Scalable Tensor Algorithms for Modern Computing Systems

Suraj KUMAR[1]

[1] Inria Paris, France

May 21, 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

Part I

Past/Ongoing Work

Collaborators

This is joint work with ...

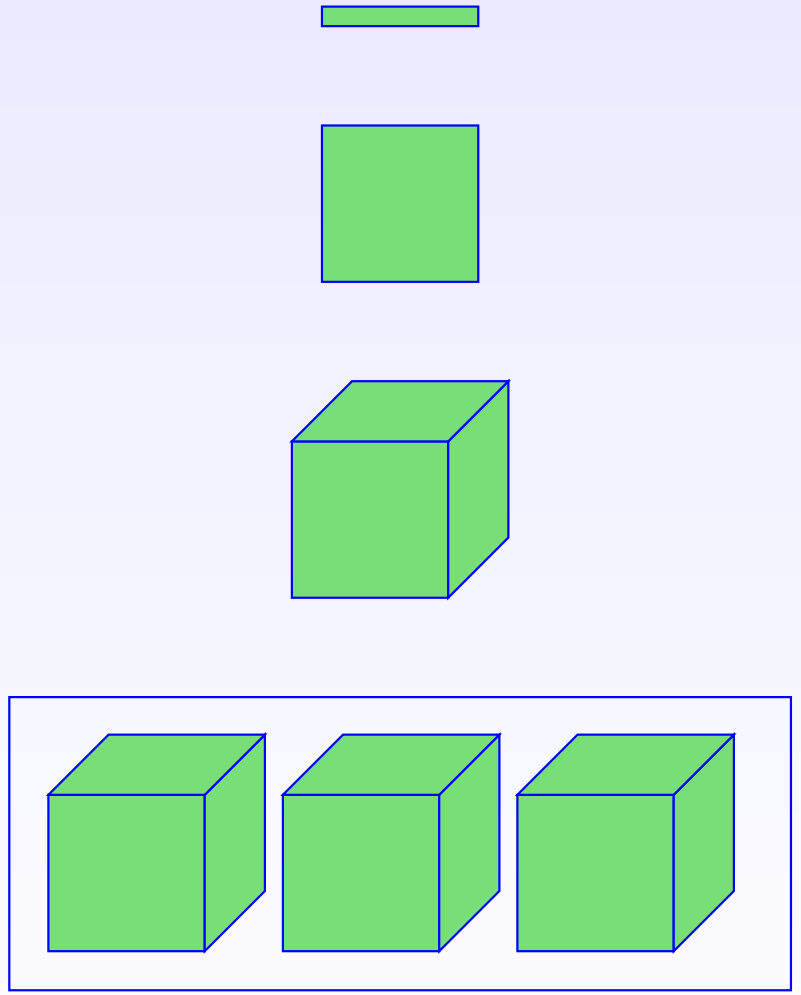
- Laura Grigori – Inria Paris, France
- Grey Ballard – Wake Forest University, USA
- Hussam Al Daas – STFC Rutherford Appleton Laboratory, UK
- Olivier Beaumont – Inria Bordeaux, France
- Sriram Krishnamoorthy – Pacific Northwest National Laboratory, USA
- Marcin Zalewski – Nvidia, USA
- Lionel Eyraud-Dubois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

Overview

1. Tensors
   - Communication Optimal Algorithms for Tensors
2. Scheduling on Heterogeneous Resources
   - Communication Computation Overlap
   - Scheduling of Dense Linear Algebra Kernels

Tensors: Multidimensional Arrays

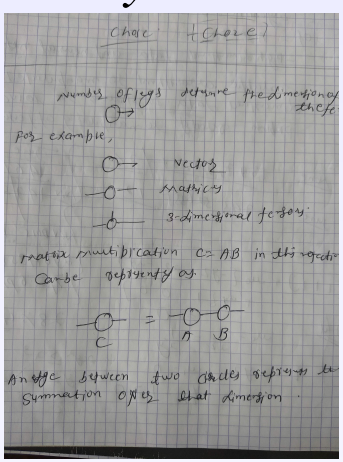
Dimension	Name
1	Vector
2	Matrix
3	3-dimensional tensor
4	4-dimensional tensor



