

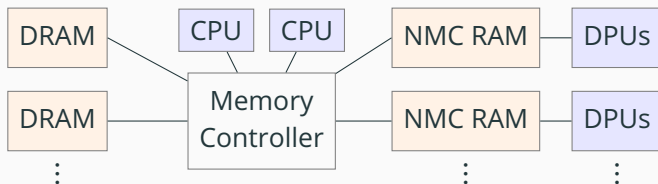
Feasibility Analysis of Semi-Permanent Database Offloading to UPMEM NMC Modules

Birte Friesel, Marcel Lütke Dreimann, Olaf Spinczyk

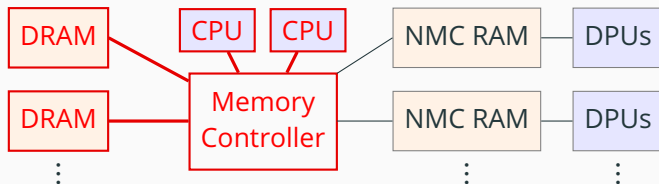
March 4th, 2025

`ess.cs.uos.de/~bf`

`birte.friesel@uos.de`

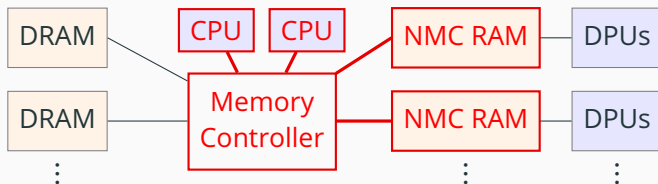


- Near-Memory Computing (NMC): move computation close to data

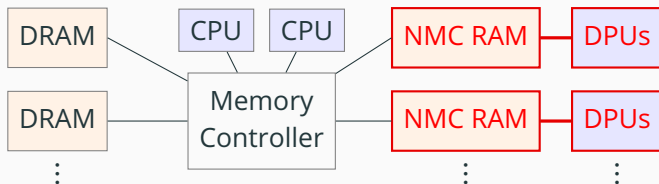


- Near-Memory Computing (NMC): move computation close to data

Near-Memory Computing

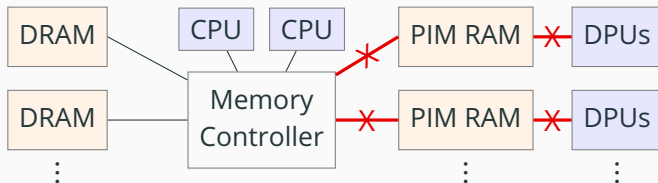


- Near-Memory Computing (NMC): move computation close to data



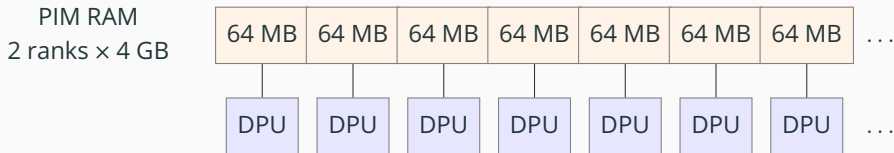
- Near-Memory Computing (NMC): move computation close to data
 - DRAM Processing Units (DPUs) have direct data access
 - Massive parallelism without contention: up to 2560 DPUs per system

Near-Memory Computing with UPMEM PIM

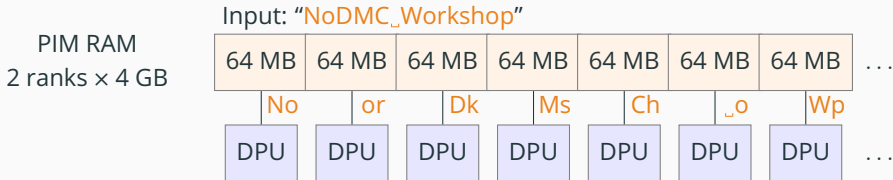


- Near-Memory Computing (NMC): move computation close to data
 - DRAM Processing Units (DPUs) have direct data access
 - Massive parallelism without contention: up to 2560 DPUs per system
- UPMEM PIM: only commercially available NMC hardware
- Behaves like an accelerator and not (ideal) NMC

