Sparse Tensor Factorization on Many-Core Processors with High-Bandwidth Memory

Shaden Smith[1*], Jongsoo Park[2], and George Karypis[1]

[1]Department of Computer Science & Engineering, University of Minnesota
[2]Parallel Computing Lab, Intel Corporation
[*]shaden@cs.umn.edu

Outline

Table of Contents

Introduction

Many applications today are *data intensive*:

- Irregular memory accesses.
- Non-uniform work distributions.
- High memory footprints.
- High memory bandwidth demands.

HPC systems are turning to *many-core processors*:

- Hundreds of concurrent threads.
- Emerging architectures feature high-bandwidth memory:
  - Intel Xeon Phi (Knights Landing – KNL)
  - NVIDIA Pascal
  - AMD Fiji

Many-core challenges

We must re-evaluate our algorithms for emerging architectures:

- Require $\approx 10\times$ more parallelism without $10\times$ more memory.
- Fine-grained synchronization is often expensive.
- Vectorization is essential for performance.
- How to best utilize the high-bandwidth memory?

We evaluate some of the above design directions via:

- KNL is our target many-core processor.
- Tensor decomposition is our data intensive application.

Intel Knights Landing (KNL)

- Up to 288 concurrent threads (72 cores $\times$ 4-way SMT)
- 16GB of on-package MCDRAM
  - $\approx$ 480GB/s memory bandwidth
  - Either managed explicitly or treated as an LLC.

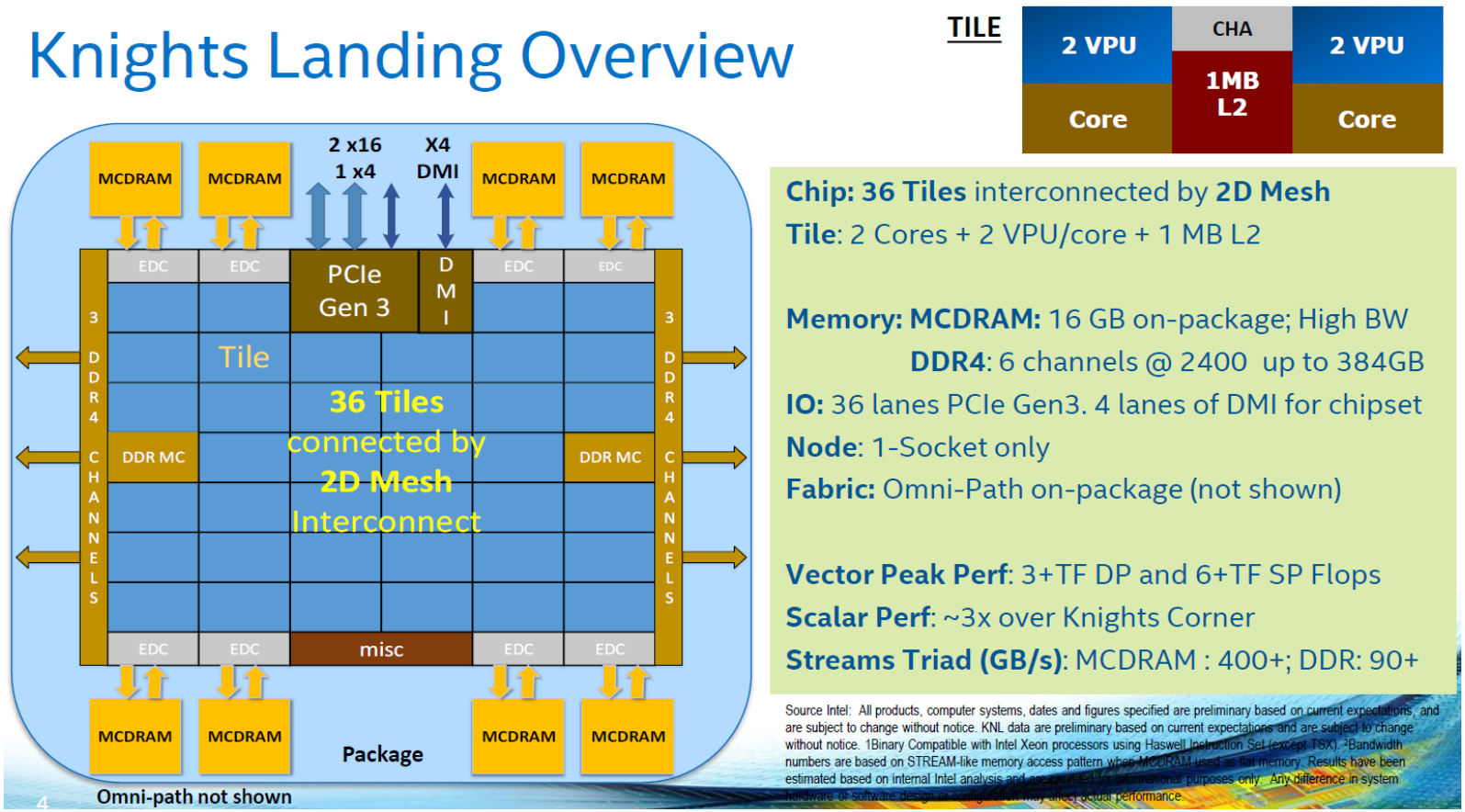
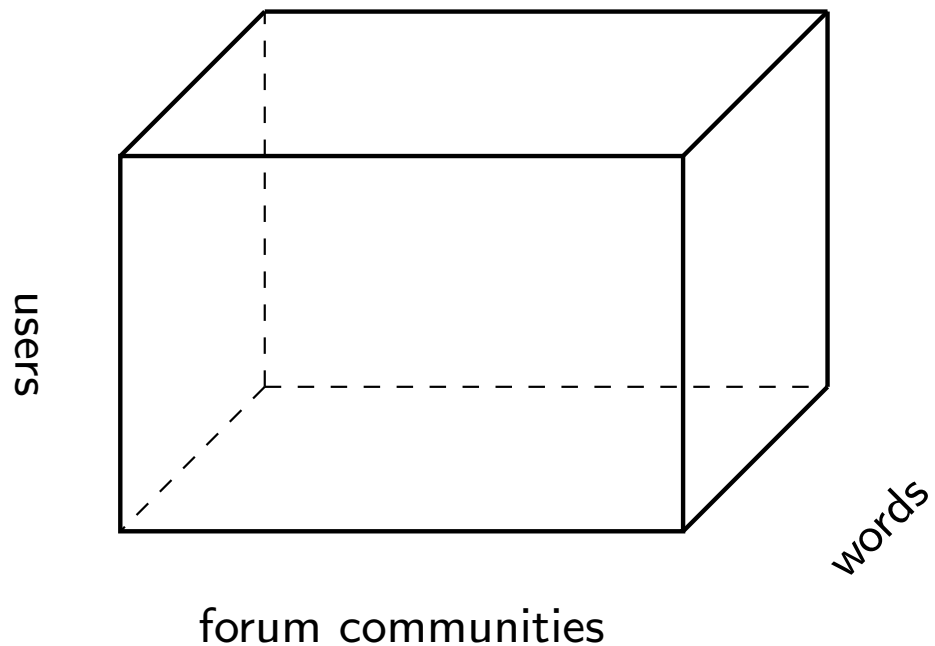


