Scalable Tensor Algorithms for Modern Computing Systems

Suraj Kumar

Alpines team, Inria Paris, France

Suraj.kumar@inria.fr

May 26, 2021

Resume

- 11/2019 – Current, Postdoc, Inria Paris, France
  - Parallel algorithms for tensor computations, Use of tensors in molecular simulations
- 05/2018 – 10/2019, Postdoc, Pacific Northwest National Laboratory, USA
  - Theoretical and empirical analysis of data transfer orders, Task-based runtime systems for tensor operations of molecular simulations
- 08/2017 – 02/2018, Senior Engineer, Ericsson, Bangalore, India
  - Use of remote GPUs in cloud
- 12/2013 – 06/2017, PhD Student, University of Bordeaux & Inria Bordeaux, France
  - Scheduling of dense linear algebra kernels on heterogeneous resources
- 07/2012 – 11/2013, Software Engineer, IBM Research, New Delhi, India
  - Performance optimizations of TTI RTM algorithm on GPUs, Schedulers for BG/P machine
- 08/2010 – 06/2012, Master Student, Indian Institute of Science, Bangalore, India
  - Automatic parallelization of programs with linked-list data structure

Part I

Past/Ongoing Work

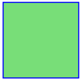
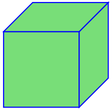
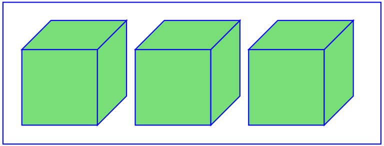
Collaborators

This is joint work with ...

- Laura Grigori – Inria Paris, France
- Grey Ballard – Wake Forest University, USA
- Hussam Al Daas – STFC Rutherford Appleton Laboratory, UK
- Olivier Beaumont – Inria Bordeaux, France
- Sriram Krishnamoorthy – Pacific Northwest National Laboratory, USA
- Marcin Zalewski – Nvidia, USA
- Lionel Eyraud-Dubois – Inria Bordeaux, France
- Samuel Thibault – Inria Bordeaux, France
- Emmanuel Agullo – Inria Bordeaux, France

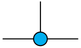
Overview

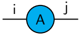
Tensors: Multidimensional Arrays

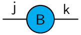
Dimension	Name	
1	Vector	
2	Matrix	
3	3-dimensional tensor	
4	4-dimensional tensor	

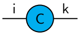
Tensor Diagram Notations

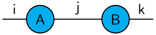
Tensors are denoted by solid shapes and number of lines coming out of the shapes denote the dimensions of the tensors.
For example,

Dimension	Name	
1	Vector	
2	Matrix	
3	3-dimensional tensor	

- Connecting two lines implies summation over the connected dimensions
- Multiplication of matrices 

and 

is represented as 

= 

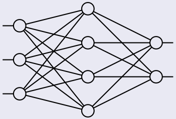
- Text notation: bold letters to denote tensors, i.e., $\mathbf{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is a d-dimensional tensor

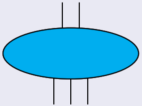
Tensors are used in Several Domains

- **Neuroscience**: Neuron $\times$ Time $\times$ Trial
- **Transportation**: Pickup $\times$ Dropoff $\times$ Time
- **Media**: User x Movie x Time x Rating
- **Ecommerce**: User x Product x Rating
- **Cyber-Traffic**: IP x IP x Port x Time
- **Social-Network**: Person x Person x Time x Interaction-Type

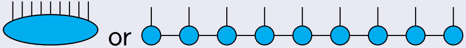
High Dimensional Tensors

- **Neural Network**:





- **Quantum or Molecular Simulation**: Wave function is represented as

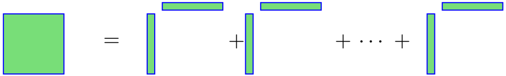


Tensor Computations

- Memory and computation requirements are exponential in the number of dimensions
  - A molecular simulation involving just 100 spatial orbitals manipulate a huge tensor with 4^{100} elements
- People work with low dimensional structure (decomposition) of the tensors
  - A tensor is represented with smaller objects
  - Useful to find patterns in massive data
  - Improves memory and computation requirements
- Limited work on parallelization of tensor algorithms

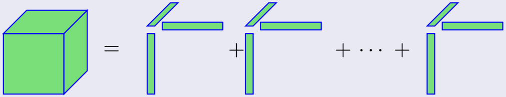
SVD and Higher Order Generalization of SVD

- SVD represents a matrix as the sum of rank one matrices, $A = U \Sigma V^T = \Sigma(i;i) U_i V_i^T$

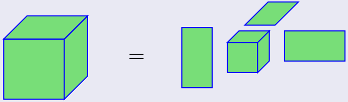


Popular Tensor Decompositions (Higher Order Generalization of SVD)

- Canonical decomposition (Also known as Canonical Polyadic or CANDECOMP/PARAFAC)

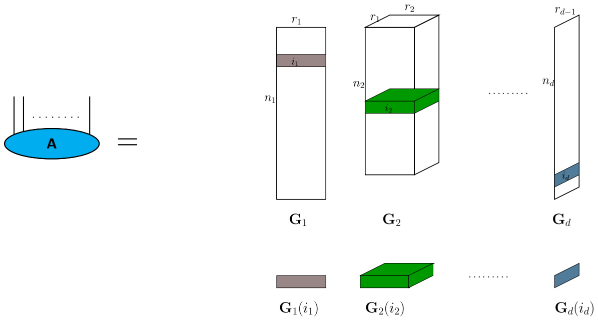


- Tucker decomposition



- Tensor Train decomposition (equivalently known as Matrix Product States)

Tensor Train Representation: Product of Matrices View

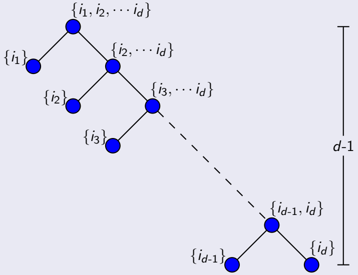


$$\mathbf{A}(i_1, i_2, \cdots, i_d) = \mathbf{G}_1(i_1)\mathbf{G}_2(i_2)\cdots\mathbf{G}_d(i_d)$$