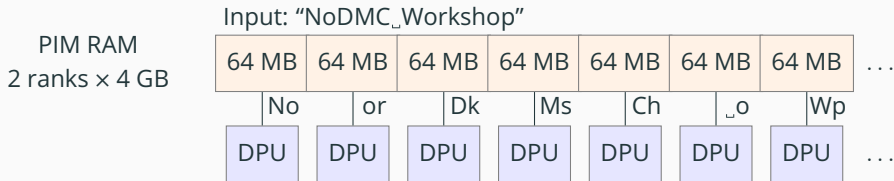
UPMEM PIM: An Accelerator for Parallel Kernels



UPMEM PIM: An Accelerator for Parallel Kernels

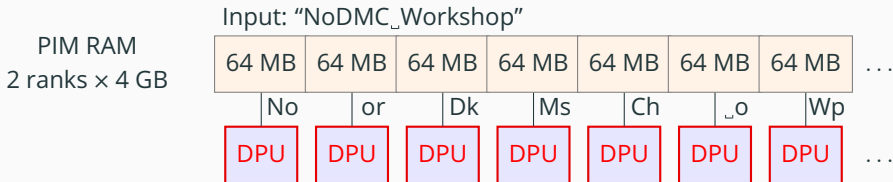


UPMEM PIM: An Accelerator for Parallel Kernels



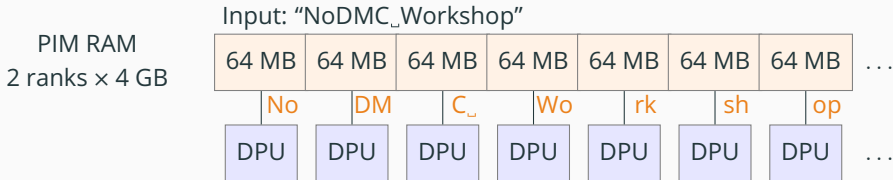
- CPU execution: `runKernel(...)`

UPMEM PIM: An Accelerator for Parallel Kernels



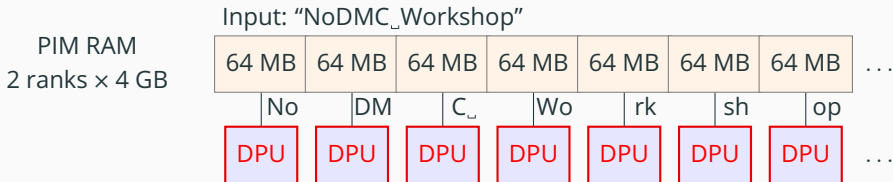
- CPU execution: `runKernel(...)`
- NMC execution: `alloc(...); writeData(...); loadKernel(...); writeQuery(...); runKernel(); readResult();`

UPMEM PIM: An Accelerator for Parallel Kernels



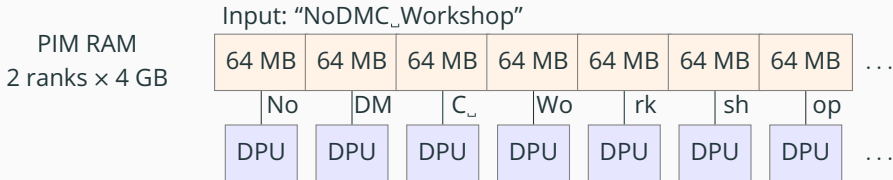
- CPU execution: `runKernel(...)`
- NMC execution: `alloc(...); writeData(...); loadKernel(...); writeQuery(...); runKernel(); readResult();`

UPMEM PIM: An Accelerator for Parallel Kernels



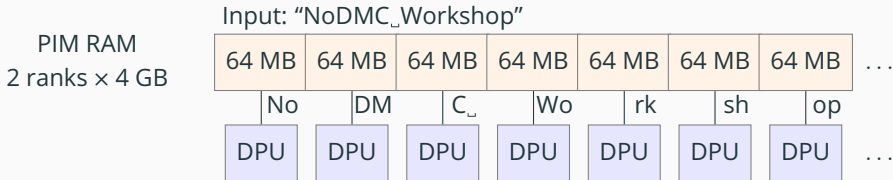
- CPU execution: `runKernel(...)`
- NMC execution: `alloc(...); writeData(...); loadKernel(...); writeQuery(...); runKernel(); readResult();`