Table of Contents

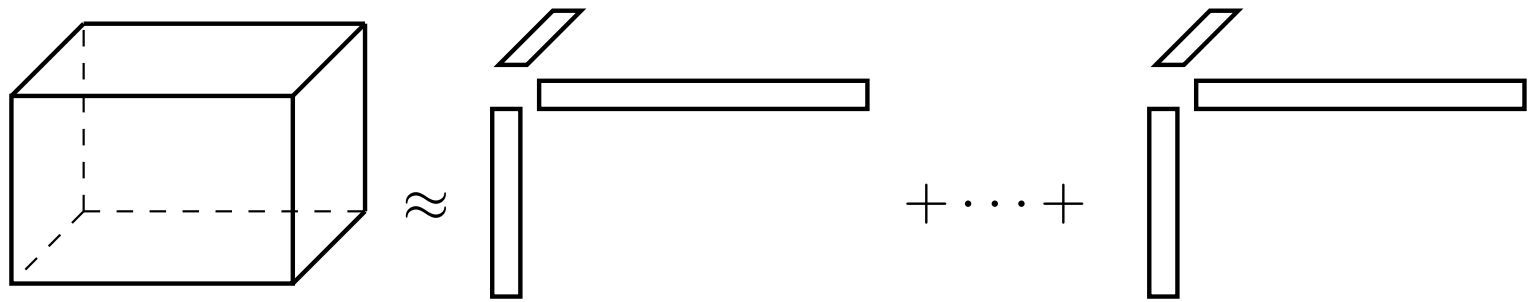
Tensors

- Tensors are the generalization of matrices to higher dimensions.
- Allow us to represent and analyze multi-dimensional data.
- Applications in precision healthcare, cybersecurity, recommender systems, ...



Canonical polyadic decomposition (CPD)

- The CPD models a tensor as the summation of rank-1 tensors.
  - A rank-1 tensor is the outer product of m vectors.



$$\boldsymbol{\mathcal{X}}(i,j,k) \approx \sum_{f=1}^{F} \mathbf{A}(i,f) \times \mathbf{B}(j,f) \times \mathbf{C}(k,f)$$

Notation

- $\mathbf{A}, \mathbf{B}, \mathbf{C}$, each with F columns, will be used to denote the factor matrices for a 3D tensor.

Alternating least squares (ALS)

ALS cyclically updates one factor matrix at a time while holding all others constant.

Algorithm 1 CPD-ALS

1: **while** not converged **do**

2: $\quad \mathbf{A}^T \leftarrow (\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1} \left(\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})\right)^T$

3: $\quad \mathbf{B}^T \leftarrow (\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^{-1} \left(\mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})\right)^T$

4: $\quad \mathbf{C}^T \leftarrow \underbrace{(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^{-1}}_{\text{Normal equations}} \underbrace{\left(\mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})\right)^T}_{\text{MTTKRP}}$

5: **end while**

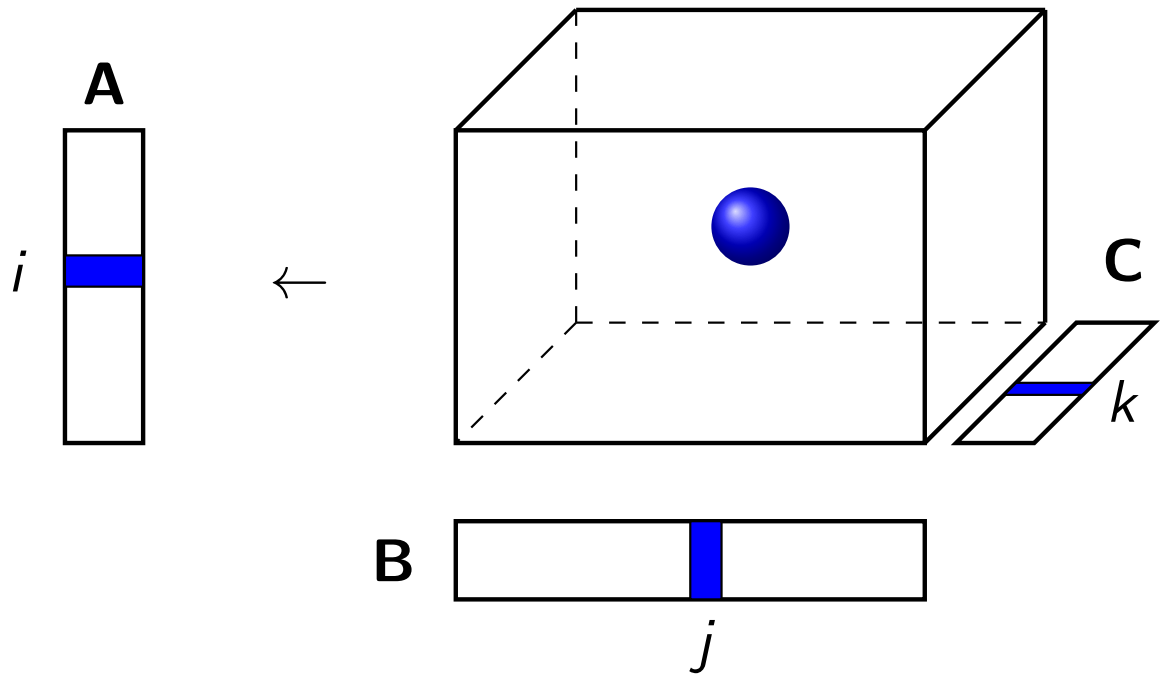
Notation

$*$ denotes the Hadamard (elementwise) product.