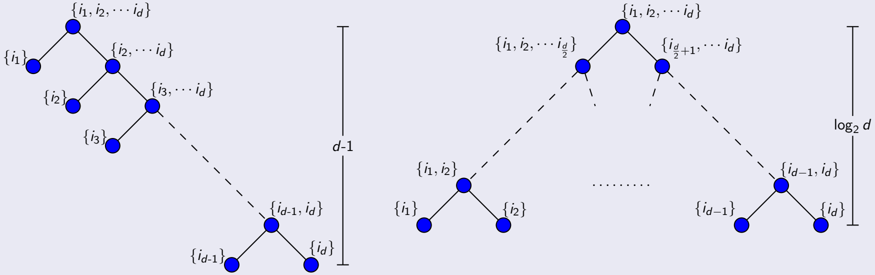
- A d-dimensional tensor is represented with 2 matrices and d-2 3-dimensional tensors.
- An entry is computed by multiplying corresponding matrix (or row/column) of each core.
- For $n_1 = \cdots = n_d = n$ and $r_1 = \cdots = r_{d-1} = r$, the number of entries $= \mathcal{O}(ndr^2)$

Table of Contents

Separatation of Dimensions in Sequential Algorithm [Oseledets, 2011]



Separatation of Dimensions for Maximum Parallelization



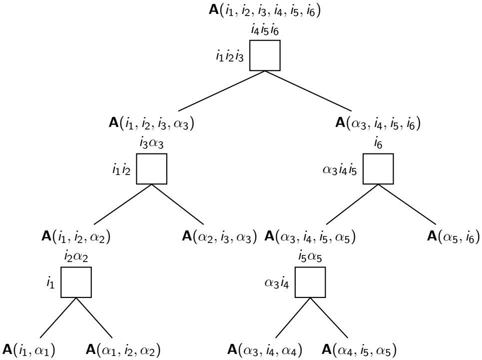
Diagramatic Representation of the Parallel Algorithm

k-th unfolding matrix

A_k denotes k-th unfolding matrix of tensor $\mathbf{A}$.

$$A_k = [A_k(i_1, i_2, \cdots, i_k; i_{k+1}, \cdots, i_d)]$$

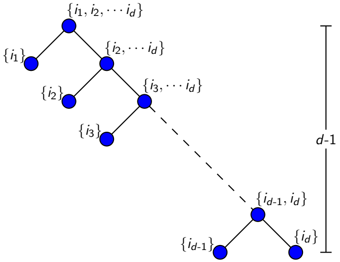
- Size of A_k is $(\prod_{l=1}^{k} n_l) \times (\prod_{l=k+1}^{d} n_l)$
- $r_k = \text{rank}(A_k)$



Theorem

The parallel algorithm produces a Tensor Train representation with ranks not higher than r_k. It implies $\alpha_i \le r_i$ in the above diagram.

Aprroximations in Sequential Tensor Train Algorithms [Oseledets, 2011]



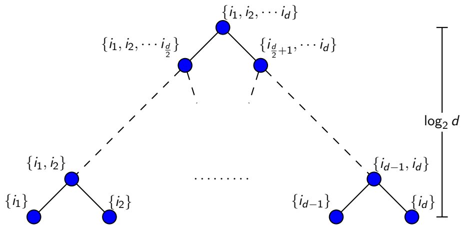
Approximation of a d dimensional tensor

To obtain approximation error not more than ϵ, at each node,

- Perform SVD of input matrix A, $A = U\Sigma V^T + E_A$
- Apply constant truncation $\frac{\epsilon}{\sqrt{d-1}}$, i.e., $||E_A||_F \leq \frac{\epsilon}{\sqrt{d-1}}$
- Reshape U as one of the tensor cores
- If ΣV^T corresponds to more than one dimension, reshape and work with it on the right node

- Frobenius norm of a matrix A is defined as, $||A||_F = \sqrt{\sum_{i,j} A(i;j)^2}$
- Frobenius norm of a d-dimensional tensor $\mathbf{A}$ is defined as, $||\mathbf{A}||_F = \sqrt{\sum_{i_1,i_2,\cdots,i_d} \mathbf{A}(i_1, i_2, \cdots, i_d)^2}$

Aprroximations in Parallel Tensor Train Algorithms



Approach 1: $X = I,\ Y = \Sigma,\ S = I$

Approach 2: $X = Y = \Sigma^{\frac{1}{2}},\ S = I$

Approach 3: $X = Y = \Sigma,\ S = \Sigma^{-1}$

Approximation of a d dimensional tensor

To obtain approximation error less than or close ϵ, at each node,

- Perform SVD of input matrix A, $A = U\Sigma V^T + E_A$
- Find diagonal matrices X, Y, and S, such that $\Sigma = XSY$
- Apply truncation, i.e., $||E_A||_F \leq \frac{\epsilon}{\sqrt{dimensions(A)-1}}$
- If U (resp. V^T) corresponds to more than one dimension, reshape UX and call left (resp. right) subtree with approximation error ϵ_1 (resp. ϵ_2)
- ϵ_1 and ϵ_2 depend on X, Y, S and ϵ

Comparison of all Approaches

We consider a *Log* tensor generated with low rank function $\log(\sum_{j=1}^{d} j i_j)$. For $d = 12$ and $i_j \in \{1, 2, 3, 4\}_{1 \leq j \leq d}$, this function produces a 12-dimensional tensor with 4^{12} elements.

Comparison of all approaches for Log tensor

- Prescribed accuracy $= 10^{-6}$
- compr: compression ratio, ne: number of elements in aprroximation, OA: approximation accuracy

Metric	Sequential Algo	Parallel Algo		
		Approach 1	Approach 2	Approach 3
compr	99.993	99.817	99.799	99.993
ne	1212	30632	33772	1212
OA	2.271e-07	3.629e-08	2.820e-08	2.265e-07

- Approach 3 performs the best among all parallel approaches – will use only this for further comparison

Alternatives to SVD

- SVD is expensive and hard to parallelize
- Good alternatives to SVD: QR factorization with column pivoting (QRCP), randomized SVD (RSVD)

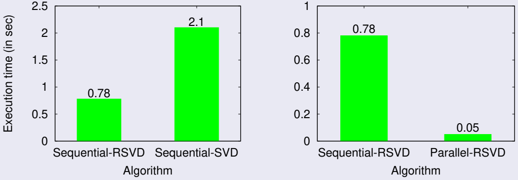
SVD vs QRCP+SVD vs RSVD for *Log* tensor

Approach	Rank	compr	ne	Sequential-OA	Parallel-OA
SVD	5	99.994	992	6.079e-06	6.079e-06
QRCP+SVD				1.016e-05	1.384e-05
RSVD				6.079e-06	6.079e-06
SVD	6	99.992	1376	1.323e-07	1.340e-07
QRCP+SVD				3.555e-07	5.737e-07
RSVD				1.322e-07	1.322e-07
SVD	7	99.989	1824	2.753e-09	2.279e-08
QRCP+SVD				6.620e-09	1.167e-08
RSVD				2.760e-09	2.774e-09