UPMEM PIM: An Accelerator for Parallel Kernels



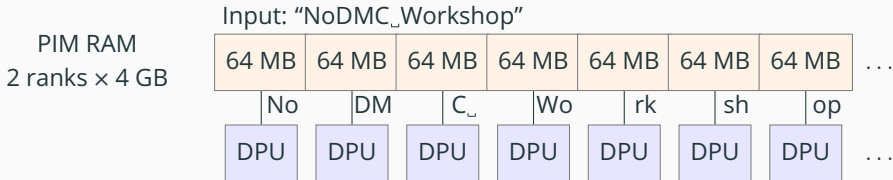
- CPU execution: `runKernel(...)`
- NMC execution: `alloc(...); writeData(...); loadKernel(...);`
`writeQuery(...); runKernel(); readResult();`

UPMEM PIM: An Accelerator for Parallel Kernels



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

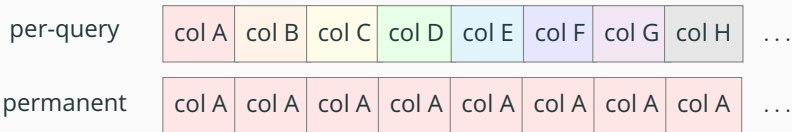
UPMEM PIM: An Accelerator for Parallel Kernels



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

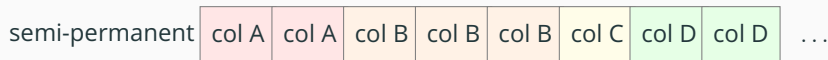


- Kernels can be $> 100\times$ faster than CPU, but high overhead
- (Partial) offloading of analytical queries [BJS23a; BJS23c; BJS23b; Lim+23]; either **per-query** or **permanent** offloading (constant data)





- Kernels can be $> 100\times$ 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)





- 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 ($= \frac{\#DPUs}{64}$)

Approach: Performance Comparison



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

⇒ offload to UPMEM PIM modules

Approach: Performance Comparison



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

⇒ offload to UPMEM PIM modules

NMC configuration space: 100 (1 . . . 100 queries)

Approach: Performance Comparison



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

⇒ offload to UPMEM PIM modules

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

Approach: Performance Comparison



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

⇒ offload to UPMEM PIM modules

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

Approach: Performance Comparison



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

⇒ offload to UPMEM PIM modules

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

Approach: Performance Models



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

⇒ offload to UPMEM PIM modules

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

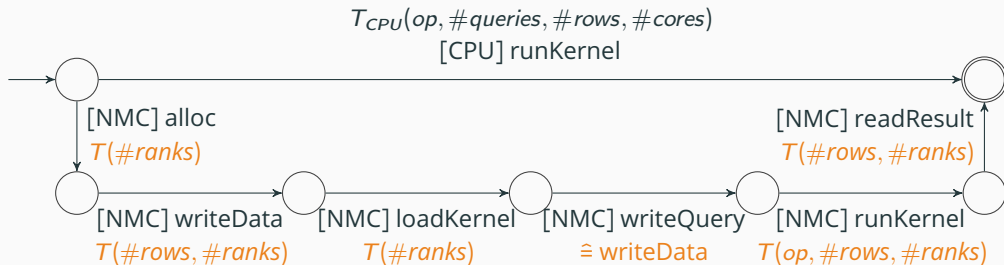
Approach: Performance-Aware Behaviour Models



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

⇒ offload to UPMEM PIM modules

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



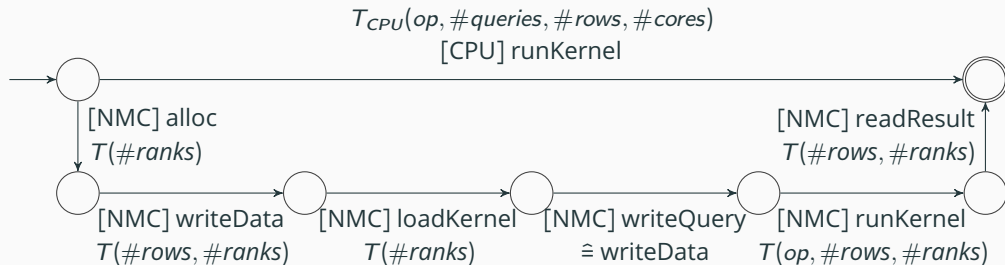
Approach: Performance-Aware Behaviour Models