MTTKRP – elementwise

Elementwise formulation:

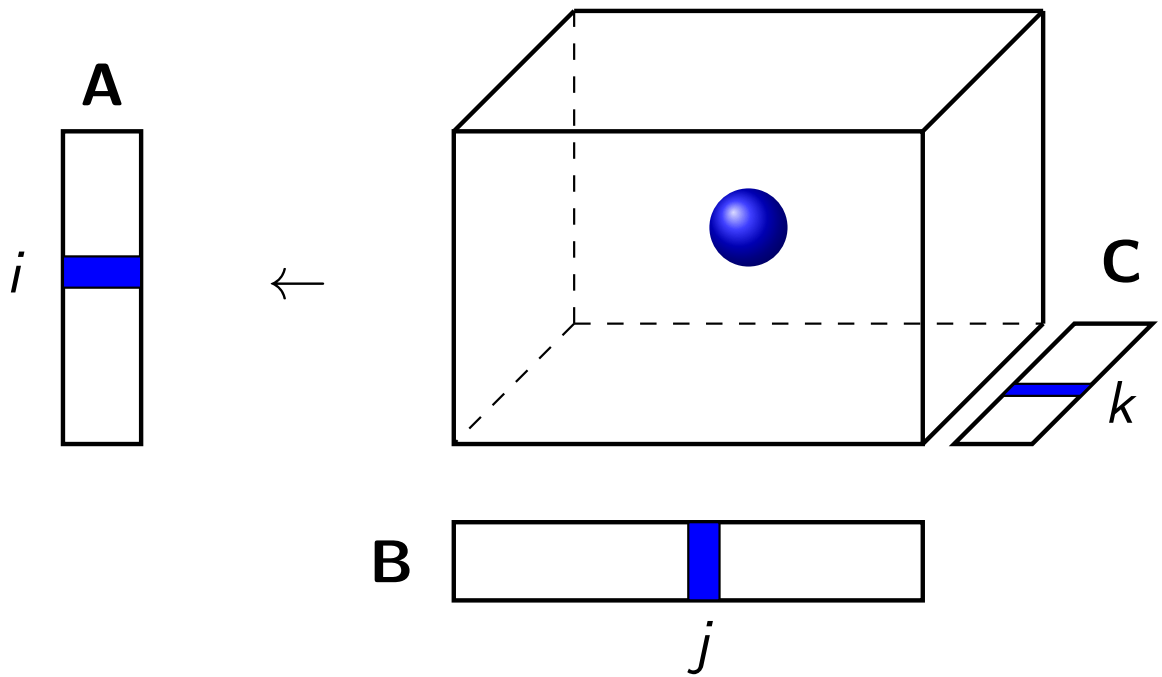
$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \boldsymbol{\mathcal{X}}(i,j,k)\,[\mathbf{B}(j,:) * \mathbf{C}(k,:)]$$



Disclaimer
This is a simplification of how MTTKRP is implemented.

MTTKRP – Parallelism

- Each *slice* of non-zeros can be processed independently.
  - Great, if we store three copies of our tensor ordered by slice.
  - Otherwise we must synchronize on $\mathbf{A}(i,:)$, $\mathbf{B}(j,:)$, etc.



Parallelism – tiling

When we cannot afford additional tensor representations:

- For p threads, do a p-way tiling of each tensor mode.
- Distributing the tiles allows us to eliminate the need for mutexes.

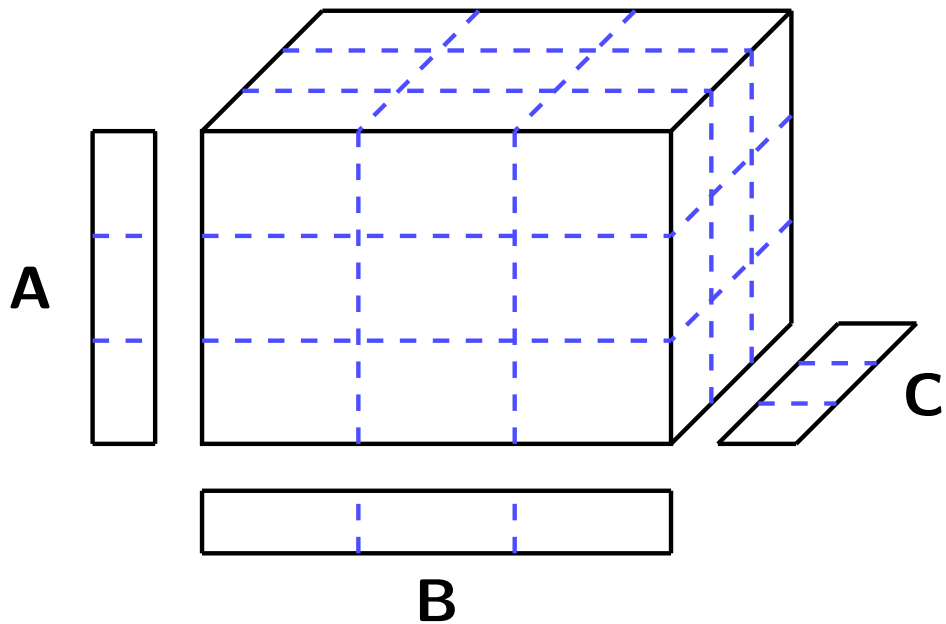


Table of Contents

Scalability challenges

Sparse tensors inherit the scalability challenges of sparse matrices:

- Unstructured sparsity patterns.
  - Fine-grained synchronizations and atomics.
- Non-uniform work distributions.
  - *Hub slices* prevent load balanced coarse-grained parallelism.

Tensors also bring unique challenges:

- Mode-centric computations.
  - We cannot always afford to optimize data structures for every mode.
  - p-way tiling for higher-order tensors is not practical.
- Mode lengths are highly variable.
  - We may have 1M users but only 5 purchase contexts.

Decomposing Hub Slices

Skewed non-zero distribution can result in load imbalance.

Example:

- A tensor of Amazon product reviews contains a slice with 6.5% of the total non-zeros.
- A 1D decomposition cannot be load balanced with more than 16 threads.

Decomposing Hub Slices

Skewed non-zero distribution can result in load imbalance.