Performance Comparison

Sequential performance with *Log* tensor

Number of computations for both RSVD algorithms $= \mathcal{O}(n^d)$



Parallel performance counts along the critical path on P processors

Algorithm	# Computations	Communications	# Messages
Sequential-RSVD	$\mathcal{O}(\frac{n^d}{P})$	$\mathcal{O}(\frac{n^{d-1}}{\sqrt{P}} \log P)$	$\mathcal{O}(d \log P)$
Parallel-RSVD	$\mathcal{O}(\frac{n^d}{P})$	$\mathcal{O}(\frac{n^{\frac{d}{2}}}{\sqrt{P}} \log P)$	$\mathcal{O}(\log d \log P)$

Table of Contents

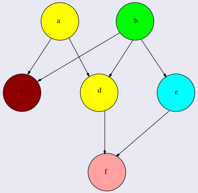
Heterogeneous Systems & Task Based Runtime Systems

Share of Accelerators: TOP500 list

Year	#Systems	% Performance share
2015	103	28
2020	147	43

Task Based Runtime Systems

- Almost impossible to develop optimized hand tune kernels for all architectures
- Task based runtime systems, Eg: StarPU, StarSs, SuperMatrix, QUARK, PaRSEC



- Application is represented as a Direct Acyclic Graph (DAG)
- Vertices represent tasks (computations)
- Edges represent dependencies among tasks

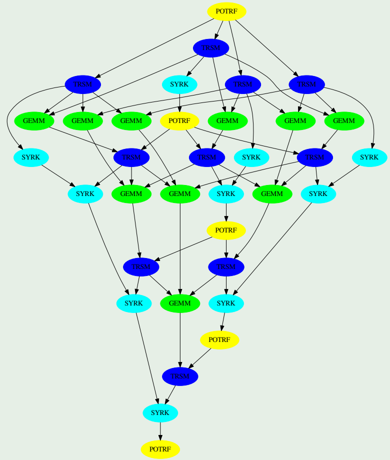
Tiled Cholesky Factorization

Input: a positive definite matrix, A with N $\times$ N tiles
Output: a lower triangular matrix L, A $=$ LL^T

Algorithm 1 Tiled Cholesky factorization

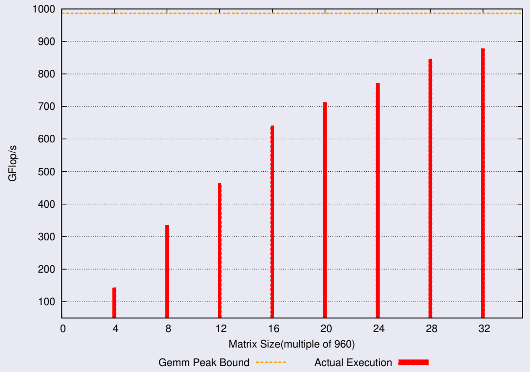
for $k = 0$ to $N-1$ **do**
  L[k][k] $\leftarrow$ POTRF(A[k][k])
  for $i = k+1$ to $N-1$ **do**
    L[i][k] $\leftarrow$ TRSM(L[k][k], A[i][k])
  end for
  for $j = k+1$ to $N-1$ **do**
    A[j][j] $\leftarrow$ SYRK(L[j][k], A[j][j])
    for $i = j+1$ to $N-1$ **do**
      A[i][j] $\leftarrow$ GEMM(L[i][k], L[j][k], A[i][j])
    end for
  end for
end for

Tiled Cholesky Task Graph (N=5)



Performance vs Bounds of Cholesky Factorization

StarPU scheduler performance



Platform description

- Heterogeneous platform description
  - 12 CPU cores (9 compute cores)
  - 3 GPUs
  - Theoretical peak = 1641 GFlop/s
  - GEMM peak = 981 GFlop/s
- Performance and bound gap is significant

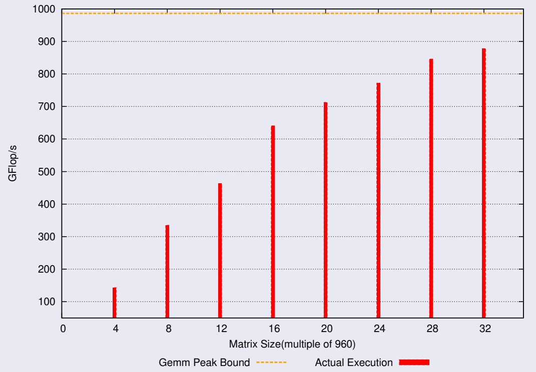
POTRF	TRSM	SYRK	GEMM
$\simeq 2\times$	$\simeq 11\times$	$\simeq 26\times$	$\simeq 29\times$

Table: GPUs relative speedup (TileSize=960)

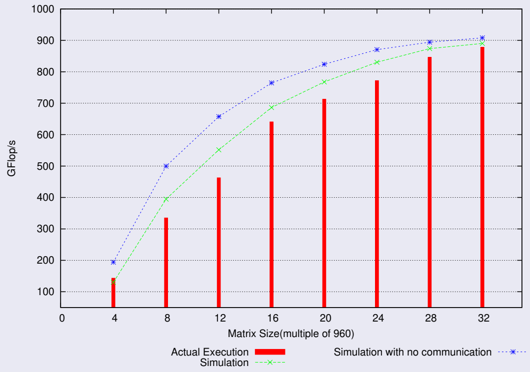
- Goal: propose approaches to enhance performance bounds and better scheduling strategies

Impact of Communication on Cholesky Performance

StarPU scheduler performance



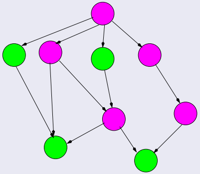
Impact of communication



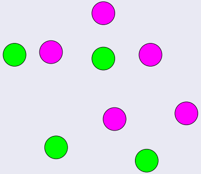
- Goal: propose approaches to enhance performance bounds and better scheduling strategies

Iterative Performance Bound

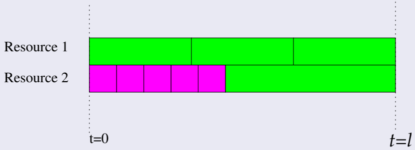
DAG



No Dependencies

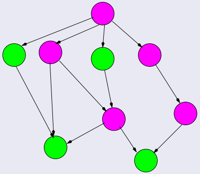


Minimum execution time (minimize l)

