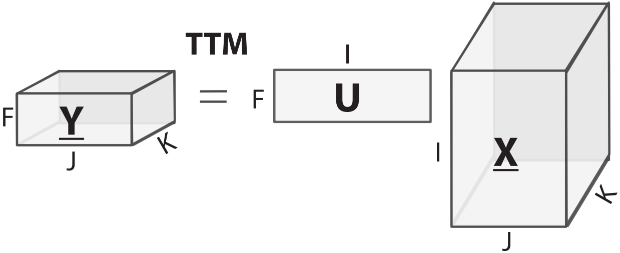
An Input-Adaptive and In-Place Approach to Dense Tensor-Times-Matrix Multiply

Jiajia Li, **Casey Battaglino**, Ioakeim Perros,
Jimeng Sun, Richard Vuduc

Computational Science & Engineering,
Georgia Institute of Technology

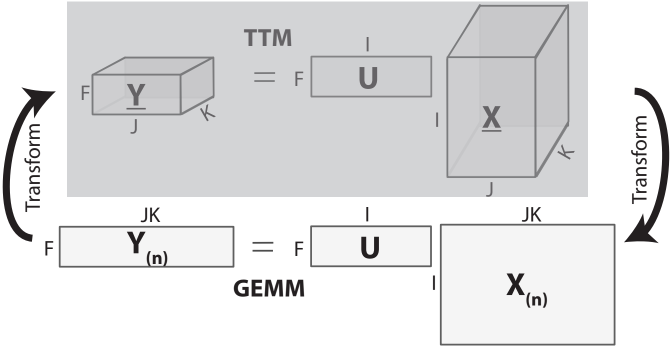
14th April 2016

The problem



$\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_n \mathbf{U}$

The problem

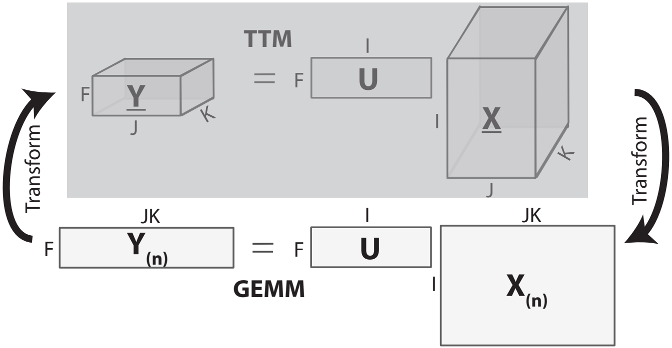


$\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_n \mathbf{U}$

$\Downarrow$

$\mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}$

The problem



$\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_n \mathbf{U}$

$\Downarrow$

$\mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}$

Transform:

70% running time.

50% space.

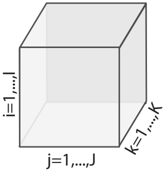
- We proposed an in-place TTM algorithm and employed auto-tuning method to adapt its parameters.

Outline

- Background
- Motivation
- InTensLi Framework
- Experiments and Analysis
- Conclusion

Tensor and Applications

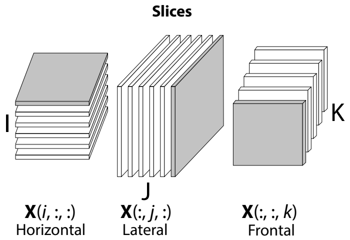
- Tensor: interpreted as a multi-dimensional array, e.g. $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$.
  - Special cases: vectors ($\mathbf{x}$) are $1D$ tensors, and matrices ($\mathbf{A}$)are $2D$ tensors.
  - Tensor dimension (N): also called mode or order.
  - We focus on dense tensors in this work.
- Applications
  - Quantum chemistry, quantum physics, signal and image processing, neuroscience, and data analytics.

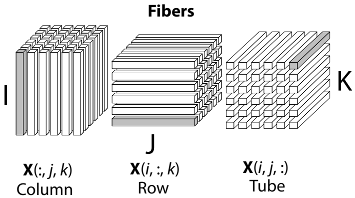


A third-order (or three-dimensional) $I \times J \times K$ tensor.

