UNIVERSITY OF OREGON

Blocking Optimizations for Sparse MTTKRP

Jee Choi
CIS, University of Oregon

Workshop on Compiler Techniques for Sparse Tensor Algebra
Cambridge, MA, January 26th, 2019

MTTKRP operation is expensive

Matricized Tensor Times Khatri-Rao Product

```
procedure CP-ALS (X, R)
      repeat

            C = X(3)(B ⊙ A)(BᵀB * AᵀA)^†
            normalize columns of C to length 1
            B = X(2)(C ⊙ A)(CᵀC * AᵀA)^†
            normalize columns of B to length 1
            A = X(1)(C ⊙ B)(CᵀC * BᵀB)^†
            store column norms of A in λ and normalize to 1

      until max iteration reached or error less than ϵ

end procedure
```

MTTKRP operation is expensive

> 90% total execution time

```
procedure CP-ALS (X, R)
      repeat

            C = X(3)(B ⊙ A)(BᵀB * AᵀA)†
            normalize columns of C to length 1
            B = X(2)(C ⊙ A)(CᵀC * AᵀA)†
            normalize columns of B to length 1
            A = X(1)(C ⊙ B)(CᵀC * BᵀB)†
            store column norms of A in λ and normalize to 1

      until max iteration reached or error less than ϵ

end procedure
```