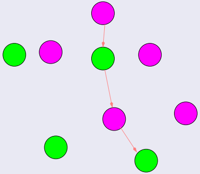


Iterative Performance Bound

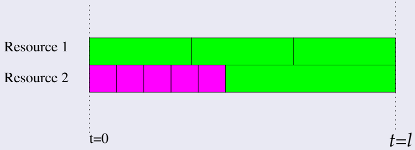
DAG



Some Dependencies



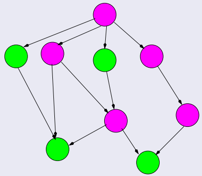
Minimum execution time (minimize l)



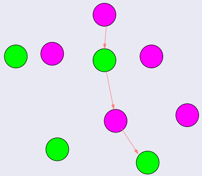
- ⋆ If any path in DAG is larger than l

Iterative Performance Bound

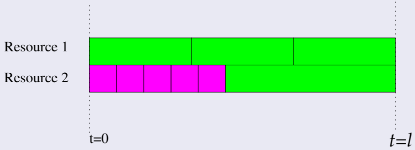
DAG



Some Dependencies

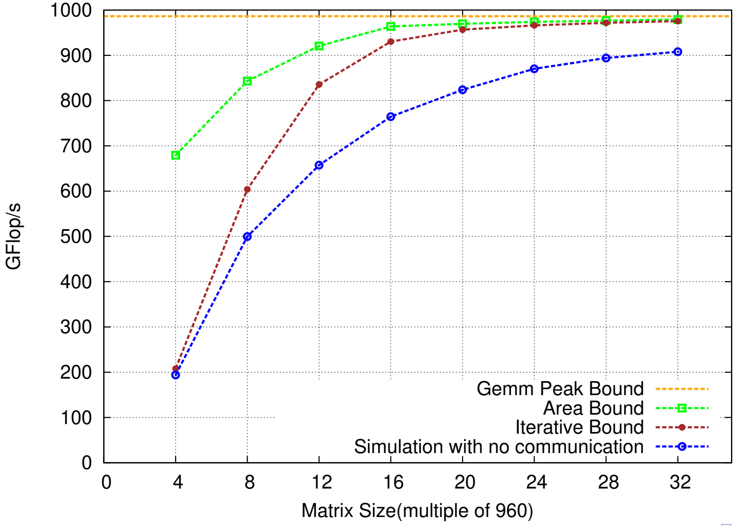


Minimum execution time (minimize l)



- ⋆ If any path in DAG is larger than l
  - add this path as a constraint and repeat the procedure

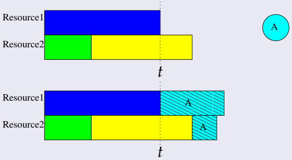
Comparison of Simulated Performance and Bounds



Scheduling strategies

Heft Scheduler [Topcuoglu et al., 2002]

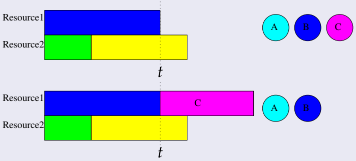
- Task centric scheduler and based on heterogeneous early finish time heuristic



- A is highest priority ready task at t
- Completion time on resource2 is minimum
- Task A is assigned to resource2

HeteroPrio Scheduler [Agullo et al., 2016]

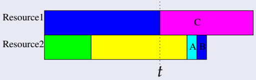
- Resource centric scheduler and based on tasks heterogeneity factors



- A is highest priority ready task at t
- Resource1 is best suited to task C
- Resource1 selects task C

HeteroPrio Scheduler

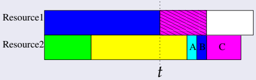
- Resource2 completes tasks A and B
- C is running on resource1, wchich may be much faster on resource2



- Nothing prevents the slow resource to execute a long task
- Suitable for small independent tasks

Generalization of HeteroPrio

- Spoliation: An idle resource restarts the highest priority task if it finishes earlier
- Resource2 spoliates task C

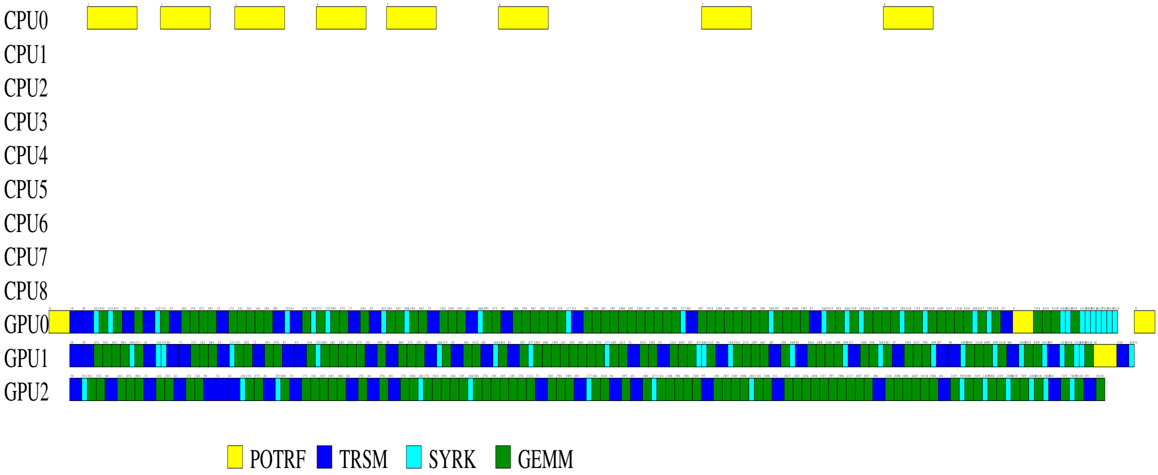


Other corrections to HeteroPrio:

- CPU selects lowest priority task among all tasks of same acceleration factor
- GPU selects highest priority task among all tasks of similar acceleration factors

heft Scheduler

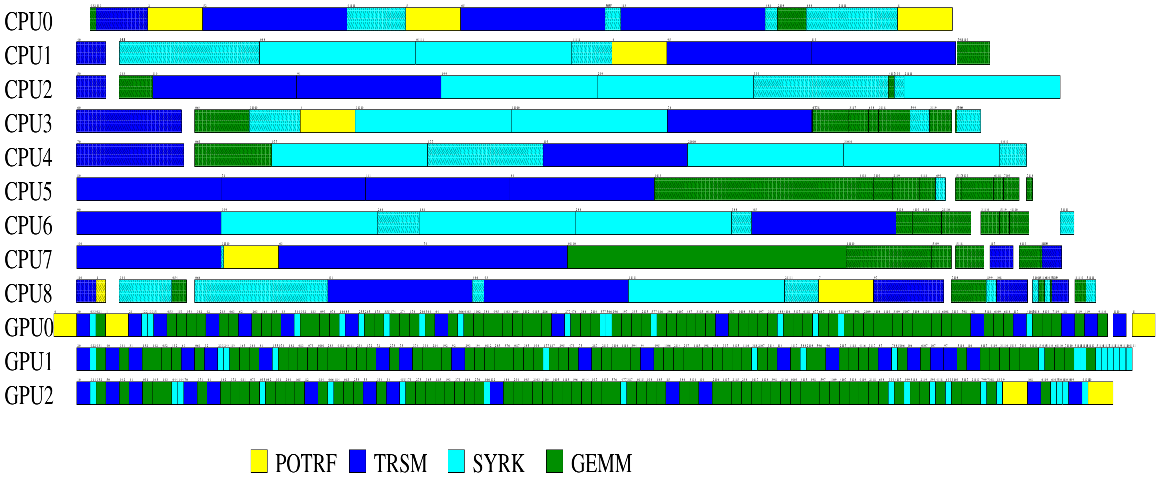
heft trace for 12 X 12 blocks of Cholesky Factorization



- Most of the CPU resources are not utilized (686 GFlop/s)

HeteroPrio scheduler

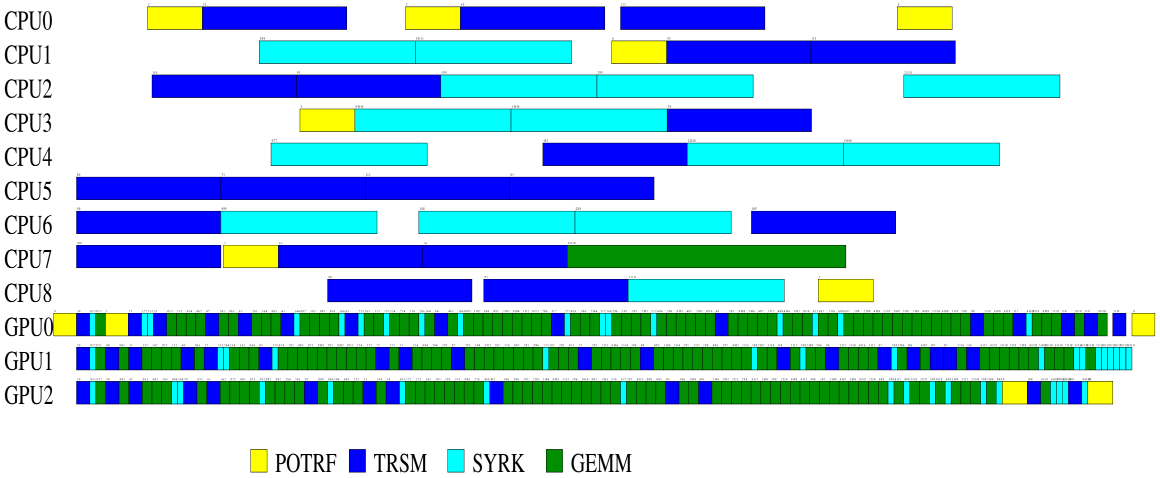
HeteroPrio trace for 12 X 12 blocks of Cholesky Factorization



Achieved performance = 760 GFLop/s, heft performance = 686 GFlop/s

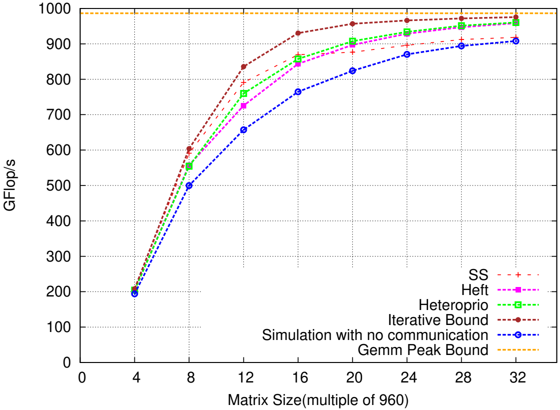
HeteroPrio scheduler

HeteroPrio trace for 12 X 12 blocks of Cholesky Factorization



Achieved performance = 760 GFLop/s, heft performance = 686 GFlop/s

Heft-Vs-HeteroPrio-Vs-Bound



- SS: static schedule obtained with constraint programming

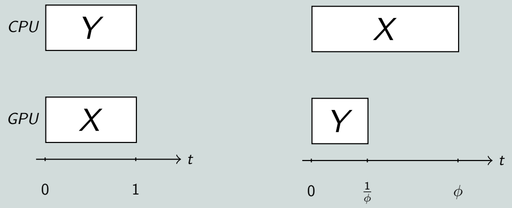
HeteroPrio Approximation Ratios and Worst Case Example Ratios

(#CPUs, #GPUs)	set of independent tasks		task graphs	
	Approximation ratio	Worst case ex.	Approximation ratio	Worst case ex.
(1,1)	$\frac{1+\sqrt{5}}{2}$	$\frac{1+\sqrt{5}}{2}$	2	2
(m,n)	$2+\sqrt{2} \approx 3.41$	$2+\frac{2}{\sqrt{3}} \approx 3.15$	$2+\max(\frac{m}{n}, \frac{n}{m})$	$1+\max(\frac{m}{n}, \frac{n}{m})$

1 CPU 1 GPU HeteroPrio Worst Case Example with two independent tasks

Task	CPU Time	GPU Time	accel ratio
X	ϕ	1	ϕ
Y	1	$\frac{1}{\phi}$	ϕ

Where $\phi = \frac{1+\sqrt{5}}{2}$



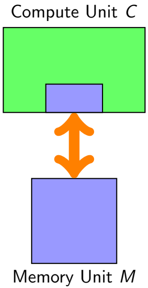
Optimal schedule HeteroPrio schedule

Table of Contents

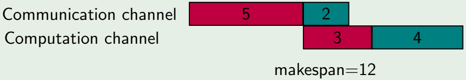
Problem Definition

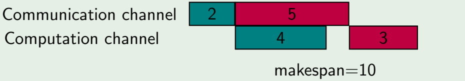
Task	Data Transfer Time	Computation Time
A	5	3
B	2	4

Problem: Given a set of tasks in what order we transfer them from M to C such that the makespan is minimal?