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

⇒ 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

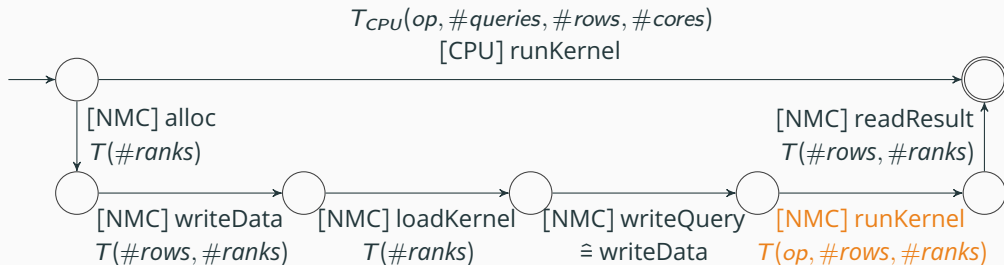
Approach: Performance-Aware Behaviour Models



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

⇒ 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**

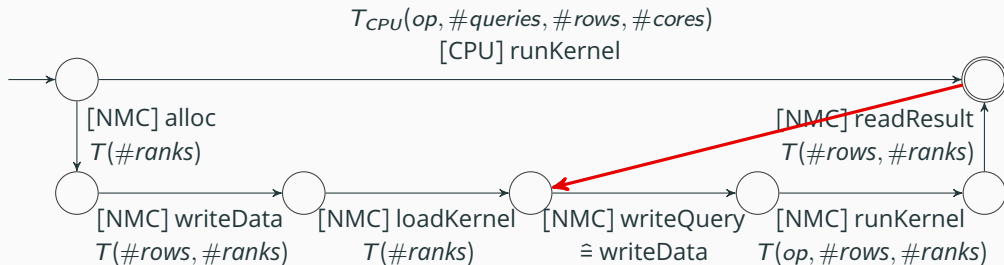
Approach: Performance-Aware Behaviour Models



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

⇒ 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

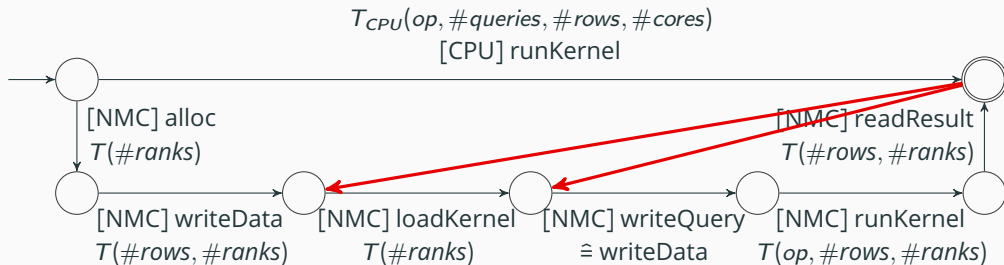
Approach: Performance-Aware Behaviour Models



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

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

CPU: $\#queries \times \text{runKernel}()$

NMC: `alloc(); writeData(); loadKernel();`

$\#queries \times \{ \text{writeQuery}(); \text{runKernel}(); \text{readResult}(); \}$

Performance Models: Kernels



COUNT

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

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

SELECT

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

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

Performance Models: Kernels



COUNT

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

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

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

SELECT

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

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

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

Performance Models: Kernels

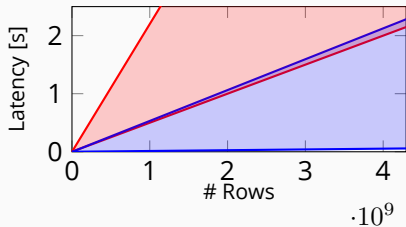


COUNT

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

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

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

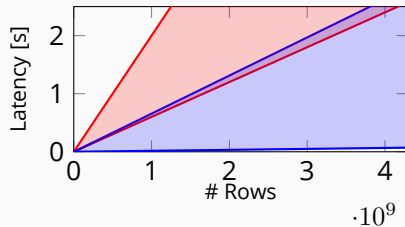


SELECT

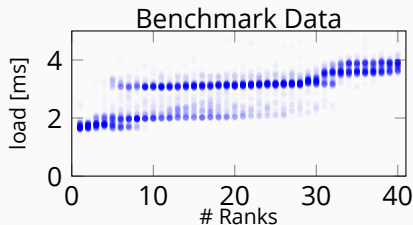
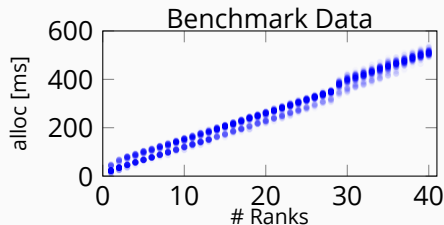
CPU (1C) $0.0 \text{ ns} + 2.0 \text{ ns} \cdot \#rows$