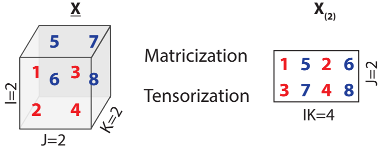
Tensor Representations

- Sub-tensor





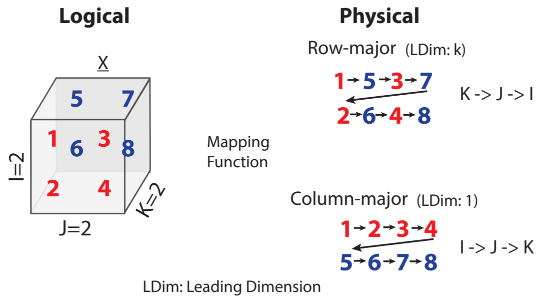
- Whole tensor



- Diff representations $\rightarrow$ Diff algorithms $\rightarrow$ Diff performance.

Memory Mapping

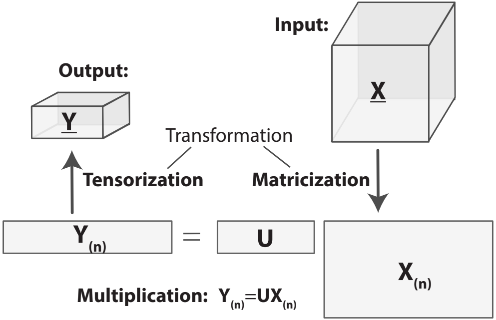
- Tensor organization
  - Multi-dimensional array – logically
  - Linear storage – physically
- Memory mapping[1].



[1] GARCIA, R.,and LUMSDAINE, A. Multiarray:A c++ library for generic programming with arrays.Software Practive Experience 35 (2004), 159–188.

TTM Algorithm

- Baseline tensor-times-matrix multiply (TTM) algorithm in TENSOR TOOLBOX and CYCLOPS Tensor Framework (CTF).



- TTM Applications
  - Low-rank tensor decomposition.
  - Tucker decomposition, e.g. TUCKER-HOOI algorithm.

$$\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_1 \mathbf{A}^{(1)T} \cdots \times_{n-1} \mathbf{A}^{(n-1)T} \times_{n+1} \mathbf{A}^{(n+1)T} \cdots \times_N \mathbf{A}^{(N)T}.$$

Main Contributions

- Proposed an in-place tensor-times-matrix multiply (InTtm) algorithm, by avoiding physical reorganization of tensors.
- Built an input-adaptive framework InTensLi to automatically adapt parameters and generate the code.
- Achieved $4\times$ and $13\times$ speedups compared to the state-of-the-art Tensor Toolbox and Ctf tools.

Observation 1: Transformation is expensive.

Notation: the number of words (Q), floating-point operations (W), last-level cache size (Z).

The relation of them is $Q \geq \frac{W}{8\sqrt{Z}} - Z$ [2] for both general matrix-matrix multiply (GEMM) and TTM.

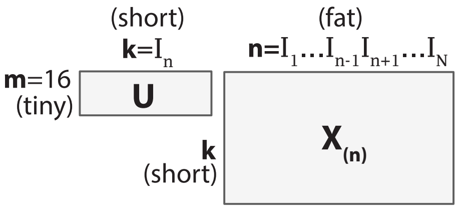
- Suppose TTM does the same number of flops as GEMM ($\hat{W} = W$), the relation of Arithmetic Intensity of GEMM and TTM: $\hat{A} \approx A/(1 + \frac{A}{m})$ when counting transformation.
  $(1 + \frac{A}{m})$ is the penalty.
- Assume cache size Z is 8MB, the penalty of a 3-D tensor is 33.

Conclusion: When TTM and GEMM do the same number of flops, Arithmetic Intensity of TTM is decreased by a penalty of 33 or more, as tensor dimension increases.

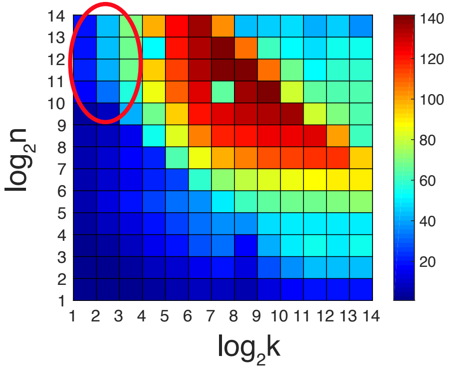
[2] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. Acta Numerica, 23:pp. 1–155, 2014.