Possible schedules





Order of Data Transfers

- Compute intensive task: Computation time $\geq$ Data transfer time
- Communication intensive task: Computation time $<$ Data transfer time

Optimal Algorithm: When memory capacity of the compute unit is not a concern

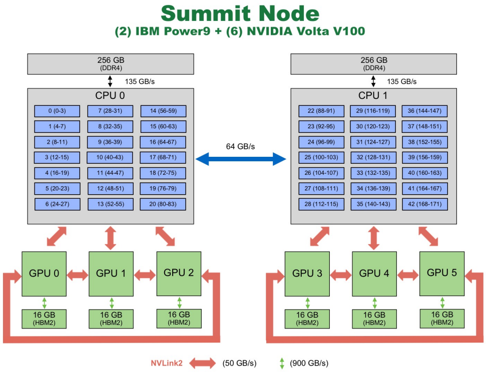
- First sort compute intensive tasks in increasing order of their data transfer time
- Then sort the communication intensive tasks in decreasing order of their computation time

Memory capacity is limited

- Proved that the problem is NP-Complete
- Proposed static and dynamic based approaches and evaluated them on traces of molecular simulations
  - Static approaches: order is computed in advance
  - Dynamic approaches: next task is chosen based on the heuristic
  - Combination of both: start with static order and switch to dynamic based on available memory

Performance evaluation on Summit supercomputer

- Implemented static approaches in Tensor Algebra for Manybody Methods (TAMM) library
- CCSD application, Ubiqtin molecule, cc-pVDZ (737 basis functions, 220 nodes), aug-cc-pVDZ (1243 basis functions, 256 nodes)



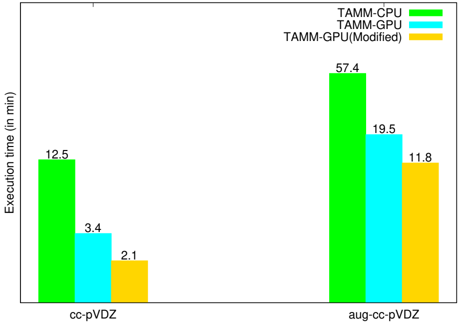


Fig source: `https://www.olcf.ornl.gov`

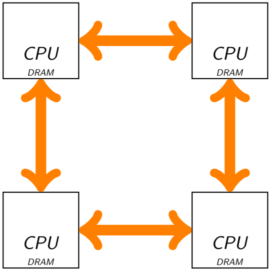
Part II

Proposed Plan

Overview

Communication and its importance in HPC

- Running time of an algorithm depends on
  - Computations
    - Number of operations * time-per-operation
  - Data movement
    - Volume of communication / Network-bandwidth
    - Number of messages * Network-latency



- Gaps growing exponentially with time (Source: Getting up to speed: The future of supercomputing)

	time-per-operation	Network-bandwidth	Network-latency
Annual improvements	59 %	26 %	15 %

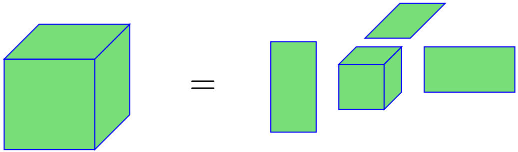
- Avoid communication to save time (and energy)

Table of Contents

Popular Tensor Algorithms

- Determine the communication lower bounds for tensor decompositions
- Analyse the popular decomposition algorithms and communications performed by them
- Propose new scalable communication optimal algorithms
  - If possible design tiles/tasks based algorithms
- Implement the proposed algorithms
  - Handle performance issues for homogeneous systems
    - Load balancing
    - Memory aware approaches
    - scheduling strategies
- Same for manipulation operations of popular tensor representations
- Extend implementation for heterogeneous systems (start with Nvidia GPUs based heterogeneous systems)
- Create a tensor library

Tucker Decomposition



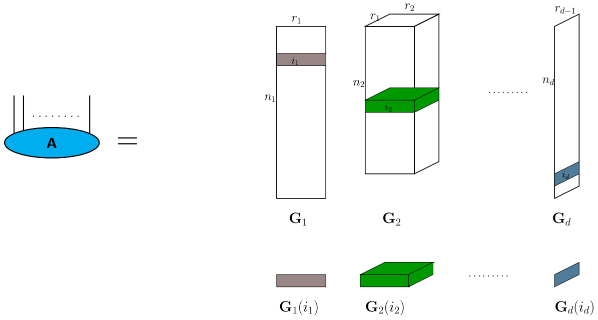
- Determine communication lower bounds for this operation
- Analyse communication performed by higher-order singular value decomposition (HOSVD) algorithm

HOSVD Algorithm

1. Obtain factor matrices by computing SVD of all unfolding matrices of the tensor
2. Obtain core tensor by multiply factor matrices with the original tensor

- Propose new scalable communication algorithms and implement them

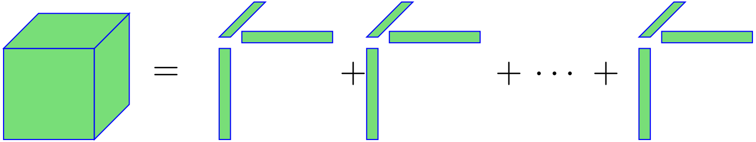
Tensor Train Decomposition



$$\mathbf{A}(i_1, i_2, \cdots, i_d) = \mathbf{G}_1(i_1)\mathbf{G}_2(i_2)\cdots\mathbf{G}_d(i_d)$$

- Determine communication lower bounds for this operation
- Analyse communication performed by the popular algorithm:Completed
- Propose new scalable communication algorithms and implement them

Canonical Decomposition



- No deterministic algorithm which guarantees to find the decomposition when it exists
- Analyse one iteration of the popular existing algorithms
- Matricized tensor times Khatri-Rao product (MTTKRP) is the most time consuming operation
    - MTTKRP: $(\mathcal{X}, \{A_1, \cdots, A_{k-1}, A_{k+1}, \cdots, A_d\}) \longrightarrow A_k$
- Determine communication lower bounds for MTTKRP operation
- Propose and implement scalable algorithms for this operation

Proving Communication Lower Bounds

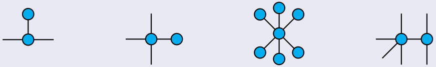
How people did it for linear algebra operations?

