OSNABRÜCK UNIVERSITY

Embedded Software Systems

Feasibility Analysis of Semi-Permanent Database Offloading to UPMEM NMC Modules

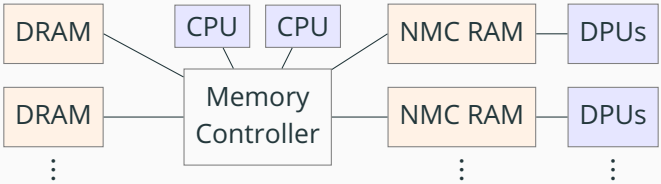
Birte Friesel, Marcel Lütke Dreimann, Olaf Spinczyk

March 4th, 2025

ess.cs.uos.de/~bf

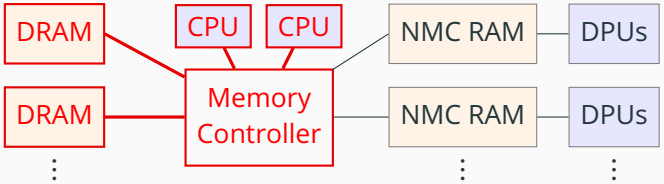
birte.friesel@uos.de

Near-Memory Computing



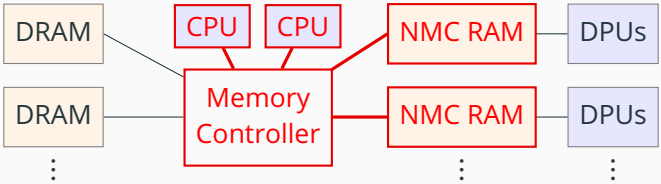
- Near-Memory Computing (NMC): move computation close to data

Near-Memory Computing



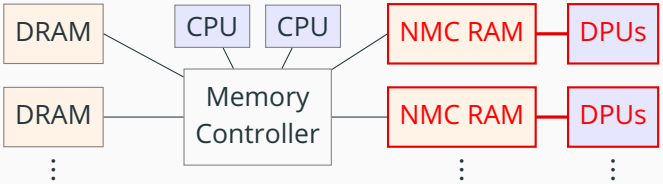
- Near-Memory Computing (NMC): move computation close to data

Near-Memory Computing



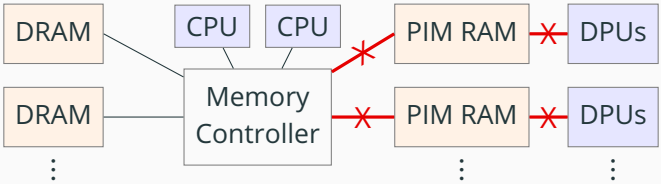
- Near-Memory Computing (NMC): move computation close to data

Near-Memory Computing



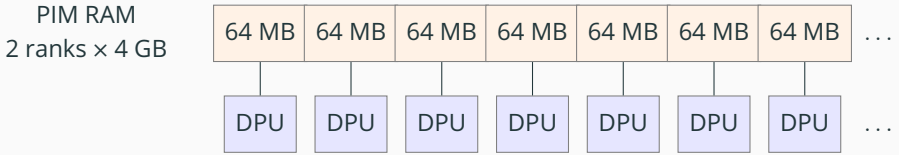
- Near-Memory Computing (NMC): move computation close to data
  - DRAM Processing Units (DPUs) have direct data access
  - Massive parallelism without contention: up to 2560 DPUs per system

Near-Memory Computing with UPMEM PIM



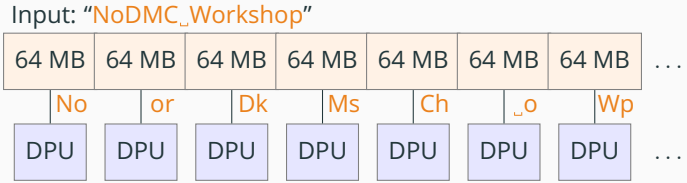
- Near-Memory Computing (NMC): move computation close to data
  - DRAM Processing Units (DPUs) have direct data access
  - Massive parallelism without contention: up to 2560 DPUs per system
- UPMEM PIM: only commercially available NMC hardware
- Behaves like an accelerator and not (ideal) NMC

UPMEM PIM: An Accelerator for Parallel Kernels

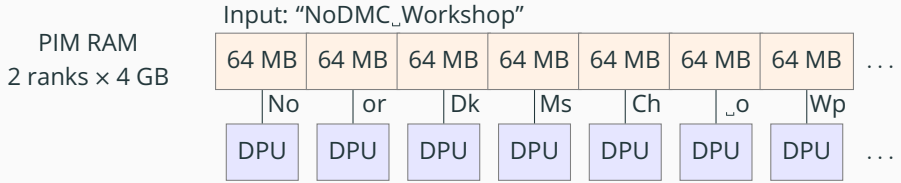


UPMEM PIM: An Accelerator for Parallel Kernels

PIM RAM
2 ranks × 4 GB

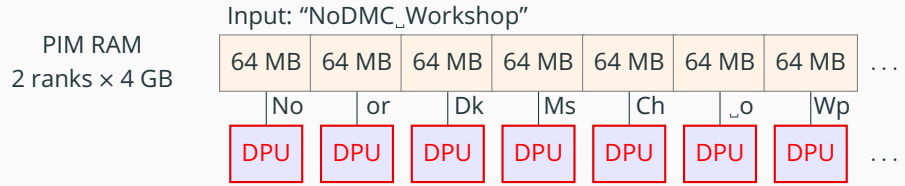


UPMEM PIM: An Accelerator for Parallel Kernels



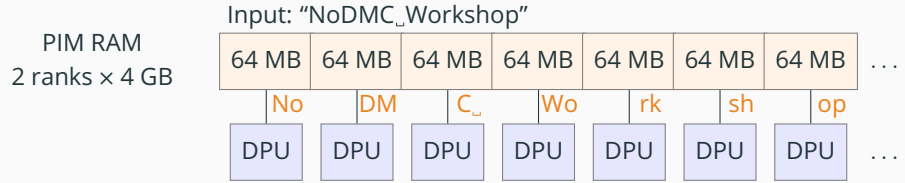
- CPU execution: runKernel(. . .)