Observation 2: Performance of the multiplication in TTM is far below peak.

- TTM algorithm involves a variety of rectangular problem sizes.



(a) TTM's multiplication.



(b) GEMM performance in Intel MKL with 4 threads.

Observation 3: TTM organization is critical to data locality.

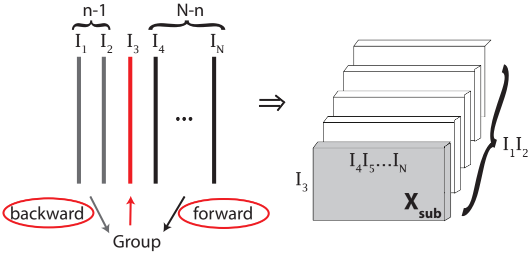
- There are many ways to organize data accesses.
- Choose slice representation.

Table 1 : Different representation forms of mode-1 TTM on a $I \times J \times K$ tensor.

Mode-1 Product Representation Forms		BLAS Level	Transformation
Full reorganization	*Tensor representation* $\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_1 \mathbf{U}$	—	—
	Matrix representation $\mathbf{Y}_{(1)} = \mathbf{U}\mathbf{X}_{(1)}$	L3	Yes
Sub-tensor extraction	*Fiber representation* $\mathbf{y}(f,:,k) = \mathbf{X}(:,:,k)\mathbf{u}(f,:)$, *Loops* : $k = 1, \cdots, K, f = 1, \cdots, F$	L2	No
	Slice representation $\mathbf{Y}(:,:,k) = \mathbf{U}\mathbf{X}(:,:,k)$, *Loops* : $k = 1, \cdots, K$	L3	No

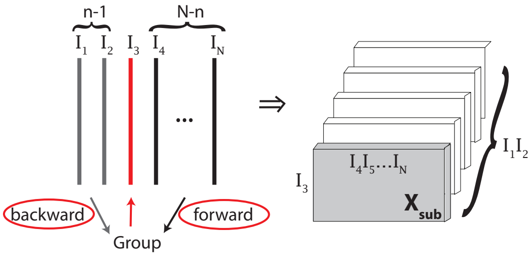
Layout

Algorithmic Strategy



- To avoid data copy,
  - Rules: 1) compress only contiguous dimensions; 2) always include the leading dimension.
  - Lemma: TTM can be performed on up to $max\{n-1, N-n\}$ contiguous dimensions without physical reorganization.

Algorithmic Strategy



- To avoid data copy,
  - Rules: 1) compress only contiguous dimensions; 2) always include the leading dimension.
  - Lemma: TTM can be performed on up to $max\{n-1, N-n\}$ contiguous dimensions without physical reorganization.
- To get high performance of GEMM,
  - Find an approximate matrix size according to computer architecture.
  - Use auto-tuning method in INTENSLI framework.

InTtm Algorithm and Comparison

- InTtm's AI: $\tilde{A} \lesssim \frac{\hat{Q}}{\frac{\hat{Q}}{8\sqrt{Z}}} = 8\sqrt{Z} \approx A$.
- Traditional Ttm's AI: $\hat{A} \approx \frac{A}{1+\frac{A}{m}}$.
- InTtm eliminates the AI by a factor $1 + \frac{A}{m}$.

Input: A dense tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, a dense matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$, and an integer n;
Output: A dense tensor $\underline{\mathbf{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$;

// Nested loops, using P_L threads
parfor $i_l = 1$ to I_l, all $i_l \in M_L$ **do**
  if M_C are on the left of i_n **then**
    $\mathbf{X}_{\text{sub}} =$ inplace-mat $(\underline{\mathbf{X}}, M_C, i_n)$;
    $\mathbf{Y}_{\text{sub}} =$ inplace-mat $(\underline{\mathbf{Y}}, M_C, j)$;
    // Matrix-matrix multiplication, using P_C threads
    $\mathbf{Y}_{\text{sub}} = \mathbf{X}_{\text{sub}}\mathbf{U}'$, $\mathbf{U}'$ is the transpose of $\mathbf{U}$.
  else
    $\mathbf{X}_{\text{sub}} =$ inplace-mat $(\underline{\mathbf{X}}, i_n, M_C)$;
    $\mathbf{Y}_{\text{sub}} =$ inplace-mat $(\underline{\mathbf{Y}}, j, M_C)$;
    // Matrix-matrix multiplication, using P_C threads
    $\mathbf{Y}_{\text{sub}} = \mathbf{U}\mathbf{X}_{\text{sub}}$
  end if
end parfor
return $\underline{Y}$;

In-place Tensor-Times-Matrix Multiply (InTtm) algorithm.

