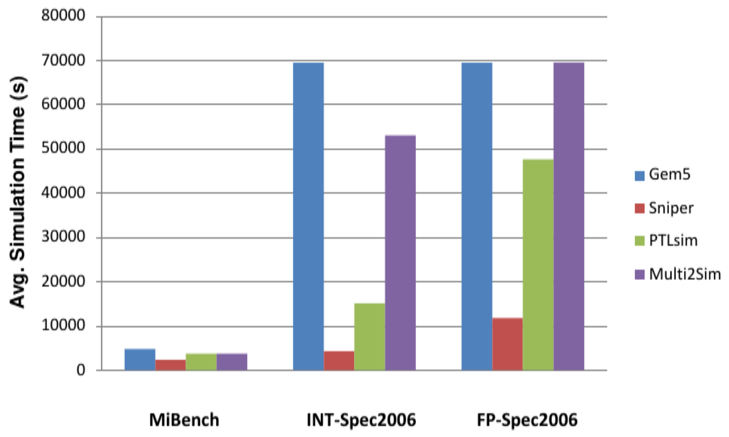


Figure 7: Average simulation time

Online Models: Literature Review

- Typical use case: NUMA-aware application behavior
  - No. of threads, thread affinity, etc.

- **Examples**
  - **hwloc** [1] – the model behind `lstopo`
  - **mctop** [3] – measurement-based topology analysis
  - **SKB** [4] – Barrelfish’s versatile „System Knowledge Base“
  - **Pythia** [8] – Smart co-location with machine learning

hwloc [1] — The Model Behind `lstopo`

- Early model from 2010
- Describes **NUMA topologies**
  - Based on data from Linux sysfs
- Entities
  - „*Node, Socket, Cache, Core,* and more“
- Attributes
  - „such as the *cache type* and *size*, or the *socket number*“

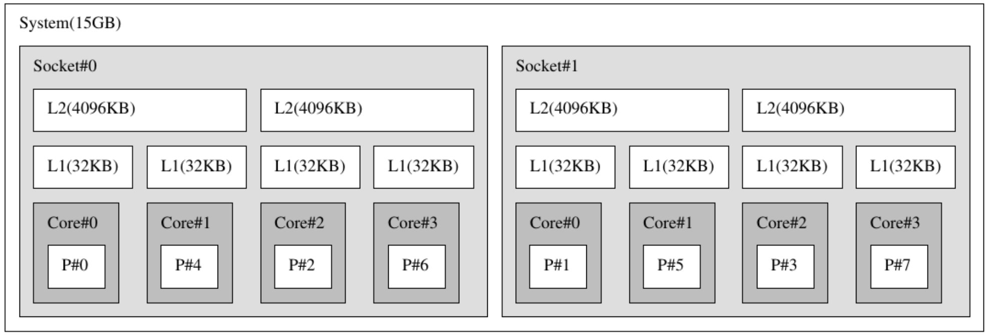


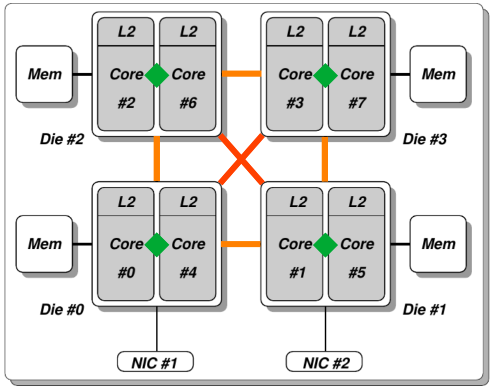
Fig. 3. Graphical output of the `lstopo` tool describing the topology of the host from Figure 1.