How data is accessed for each non-zero in the tensor

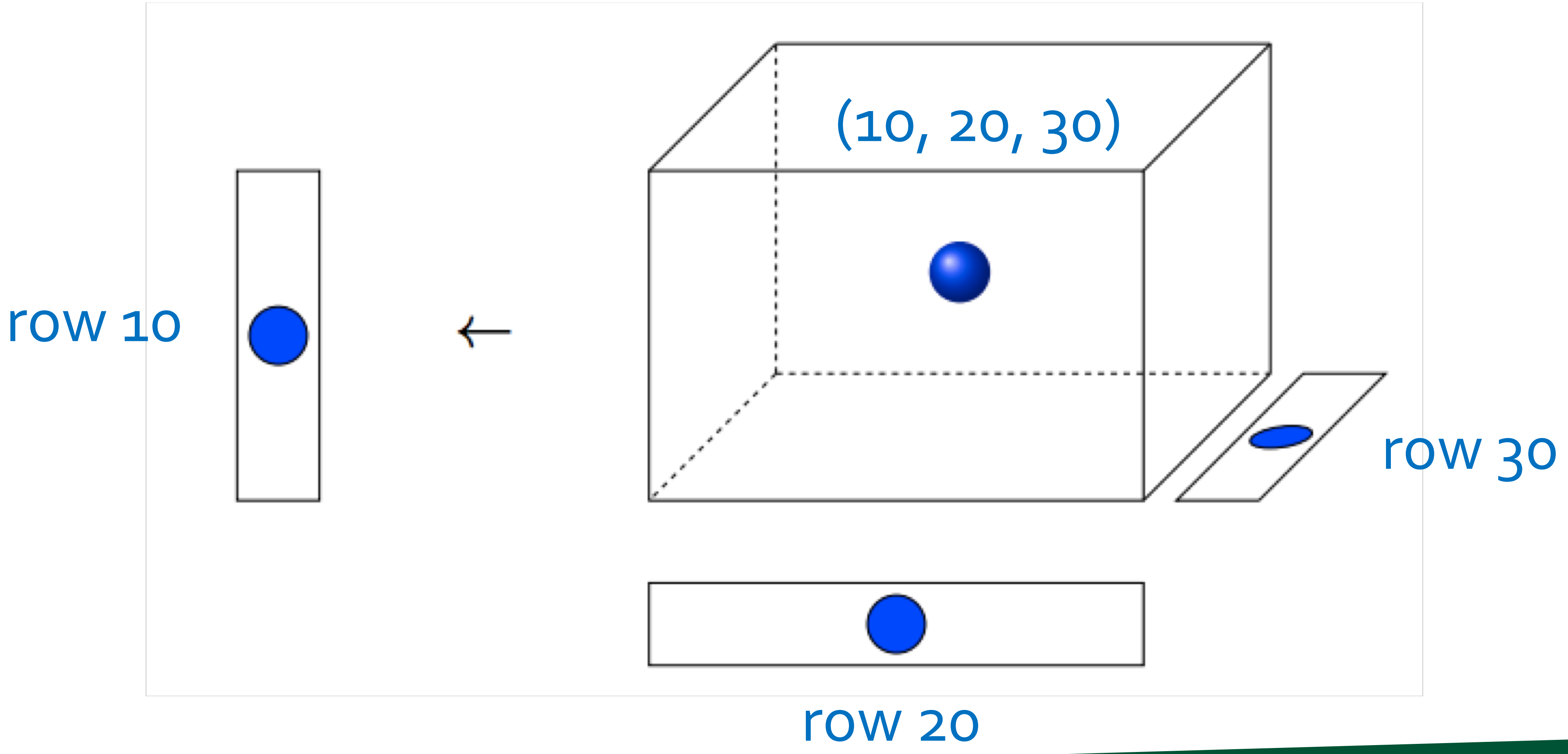
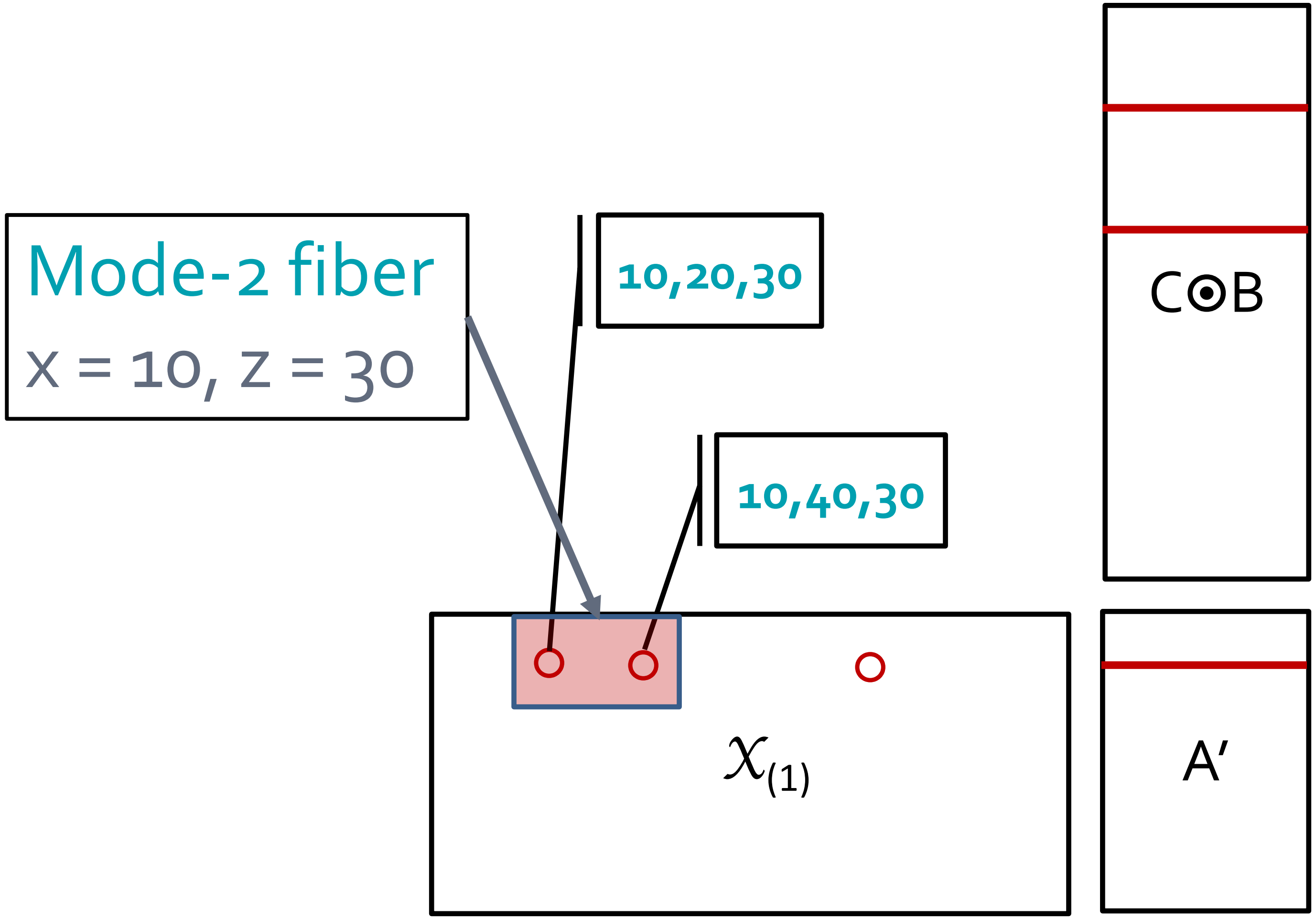
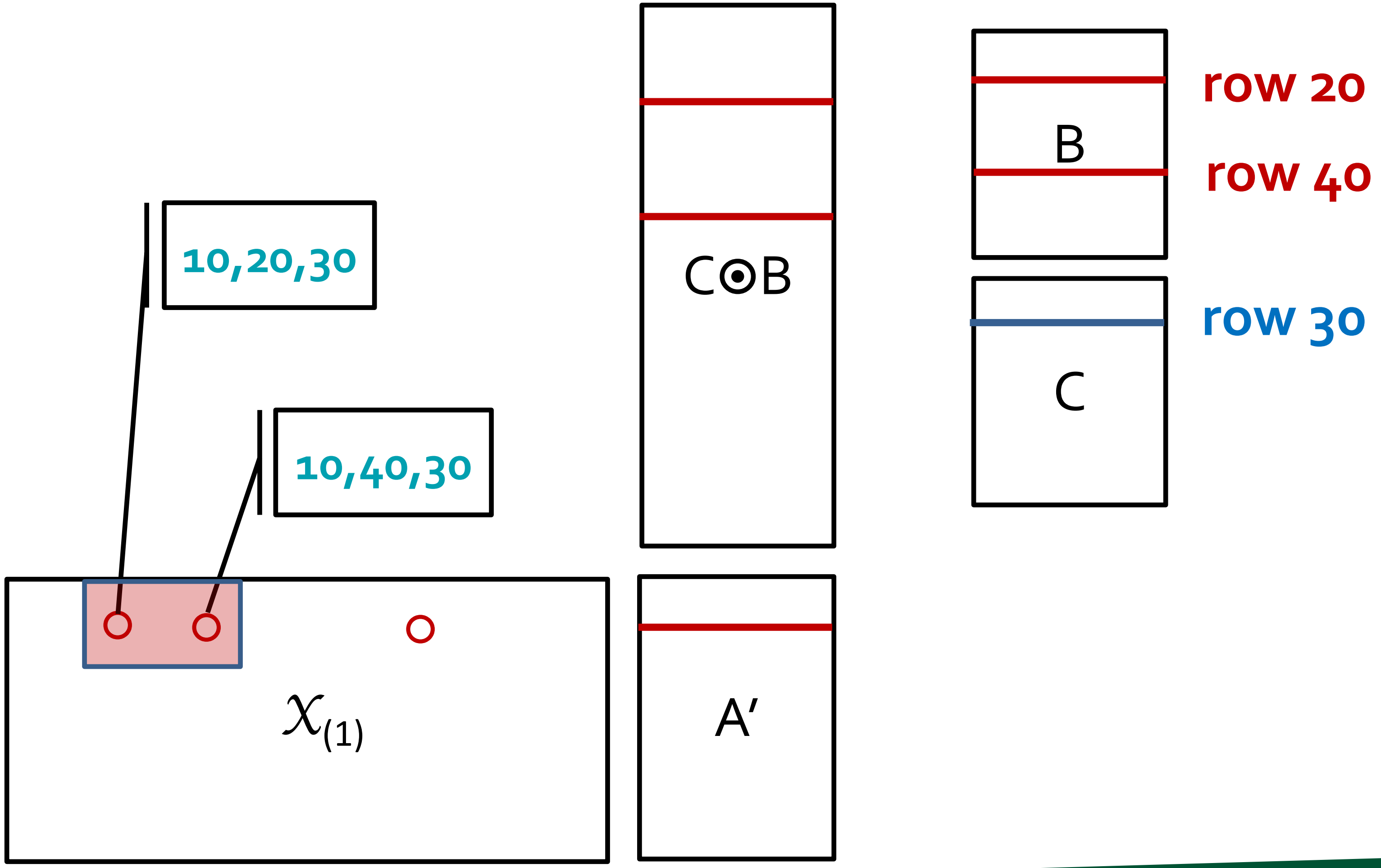


Image source Shaden Smith, et al., ***SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication***, *IPDPS 2015*

Reduce computation by processing fibers (CSF)



Reduce computation by processing fibers (CSF)