Tensor Diagram Notations

Tensorsaredenotedbysolidshapesaandnumberoflinescomingoutoftheshapesdeno

For example,

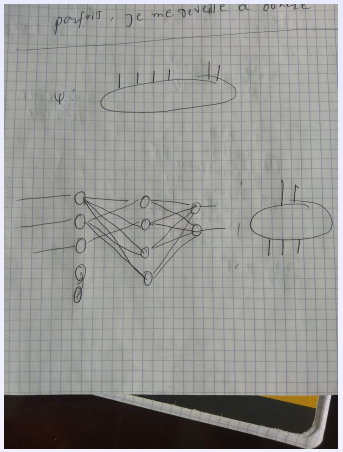


- Connecting two lines implies summation over the connected dimensions

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

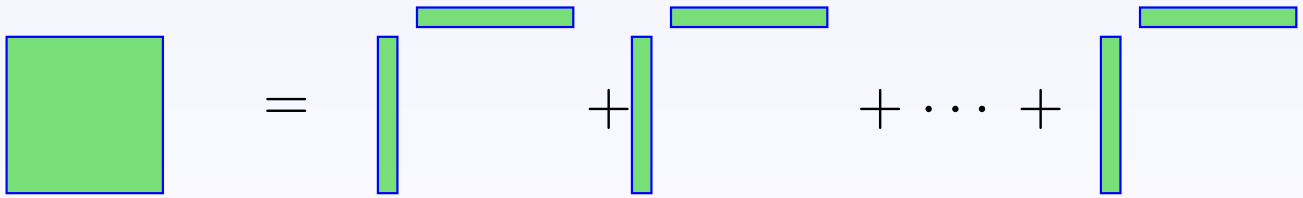
- **Neural Network**: 
- **Quantum or Molecular Simulation**: Approximation of ψ in the Schrodinger equation, $\hat{H}\psi = E\psi$. $\hat{H}$ is an operator which corresponds to the total energy of the system. ψ represents the quantum state of the system. It is a huge tensor with 2^n elements where n is the number of electrons in the system or with 4^k elements where k is the number of spatial orbitals in the consideration.

Tensors

- 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

Singular Value Decomposition (SVD) of Matrices

- It decomposes a matrix $A \in \mathbb{R}^{m \times n}$ to the form $U \Sigma V^T$
  - U is an $m \times m$ orthogonal matrix
  - V is an $n \times n$ orthogonal matrix
  - Σ is an $m \times n$ rectangular diagonal matrix
- It represents a matrix as the sum of rank one matrices
  - $A = \sum_i \Sigma(i;i) U_i V_i^T$
  - Minimum number of rank one matrices required in the sum is called the rank of the original matrix

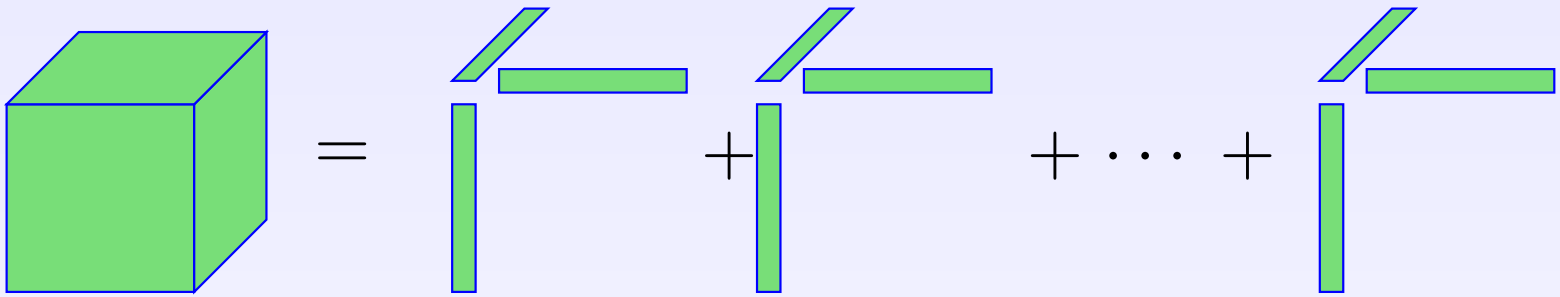


- Canonical decomposition (equivalently known as Canonical Polyadic or CANDECOMP or PARAFAC)
- Tucker decomposition
- Tensor Train decomposition (equivalently known as Matrix Product States)

Tensor Notations

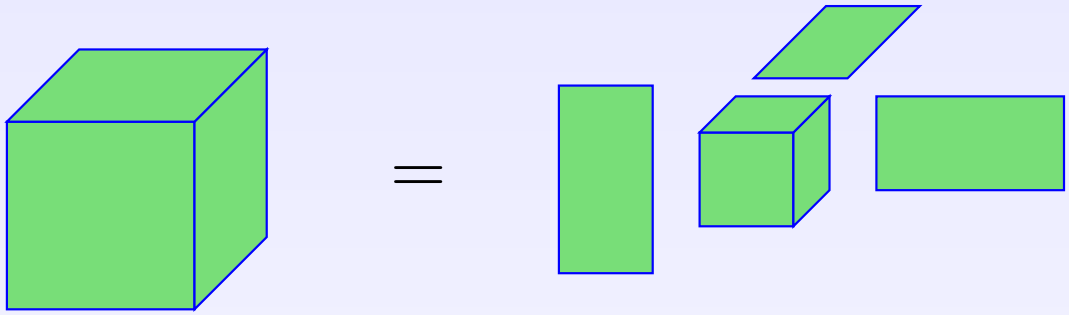
- $\mathbf{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ is a d-dimensional tensor
- $\mathbf{A}(i_1, \cdots, i_d)$ represent elements of $\mathbf{A}$
- Use bold letters to denote tensors

Canonical Representation



- $\mathbf{A}(i_1, \cdots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(i_2, \alpha) \cdots U_d(i_d, \alpha)$
- (+) For $n_1 = n_2 = \cdots n_d = n$, the number of entries = $\mathcal{O}(nrd)$
- (-) Determining the minimum value of r is an NP-complete problem
- (-) No robust algorithms to compute this representation