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

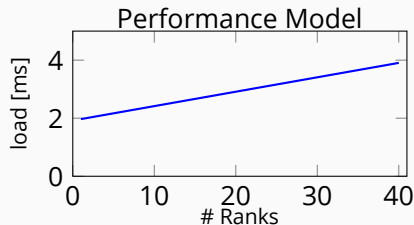
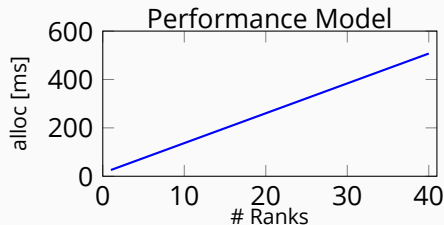
NMC $203 \mu\text{s} + 0.66 \text{ ns} \cdot \frac{\#rows}{\#ranks}$



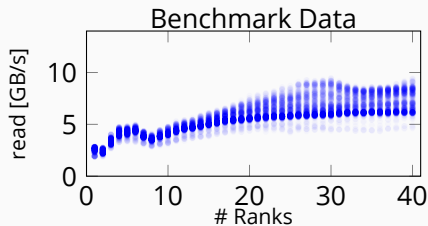
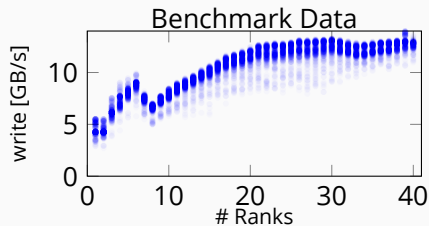
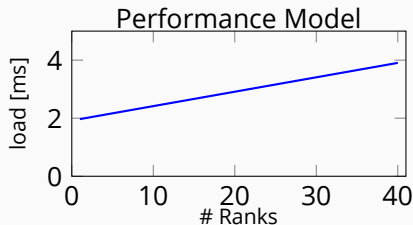
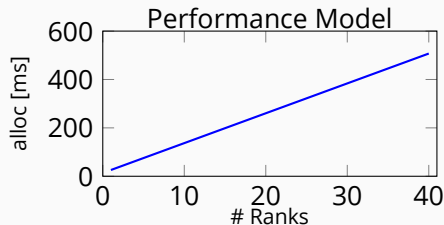
Performance Models: UPMEM Overhead



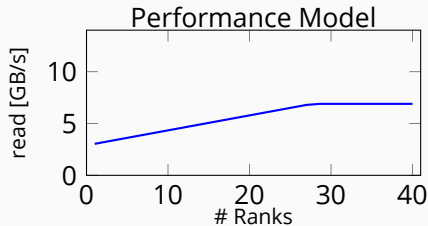
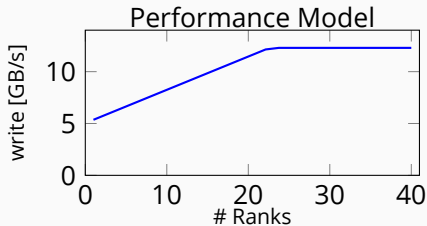
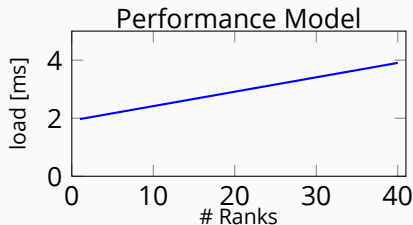
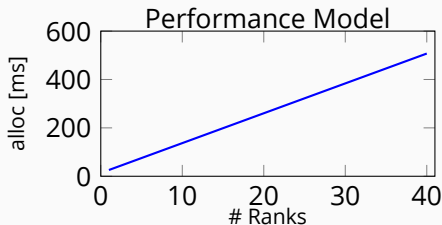
Performance Models: UPMEM Overhead



Performance Models: UPMEM Overhead



Performance Models: UPMEM Overhead



Simulation Results: COUNT (1 CPU Core)