UPMEM PIM: An Accelerator for Parallel Kernels



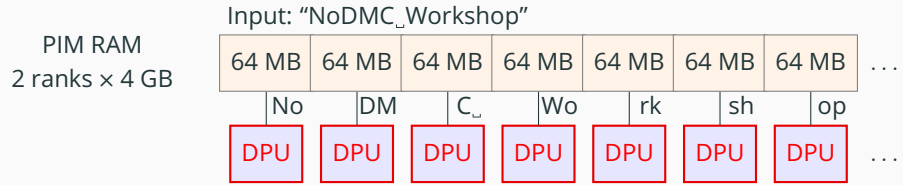
- CPU execution: runKernel(. . .)
- NMC execution: alloc(. . .); writeData(. . .); loadKernel(. . .); writeQuery(. . .); runKernel(); readResult();

UPMEM PIM: An Accelerator for Parallel Kernels



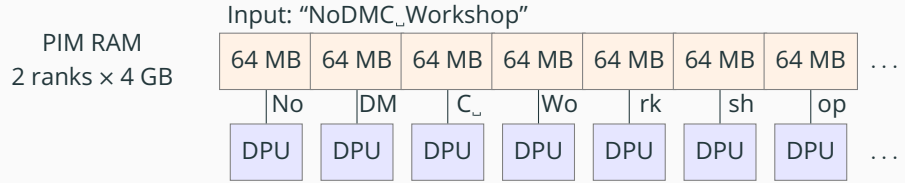
- CPU execution: runKernel(. . .)
- NMC execution: alloc(. . .); writeData(. . .); loadKernel(. . .); writeQuery(. . .); runKernel(); readResult();

UPMEM PIM: An Accelerator for Parallel Kernels



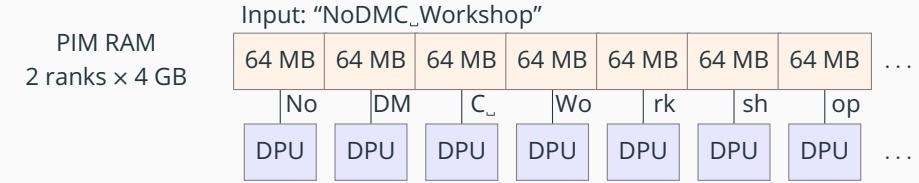
- CPU execution: runKernel(. . .)
- NMC execution: alloc(. . .); writeData(. . .); loadKernel(. . .); writeQuery(. . .); runKernel(); readResult();

UPMEM PIM: An Accelerator for Parallel Kernels



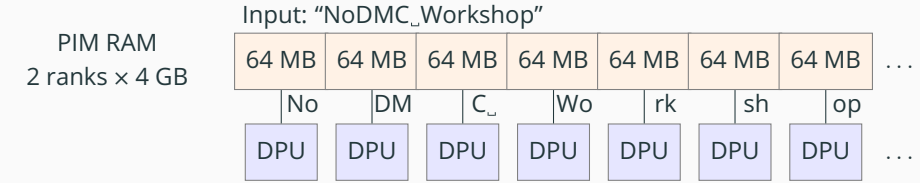
- CPU execution: runKernel(. . .)
- NMC execution: alloc(. . .); writeData(. . .); loadKernel(. . .); writeQuery(. . .); runKernel(); readResult();

UPMEM PIM: An Accelerator for Parallel Kernels



- CPU execution: runKernel(. . .)
- NMC execution: alloc(. . .); writeData(. . .); loadKernel(. . .); writeQuery(. . .); runKernel(); readResult();

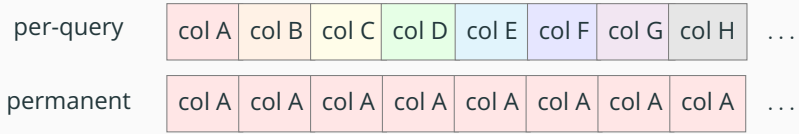
UPMEM PIM: An Accelerator for Parallel Kernels



- CPU execution: runKernel(. . .)
- NMC execution: alloc(. . .); writeData(. . .); loadKernel(. . .); writeQuery(. . .); runKernel(); readResult();

UPMEM PIM in Database Systems

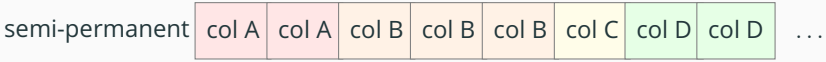
- Kernels can be > 100× faster than CPU, but high overhead
- (Partial) offloading of analytical queries [BJS23a; BJS23c; BJS23b; Lim+23]; either per-query or permanent offloading (constant data)



UPMEM PIM in Database Systems

- Kernels can be > 100× faster than CPU, but high overhead
- (Partial) offloading of analytical queries [BJS23a; BJS23c; BJS23b; Lim+23]; either per-query or permanent offloading (constant data)

→ Middle ground: semi-permanent offloading
(several consecutive queries on data residing in UPMEM)



UPMEM PIM in Database Systems

- Kernels can be > 100× faster than CPU, but high overhead
- (Partial) offloading of analytical queries [BJS23a; BJS23c; BJS23b; Lim+23]; either per-query or permanent offloading (constant data)

→ Middle ground: semi-permanent offloading
(several consecutive queries on data residing in UPMEM)

- Generic approach for identifying UPMEM-compatible workloads:
  - Determine break-even point: #queries where NMC outperforms CPU
  - Variable operators, #queries, #rows, #cores, #ranks (= $\frac{\#\text{DPUs}}{64}$)

Approach: Performance Comparison

$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$

$\Rightarrow$ offload to UPMEM PIM modules

Approach: Performance Comparison

$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$

$\Rightarrow$ offload to UPMEM PIM modules

NMC configuration space: 100 $(1 \ldots 100$ queries$)$

Approach: Performance Comparison

$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$

$\Rightarrow$ offload to UPMEM PIM modules

NMC configuration space: $100 \cdot 16$ ($2^{16} \ldots 2^{32}$ rows)

Approach: Performance Comparison

$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$

$\Rightarrow$ offload to UPMEM PIM modules

NMC configuration space: $100 \cdot 16 \cdot 40$ ($1 \ldots 40$ ranks)