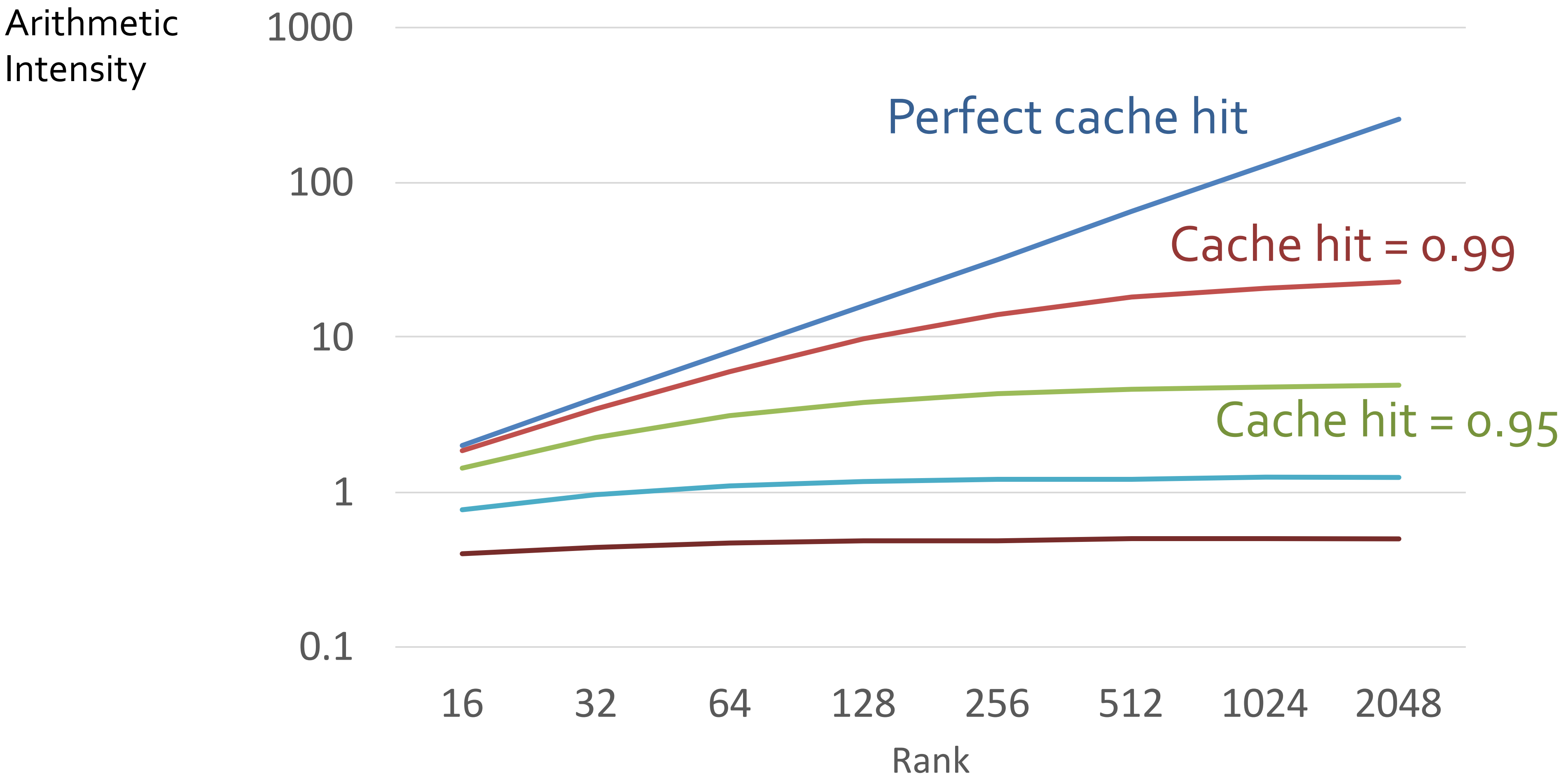


Roofline model applied to CSF MTTKRP

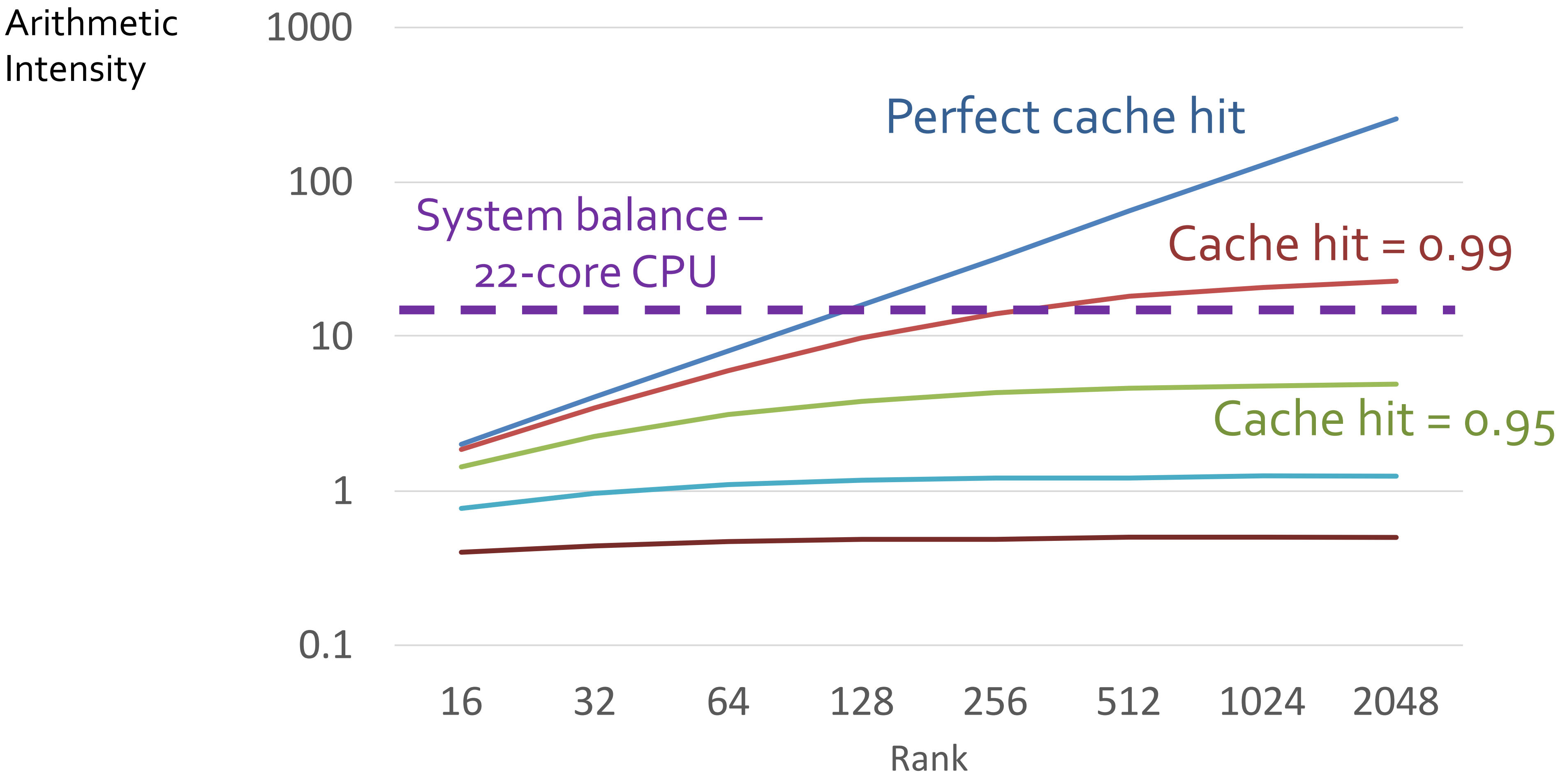
- Let's calculate the # of flops and # of bytes and compare
  - Flops: W = **2R(m + P)**
  - Data: Q = **2m** (value + mode-2 index) + **2P** (mode-3 index + mode-3 pointer) + **(1-α)Rm** (mode-2 factor) + **(1-α)RP** (mode-3 factor)
- Arithmetic Intensity
  - Ratio of work to communication **I = W/Q**
  - I = W / (Q * 8 Bytes) = **R / (8 + 4R(1-α))**

m = # of nonzeros
P = # of non-empty fibers
R = rank
α = cache hit rate

Arithmetic intensity vs. system balance (on the latest CPU)