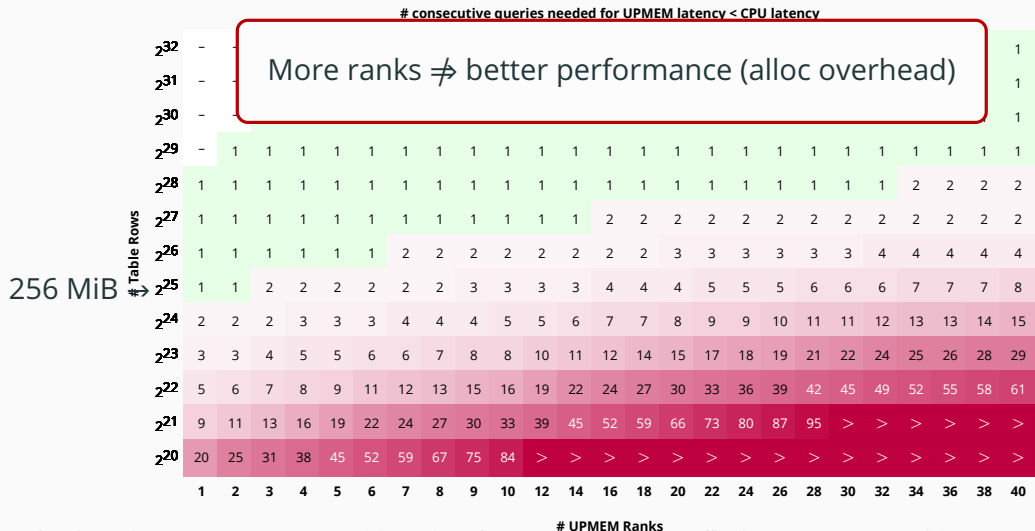
		# consecutive queries needed for UPMEM latency < CPU latency																											
# Table Rows	2 ³²	-	-	-	-	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2 ³¹	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2 ³⁰	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2 ²⁹	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2 ²⁸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
	2 ²⁷	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	2 ²⁶	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	4	4	4	4	4	4
	2 ²⁵	1	1	2	2	2	2	2	2	3	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	7	8	8	8
	2 ²⁴	2	2	2	3	3	3	4	4	4	5	5	6	7	7	8	9	9	10	11	11	12	13	13	14	15	15	15	15
	2 ²³	3	3	4	5	5	6	6	7	8	8	10	11	12	14	15	17	18	19	21	22	24	25	26	28	29	29	29	29
	2 ²²	5	6	7	8	9	11	12	13	15	16	19	22	24	27	30	33	36	39	42	45	49	52	55	58	61	61	61	61
	2 ²¹	9	11	13	16	19	22	24	27	30	33	39	45	52	59	66	73	80	87	95	>	>	>	>	>	>	>	>	>
	2 ²⁰	20	25	31	38	45	52	59	67	75	84	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
			1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	40	40
		# UPMEM Ranks																											

UPMEM faster than CPU for $> 2^{25}$ elements

Table Rows

[illegible]

UPMEM Ranks



Simulation Results: COUNT (4 CPU Cores)



consecutive queries needed for UPMEM latency < CPU latency

# Table Rows	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{32}	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{31}	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{30}	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{29}	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3
2^{28}	7	3	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4
2^{27}	8	3	3	3	3	3	3	3	3	3	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7
2^{26}	9	4	4	4	4	4	4	4	5	5	5	6	7	7	8	8	9	9	10	10	11	12	12	13	13
2^{25}	13	5	5	6	6	6	7	7	8	9	10	11	12	13	14	15	17	18	19	20	21	22	24	25	26
2^{24}	24	9	9	10	10	12	13	14	15	16	18	21	23	26	28	31	33	36	39	41	44	47	50	52	54
2^{23}	50	16	17	18	20	23	25	27	30	33	38	43	49	55	61	67	73	79	86	93	>	>	>	>	>
2^{22}	>	34	34	38	43	48	54	60	66	72	85	>	>	>	>	>	>	>	>	>	>	>	>	>	>
2^{21}	>	92	89	98	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
2^{20}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>