Example:

- A tensor of Amazon product reviews contains a slice with 6.5% of the total non-zeros.
- A 1D decomposition cannot be load balanced with more than 16 threads.

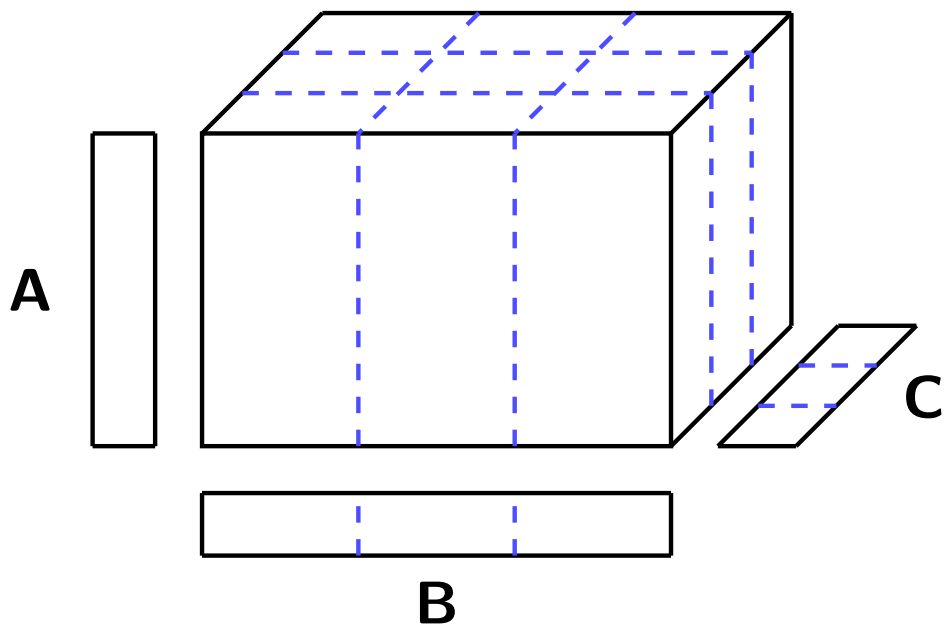
Solution:

- Extract the hub slices and use coarse-grained parallelism for the remaining ones.
- Fine-grained (i.e., non-zero based) parallelism used for hub slices.

Partial tiling

Constructing p^N tiles is not practical for high thread counts (p) or tensor dimensionality (N).

- Tile a few modes and selectively use mutexes for the remaining ones.
- Writes to **A** must be synchronized.
- Blocks of **B** and **C** are distributed among threads.



Privatization

Short modes will suffer from high lock contention.

- Give each thread a private factor matrix and aggregate at the end.

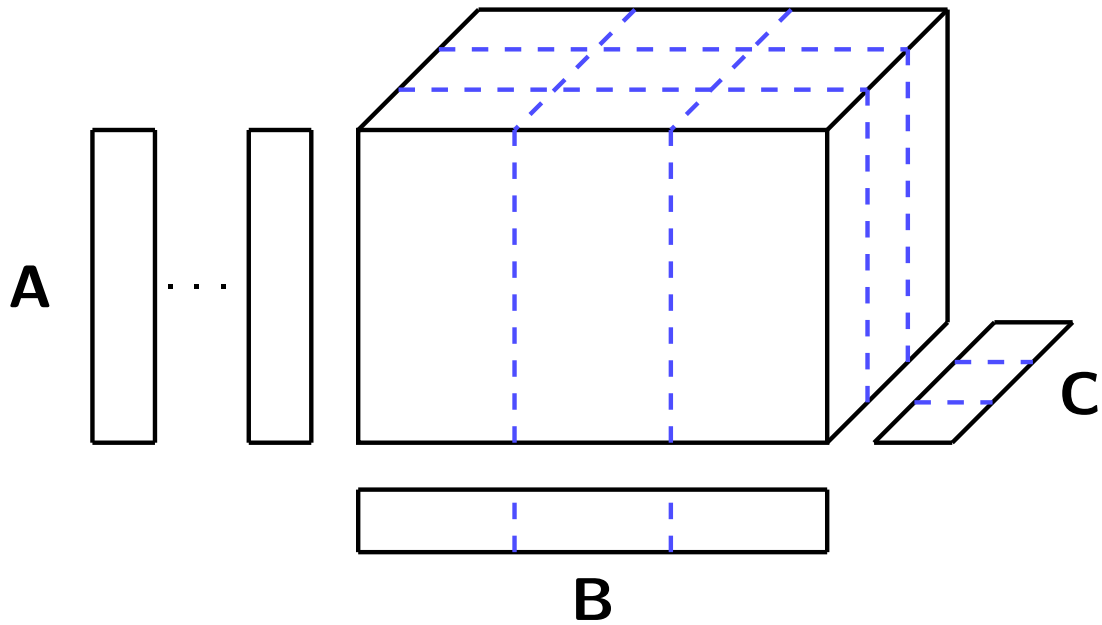


Table of Contents

Datasets

Dataset	NNZ	Dimensions	Size (GB)
Outpatient	87M	1.6M, 6K, 13K, 6K, 1K, 192K	4.1
Netflix	100M	480K, 18K, 2K	1.6
Delicious	140M	532K, 17M, 3M	2.7
NELL	143M	3M, 2M, 25M	2.4
Yahoo	262M	1M, 624K, 133	4.3
Reddit	924M	1.2M, 23K, 1.3M	15.0
Amazon	1.7B	5M, 18M, 2M	36.4

Experimental Setup

Software:

- Modified SPLATT library v1.1.1
- Open source C++ code with OpenMP parallelism
- Modified to use AVX-2 and AVX-512 intrinsics for vectorization.