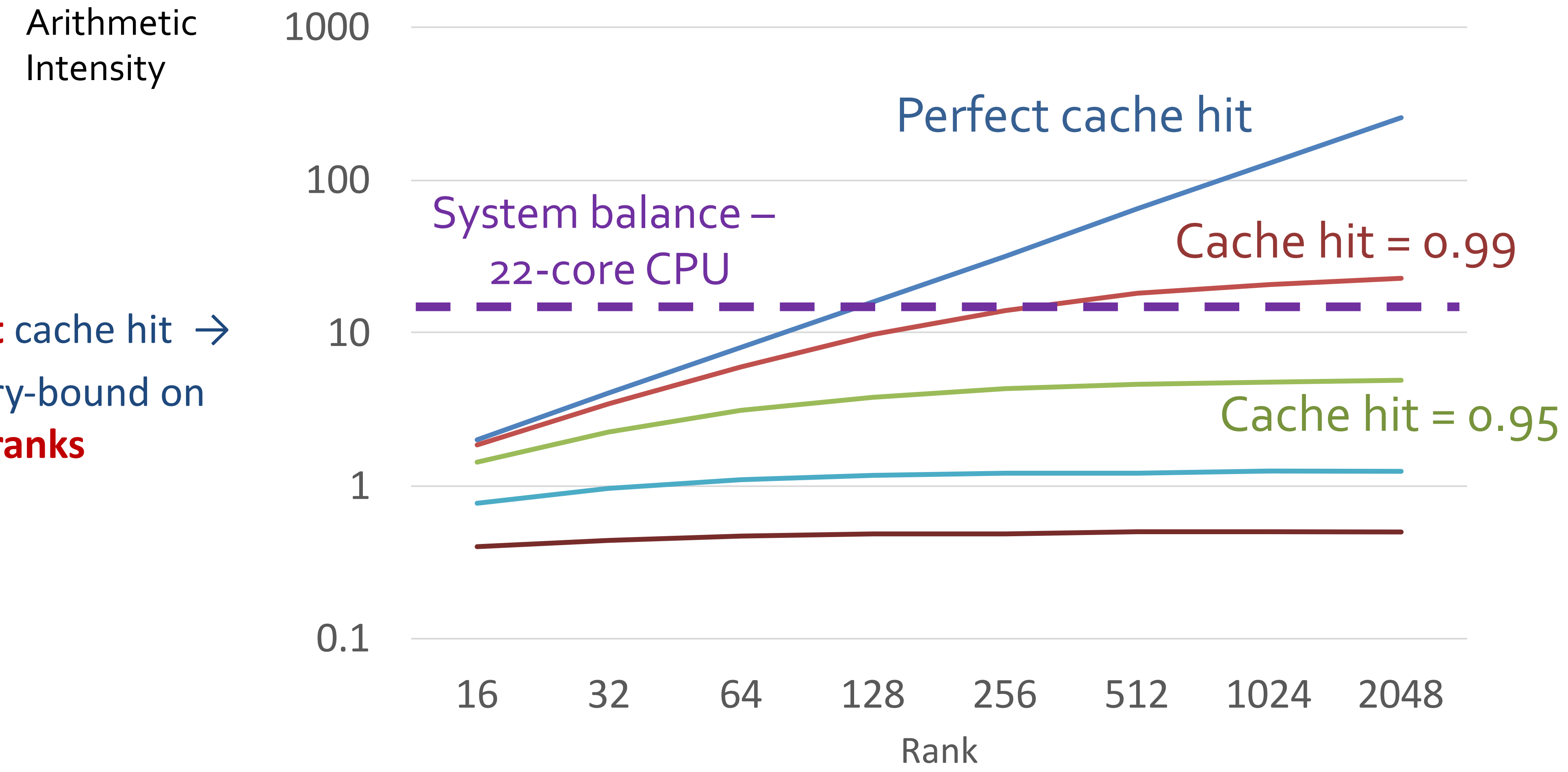


Arithmetic intensity vs. system balance (on the latest CPU)