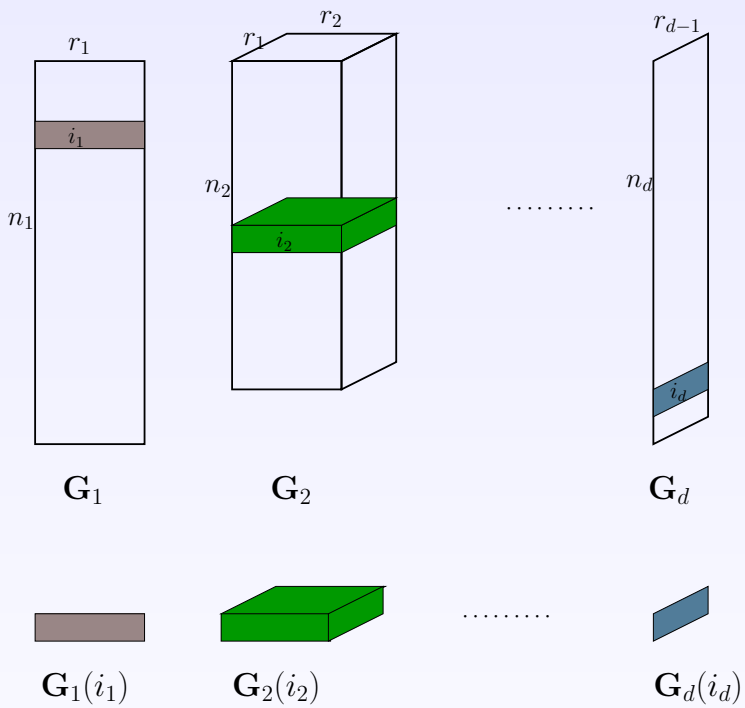
Tucker Representation



- Represents a tensor with d matrices and a small core tensor
- $\mathbf{A}(i_1, \cdots, i_d) = \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_d=1}^{r_d} \mathbf{g}_{\alpha_1 \cdots \alpha_d} U_1(i_1, \alpha_1) \cdots U_d(i_d, \alpha_d)$
- (+) SVD based stable algorithms to compute this representation
- (-) For $n_1 = n_2 = \cdots n_d = n$ and $r_1 = r_2 = \cdots = r_d = r$, the number of entries = $\mathcal{O}(ndr + r^d)$

Tensor Train Representation: Product of Matrices View

- A d-dimensional tensor is represented with 2 matrices and d-2 3-dimensional tensors.



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

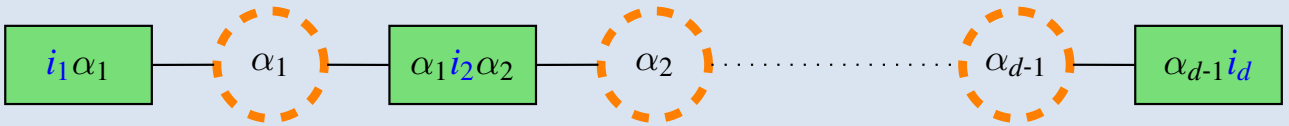
An entry of $\mathbf{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is computed by multiplying corresponding matrix (or row/column) of each core.

Tensor Train Representation

$\mathbf{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is represented with cores $\mathbf{G}_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$, k=1, 2, $\cdots d$, r_0=r_d=1 and its elements satisfy the following expression:

$$\mathbf{A}(i_1, \cdots, i_d) = \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_d=1}^{r_d} \mathbf{G}_1(\alpha_0, i_1, \alpha_1) \cdots \mathbf{G}_d(\alpha_{d-1}, i_d, \alpha_d)$$

$$= \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_{d-1}=1}^{r_{d-1}} \mathbf{G}_1(1, i_1, \alpha_1) \cdots \mathbf{G}_d(\alpha_{d-1}, i_d, 1)$$



- For $n_1 = n_2 = \cdots = n_d = n$ and $r_1 = r_2 = \cdots = r_{d-1} = r$, the number of entries = $\mathcal{O}(ndr^2)$

Table of Contents

Unfolding Matrices of a Tensor & Notations

- 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 **A** is defined as,
  $||\mathbf{A}||_F = \sqrt{\sum_{i_1,i_2,\cdots,i_d} \mathbf{A}(i_1, i_2, \cdots, i_d)^2}$