- **no** meta model,
  no accelerators,
  no performance metrics,
  no DMTs
- Nevertheless, ***useful***!

hwloc — Useful Despite Its Deficiencies

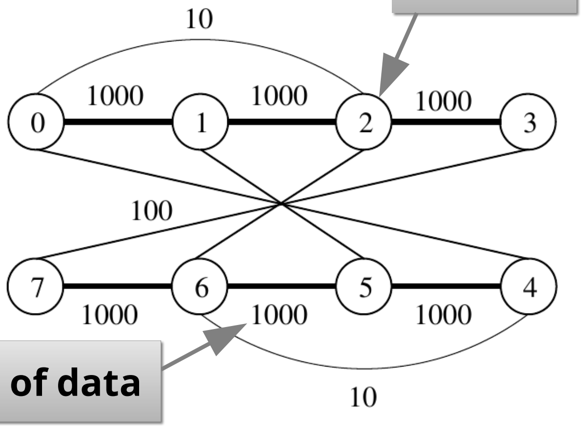
- Simple performance model [2]

bandwidth (kind of)
same die: 1000
1 hop: 100
2 hops: 10



- Application model

process



amount of data

Fig. 8. NAS LU (class B, 8 processes) communication pattern representation. The coefficients on the edges represent the magnitudes in the amounts of data exchanged between processes.

- Locality pays off

TABLE III
EXECUTION TIMES (IN SECONDS) FOR NAS CG KERNEL (64 PROCESSES).

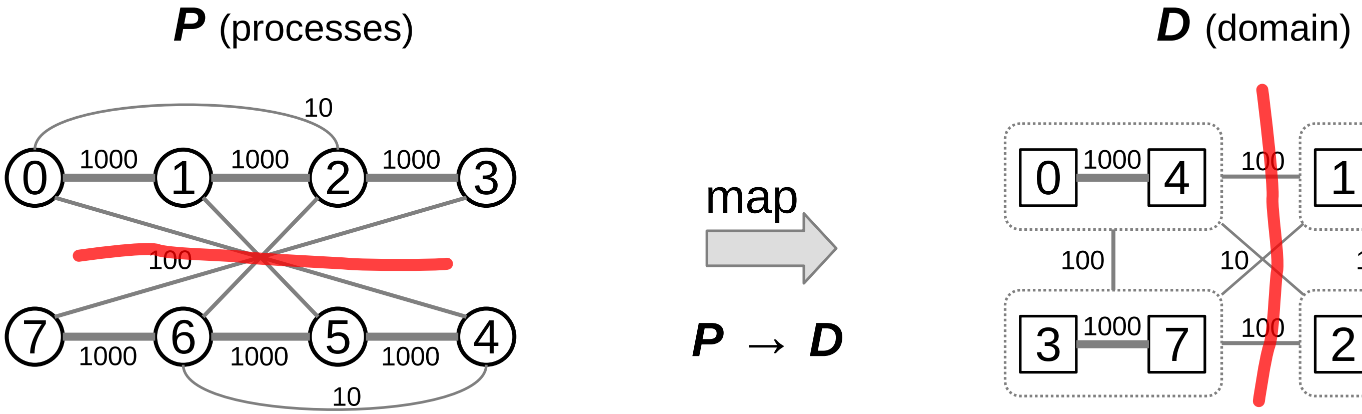
	Round-Robin	Placed	Improvement
CG (Class C)	21.16	15.6	**26%**
CG (Class D)	920.6	848.4	**8%**