Approach: Performance Comparison

$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$

$\Rightarrow$ offload to UPMEM PIM modules

NMC configuration space: $100 \cdot 16 \cdot 40 \geq 64000$ per operation

Approach: Performance Models

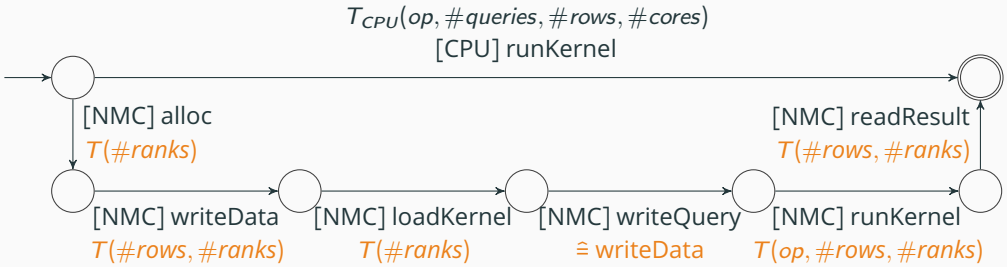
$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$

$\Rightarrow$ offload to UPMEM PIM modules

NMC configuration space: $100 \cdot 16 \cdot 40 \geq 64000$ per operation

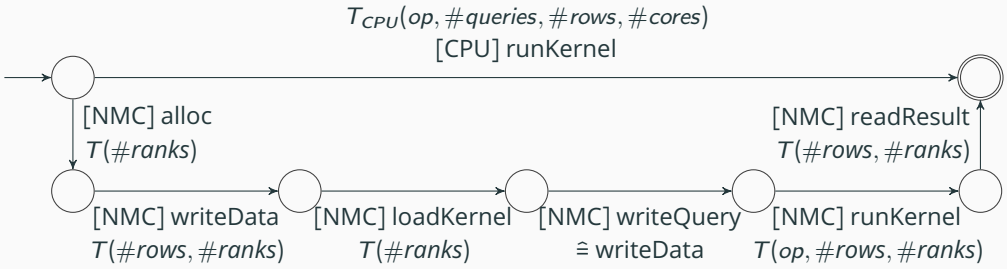
Approach: Performance-Aware Behaviour Models

$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$
$\Rightarrow$ offload to UPMEM PIM modules
NMC configuration space: $100 \cdot 16 \cdot 40 \geq 64000$ per operation



Approach: Performance-Aware Behaviour Models

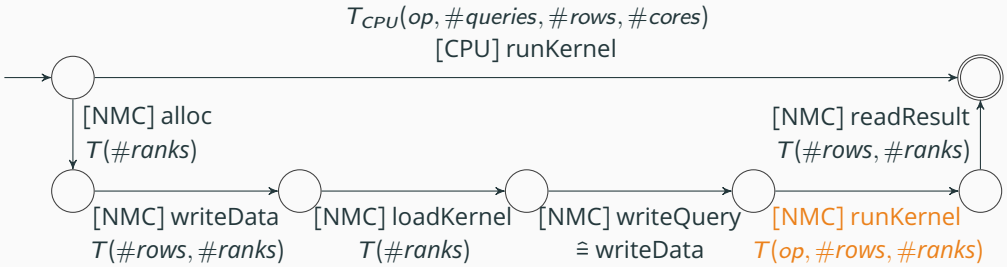
$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$
$\Rightarrow$ offload to UPMEM PIM modules
NMC configuration space: $100 \cdot 16 \cdot 40 \geq 64000$ per operation



NMC configuration space: $40 + 16 \cdot 40 + 40 + 16 \cdot 40 \geq 1360$ per operation

Approach: Performance-Aware Behaviour Models

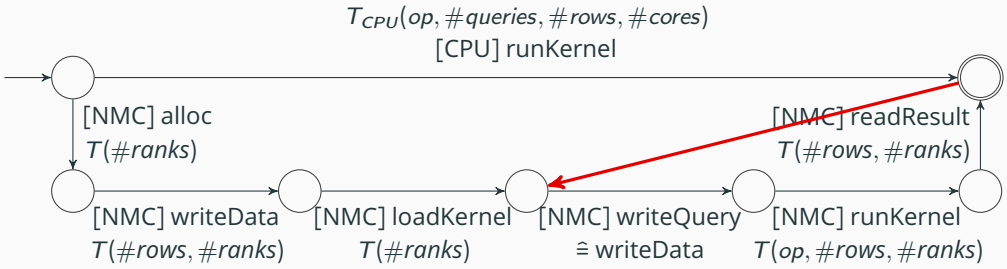
$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$
$\Rightarrow$ offload to UPMEM PIM modules
NMC configuration space: $100 \cdot 16 \cdot 40 \geq 64000$ per operation



NMC configuration space: $40 + 16 \cdot 40 + 40 + 16 \cdot 40 \geq 1360$ per operation

Approach: Performance-Aware Behaviour Models

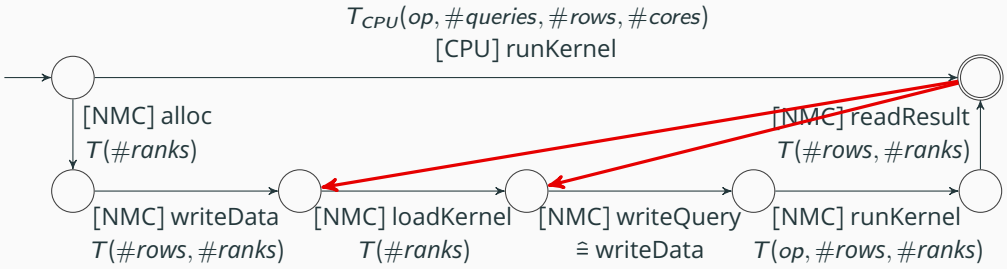
$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$
$\Rightarrow$ offload to UPMEM PIM modules
NMC configuration space: $100 \cdot 16 \cdot 40 \geq 64000$ per operation



NMC configuration space: $40 + 16 \cdot 40 + 40 + 16 \cdot 40 \geq 1360$ per operation

Approach: Performance-Aware Behaviour Models

$T_{NMC}(op, \#queries, \#rows, \#ranks) < T_{CPU}(op, \#queries, \#rows, \#cores)$
$\Rightarrow$ offload to UPMEM PIM modules
NMC configuration space: $100 \cdot 16 \cdot 40 \geq 64000$ per operation



NMC configuration space: $40 + 16 \cdot 40 + 40 + 16 \cdot 40 \geq 1360$ per operation

Determining the Break-Even Point

Setup:

- Benchmarks → models for NMC alloc, load kernel, write data, read data
- Benchmarks → models for CPU/NMC kernel latency:
  - `SELECT COUNT(*) FROM` *col* (uint64) `WHERE` *predicate* → uint64
  - `SELECT * FROM` *col* (uint64) `WHERE` *predicate* → bitmask