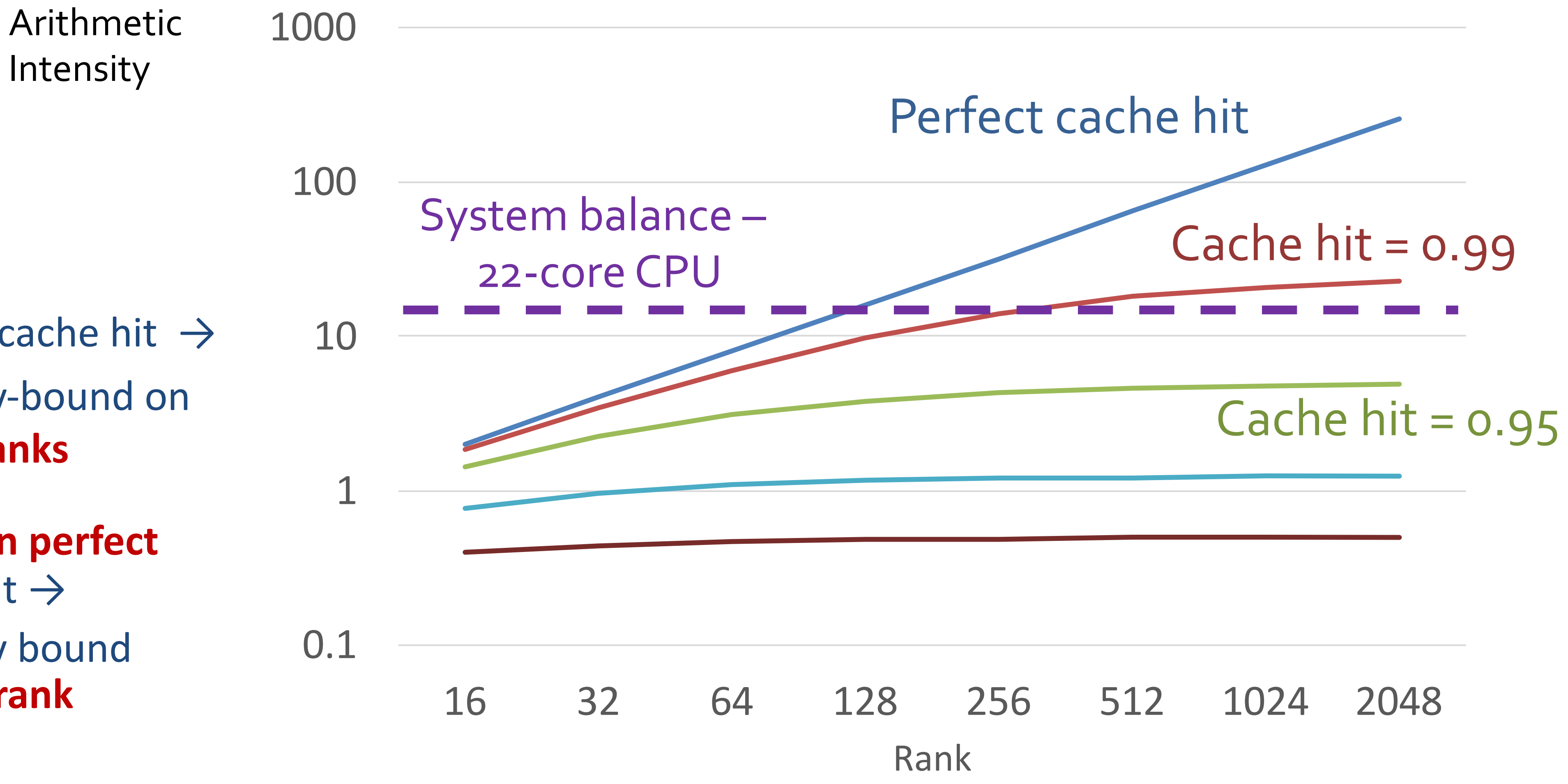


Arithmetic intensity vs. system balance (on the latest CPU)

perfect cache hit →
memory-bound on
lower ranks



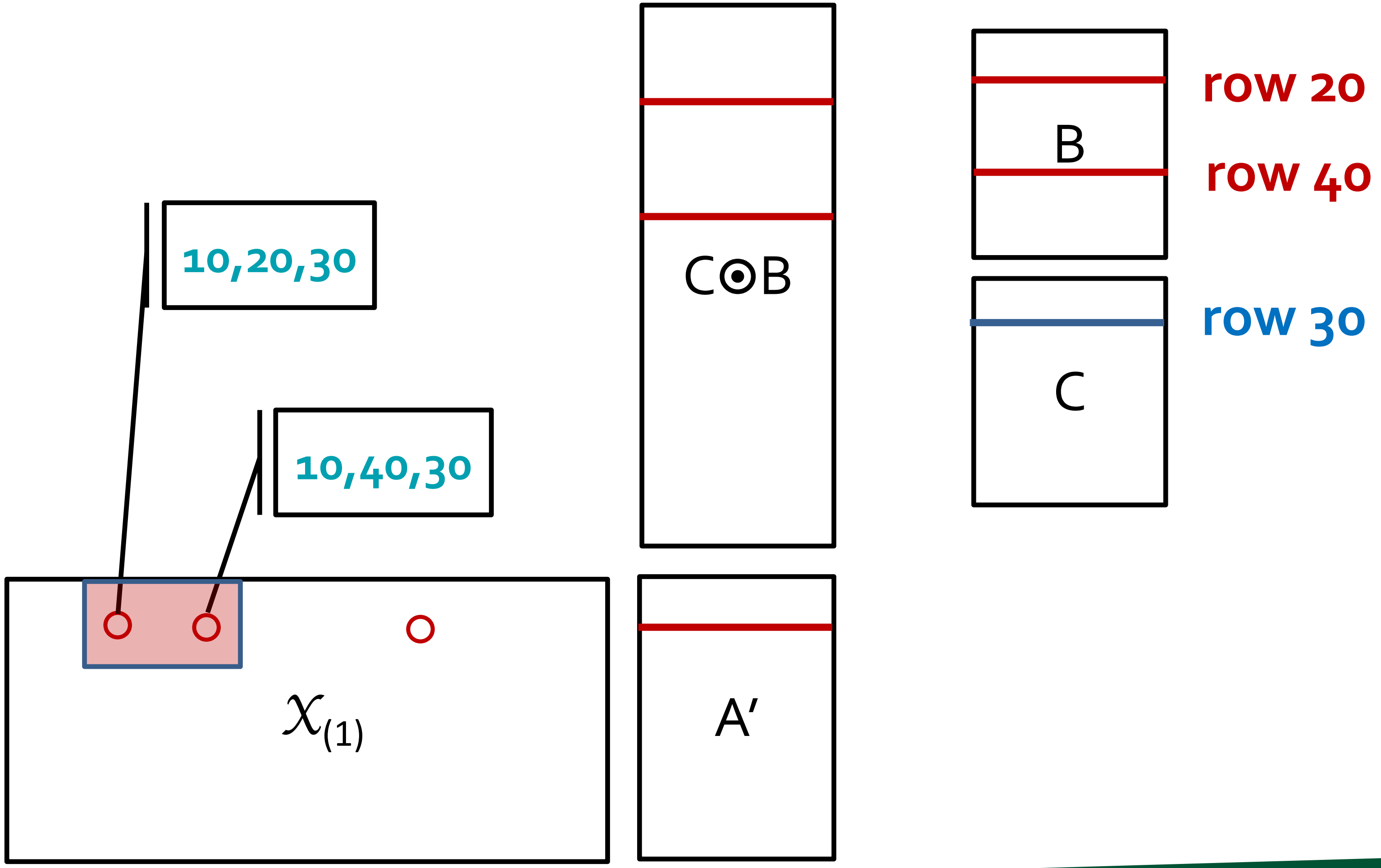
Arithmetic intensity vs. system balance (on the latest CPU)



perfect cache hit → memory-bound on **lower ranks**

less than perfect cache hit → memory bound for **any rank**

Reduce computation by processing fibers (CSF)



A pressure point analysis reveals the bottleneck

Time	Pressure point
2.6s	Baseline (2R(m + P) flops)

Using COO instead of CSF only increases exec. time by < 2%

Time	Pressure point
2.6s	Baseline (2R(m + P) flops)
2.64s	Move flops to inner loop (3 * m * R flops)

Increasing flops only changes time by < 2%

Removing access to C: exec. time down by 7%

Time	Pressure point
2.6s	Baseline (2R(m + P) flops)
2.64s	Move flops to inner loop (3 * m * R flops)
2.43s	Access to C removed

Removing per-fiber access to matrix C has a bigger impact than increasing flops

Memory access to B is the primary bottleneck

Time	Pressure point
2.6s	Baseline (2R(m + P) flops)
2.64s	Move flops to inner loop (3 * m * R flops)
2.43s	Access to C removed
1.81s	Access to B limited to L1 cache

Eliminating our suspect has a huge impact

Completely removing it give us an extra 6% - why?