static mapping calculated with *SCOTCH* library

hwloc — Static Process Mapping

- SCOTCH library [7]: **dual recursive bi-partitioning** heuristic

application processes

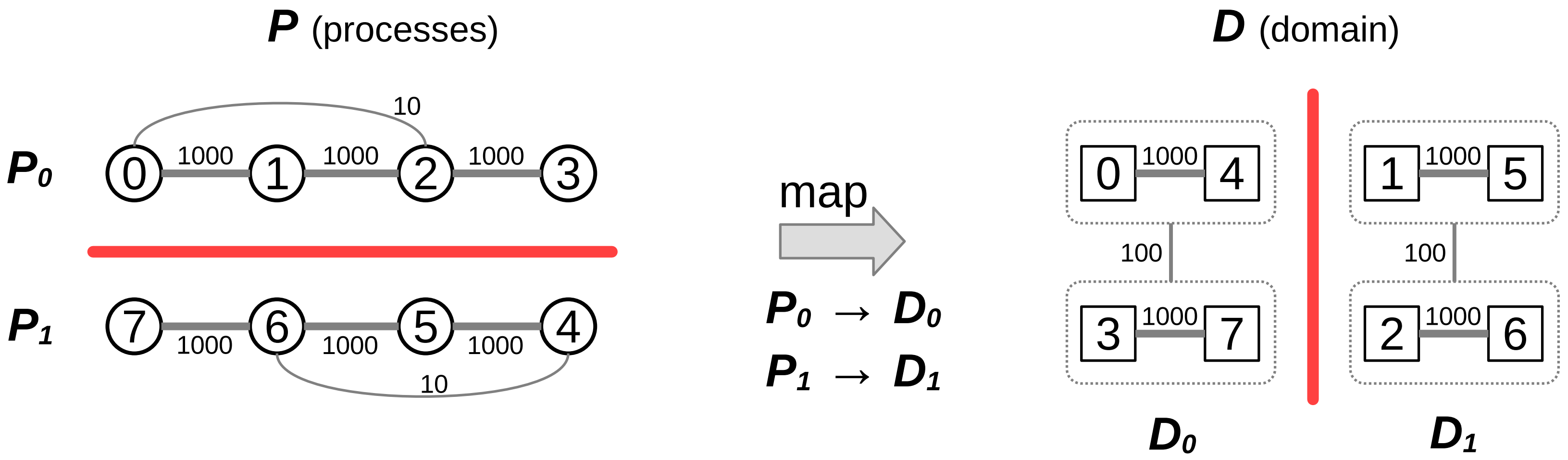


CPU cores of target machine

1. Find a **cut** in ***D*** so that both partitions are **well-balanced** and have minimal cross-partition **communication costs**.
2. Find a **cut** in ***P*** so that both partitions are **well-balanced** and have minimal cross-partition **data exchange**.

hwloc — Static Process Mapping

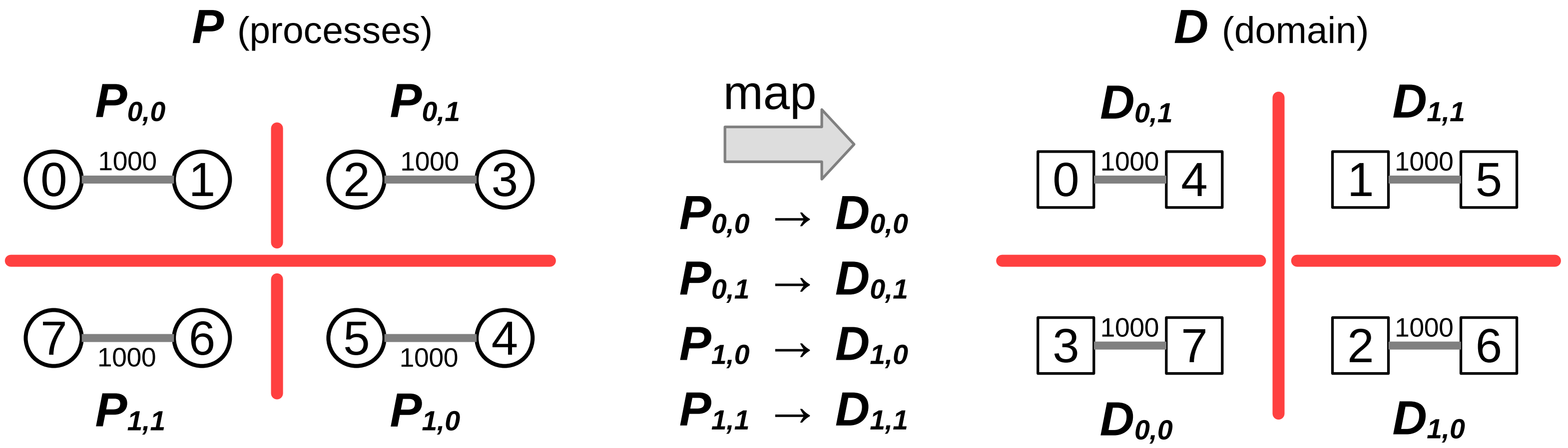
- SCOTCH library [7]: **dual recursive bi-partitioning** heuristic