k-th unfolding matrix

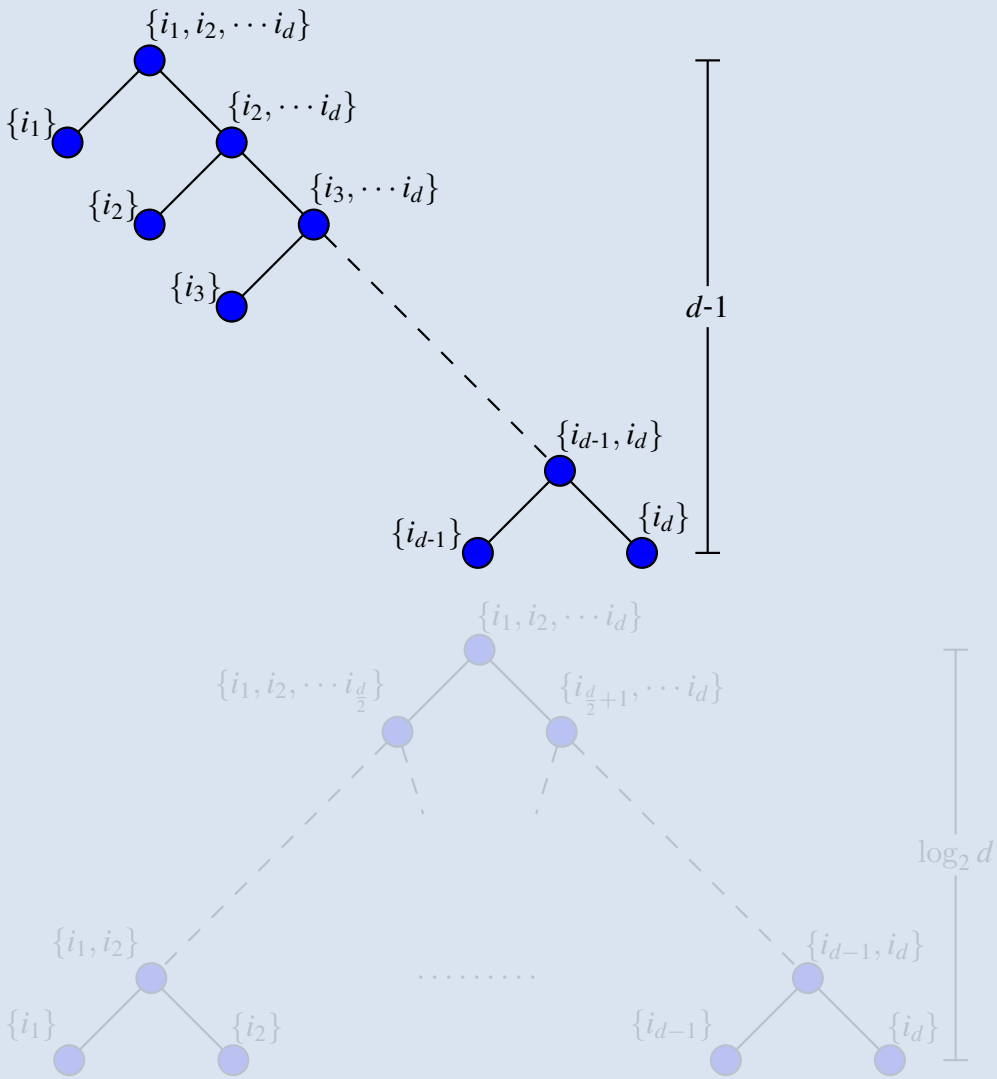
A_k denotes k-th unfolding matrix of tensor $\mathbf{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$.

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

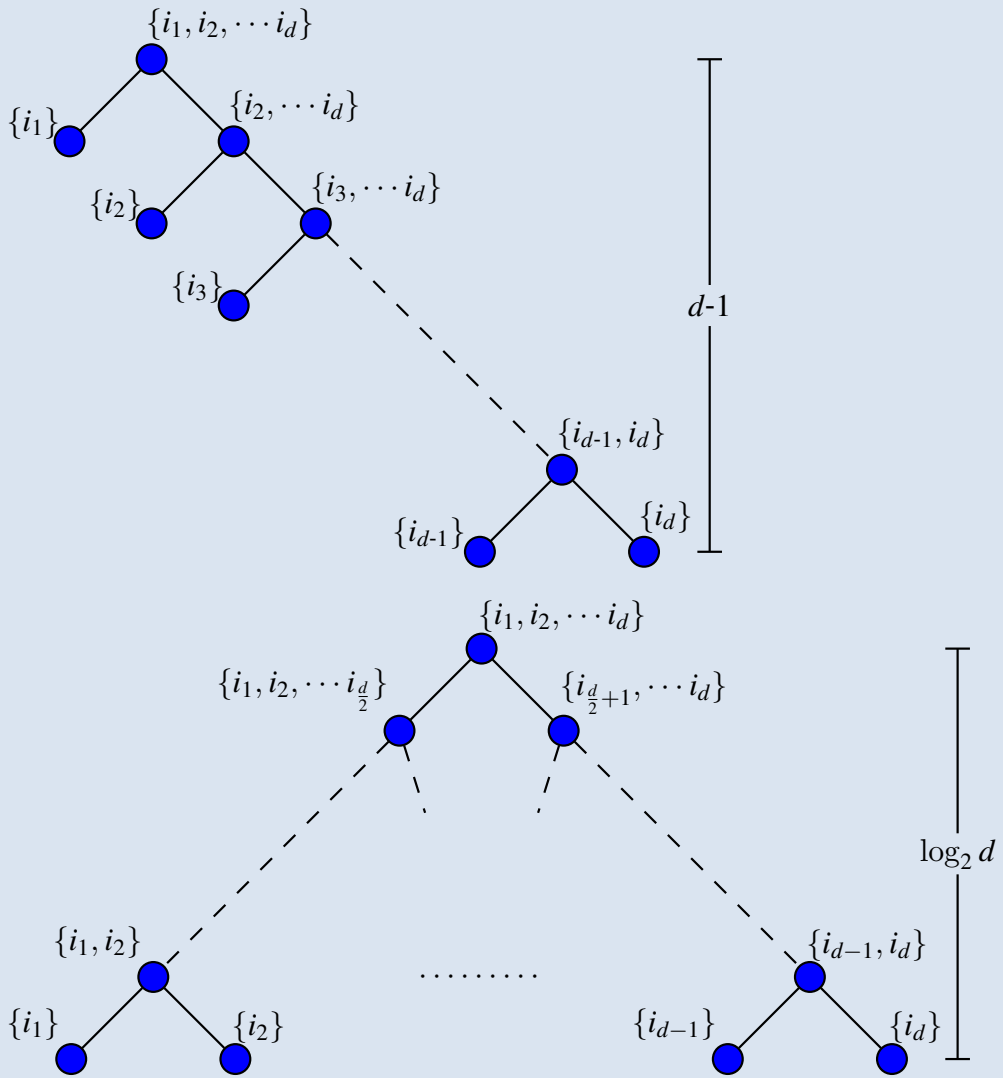
- Size of A_k is $(\prod_{l=1}^{k} n_l) \times (\prod_{l=k+1}^{d} n_l)$
- r_k denotes the rank of A_k.