Knights Landing:

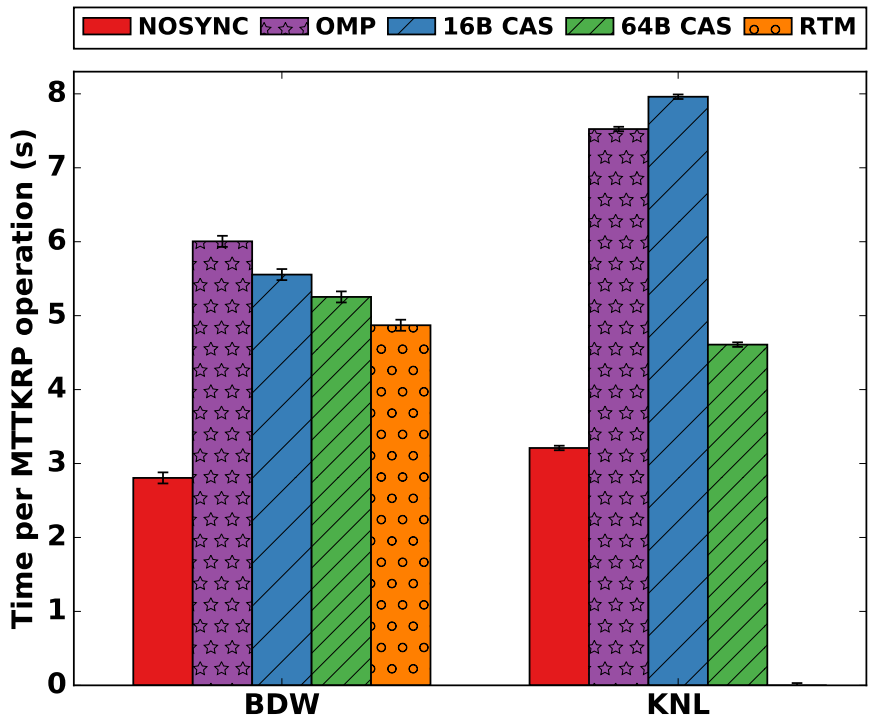
- KNL 7250 at TACC (Stampede).
- 68 cores (up to 272 threads)
- 16GB MCDRAM
- 94GB DDR4

Xeon:

- 2× 22-core Intel Xeon E5- 2699v4 Broadwell
- 2× 55MB last-level cache
- 128GB DDR4

Synchronization primitives

Synchronization overheads on Outpatient.



64B CAS simulates having CAS as wide as an AVX-512 vector.

MCDRAM

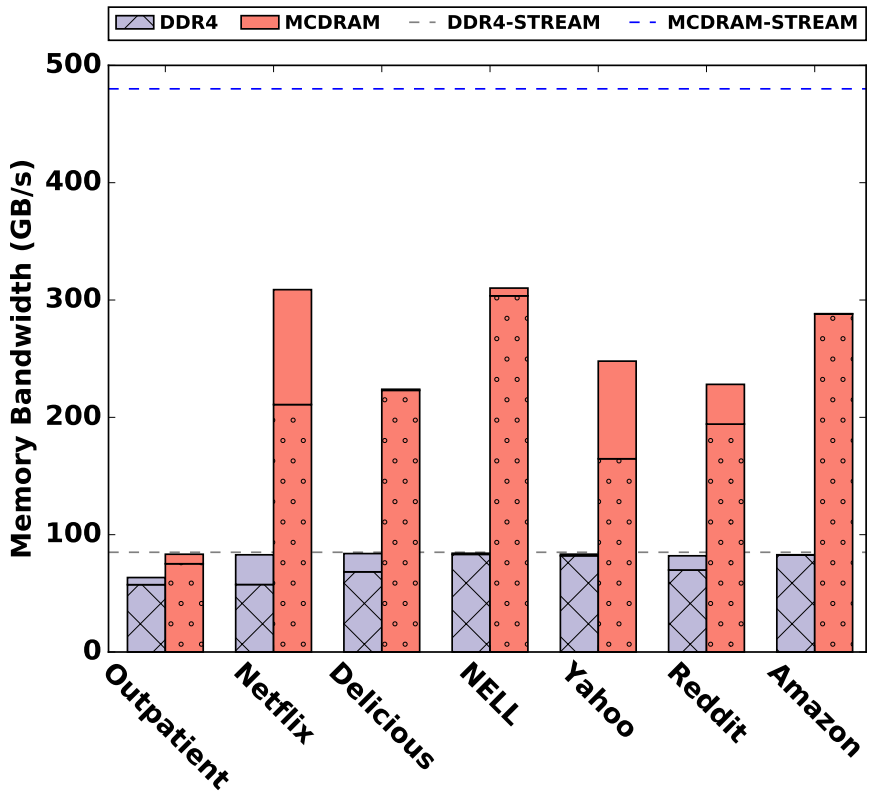
Non-zeros use $O(1)$ storage but spawn $O(F)$ accesses to the factors.

- Focus on placing the factors in MCDRAM.

MCDRAM

Non-zeros use $O(1)$ storage but spawn $O(F)$ accesses to the factors.

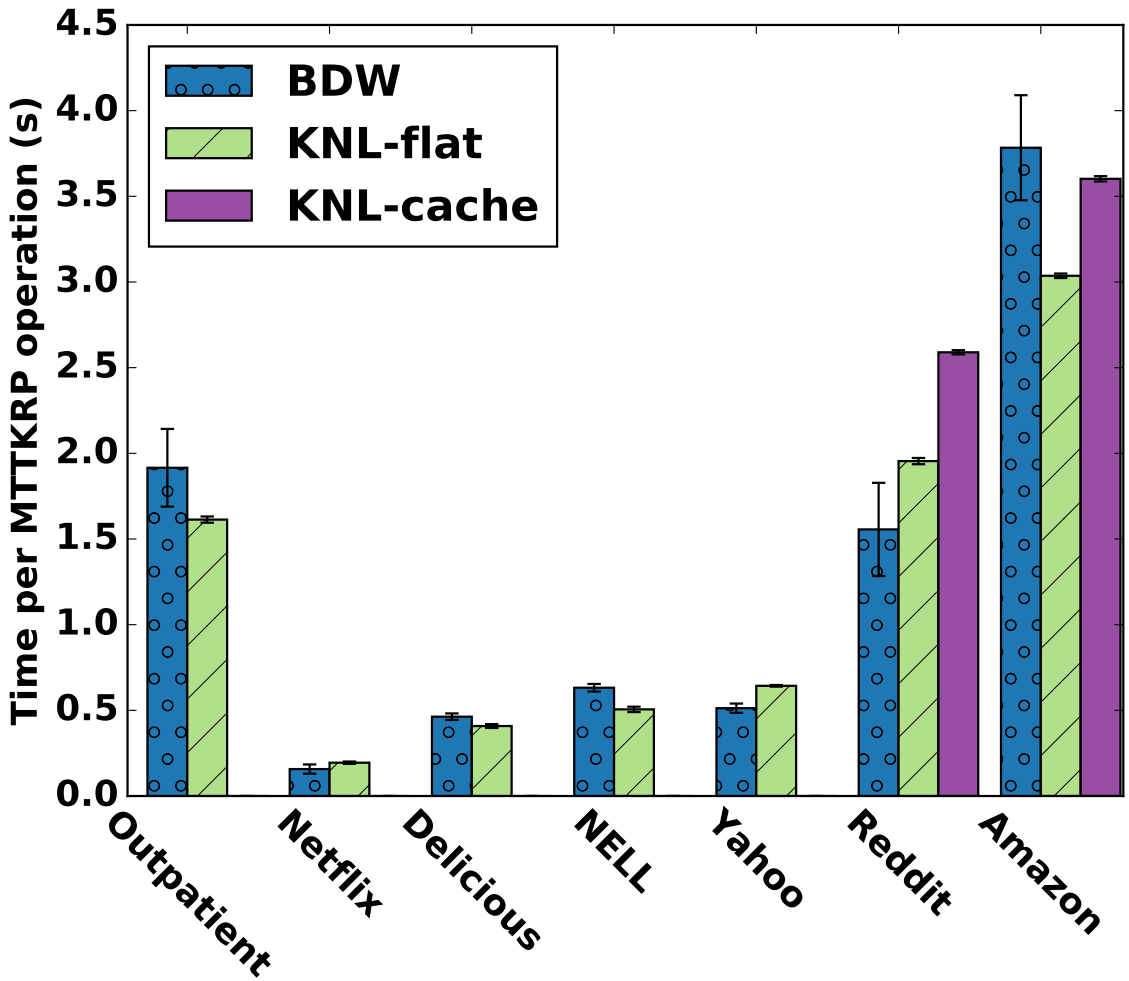
- Focus on placing the factors in MCDRAM.