Time	Pressure point
2.6s	Baseline (2R(m + P) flops)
2.64s	Move flops to inner loop (3 * m * R flops)
2.43s	Access to C removed
1.81s	Access to B limited to L1 cache
1.63s	Access to B removed completely

Unexplained 6%
decrease in exec. time

Conclusions from our empirical analysis

- Flops aren't the issue
- Bottlenecks
  1. Data access to factor matrix B (and not the tensor, e.g., SpMV)
  2. Load instructions (why previous attempt at cache blocking was not successful)

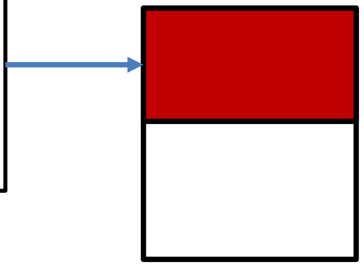
Cache/register blocking should help alleviate these bottlenecks

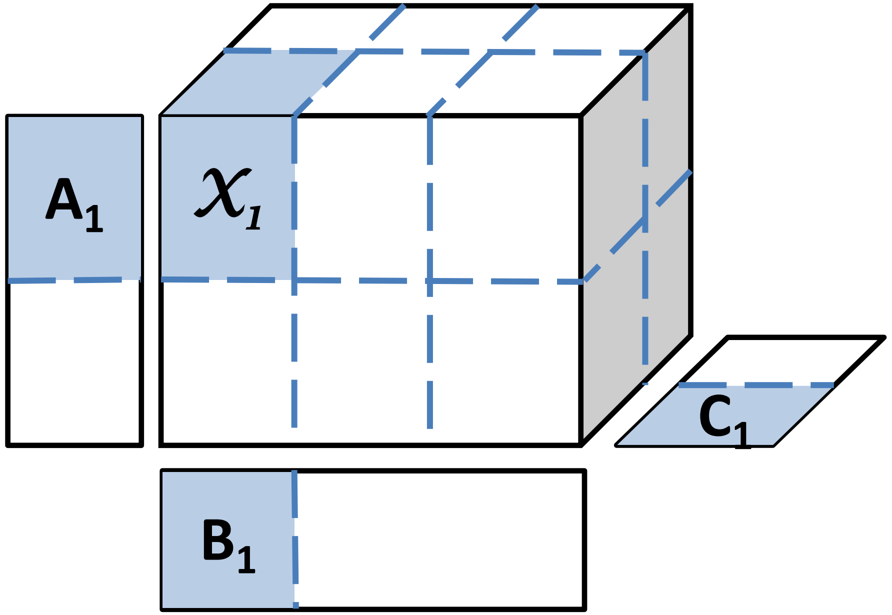
- Flops aren't the issue
- Bottlenecks
  1. Data access to factor matrix B → cache blocking
  2. Load instructions → register blocking

We use n-D blocking and rank blocking

- Multi-dimensional blocking
  - 3D blocking – maximize re-use of both matrix B and C
  - Multiple access to the factor matrices

Make sure this fits in the LLC

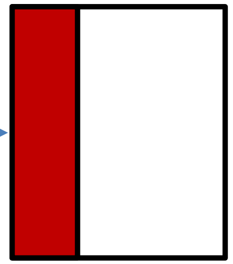




We use n-D blocking and rank blocking

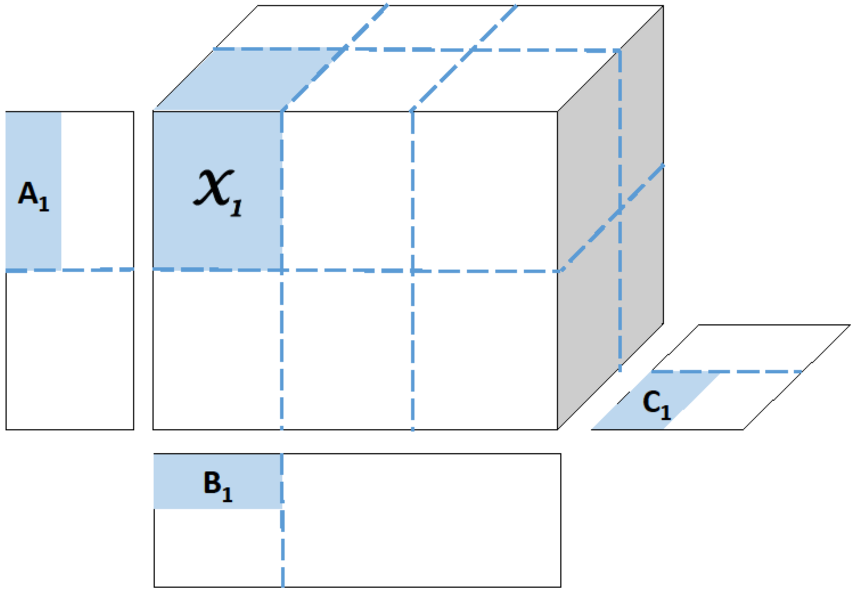
- Multi-dimensional blocking
  - 3D blocking – maximize re-use of both matrix B and C
  - Multiple access to the factor matrices
- Rank blocking
  - Agnostic to tensor sparsity
  - Similar to register blocking
  - Tensor replication

Increase the **chance** of finding rows in cache



We can combine n-D blocking with rank blocking

- Multi-dimensional + rank blocking
  - Partial replication
  - "Best of both worlds" re-use
  - Even more repeated accesses to tensor/factor