Determining the Break-Even Point

Setup:

- Benchmarks → models for NMC alloc, load kernel, write data, read data
- Benchmarks → models for CPU/NMC kernel latency:
  - `SELECT COUNT(*) FROM` *col* (uint64) `WHERE` *predicate* → uint64
  - `SELECT * FROM` *col* (uint64) `WHERE` *predicate* → bitmask
- Model-based simulation → total latency for CPU/NMC execution

CPU: #*queries* × runKernel()

NMC: alloc(); writeData(); loadKernel();
#*queries* × { writeQuery(); runKernel(); readResult(); }

Performance Models: Kernels

	COUNT		SELECT
CPU (1C)	$4.5\,\text{ns} + 2.2\,ns \cdot \#rows$	CPU (1C)	$0.0\,\text{ns} + 2.0\,ns \cdot \#rows$
CPU (≥4C)	$0.9\,\text{ns} + 0.5\,ns \cdot \#rows$	CPU (≥4C)	$4.8\,\text{ns} + 0.6\,ns \cdot \#rows$

Performance Models: Kernels

	COUNT		SELECT
CPU (1C)	$4.5\,\text{ns} + 2.2\,ns \cdot \#rows$	CPU (1C)	$0.0\,\text{ns} + 2.0\,ns \cdot \#rows$
CPU (≥4C)	$0.9\,\text{ns} + 0.5\,ns \cdot \#rows$	CPU (≥4C)	$4.8\,\text{ns} + 0.6\,ns \cdot \#rows$
NMC	$242\,\mu\text{s} + 0.53\,ns \cdot \frac{\#rows}{\#ranks}$	NMC	$203\,\mu\text{s} + 0.66\,ns \cdot \frac{\#rows}{\#ranks}$

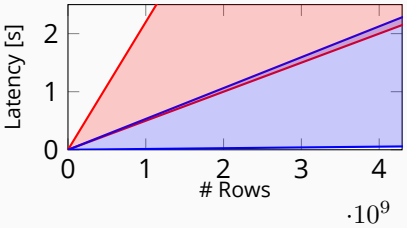
Performance Models: Kernels

COUNT

CPU (1C) $4.5\,\text{ns} + 2.2\,ns \cdot \#rows$

CPU (≥4C) $0.9\,\text{ns} + 0.5\,ns \cdot \#rows$

NMC $242\,\mu\text{s} + 0.53\,ns \cdot \frac{\#rows}{\#ranks}$

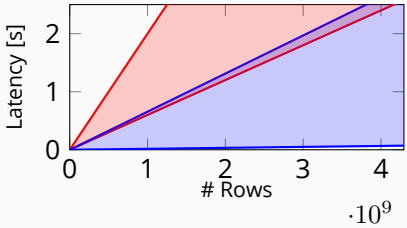


SELECT

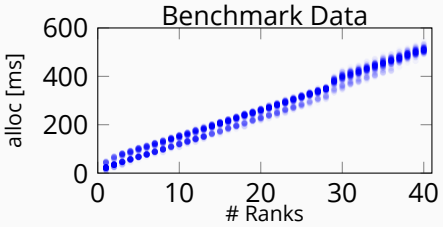
CPU (1C) $0.0\,\text{ns} + 2.0\,ns \cdot \#rows$

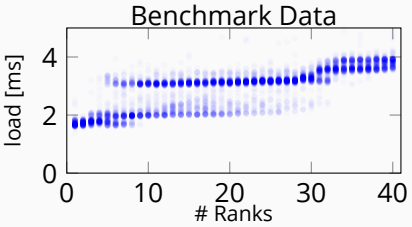
CPU (≥4C) $4.8\,\text{ns} + 0.6\,ns \cdot \#rows$