InTtm Algorithm and Comparison

- InTtm's AI: $\tilde{A} \lesssim \frac{\hat{Q}}{\frac{\hat{Q}}{8\sqrt{Z}}} = 8\sqrt{Z} \approx A$.
- Traditional Ttm's AI: $\hat{A} \approx \frac{A}{1+\frac{A}{m}}$.
- InTtm eliminates the AI by a factor $1 + \frac{A}{m}$.

Input: A dense tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, a dense matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$, and an integer n;
Output: A dense tensor $\underline{\mathbf{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$;

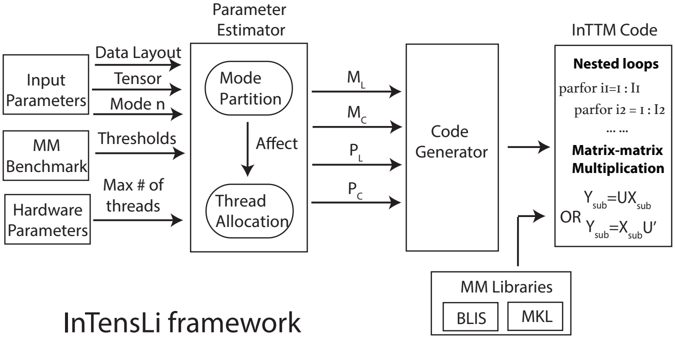
```
// Nested loops, using [P_L] threads
parfor i_l = 1 to I_l, all i_l ∈ [M_L] do
  if [M_C] are on the left of i_n then
    X_sub = inplace-mat (X, M_C, i_n);
    Y_sub = inplace-mat (Y, M_C, j);
    // Matrix-matrix multiplication, using [P_C] threads
    Y_sub = X_sub U', U' is the transpose of U.
  else
    X_sub = inplace-mat (X, i_n, M_C);
    Y_sub = inplace-mat (Y, j, M_C);
    // Matrix-matrix multiplication, using P_C threads
    Y_sub = U X_sub
  end if
end parfor
return Y;
```

In-place Tensor-Times-Matrix Multiply (InTtm) algorithm.

Layout

InTensLi Framework

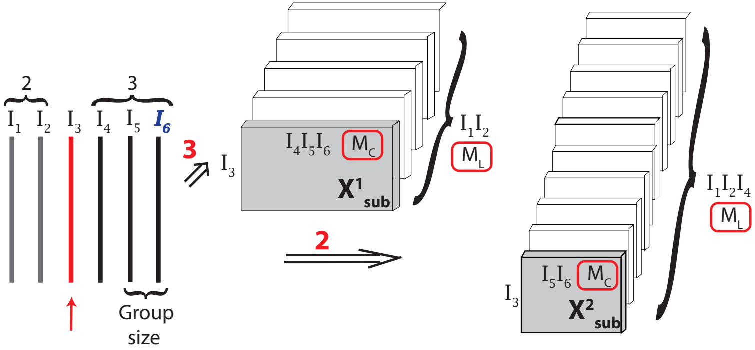
- Input: tensor features, hardware configuration, and MM benchmark.
- Parameter estimation
  - Mode partitioning: M_L and M_C.
  - Thread allocation: P_L and P_C.
- Code generation



InTensLi framework

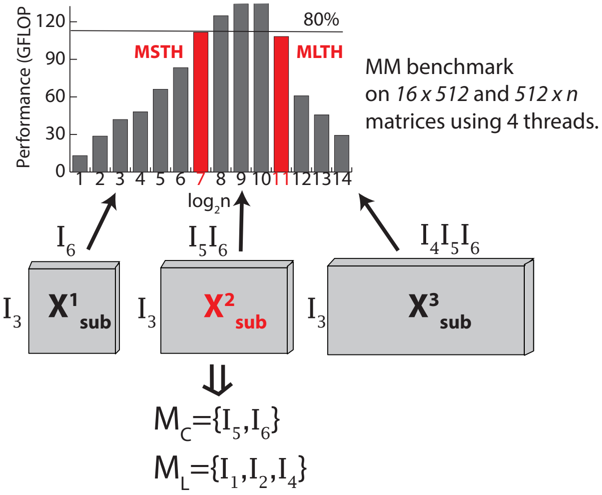
Parameter Estimation – Mode Partitioning

- Determine grouping direction
  - Row-major ↔ forward
  - Column-major ↔ backward
- Group size decides InTtm algorithm.