- $(r_1, r_2, \cdots, r_{d-1})$ denotes the ranks of unfolding matrices of the tensor.

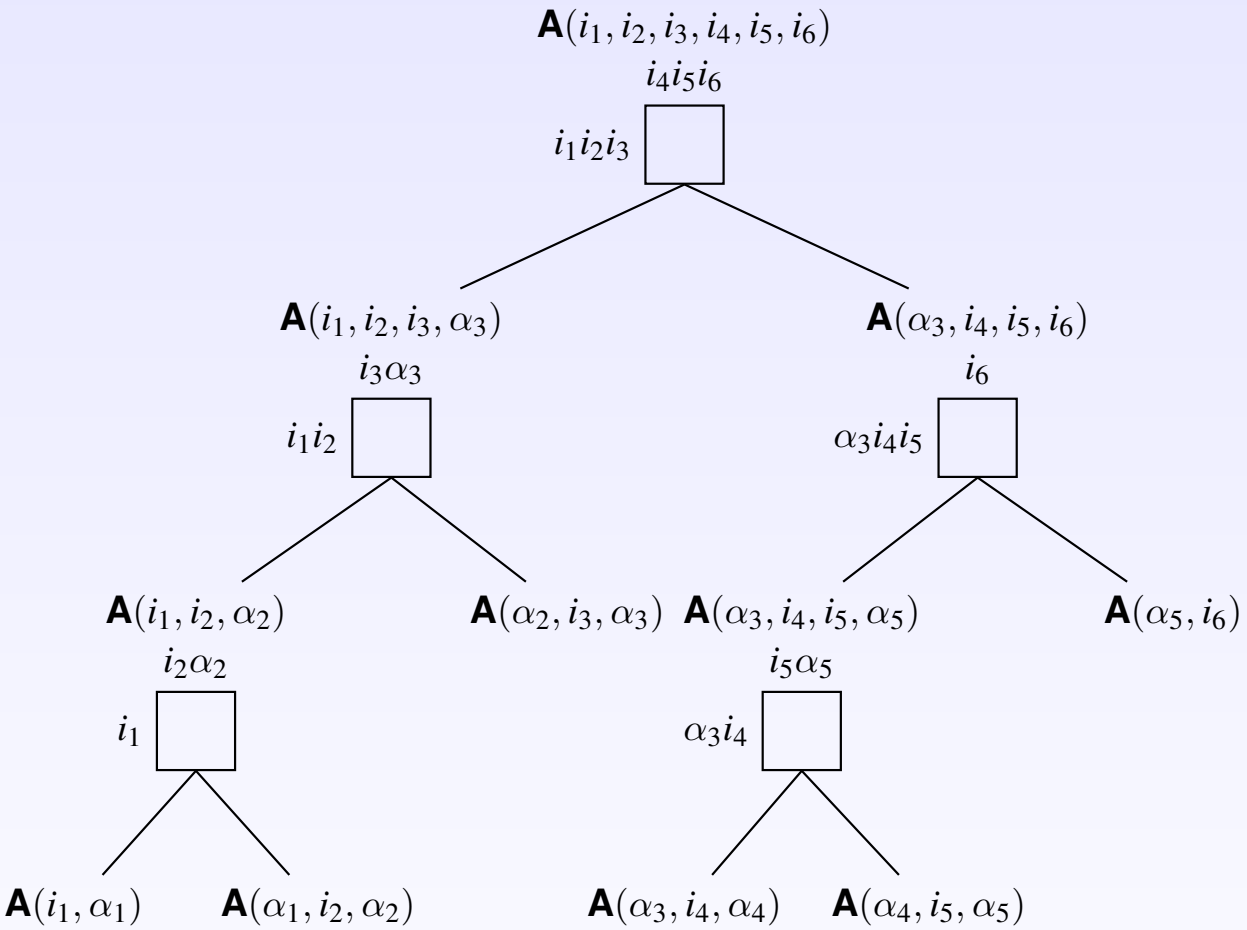
Separatation of Dimensions in Sequential Algorithms