NMC $203\,\mu\text{s} + 0.66\,ns \cdot \frac{\#rows}{\#ranks}$

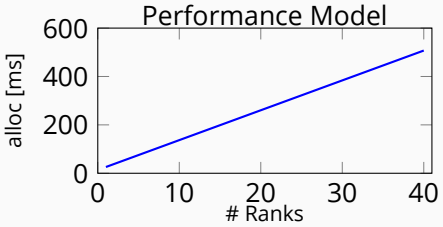


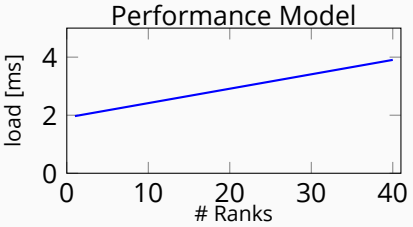
Performance Models: UPMEM Overhead



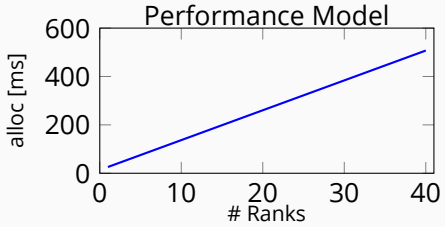


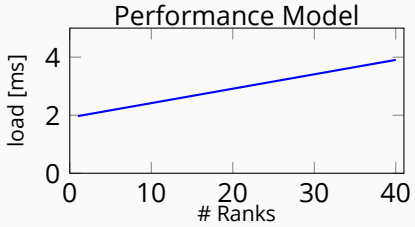
Performance Models: UPMEM Overhead

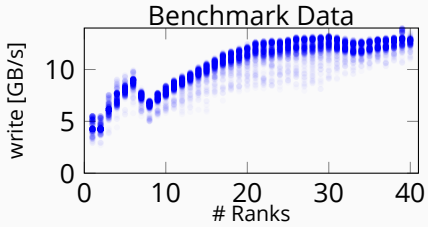


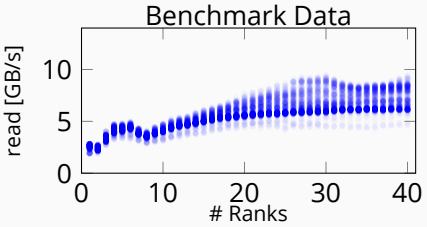


Performance Models: UPMEM Overhead

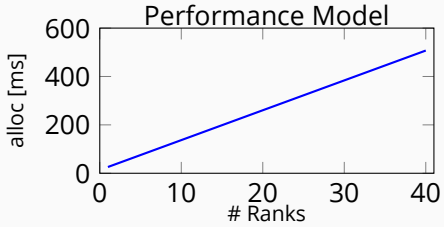


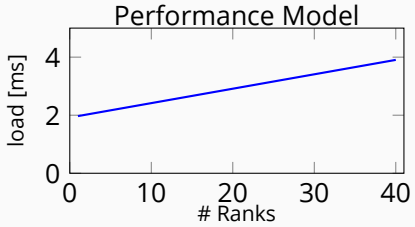


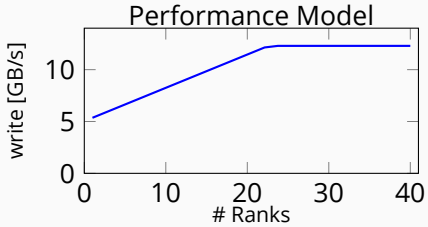


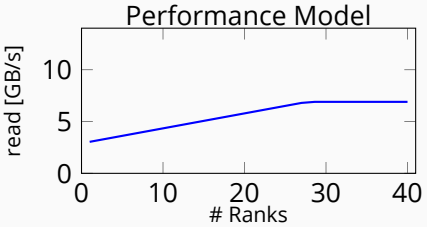


Performance Models: UPMEM Overhead

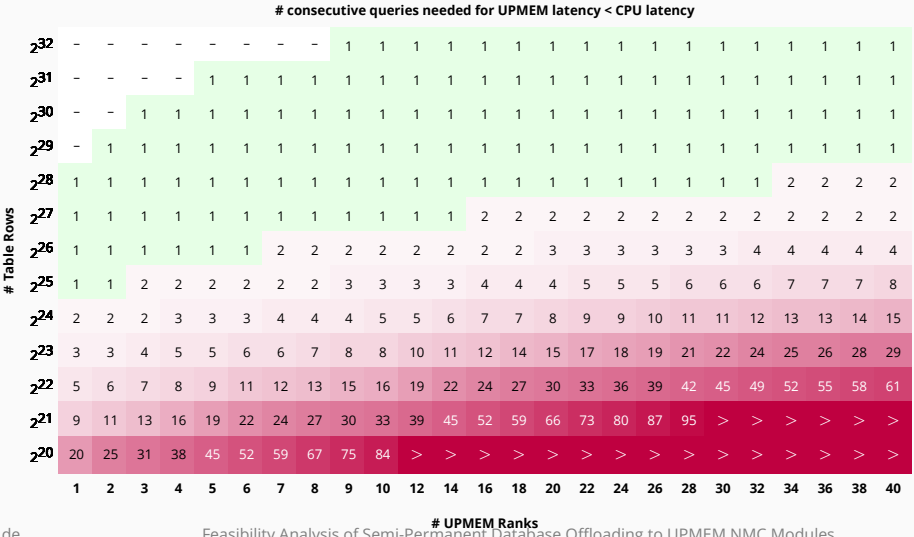




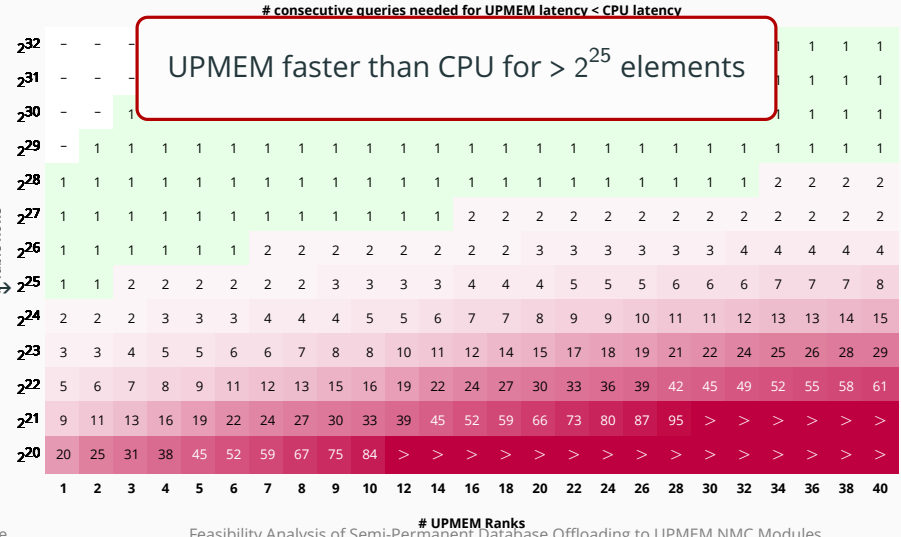




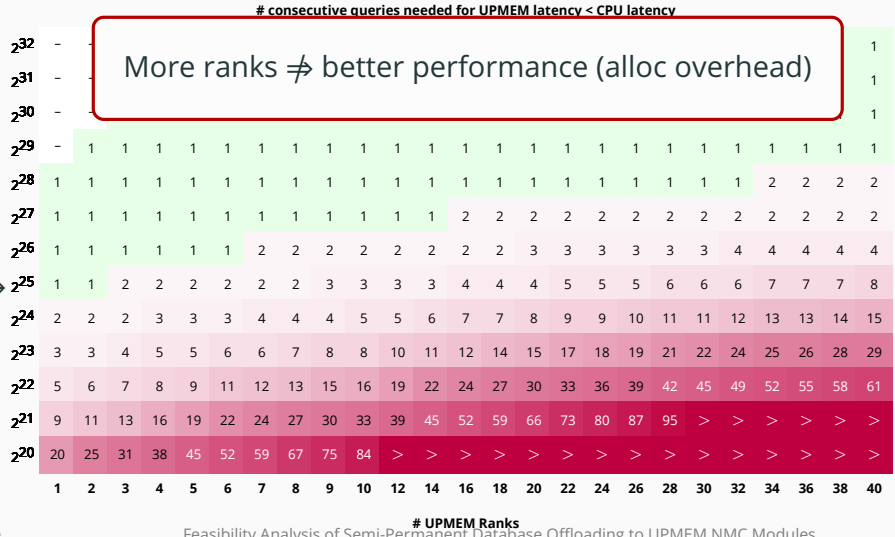
Simulation Results: COUNT (1 CPU Core)