UPMEM Ranks

Simulation Results: COUNT (4 CPU Cores)



consecutive queries needed for UPMEM latency < CPU latency

# Table Rows	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{32}	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{31}	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{30}	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{29}	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{28}	7	3	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4
2^{27}	8	3	3	3	3	3	3	3	3	3	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7
2^{26}	9	4	4	4	4	4	4	4	5	5	5	6	7	7	8	8	9	9	10	10	11	12	12	13	13
2^{25}	13	5	5	6	6	6	7	7	8	9	10	11	12	13	14	15	17	18	19	20	21	22	24	25	26
2^{24}	24	9	9	10	10																		50	52	54
2^{23}	50	16	17	18	20																		>	>	>
2^{22}	>	34	34	38	43																		>	>	>
2^{21}	>	92	89	98	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
2^{20}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>

At least 2 consecutive queries required

Simulation Results: COUNT (4 CPU Cores)



consecutive queries needed for UPMEM latency < CPU latency

# Table Rows	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{32}	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{31}	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{30}	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{29}	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3
2^{28}	7	3	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4
2^{27}	8	3	3	3	3	3	3	3	3	3	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7
2^{26}	9	4	4	4	4	4	4	4	5	5	5	6	7	7	8	8	9	9	10	10	11	12	12	13	13
2^{25}	13	5	5	6	6	6	7	7	8	9	10	11	12	13	14	15	17	18	19	20	21	22	24	25	26
2^{24}	24	9	9	10	10	11	12	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	54
2^{23}	50	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	>
2^{22}	>	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	>
2^{21}	>	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	>
2^{20}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>

More ranks $\nRightarrow$ better performance (alloc overhead)
Less ranks $\nRightarrow$ better performance (kernel latency)

Simulation Results: SELECT (1 CPU Core)



consecutive queries needed for UPMEM latency < CPU latency

# Table Rows	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{32}	-	-	-	-	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{31}	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{30}	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{29}	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{28}	2	2	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2
2^{27}	>	>	>	9	6	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4
2^{26}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	52	29	23	24	26	26	29	30	30
2^{25}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
# UPMEM Ranks	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40

Simulation Results: SELECT (1 CPU Core)



consecutive queries needed for UPMEM latency < CPU latency

# Table Rows	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{32}	-	-	-	-	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{31}	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{30}	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{29}	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{28}	2	2	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2
2^{27}	>	>	>	9	6	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4
2^{26}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	52	29	23	24	26	26	29	30	30
2^{25}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
	# UPMEM Ranks																								

- More complex $\rightarrow \geq 2^{28}$ rows (2 GiB) needed

Simulation Results: SELECT (1 CPU Core)



consecutive queries needed for UPMEM latency < CPU latency

# Table Rows	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{32}	-	-	-	-	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{31}	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{30}	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{29}	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2^{28}	2	2	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2
2^{27}	>	>	>	9	6	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4
2^{26}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	52	29	23	24	26	26	29	30	30
2^{25}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>

UPMEM Ranks

- More complex $\rightarrow \geq 2^{28}$ rows (2 GiB) needed
- Sweet spot for #ranks: neither min nor max ($\hat{=}$ COUNT)

Simulation Results: SELECT (4 CPU Cores)



consecutive queries needed for UPMEM latency < CPU latency

# Table Rows	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{32}	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{31}	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2^{30}	-	-	4	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3
2^{29}	-	>	>	14	8	6	5	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
2^{28}	>	>	>	>	>	>	>	>	>	>	>	58	25	18	14	12	11	11	11	11	11	12	12	13	13
2^{27}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
2^{26}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
2^{25}	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
# UPMEM Ranks	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40

Simulation Results: SELECT (4 CPU Cores)



		# consecutive queries needed for UPMEM latency < CPU latency																											
# Table Rows	2 ³²	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	2 ³¹	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	2 ³⁰	-	-	4	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3
	2 ²⁹	-	>	>	14	8	6	5	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
	2 ²⁸	>	>	>	>	>	>	>	>	>	>	>	58	25	18	14	12	11	11	11	11	11	12	12	13	13	13	13	13
	2 ²⁷	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
	2 ²⁶	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
	2 ²⁵	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>
			1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	40	40
		# UPMEM Ranks																											

- At least 2 consecutive queries required ($\hat{=}$ COUNT)
- Sweet spot for #ranks: neither min nor max ($\hat{=}$ COUNT)

Validation of Simulation Results



COUNT Validation - simulated break-even point vs. real-world break-even point																										
# Table Rows	2 ²⁹	-	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2 ²⁸	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	2 ²⁷	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	-	0	0	-	0
	2 ²⁶	1	1	1	1	1	1	0	0	0	0	0	0	0	-	-	0	-	0	0	-	-	0	0	-	-
		1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
# UPMEM Ranks																										