- People obtain results for matrix multiplication operations
- Same lower bound apply to almost all direct linear linear algebra operations using reduction [Ballard et. al., 09] , for instance, bound for LU factorization

$$\begin{pmatrix} I & & -B \\ A & I & \\ & & I \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ & & I \end{pmatrix} \begin{pmatrix} I & & -B \\ & I & AB \\ & & I \end{pmatrix}$$

Approach to compute lower bounds for tensor computations

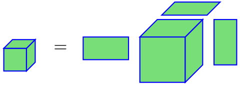
- Obtain bounds for basic tensor operations: Tensor times matrix (TTM), Multiple tensor times matrix (Multi-TTM), Tensor contraction



- Express decompositions and manipulations in terms of these basis operations

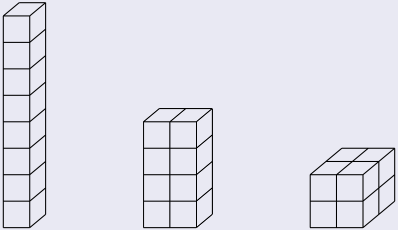
Table of Contents

Communication lower bound of 3-dimensional Multi-TTM computation



- It is an ongoing work
- We revisited lower bounds for matrix multiplication
- Our goal was to express existing approaches in the form suitable for tensors

Different arrangements of 8 processors



Communication lower bounds for Matrix Multiplications on P Processors

- Each processor asymptotically performs the same amount of computation
- One copy of data is distributed among processor

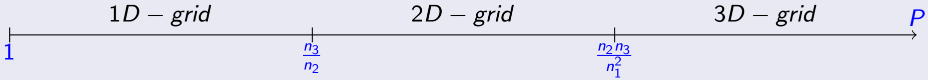
Square matrix multiplication

- Communication optimal algorithms consider 3-dimensional processor arrangement
- Rediscovered many times in literature

Rectangular matrix multiplication $C = AB$

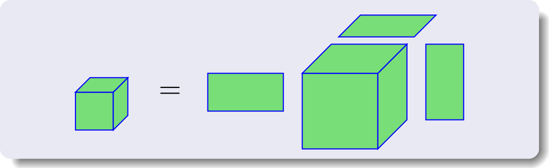
Assume $n_1 \leq n_2 \leq n_3$ and dimensions of A, B, and C are $n_1 \times n_2$, $n_2 \times n_3$ and $n_1 \times n_3$.

- Lower bounds depend on the dimensions of the matrices
- Requires to solve a linear program
- Lower bounds also instruct the arrangement of processors in optimal algorithms



- Corrected the constants in the existing ranges of P (Demmel et.al [IPDPS 2013])

Let **G** and **X** are output and input tensors. A, B and C are factor matrices, which are multiplied to the input tensor in 1st, 2nd and 3rd dimensions. Sizes of **G**, **X**, A, B, C are $r_1 \times r_2 \times r_3$, $n_1 \times n_2 \times n_3$, $n_1 \times r_1$, $n_2 \times r_2$ and $n_3 \times r_3$ respectively. Sequential code for this computation can be written as..



for $i_1 = 1 : n_1$ **do**
  for $i_2 = 1 : n_2$ **do**
    for $i_3 = 1 : n_3$ **do**
      for $j_1 = 1 : r_1$ **do**
        for $j_2 = 1 : r_2$ **do**
          for $j_3 = 1 : r_3$ **do**
            $\mathbf{G}(j_1, j_2, j_3) += \mathbf{X}(i_1, i_2, i_3) * A(i_1, j_1) * B(i_2, j_2) * B(i_3, j_3)$
          end for
        end for
      end for
    end for
  end for
end for

Lower communication bounds for 3-dimensional Multi-TTM

- We apply similar approach for this computation
- Lower bounds suggest to consider the following arrangements of processors for the optimal algorithms

Processor Range	Type of Arrangement
$P < \min\left(\frac{n_3 r_3}{n_2 r_2}, \frac{n_1 n_2 n_3}{r_1 r_2 r_3}\right)$	1D-grid
$\frac{n_1 n_2 n_3}{r_1 r_2 r_3} < P \leq \frac{n_3 r_3}{n_2 r_2}$	2D-grid
$\frac{n_3 r_3}{n_2 r_2} \leq P < \min\left(\frac{n_1 n_2 n_3}{r_1 r_2 r_3}, \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}\right)$	2D-grid
$\max\left(\frac{n_3 r_3}{n_2 r_2}, \frac{n_1 n_2 n_3}{r_1 r_2 r_3}\right) \leq P < \frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}$	4D-grid
$\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2} \leq P < \frac{n_1 n_2 n_3}{r_1 r_2 r_3}$	3D-grid
$\max\left(\frac{n_2 n_3 r_2 r_3}{n_1^2 r_1^2}, \frac{n_1 n_2 n_3}{r_1 r_2 r_3}\right) \leq P$	6D-grid

Table of Contents

2×2 recursive Matrix Multiplication $C = AB$

- Matrix is divided into 2×2 blocks

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Traditional Algorithm

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Operation count recurrence,

$$T(n) = 8T\left(\frac{n}{2}\right) + \mathcal{O}(n^2),\ T(n) = 1$$

After solving, we obtain $T(n) = \mathcal{O}(n^3)$

Strassen's Algorithm

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22})B_{11} \\ M_3 &= A_{11}(B_{12} - B_{22}) \\ M_4 &= A_{22}(B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12})B_{22} \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

Operation count recurrence, $T(n) = 7T\left(\frac{n}{2}\right) + \mathcal{O}(n^2),\ T(n) = 1$

After solving, we obtain $T(n) = \mathcal{O}(n^{2.81})$

2×2 Matrix multiplication as a tensor operation

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

We can write this multiplication as a tensor operation,

$$\mathbf{T} \times_1 \begin{pmatrix} A_{11} \\ A_{12} \\ A_{21} \\ A_{22} \end{pmatrix} \times_2 \begin{pmatrix} B_{11} \\ B_{12} \\ B_{21} \\ B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} \\ C_{12} \\ C_{21} \\ C_{22} \end{pmatrix}$$

Where $\mathbf{T}$ is a $4 \times 4 \times 4$ tensor with the following slices:

$$T_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} T_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad T_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} T_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Canonical rank of $\mathbf{T} = 7$, which determines the complexity of Strassen's algorithm

Tensor contractions

$\mathbf{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_{d_1}}$, $\mathbf{B} \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_{d_2}}$ and we want to compute $\mathbf{A} \times_{n_i} \mathbf{B}$ where $n_i = m_j$.