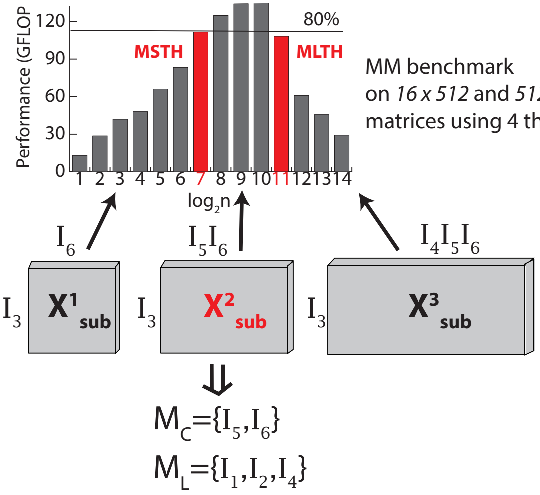
 InTensLi Framework

Choosing Group Size



- *MSTH* and *MLTH*: Thresholds of GEMM's size, the size of all the three operating matrices.
- $MSTH = 1.04MB$ and $MLTH = 7.04MB$ in our experiments.

Choosing Group Size



MM benchmark on *16 x 512* and *512 x n* matrices using 4 threads.

- *MSTH* and *MLTH*: Thresholds of GEMM's size, the size of all the three operating matrices.
- $MSTH = 1.04MB$ and $MLTH = 7.04MB$ in our experiments.
- Decide M_C: Use *MSTH* and *MLTH* to decide group size, then decide M_C.
- Decide M_L: The rest modes of M_C, except mode-n.

Thread Allocation and Code Generation

- Thread allocation
  - In most cases, maximum performance is obtained by only two configurations:
    - Small matrices: all threads are allocated to nested loops.
    - Large matrices: all threads are allocated to GEMM operation.
  - A threshold *PTH* is set to distinguish the GEMM size, which is 800 KB in our tests.
- Code generation
  - Generate nested loops and wrappers for the GEMM kernel.
  - Code generated in C++, using OpenMP with the collapse directive.

Experimental Platforms

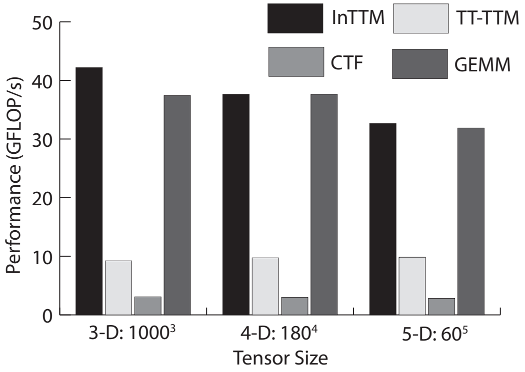
- Double-precision is adopted in our experiments.
- We employ 8 and 32 threads on the two platforms respectively, considering hyper-threading.
- Xeon E7-4820 has a relatively large memory (512 GiB), allowing us to test a larger range of (dense) tensor sizes than has been common in prior single-node studies.