- Stacked bars encode read-BW (bottom) and write-BW (top).
- KNL's maximum read-BW out of MCDRAM is 380 GB/s.

Comparison against Broadwell (rank 16)

- Up to 25% speedup over a 44-core Intel Xeon system.
- Managing MCDRAM gives us 30% speedup when dataset is larger than 16GB.



Scaling rank on Yahoo

Tensors with short modes fit inside the large cache of BDW systems.

- KNL is 2× faster as we scale the CPD rank.

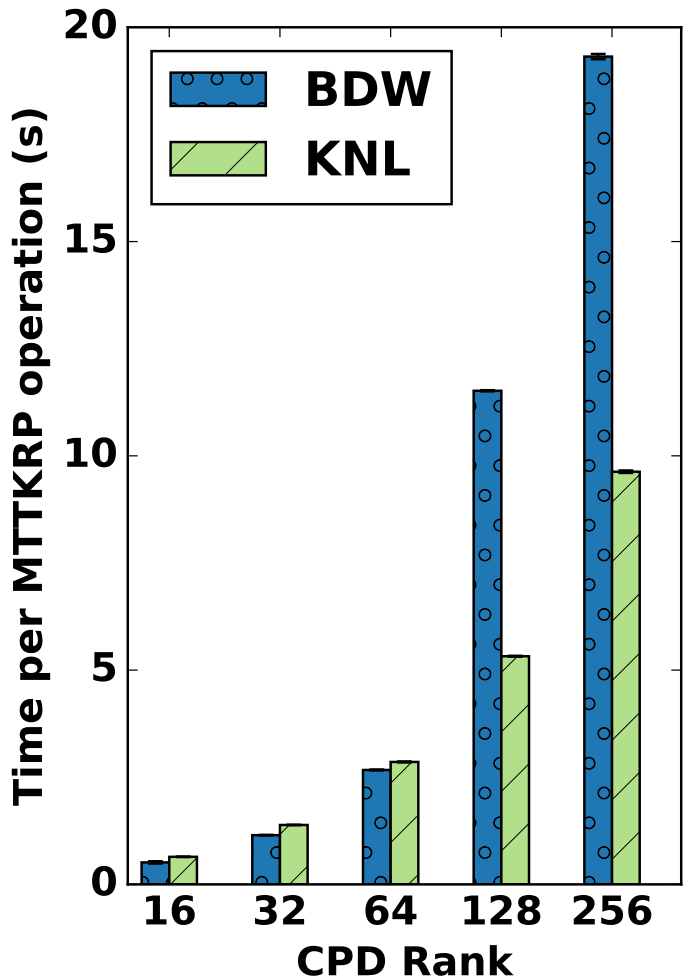


Table of Contents

Wrapping up

- Many-core processors with high-bandwidth memory can accelerate data-intensive applications.
- We have to revisit some algorithms to expose parallelism, reduce synchronization, and improve load balance.
- Managing MCDRAM can be helpful for large problem sizes.

All of our work is open source:

`http://cs.umn.edu/~splatt/`
`https://github.com/ShadenSmith/splatt-ipdps17`

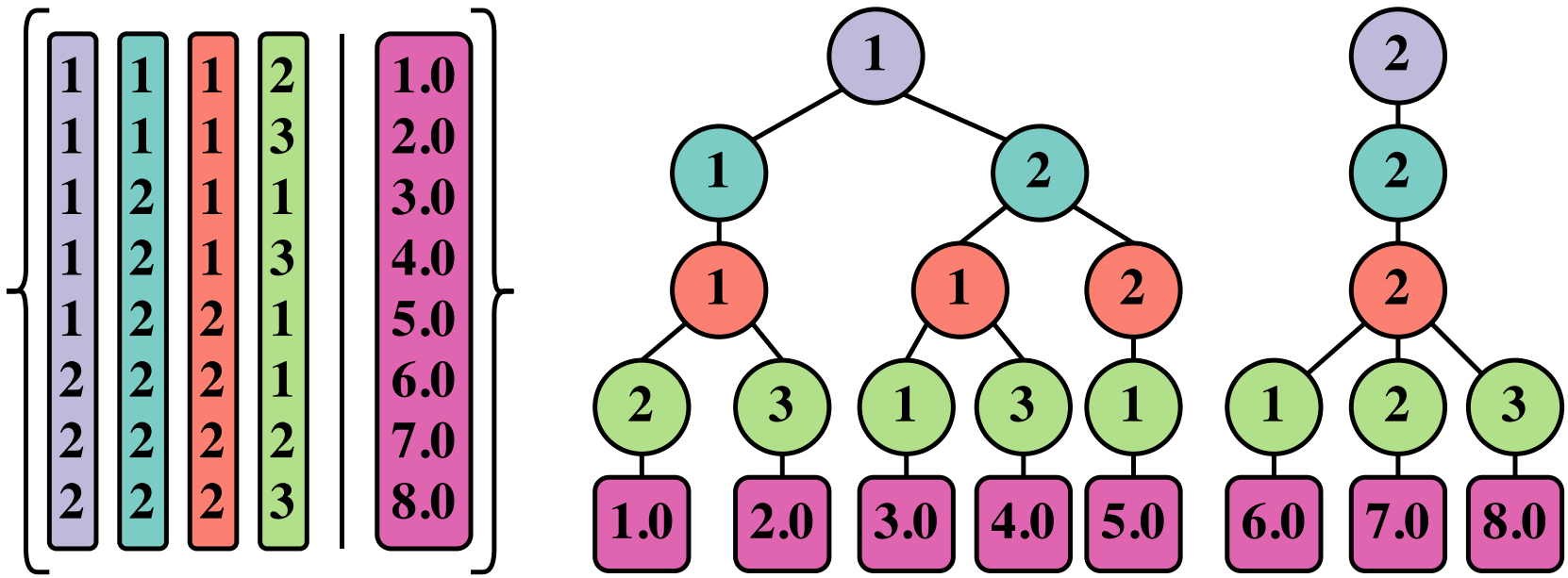
Datasets:

`http://frostt.io/`

Backup Slides

Compressed sparse fiber (CSF)

- Modes are recursively compressed.
- Paths from roots to leaves encode non-zeros.
- The tree structure encodes opportunities for savings.



MTTKRP with CSF

```
/* foreach outer slice */
for(int i=0; i < I; ++i) {
  /* foreach fiber in slice */
  for(int s = s_ptr[i]; s < s_ptr[i+1]; ++s) {
    accum[0:r] = 0;

    /* foreach  nnz in fiber */
    for(int nnz = f_ptr[s]; nnz < f_ptr[s+1]; ++nnz) {
      int k = f_ids[nnz];
      accum[0:r] += vals[nnz] * C[k][0:r];
    }

    int j = s_ids[s];
    A[i][0:r] += accum[0:r] * B[s][0:r];
  }
}
```

Hub Slices

Load imbalance on the Amazon dataset.

	BDW		KNL	
Mode	**slice**	**hub**	**slice**	**hub**
1	0.72	**0.04**	0.84	**0.05**
2	0.13	**0.04**	0.05	**0.03**
3	0.07	**0.03**	0.24	**0.18**

$$\text{imbalance} = \frac{t_{max} - t_{avg}}{t_{max}}.$$

Tile Depth

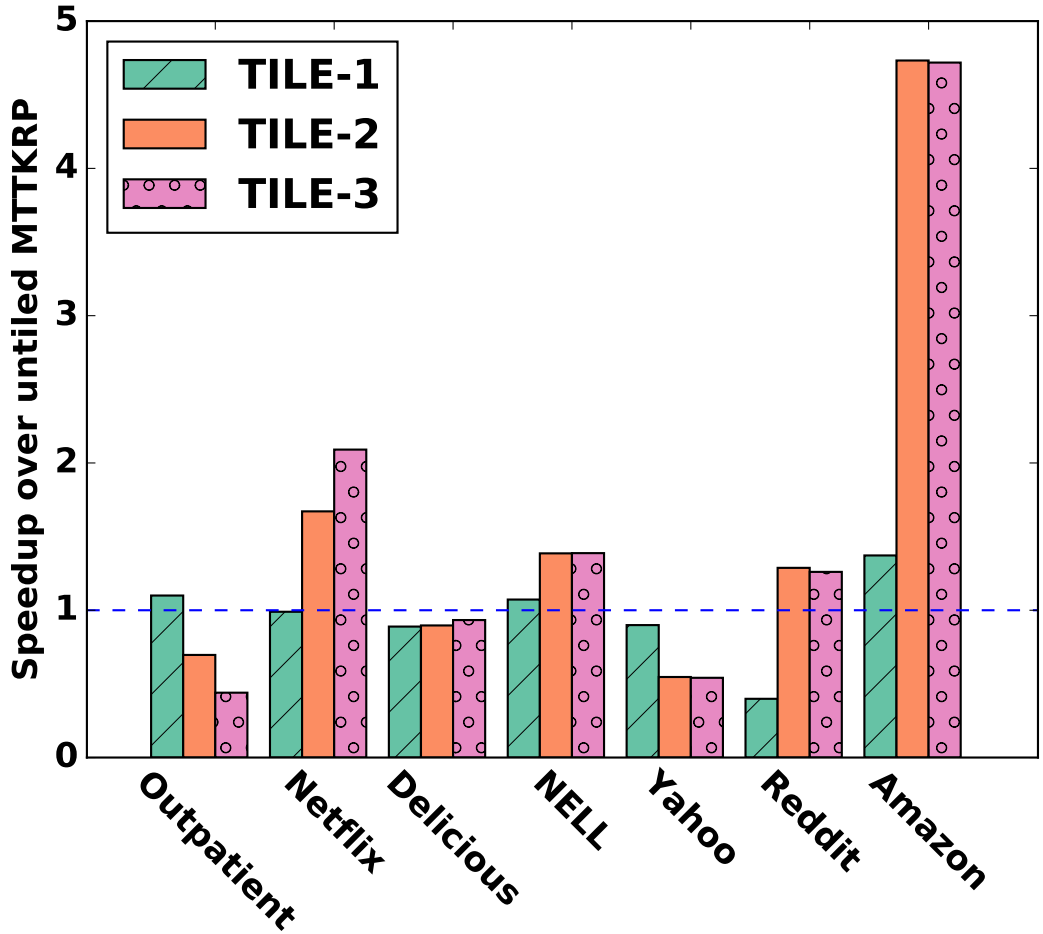


Figure: Speedup over untiled MTTKRP while tiling the longest (**Tile-1**), two longest (**Tile-2**), and three longest modes (**Tile-3**).