- $d_1 = 3,\ d_2 = 2,\ n_1 = n_2 = n_3 = m_1 = m_2 = n,\ i = 3$, and $j = 1$
- Output tensor is $\mathbf{G}$ and its all elements are initialized to zero

for $i_1 = 1 : n$ **do**
  for $i_2 = 1 : n$ **do**
    for $i_3 = 1 : n$ **do**
      for $j_2 = 1 : n$ **do**
        $\mathbf{G}(i_1, i_2, j_2) = \mathbf{G}(i_1, i_2, j_2) + \mathbf{A}(i_1, i_2, i_3) * \mathbf{B}(i_3, j_2)$
      end for
    end for
  end for
end for

- Total $\mathcal{O}(n^4)$ operations
- Applying Strassen's algorithm for each i_1, total $\mathcal{O}(n^{3.81})$ operations

Tensor contractions with Strassen's concept

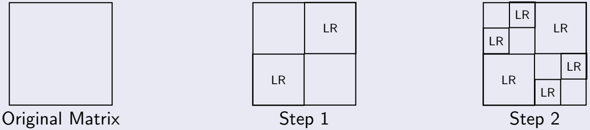
- Expressing this computation as canonical low rank decomposition of $8 \times 8 \times 4$ tensor can further reduce the number of operations

$$\mathbf{T} \times_1 \begin{pmatrix} \mathbf{A}_{111} \\ \mathbf{A}_{112} \\ \mathbf{A}_{121} \\ \mathbf{A}_{122} \\ \mathbf{A}_{211} \\ \mathbf{A}_{212} \\ \mathbf{A}_{221} \\ \mathbf{A}_{222} \end{pmatrix} \times_2 \begin{pmatrix} B_{11} \\ B_{12} \\ B_{21} \\ B_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{G}_{111} \\ \mathbf{G}_{112} \\ \mathbf{G}_{121} \\ \mathbf{G}_{122} \\ \mathbf{G}_{211} \\ \mathbf{G}_{212} \\ \mathbf{G}_{221} \\ \mathbf{G}_{222} \end{pmatrix}$$

Hierarchical Matrix concepts to Tensors

Hierarchical Matrices

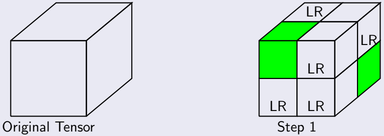
- Data sparse approximation of non-sparse matrices



LR: low rank block

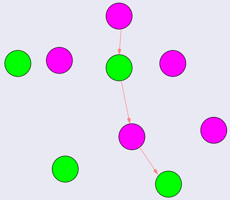
Tensors

- $f(i,j,k) = \frac{1}{|i-j|+|j-k|+|k-i|}$
- Value is small if difference of any pair is large
- Formalize and evaluate this approach for tensors

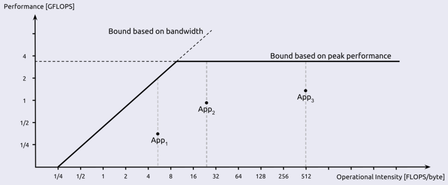


Roofline model with Dependencies

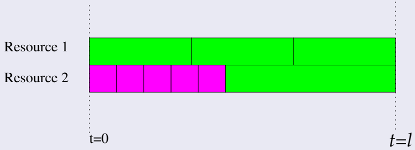
Iterative bound



Roofline Model



Minimum execution time (minimize l)



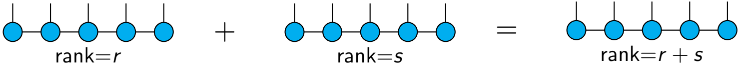
- ⋆ If any path in DAG is larger than l
  - add this path with data transfer cost as a constraint and repeat the procedure

- Take minimum of both values as the lower bound of the application

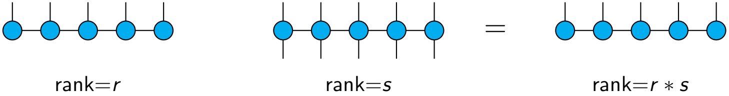
Table of Contents

New Tensor Representations

- Tensor Train is the popular representation to work with high dimensional tensors
- Add tensors in this representation

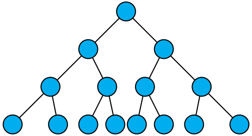


- Applying an operator in this representation

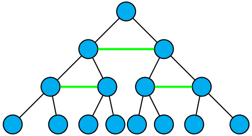


- Requires a truncation process
  - Iterate over cores one by one
- This representation is not much suited to work in parallel

- Look at new representations in tree format – suitable for parallelization



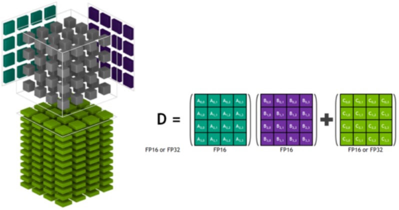
or

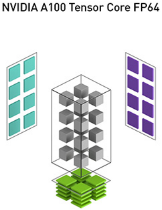


  - Data will be stored at the leaf nodes
  - Interal nodes will help to manipulate tensors in parallel

- Some tensor representations exhibit tree structure
  - Mainly designed to reduce storage or model long range interactions
  - Not suitable to work with them in parallel

Architecture Aware Algorithms



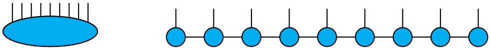


- Most linear algebra computations do not take advantages of these units
- Design algorithms which take architecture details into account

Fig source: `www.nvidia.com`

Randomization in Tensor Computations

- Randomized SVD and UTV factorization are now well established
- Apply randomization to tensors
- Perform factorizations of tensors
    - For example: QR like factorization of tensor



Integration in the Team

- **Bora Ucar**: Expert in sparse tensor computations. Work with him on the design and implementation of scalable algorithms
- **Loris Marchal**: Expert in the design of memory and communication oriented algorithms. Work with him on extending the roofline model with dependencies and memory aware algorithms to schedule tasks on HPC platforms
- **Anne Benoit**, **Yves Robert** and **Frederic Vivien**: Experts in designing parallel algorithms and scheduling strategies. Work with them on the design of scalable parallel algorithms and scheduling strategies for homogeneous/heterogeneous systems
- **Grgoire Pichon**: Works on low rank compression of matrices. Work with him on extending the concept of hierarchical matrices to tensors

Bringing additional skills in the team

- Designing strategies to work with high dimensional tensor computations
- Determining communication lower bounds for linear algebra computations
- Designing scalable approaches for large HPC systems
- Familiar with molecular and quantum simulations and how tensors are used in these domains

Thank You!