Separatation of Dimensions for Maximum Parallelization



Diagramatic Representation of the Algorithm



Theorem

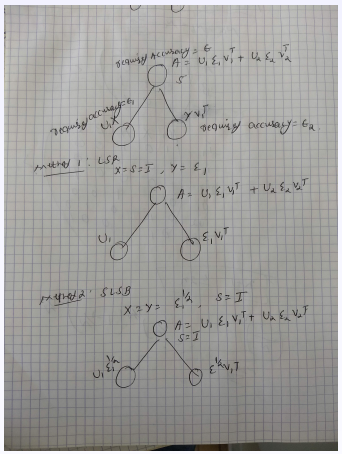
*If for each unfolding A_k of a d-dimensional tensor **A**, $rank(A_k) = r_k$, then our parallel algorithm produces a Tensor Train representation with ranks not higher than*

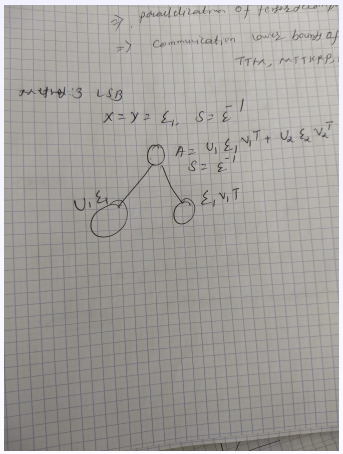
Sequential Tensor Train Approximation

- Uniform truncation of $\Delta = \frac{\epsilon}{\sqrt{d-1}}$ is applied at each step
- Σ_2 is selected based on Δ
- Approximation error of this approach is bounded by ϵ

Parallel Tensor Train Approximation Algorithms

- We choose X, S, and Y such that $XSY = \Sigma_1$
- We assume $\frac{\epsilon_1^2}{d_1^2} = \frac{\epsilon_2^2}{d_2^2}$
- Σ_2 is selected based on $\Delta = \frac{\epsilon}{\sqrt{d-1}}$
- ϵ_1 = f$(\epsilon, \epsilon_2, \Delta, X, S, Y)$





Accuracy of parallel approximation algorithms

- *Leading Singular values to Right subtensor* (*LSR*)
- *Square root of Leading Singular values to Both subtensors* (*SLSB*)
- *Leading Singular values to Both subtensors* (*LSB*)

Low Rank Functions

Log	$\log(\sum_{j=1}^{N} j i_j)$
Sin	$\sin(\sum_{j=1}^{N} i_j)$
Inverse-Square-Root (*ISR*)	$\frac{1}{\sqrt{\sum_{j=1}^{N} i_j^2}}$
Inverse-Cube-Root (*ICR*)	$\frac{1}{\sqrt[3]{\sum_{j=1}^{N} i_j^3}}$
Inverse-Penta-Root (*IPR*)	$\frac{1}{\sqrt[5]{\sum_{j=1}^{N} i_j^5}}$

We consider $N = 12$ and $i_j \in \{1, 2, 3, 4\}_{1 \leq j \leq N}$. This setting produces a 12-dimensional tensor with 4^{12} elements for each low rank function. Her we show results only for *Log* tensor.

Comparison of All Approaches for the Log tensor

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

Approach	compr	ne	OA
STTA	99.993	1212	2.271e-07
LSR	99.817	30632	3.629e-08
SLSB	99.799	33772	2.820e-08
LSB	99.993	1212	2.265e-07

Accuracy Results

- SVD is expensive – we also experimented with QRCP and Randomized SVD

Approach	Rank	compr	ne	OA
SVD	5	99.994	992	6.079e-06
QRCP	5	99.994	992	1.016e-05
RSVD	5	99.994	992	6.0791e-06
SVD	6	99.992	1376	1.323e-07
QRCP	6	99.992	1376	3.555e-07
RSVD	6	99.992	1376	1.3228e-07
SVD	7	99.989	1824	2.753e-09
QRCP	7	99.989	1824	6.620e-09
RSVD	7	99.989	1824	2.7602e-09

Performance Comparison

- Computation cost of the parallel algorithm decreases exponentially at each step
- Each algorithm runs on P processors

Cost along the critical path:

Algorithm	Computation cost	Communication volume	Number of
Sequential	$\mathcal{O}(\frac{N^d}{P})$	$\mathcal{O}(\frac{N^d}{\sqrt{P}} \log P)$	$\mathcal{O}(d \log$
Parallel	$\mathcal{O}(\frac{N^d}{P})$	$\mathcal{O}(\frac{N^d}{\sqrt{P}} \log P)$	$\mathcal{O}(\log$