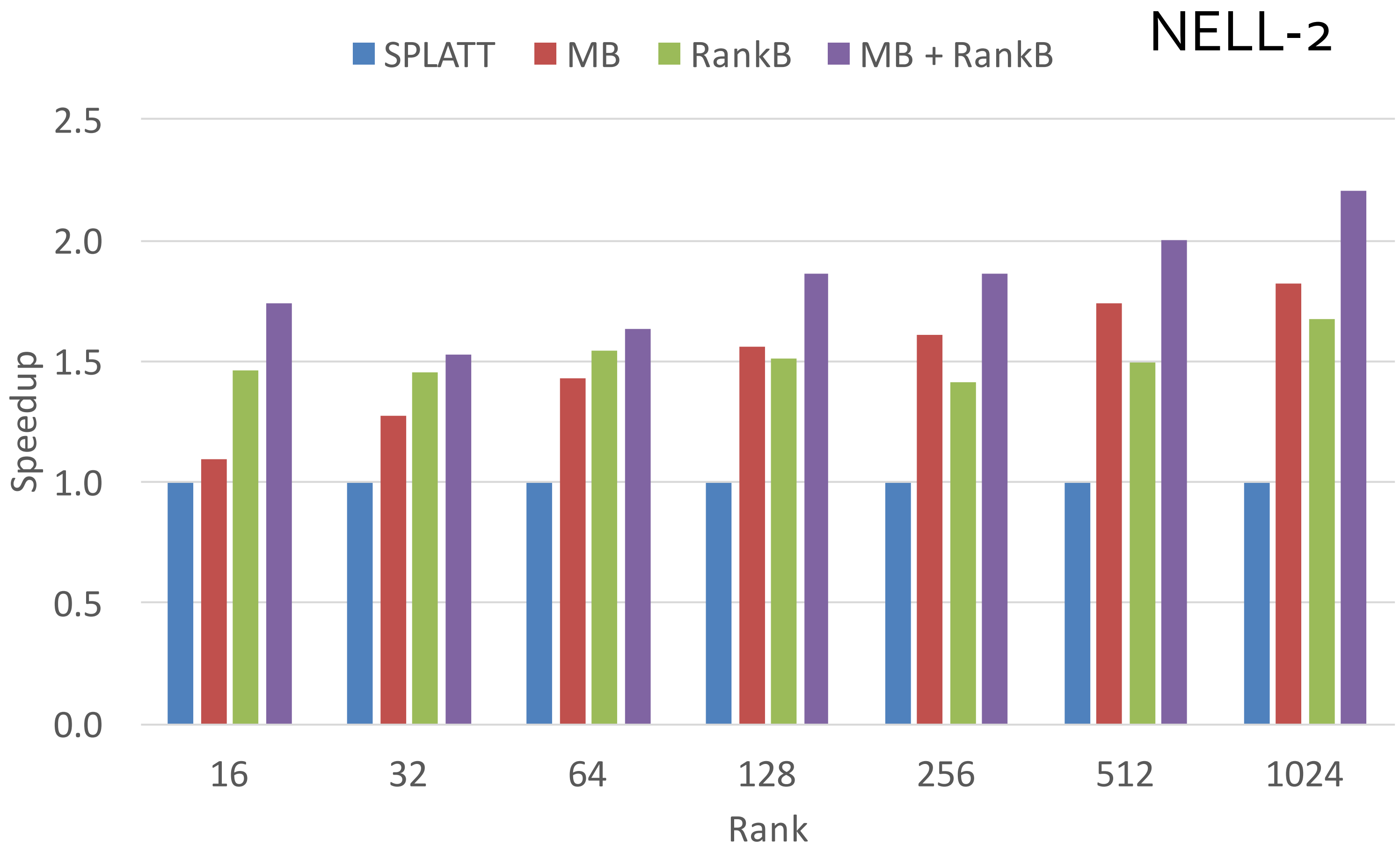


Performance **summary** for single node

Data set	Dimensions	nnz	Sparsity	Speedup
Poisson2	2K×16K×2K	121M	1.9e-3	2.0×
Poisson3	30K×30K×30K	135M	5.0e-6	1.7×
Netflix	480K×18K×80	80M	1.2e-4	3.1×
NELL-2	12K×9K×29K	77M	2.4e-5	2.2×
Reddit	1.2M×23K×1.3M	924M	2.8e-8	2.1×
Amazon	4.8M×1.8M×1.8M	1.7B	2.5e-8	3.5×

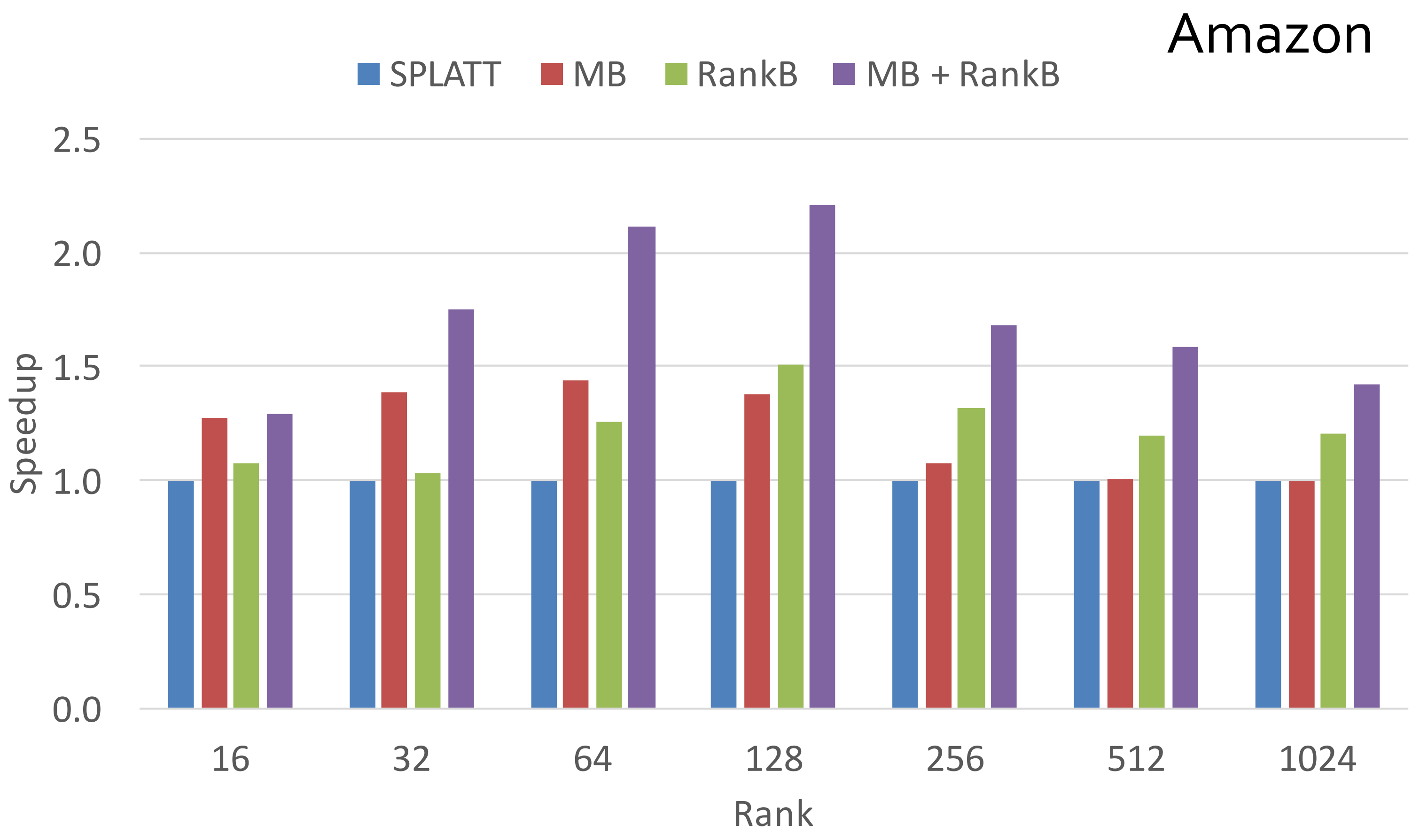
For small tensors, blocking becomes more effective at higher rank sizes

- With small dimension sizes, there is already good cache re-use without explicit blocking
- Only when rank size is large enough, do we see significant benefit from blocking