Privatization

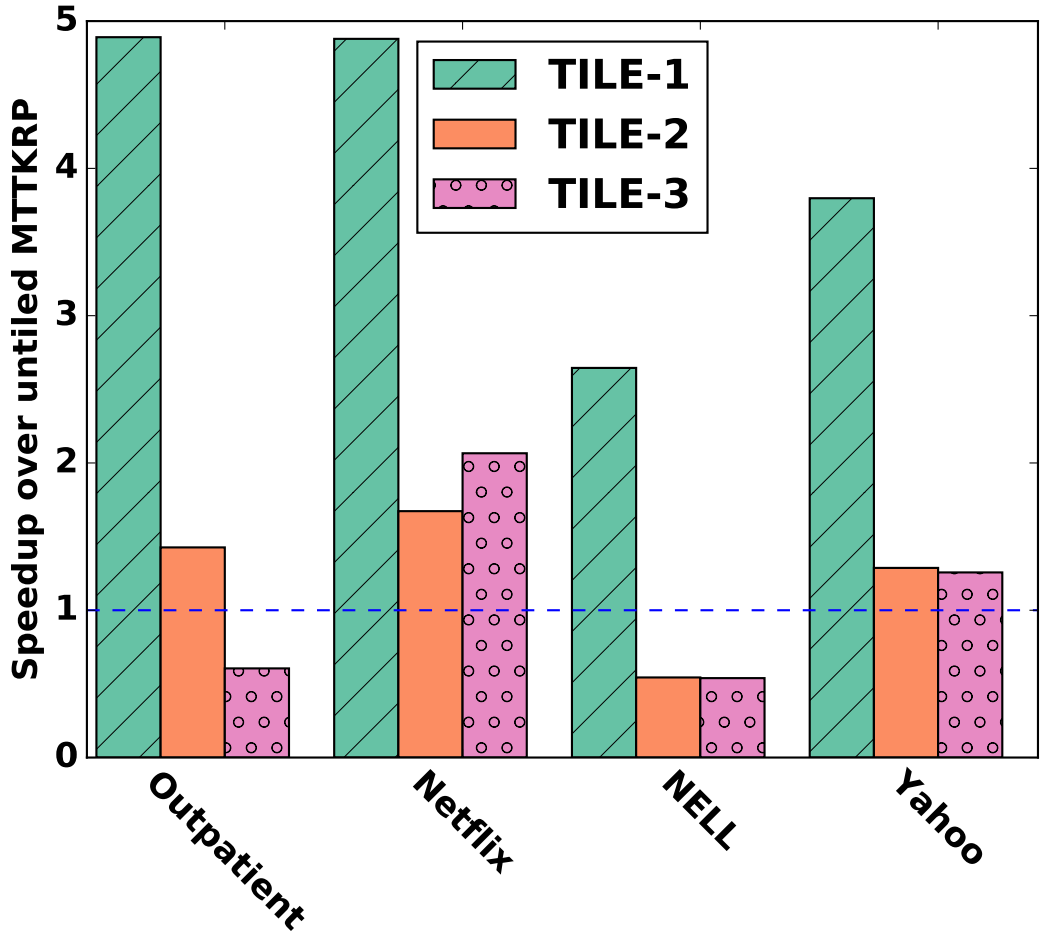


Figure: Speedup over untiled MTTKRP using one, two, and three tiled modes with privatization for synchronization. Privatized modes were selected with γ=0.2.

Number of CSF

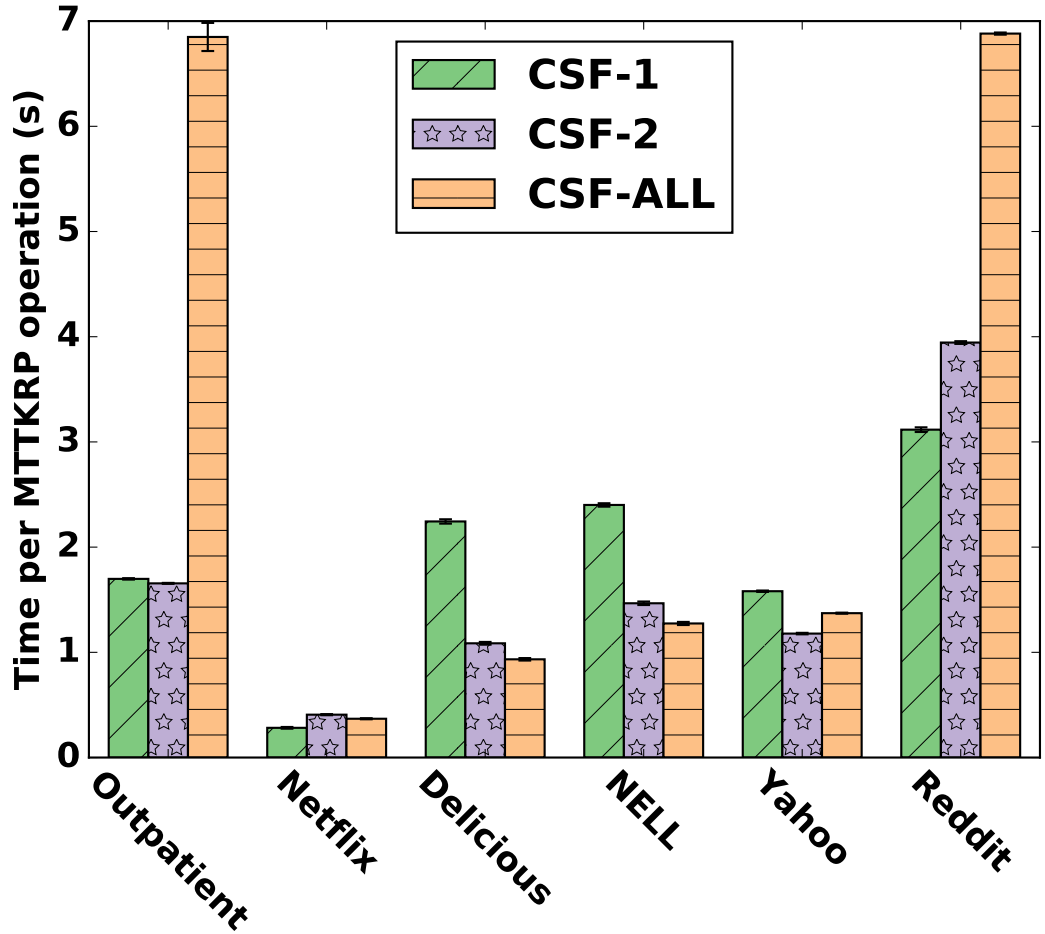


Figure: Effects of the number of CSF representations on MTTKRP runtime, using 1, 2, and M representations. Amazon is omitted due to memory constraints.