3. Map the corresponding partitions **recursively** until the target domain consists of only one CPU core.

hwloc — Static Process Mapping

- Scotch library [7]: **dual recursive bi-partitioning** heuristic

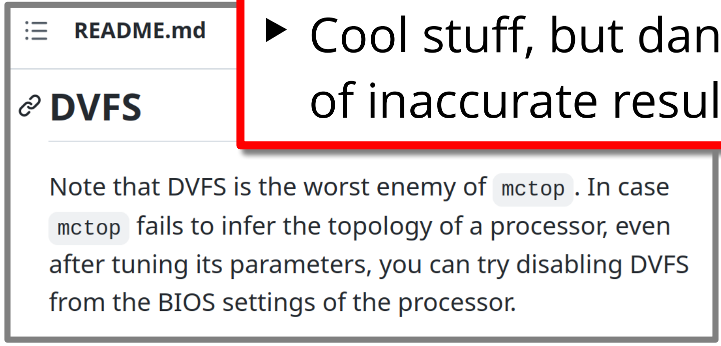

P (processes)
$P_{0,0}$
$P_{0,1}$
$P_{1,1}$
$P_{1,0}$
0
1
2
3
4
5
6
7
1000
map
$P_{0,0} \rightarrow D_{0,0}$
$P_{0,1} \rightarrow D_{0,1}$
$P_{1,0} \rightarrow D_{1,0}$
$P_{1,1} \rightarrow D_{1,1}$
D (domain)
$D_{0,1}$
$D_{1,1}$
$D_{0,0}$
$D_{1,0}$

4. Result:

process	0	1	2	3	4	5	6	7
core	3	7	4	0	6	2	5	1

MCTOP [3] — A Multicore Performance Model

- **Measurement-based** topology inference
  - Fancy **MCTOP-ALG** algorithm:
    Latency matrix → clustering → components → roles
  - But:



README.md

DVFS

Note that DVFS is the worst enemy of `mctop`. In case `mctop` fails to infer the topology of a processor, even after tuning its parameters, you can try disabling DVFS from the BIOS settings of the processor.

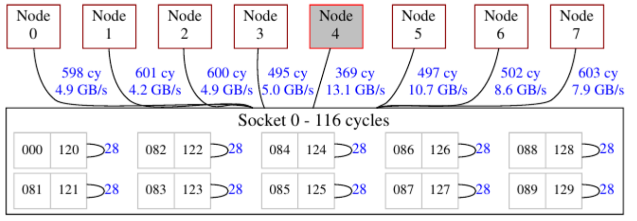
▶ Cool stuff, but danger of inaccurate results

- Portable and extensible by plugins

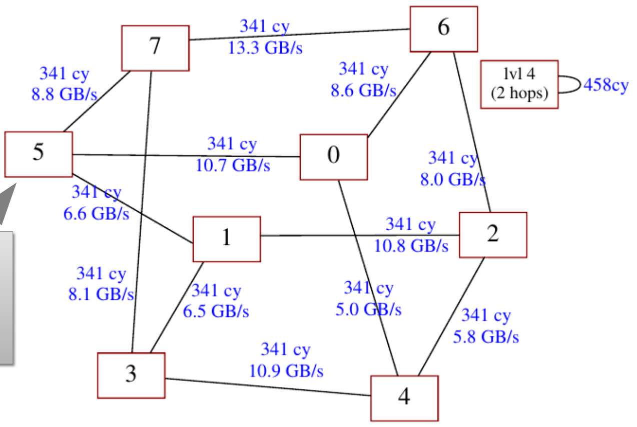
accessible memory

“execution contexts”

NUMA node



(a) Intra-socket topology of a socket.



(b) Cross-socket topology.

Figure 2: MCTOP of an 8-socket Intel processor.