Table 2 : Experimental Platforms Configuration

Parameters	Intel Core i7-4770K	Intel Xeon E7-4820
Microarchitecture	Haswell	Westmere
Frequency	3.5 GHz	2.0 GHz
# of physical cores	4	16
Hyper-threading	On	On
Peak GFLOP/s	224	128
Last-level cache	8 GiB	18 GiB
Memory size	32 GiB	512 GiB
Memory bandwidth	25.6 GB/s	34.2 GB/s
# of memory channels	2	4
Compiler	icc 15.0.2	icc 15.0.0

Performance Comparison

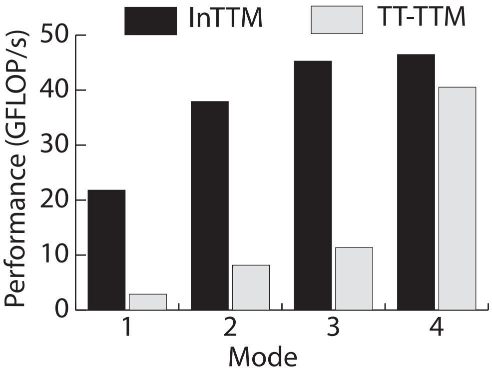
- Implementations
  - InTtm: InTensLi generated C++ code with OpenMP.
  - TT-TTM: Tensor Toolbox library in MATLAB.
  - Ctf: C++ code, supporting MPI+OpenMP parallelization.
  - Gemm: C++ code, baseline Ttm algorithm without transformation.
- Speedup
  - Obtain 4× and 13× speedup compared to Tensor Toolbox and Ctf.
  - Get close to Gemm-only's performance.



Performance comparison of Ttm on mode-2 over diverse dimensional tensors.

Analysis

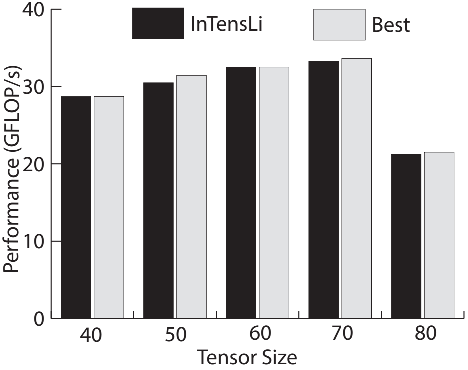
- Performance of different modes.
  - InTensLi is stable for different mode-n products, while Tensor Toolbox is not.



Performance behavior of InTtm against Tensor Toolbox (TT-TTM) for different mode products on a $160 \times 160 \times 160 \times 160$ tensor.

Analysis

- Parameter selection
  - Compare InTensLi with exhaustive search, the performance is close to optimal.



Comparison between the performance of Ttm on mode-1 with predicted configuration and the actually highest performance on 5th-order tensors.

Conclusion

Summary

- Proposed an in-place tensor-times-matrix multiply (InTtm) algorithm, by avoiding physical reorganization of tensors.
- Built an input-adaptive framework InTensLi to automatically do optimization and generate the code.
- Achieved $4\times$ and $13\times$ speedups compared to the state-of-the-art Tensor Toolbox and Ctf tools.

Source code

- `https://github.com/hpcgarage/InTensLi`
- Contact: Jiajia Li (jiajiali@gatech.edu)

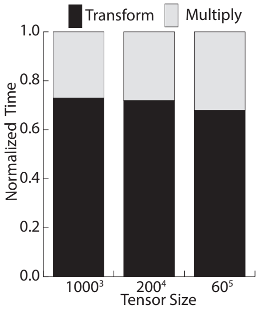
References

- E. Solomonik, D. Matthews, J. Hammond, and J. Dem- mel. Cyclops tensor framework: reducing commu- nication and eliminating load imbalance in massively parallel contractions. Technical Report UCB/EECS- 2012-210, EECS Department, University of California, Berkeley, Nov 2012.
- B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 2.5. Available from `http://www.sandia.gov/~tgkolda/TensorToolbox/index-2.6.html`, January 2012
- T. Kolda and B. Bader. Tensor decompositions and applications. SIAM Review, 51(3):455–500, 2009.
- ...

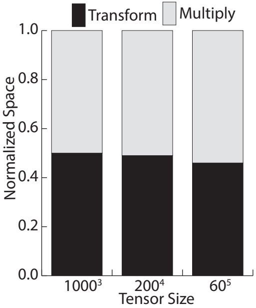
Backup Slides

Observation 1: Transformation is expensive.

- Transformation takes about 70% of the total run-time, and close to 50% of the total storage.



(a) Time Profiling



(b) Space Profiling

Profiling of TTM algorithm on mode-2 product on 3rd, 4th, and 5th-order tensors, where the output tensors are low-rank representations of corresponding input tensors.

Observation 3: TTM organization is critical to data locality.

- There are many ways to organize data accesses.