SELECT Validation – simulated break-even point vs. real-world break-even point																											
# Table Rows	2 ²⁹	-	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2 ²⁸	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0
	2 ²⁷	-	-	-	-7	-5	-3	-3	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-	-2	-2	-	-2	-	-2
		1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	
# UPMEM Ranks																											

Validation of Simulation Results



COUNT Validation - simulated break-even point vs. real-world break-even point

# Table Rows		1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{29}	-	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2^{28}	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
2^{27}	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	-	0	0	-	0	0
2^{26}	1	1	1	1	1	1	0	0	0	0	0	0	0	0	-	-	0	-	0	0	-	-	0	0	-	-

UPMEM Ranks

SELECT Validation - simulated break-even point vs. real-world break-even point

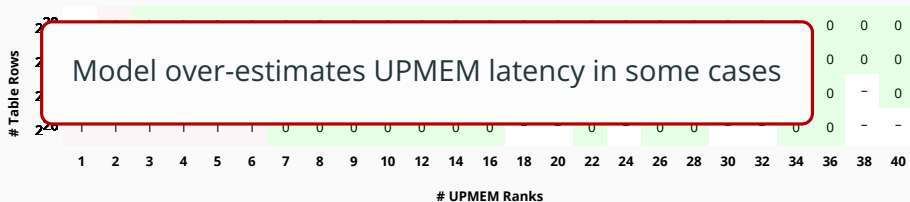
# Table Rows		1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{29}																										0
2^{28}																										0
2^{27}																										-2

UPMEM Ranks

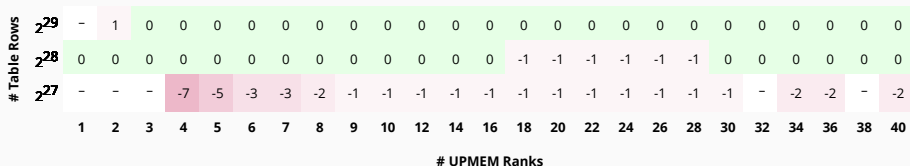
Validation of Simulation Results



COUNT Validation - simulated break-even point vs. real-world break-even point



SELECT Validation - simulated break-even point vs. real-world break-even point



Validation of Simulation Results



COUNT Validation - simulated break-even point vs. real-world break-even point

# Table Rows		1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{29}	-	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2^{28}	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
2^{27}	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	-	0	0	-	0	0
2^{26}	1	1	1	1	1	1	0	0	0	0	0	0	0	0	-	-	0	-	0	0	-	-	0	0	-	-

UPMEM Ranks

SELECT Validation - simulated break-even point vs. real-world break-even point

# Table Rows		1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
2^{29}	-	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2^{28}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0
2^{27}	-	-	-	-7	-5	-3	-3	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-	-2	-2	-	-2

UPMEM Ranks

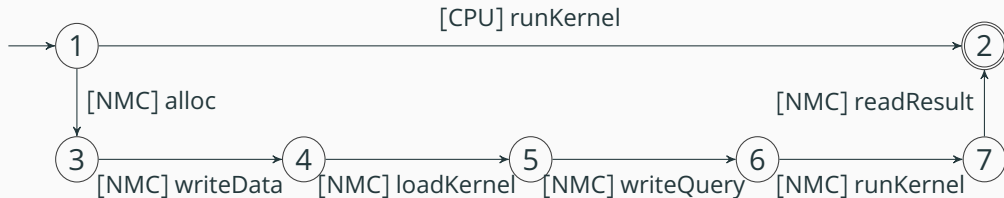
Low error, mostly in range of measurement uncertainty



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



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





- Offloading to UPMEM sensible for
 - simple kernels with a good chance of serving consecutive queries
 - on medium to large database columns (COUNT $\geq$ 256 MiB / SELECT $\geq$ 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 $\rightarrow$ only repeat corresponding benchmarks



- Offloading to UPMEM sensible for
 - simple kernels with a good chance of serving consecutive queries
 - on medium to large database columns (COUNT $\geq$ 256 MiB / SELECT $\geq$ 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 $\rightarrow$ only repeat corresponding benchmarks
- Artifacts and source code referenced in the paper



- [B]S23a] 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.
- [B]S23b] 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.



- [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.