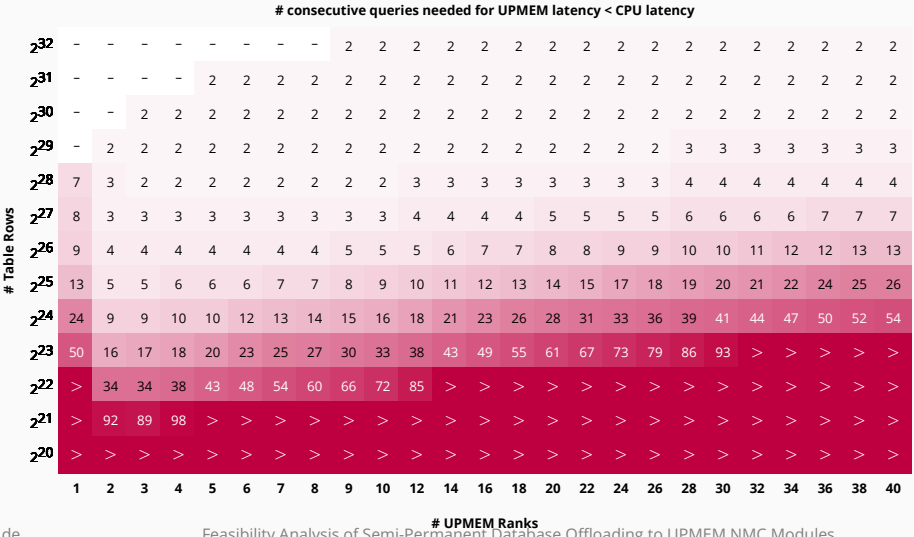
Simulation Results: COUNT (1 CPU Core)



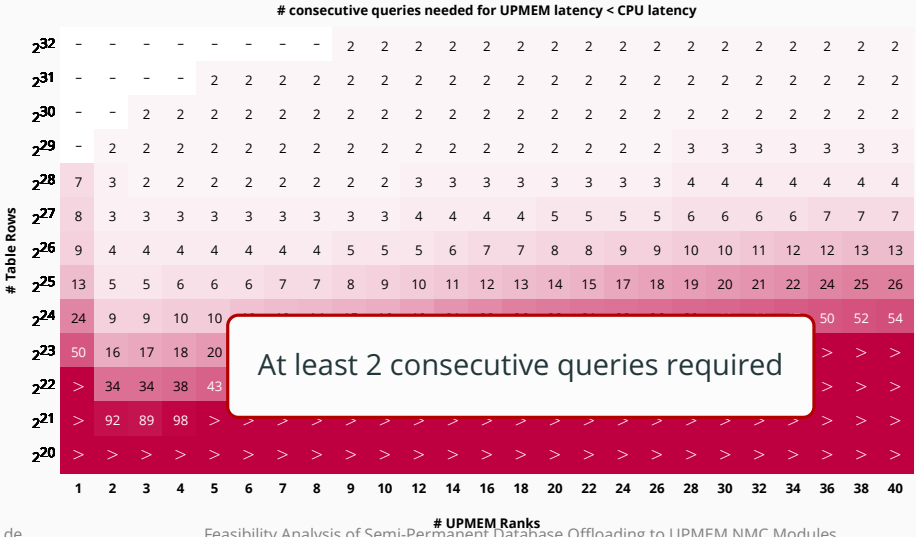
Simulation Results: COUNT (1 CPU Core)



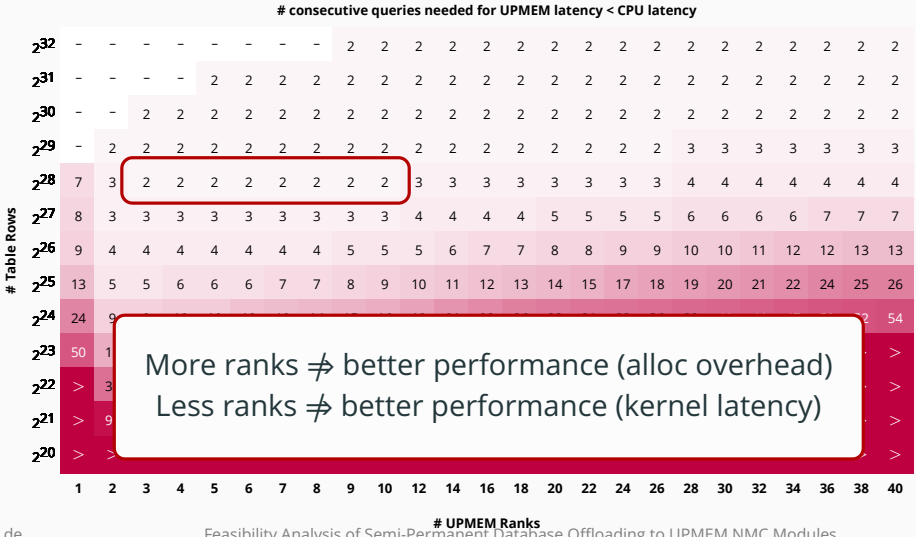
Simulation Results: COUNT (4 CPU Cores)



Simulation Results: COUNT (4 CPU Cores)

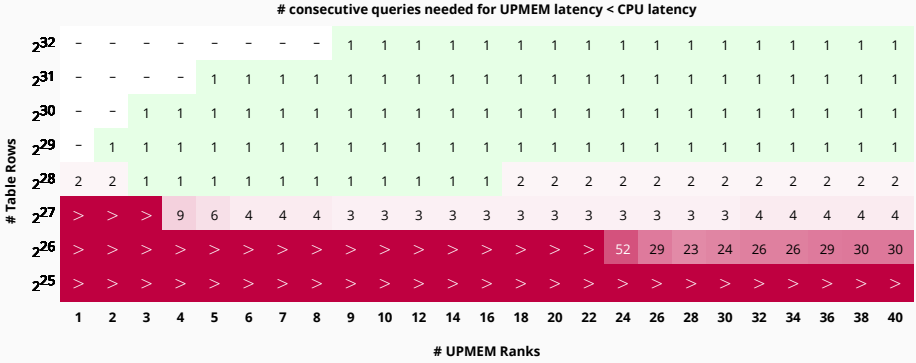


Simulation Results: COUNT (4 CPU Cores)