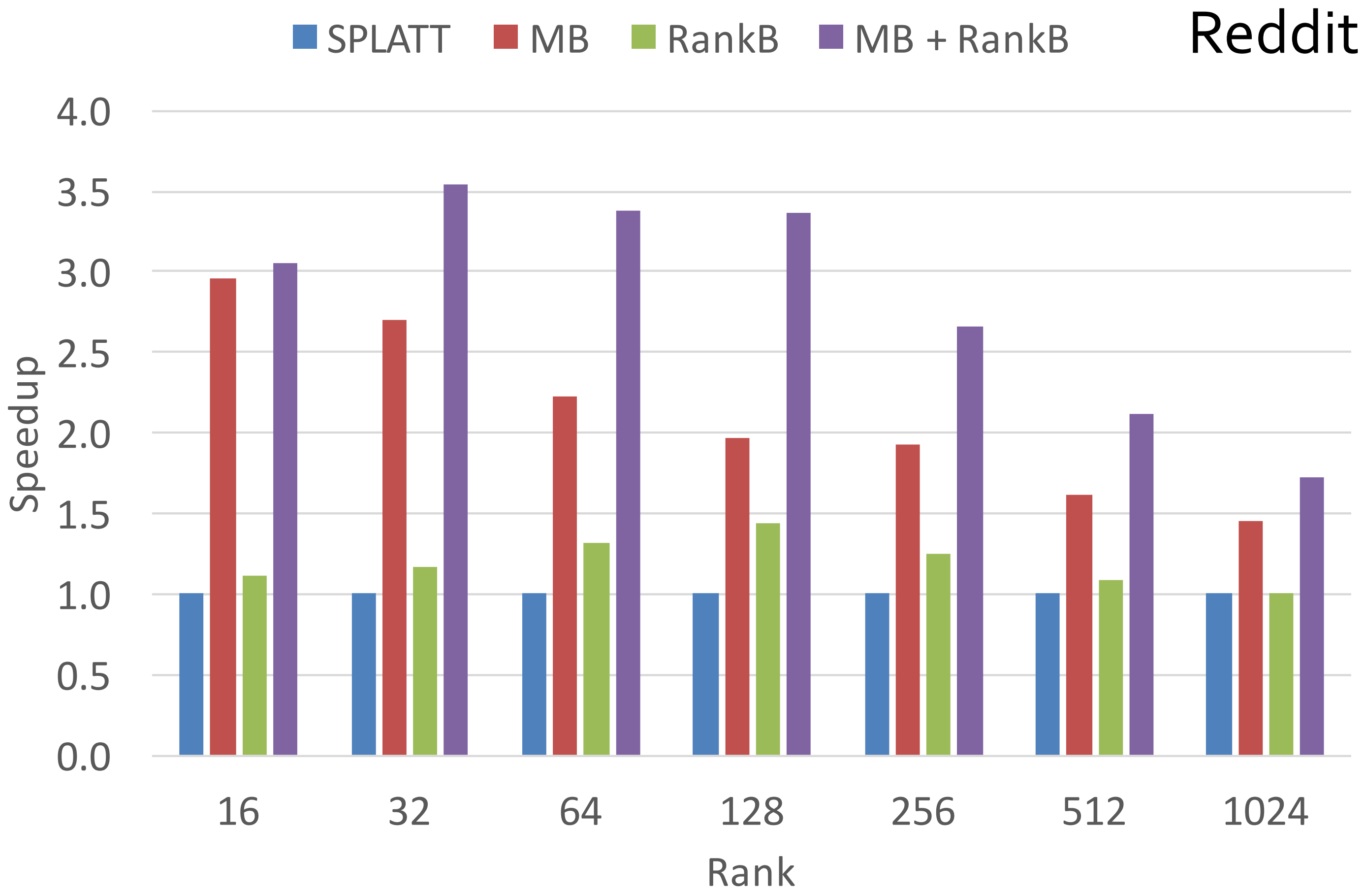
For large tensors, blocking becomes less effective at higher ranks

- With large dimension sizes **and** large ranks, data sets are so big large number of blocks are required, and the **overhead** of blocking outweighs the benefit



More potential benefit from blocking with real data sets

- Real data sets have clustering patterns which lead to higher speedups from blocking
- Combining rank blocking with n-D blocking yields the highest speedup

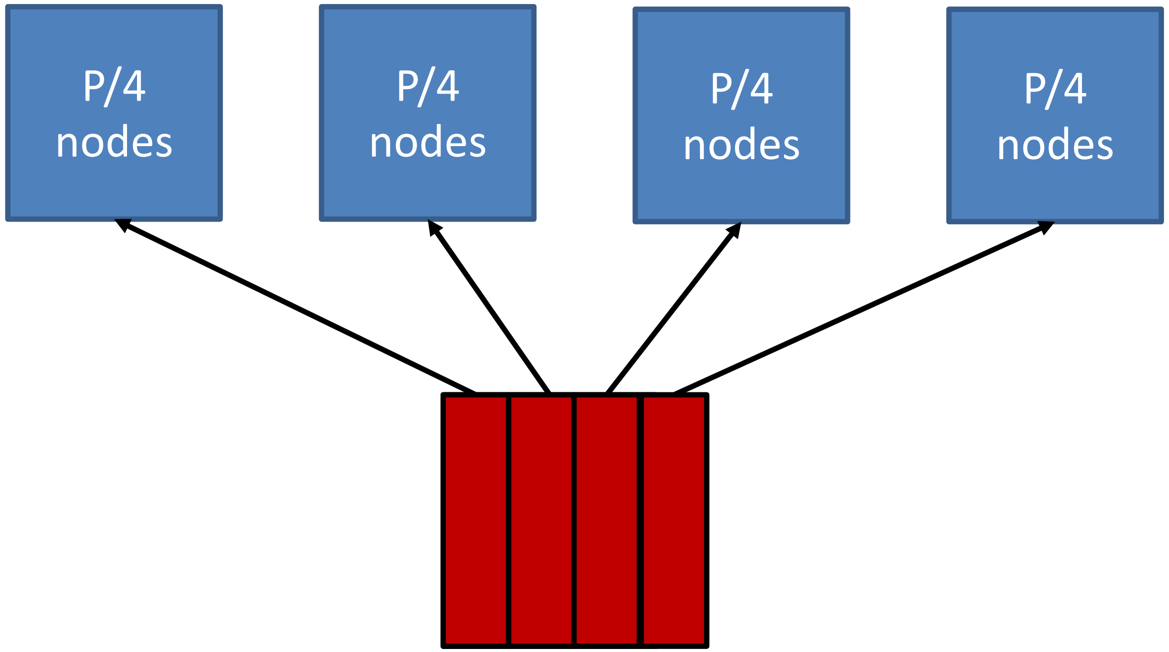


Rank blocking on distributed systems

- Strong scalability problems with traditional partitioning
  - Fewer non-zero per node -> lower efficiency & higher comm. cost -> poor scalability

Rank blocking on distributed systems

- Scalability problems
  - Fewer non-zero per node -> lower efficiency & higher comm. cost -> poor scalability
- Rank blocking
  - No comm. between proc. sets
  - Tensor replication



Rank blocking on distributed systems

Nodes	NELL2					Netflix				
	SPLATT	3D grid	3D time	4D grid	4D time	SPLATT	3D grid	3D time	4D grid	4D time
1	1.028	1x1x2	0.718	1x1x1x2	0.826	3.025	2x1x1	1.554	1x1x1x2	1.447
2	0.540	1x1x4	0.367	1x1x1x4	0.423	1.158	4x1x1	0.727	1x1x1x4	0.720
4	0.286	2x1x4	0.208	1x1x1x8	0.217	0.519	8x1x1	0.403	1x1x1x8	0.401
8	0.138	2x2x4	0.107	1x1x1x16	0.124	0.256	16x1x1	0.194	1x1x1x16	0.190
16	0.087	2x2x8	0.058	1x1x2x16	0.065	0.113	32x1x1	0.103	2x1x1x16	0.100
32	0.056	4x2x8	0.043	1x1x4x16	0.034	0.083	32x2x1	0.056	4x1x1x16	0.055
64	0.030	4x4x8	0.028	2x1x4x16	0.022	0.048	64x2x1	0.037	8x1x1x16	0.030