Cost along the critical path after the first step,

Algorithm	Computation cost	Communication volume	Number of
Sequential	$\mathcal{O}(\frac{N^{d-1}}{P})$	$\mathcal{O}(\frac{N^{d-1}}{\sqrt{P}} \log P)$	$\mathcal{O}(d \log$
Parallel	$\mathcal{O}(\frac{N^{\frac{d}{2}}}{P})$	$\mathcal{O}(\frac{N^{\frac{d}{2}}}{\sqrt{P}} \log P)$	$\mathcal{O}(\log$

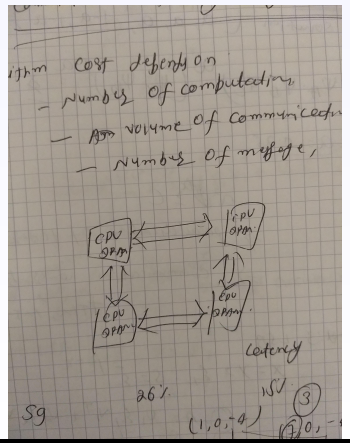
Table of Contents

Why Communication is Important?

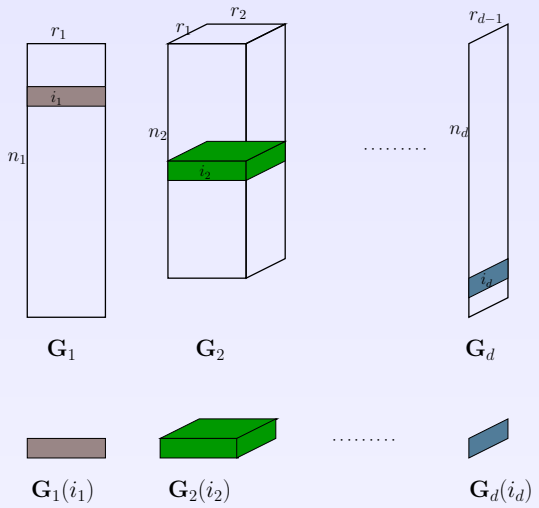
- Running time of an algorithm depends on
  - Number of computations * time-per-computation
  - Data movement
    - Volume of communication / Network-bandwidth
    - Number of messages * Network-latency
- Gaps growing exponentially with time (Source: Getting up to speed: The future of supercomputing)

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

- Avoid communication to save time (and energy)



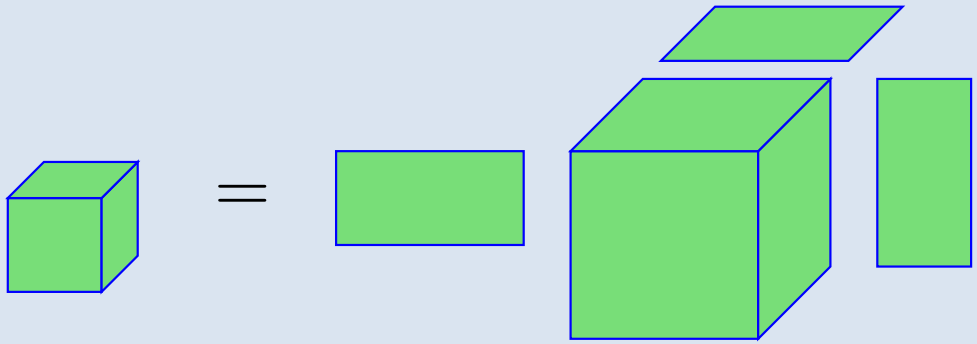
Communication Optimal Algorithms for Tensors



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

- This is a hard problem to start with
- We look similar computation which has simpler structure than this

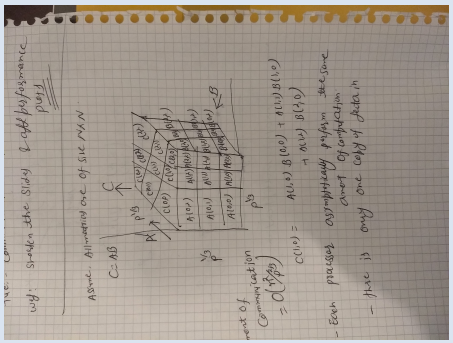
3-dimensional Multiple Tensor Times Matrix (Multi TTM)



Revisiting Matrix Multiplication Algorithms

- We look at the existing approaches
- Communication lower bounds fo many computations are mostly derived based on matrix multiplication bounds
- Our goal is to express existing approaches in the form suitable for tensors

Communication Optimal Matrix Multiplication Algorithm

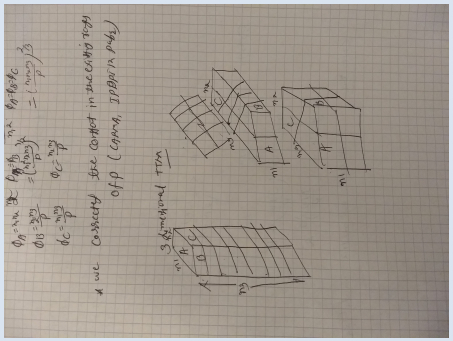


- Reinvented many times: Dekel, Nassimi, Sahni [81], Bernsten [89], Agarwal, Chandra, Snir [90], Johnson [93], Agarwal, Balle, Gustavson, Joshi, Palkar [95]