“system knowledge base”

Barrelfish SKB [4] — A Versatile Framework

- **Constraint Logic Programming** (Prolog-based)
- **datagatherer** provides „facts“
  - CPUID → cores, cache sizes, ...
  - ACPI tables → memory regions, ...
  - PCIe/USB enumeration → device list
  - measurements → performance metrics
  - any device driver can add facts
- **client library** supports queries and algorithms
  - examples: PCI configuration, NUMA-aware multicast, global resource mgnt.

```
% the bridge with a hotplug-capable slot under it
bridge(pcie, addr(3, 0, 0), 0x1033, 0x125, 6, 4, 0,
        secondary(4)).

% artificial device with vendor set to 0xffff and all
% other fields to 0
device(pcie, addr(4, 3, 0), 0xffff, 0, 0, 0, 0, 0).

% three small BARs, one in each space
bar(addr(4, 3, 0), 0, 0, 8192, mem, prefetchable, 64).
bar(addr(4, 3, 0), 0, 0, 8192, mem, non-prefetchable, 32).
bar(addr(4, 3, 0), 0, 0, 256, io, non-prefetchable, 32).
```

▶ Nice, but the infrastructure is not for free!

Pythia [8] — Smart Co-Location with ML

- **Regression model** predicts contention on shared resources
  - For latency-sensitive workload combined with K batch workloads $W_0,\ldots,W_{K-1}$

combined interference

$$B_S = \sum_{i=0}^{K-1} c_{W_i} B_{W_i}$$

individual interference

***trained* coefficients: interference vulnerability**

  - High accuracy even with sparsely sampled combination space
- Usage scenario:

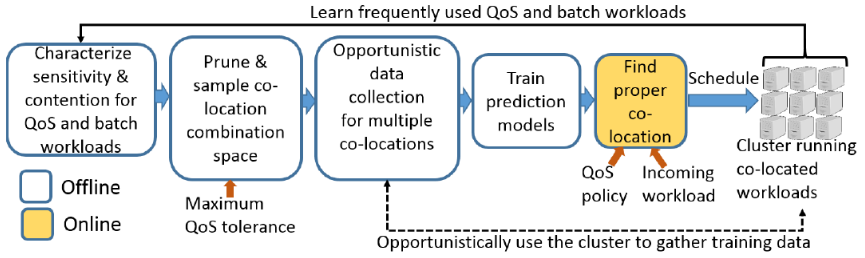

Learn frequently used QoS and batch workloads
Characterize sensitivity & contention for QoS and batch workloads
Prune & sample co-location combination space
Opportunistic data collection for multiple co-locations
Train prediction models
Find proper co-location
Schedule
Cluster running co-located workloads
Offline
Online
Maximum QoS tolerance
QoS policy
Incoming workload
Opportunistically use the cluster to gather training data

Figure 8: The complete workflow of PYTHIA.

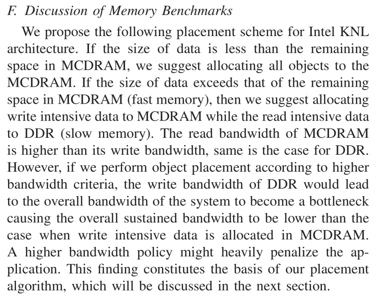
▶ Very compact system ***and*** application model!

Outline

- **Benchmarks and models for individual components**
  CPU + DRAM + disruptive memory technology
- **System- and application-level performance models**
  interaction and contention between components and applications
- **Challenges and conclusion**

Defining “Performance Model”

Prose



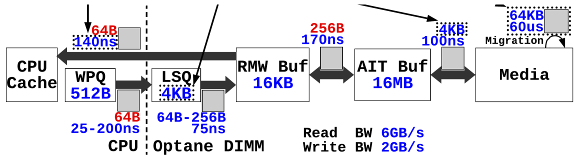
F. Discussion of Memory Benchmarks

We propose the following placement scheme for Intel KNL architecture. If the size of data is less than the remaining space in MCDRAM, we suggest allocating all objects to the MCDRAM. If the size of data exceeds that of the remaining space in MCDRAM (fast memory), then we suggest allocating write intensive data to MCDRAM while the read intensive data to DDR (slow memory). The read bandwidth of MCDRAM is higher than its write bandwidth, same is the case for DDR. However, if we perform object placement according to higher bandwidth criteria, the write bandwidth of DDR would lead to the overall bandwidth of the system to become a bottleneck causing the overall sustained bandwidth to be lower than the case when write intensive data is allocated in MCDRAM. A higher bandwidth policy might heavily penalize the application. This finding constitutes the basis of our placement algorithm, which will be discussed in the next section.