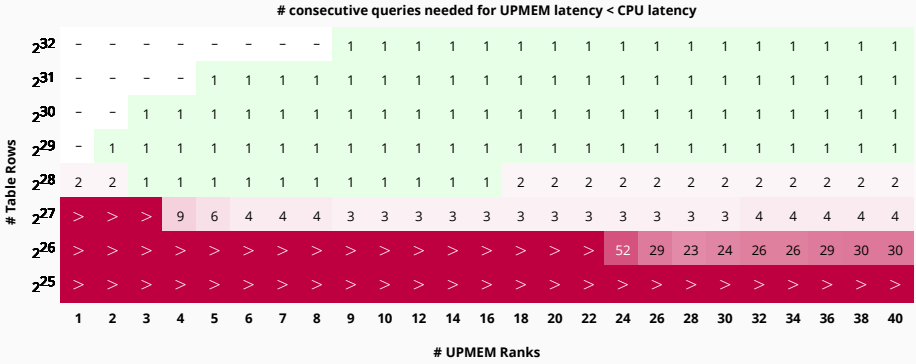


More ranks ⇏ better performance (alloc overhead)
Less ranks ⇏ better performance (kernel latency)

Simulation Results: SELECT (1 CPU Core)

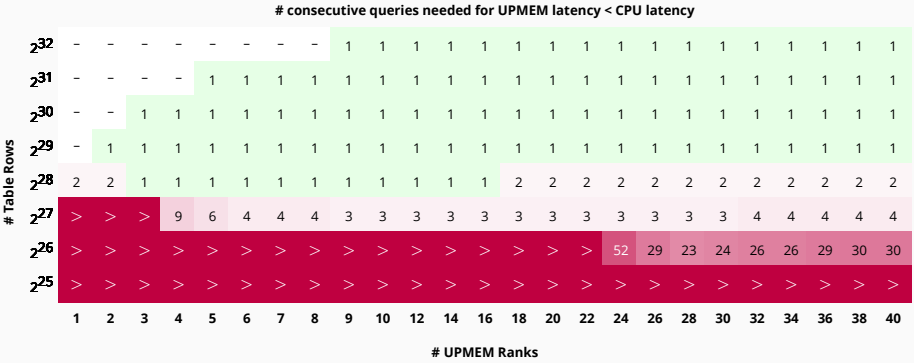


Simulation Results: SELECT (1 CPU Core)



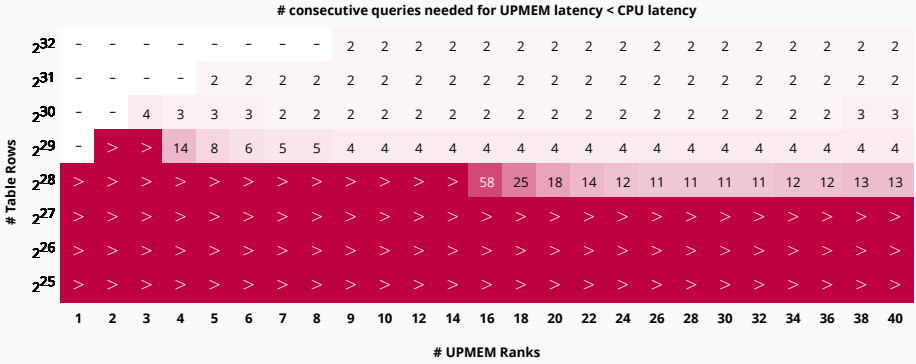
- More complex →≥ 2^{28} rows (2 GiB) needed

Simulation Results: SELECT (1 CPU Core)

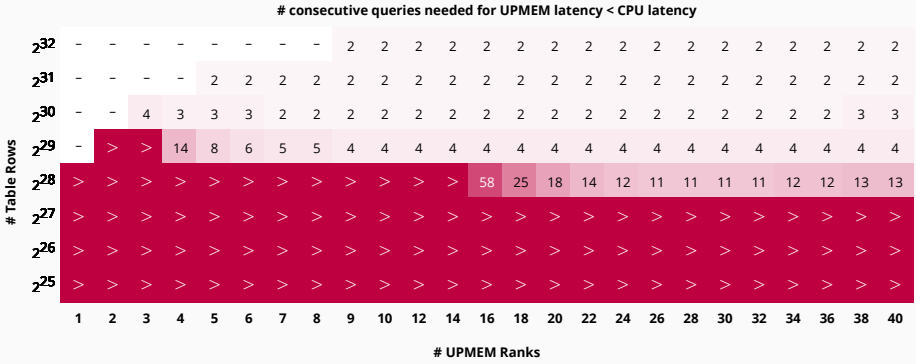


- More complex →≥ 2^{28} rows (2 GiB) needed
- Sweet spot for #ranks: neither min nor max (≙ COUNT)

Simulation Results: SELECT (4 CPU Cores)

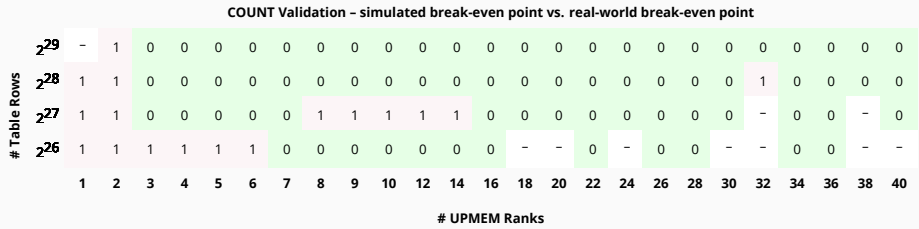


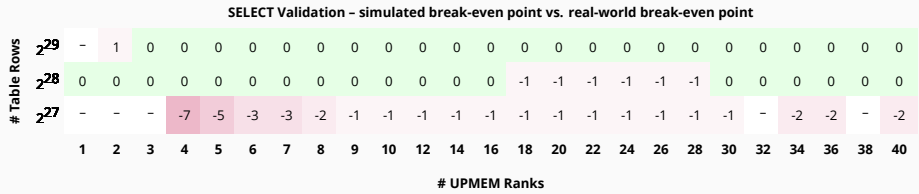
Simulation Results: SELECT (4 CPU Cores)



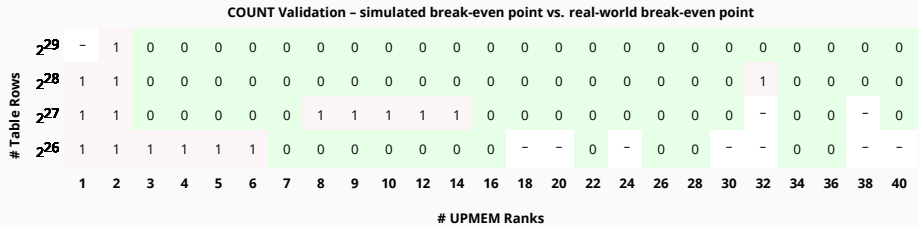
- At least 2 consecutive queries required (≙ COUNT)
- Sweet spot for #ranks: neither min nor max (≙ COUNT)