HBM (Laghari et al.)

- Manual generation: Benchmark analysis, data sheets
- Manual application
- Can include simulators

Visual

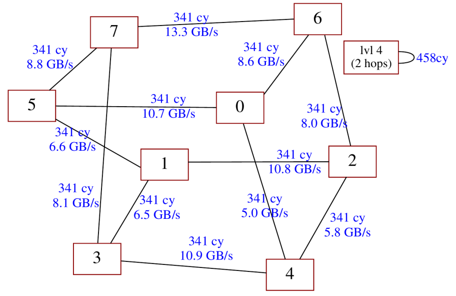


NVM (Wang et al.)

- Partially automated
- Can be built into placement algorithms
- manual post-processing

Automated model usage requires formal **meta models**

Formal

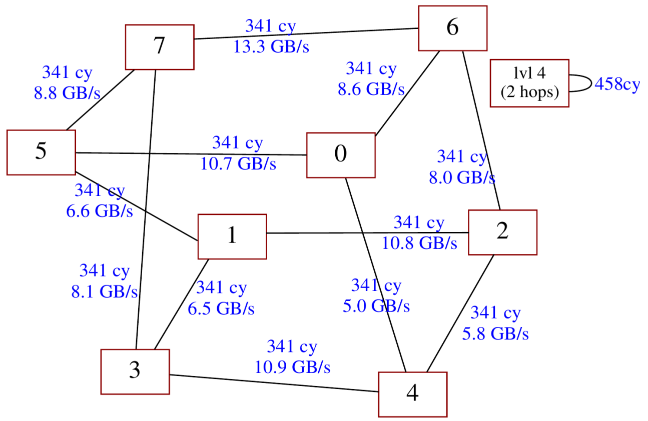


mctop (Chatzopoulos et al.)

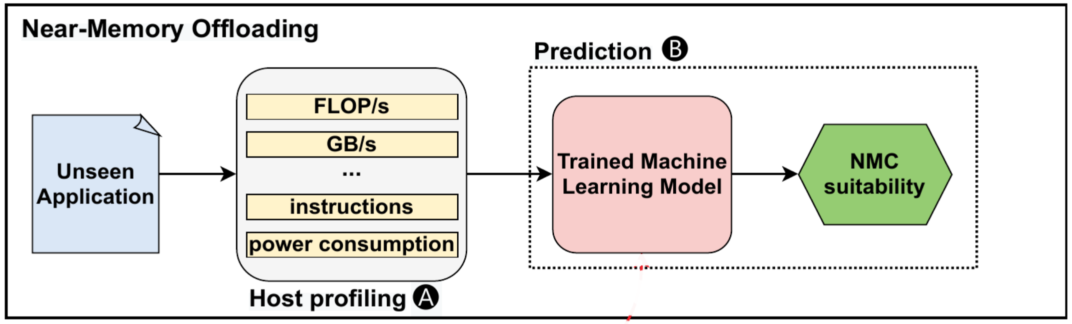
E.g. weighted graph (V, E, W)
v ∈ V : NUMA nodes
(u, v) ∈ E ⇔ Interconnect u – v
w(u, v) = (latency, bandwidth)

Model Design

- Knowledge base
  - Querying topology / performance attributes
  - E.g. HBM channels, LENS, SKB, hwloc, mctop, …
- Machine learning
  - Pythia: Batch workload scheduling
  - NMPO: Offloading suitability



Can (and should) we **combine** Knowledge bases and ML?



Model Use Cases

Speed →

HH:MM	ms	µs	ns
Simulation	Graph, Network	Regression, heuristic	Hardware heuristic

- Offline
  - Verification
  - Evaluation
- Simulation must be correct

- Workload placement

- Task placement

- Short-term data placement, e.g. caching

Workload/Task model?
Profiling required?

How to design **scalable models**?

Conclusion

- Disruptive Memory Technologies challenge design assumptions
- System engineers should adjust models and placement algorithms
- Modeling methods, assumptions, and findings vary
  - Lack of **meta models** in the literature
  - Inconsistent **contention** handling
  - **Data source**: data sheets vs. micro-benchmarks
  - Profiling / Describing **application requirements**

⇒ Lots of room for improvement

SPP 2377

References (1)

[1] F. Broquedis, et al., *hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications*, in 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2010), Pisa, 2010, doi: 10.1109/PDP.2010.67

[2] G. Mercier, J. Clet-Ortega. *Towards an efficient process placement policy for MPI applications in multicore environments*. Europvm/mpi 2009, Sep 2009, Espoo, Finland. pp 104--115, ⟨10.1007/978-3-642-03770-2_17⟩.

[3] G. Chatzopoulos, R. Guerraoui, T. Harris, and V. Trigonakis. 2017. *Abstracting Multi-Core Topologies with MCTOP*. In Proceedings of the Twelfth European Conference on Computer Systems (EuroSys '17). Association for Computing Machinery, New York, NY, USA, 544–559. https://doi.org/10.1145/3064176.3064194

References (2)

[4] A. L. Schüpbach. 2012. *Tackling OS Complexity with Declarative Techniques*. ETH Zürich.

[5] X. Wei, X. Xie, R. Chen, H. Chen, and B. Zang. 2021. *Characterizing and Optimizing Remote Persistent Memory with RDMA and NVM*. USENIX Annual Technical Conference.

[6] A. Akram and L. Sawalha. 2016. *A Comparison of x86 Computer Architecture Simulators*. Computer Architecture and Systems Research Laboratory (CASRL). 1. https://scholarworks.wmich.edu/casrl_reports/1

[7] F. Pellegrini. 2008. *Scotch and libScotch 5.1 User's Guide*. INRIA Bordeaux Sud-Ouest.

References (3)

[8] R. Xu, S. Mitra, J. Rahman, P. Bai, B. Zhou, G. Bronevetsky, and S. Bagchi. 2018. *Pythia: Improving Datacenter Utilization via Precise Contention Prediction for Multiple Co-located Workloads*. In Proceedings of the 19th International Middleware Conference (Middleware '18). Association for Computing Machinery, New York, NY, USA, 146–160. https://doi.org/10.1145/3274808.3274820

[9] R. Das, et al. 2020. *How to Manage High-Bandwidth Memory Automatically*. In Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA’20)

[10] L. Zhang, R. Karimi, I. Ahmad, Y. Givfusson. 2020. *Optimal Data Placement for Heterogeneous Cache, Memory, and Storage Systems*. In Proc. ACM Meas. Anal. Comput. Syst., Vol. 4, No. 1

References (4)

[11] Z. Wang, et al. 2020. *Characterizing and Modeling Non-Volatile Memory Systems*. In Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)

[12] Z. Wang, et al. 2023. *NVLeak: Off-Chip Side-Channel Attacks via Non-Volatile Memory Systems*. In USENIX Security Symposium

[13] M. Laghari, D. Unat. 2017. *Object Placement for High Bandwidth Memory Augmented with High Capacity Memory*. In Proceedings of the 29th International Symposium on Computer Architecture and High Percormance Computing (SBAC-PAD)

[14] D. Shen, et al. 2020. *Distributed and Optimal RDMA Resource Scheduling in Shared Data Center Networks*. In IEEE Conference on Computer Communications (INFOCOM)

References (5)

[15] H. Qiu, et al. 2018. *Toward Effective and Fair RDMA Resource Sharing*. In APNET ,18: 2nd Asia-Pacific Workshop on Networking

[16] J. Gómez-Luna, et al. 2022. *Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System*. In IEEE Access, Vol. 10, 2022

[17] S. Corda, et al. 2021. *NMPO: Near-memory Computing Profiling and Offloading*. In 24th Euromicro Conference on Digital System Design (DSD)