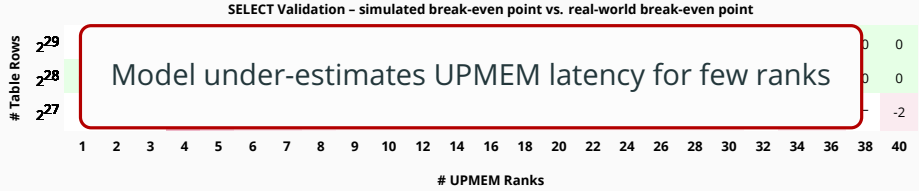
Validation of Simulation Results





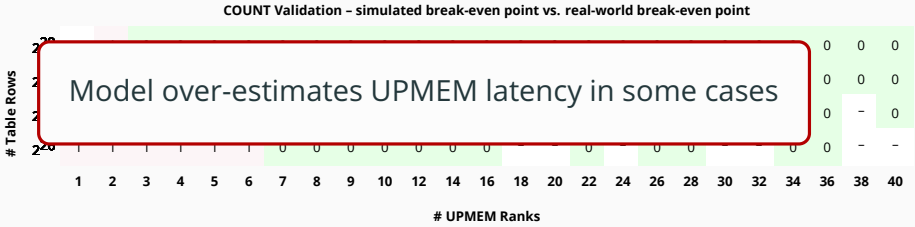
Validation of Simulation Results

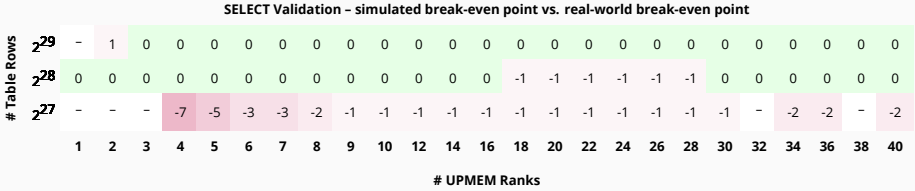




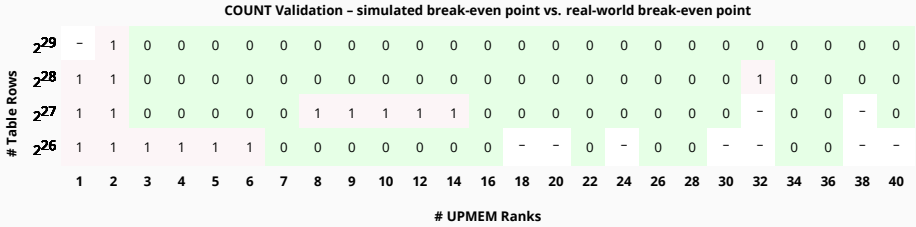
Model under-estimates UPMEM latency for few ranks

Validation of Simulation Results





Validation of Simulation Results



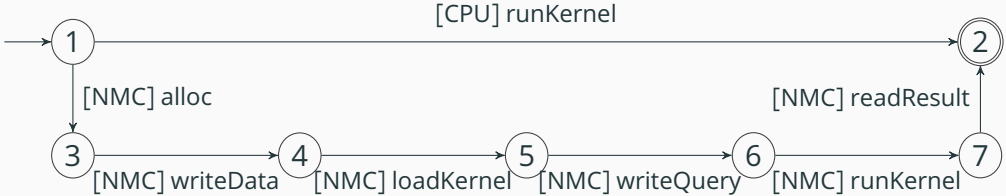
Low error, mostly in range of measurement uncertainty

Conclusion

- Offloading to UPMEM sensible for
  - simple kernels with a good chance of serving consecutive queries
  - on medium to large database columns (COUNT ≥ 256 MiB / SELECT ≥ 2 GiB)

Conclusion

- Offloading to UPMEM sensible for
  - simple kernels with a good chance of serving consecutive queries
  - on medium to large database columns (COUNT ≥ 256 MiB / SELECT ≥ 2 GiB)
- Analysis feasible thanks to performance-aware behaviour models
  - Manageable configuration space thanks to model decomposition



Conclusion

- Offloading to UPMEM sensible for
  - simple kernels with a good chance of serving consecutive queries
  - on medium to large database columns (COUNT ≥ 256 MiB / SELECT ≥ 2 GiB)
- Analysis feasible thanks to performance-aware behaviour models
  - Manageable configuration space thanks to model decomposition
  - Performance models give background for interpreting results
  - SDK or kernel changes → only repeat corresponding benchmarks

Conclusion

- Offloading to UPMEM sensible for
  - simple kernels with a good chance of serving consecutive queries
  - on medium to large database columns (COUNT ≥ 256 MiB / SELECT ≥ 2 GiB)
- Analysis feasible thanks to performance-aware behaviour models
  - Manageable configuration space thanks to model decomposition
  - Performance models give background for interpreting results
  - SDK or kernel changes → only repeat corresponding benchmarks
- Artifacts and source code referenced in the paper

References i

[BJS23a] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. **“Accelerating Large Table Scan using Processing-In-Memory Technology”.** In: BTW 2023. Gesellschaft für Informatik e.V., 2023, pp. 797–814. DOI: 10.18420/BTW2023-51.

[BJS23b] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. **“Adaptive Query Compilation with Processing-in-Memory”.** In: Proceedings of the 39th International Conference on Data Engineering Workshops. ICDEW ’23. 2023, pp. 191–197. DOI: 10.1109/ICDEW58674.2023.00035.

References ii

[BJS23c] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. **“Processing-in-Memory for Databases: Query Processing and Data Transfer”.** In: Proceedings of the 19th International Workshop on Data Management on New Hardware. DaMoN ’23. Association for Computing Machinery, 2023, pp. 107–111. DOI: 10.1145/3592980.3595323.

[Lim+23] Chaemin Lim et al. **“Design and Analysis of a Processing-in-DIMM Join Algorithm: A Case Study with UPMEM DIMMs”.** In: Proc. ACM Manag. Data 1.2 (June 2023). DOI: 10.1145/3589258.