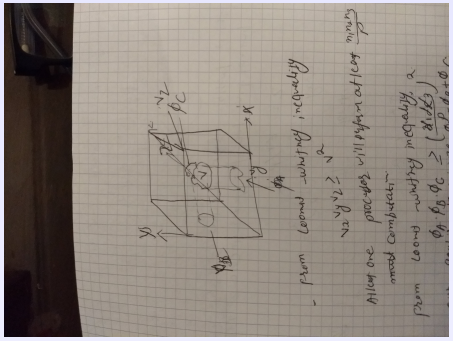
Processors Arranged in Different Ways

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

Three different arrangements of processor grids



Lower Bounds for Matrix Multiplication Algorithms



From Loomis-Whitney inequality,

$$V_x V_y V_z \geq V^2$$

Atleast one processor will perform atleast $\frac{n_1 n_2 n_3}{P}$ computations.

From Loomis-Whitney inequality, $\phi_A \phi_B \phi_C \geq \left(\frac{n_1 n_2 n_3}{P}\right)^2$.

Here ϕ_A, ϕ_B and ϕ_C denote the projections of computations on A, B and C matrices. Our goal is to minimize $\phi_A + \phi_B + \phi_C$.

We also added other inequalities.

- To perform atleast $\frac{1}{P}$th computation, the processor must access atleast $\frac{1}{P}$th data of each matrix
- Projectin on each matrix can not be larger than the original matrix

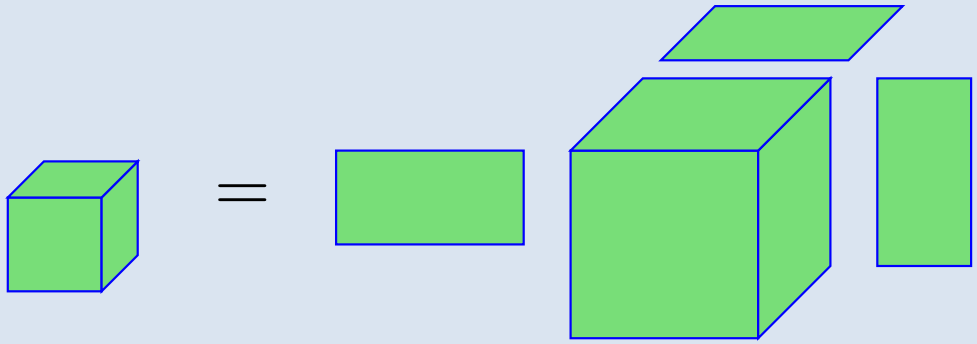
We obtain the following additional inequalities.

Lower Bounds for Matrix Multiplication Algorithms

After solving the minimization problem we obtain the following ranges for different arrangement of process grids.

- We corrected the constants in the existing ranges of P (Demmel, Eliahu, Fox, Kamil, Lipshitz, Schwartz, Spillinger [IPDPS 2013])

3-dimensional Multiple Tensor Times Matrix (Multi TTM)



Let G and χ 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. Sequential code for this computation can be written as..

3-dimensional Multi TTM Computation

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

Multi TTM Computation

ϕ_x denotes the projection of iteration space on variable x. Here $x \in G, \mathcal{X}, A, B, C$. Dimension of **G**, $\mathcal{X}$, 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. We assume $r_1 r_2 r_3 \leq n_1 n_2 n_3$ and $n_1 r_1 \leq n_2 r_2 \leq n_3 r_3$.

We need to solve the following linear program:

Minimize $\phi_G + \phi_X + \phi_A + \phi_A + \phi_B + \phi_C$ subject to

$$\phi_G \phi_X \geq \frac{n_1 n_2 n_3 r_1 r_2 r_3}{P}$$

$$\phi_A \phi_B \phi_C \geq \frac{n_1 n_2 n_3 r_1 r_2 r_3}{P}$$

$$\frac{n_1 n_2 n_3}{P} \leq \phi_X \leq n_1 n_2 n_3$$

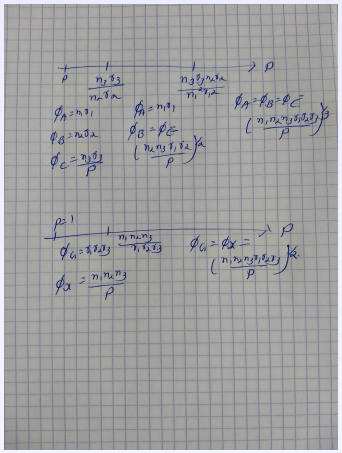
$$\frac{r_1 r_2 r_3}{P} \leq \phi_G \leq r_1 r_2 r_3$$

$$\frac{n_1 r_1}{P} \leq \phi_A \leq n_1 r_1$$

$$\frac{n_2 r_2}{P} \leq \phi_B \leq n_2 r_2$$

$$\frac{n_3 r_3}{P} \leq \phi \leq$$

Multi TTM Computation



Lower bound on the communication for the processor $\geq \phi_G + \phi_X + \phi_A + \phi_A + \phi_B + \phi_C$ - *Portion of the variables already present in the memory*

Table of Contents

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?

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 of the compute unit is not a concern

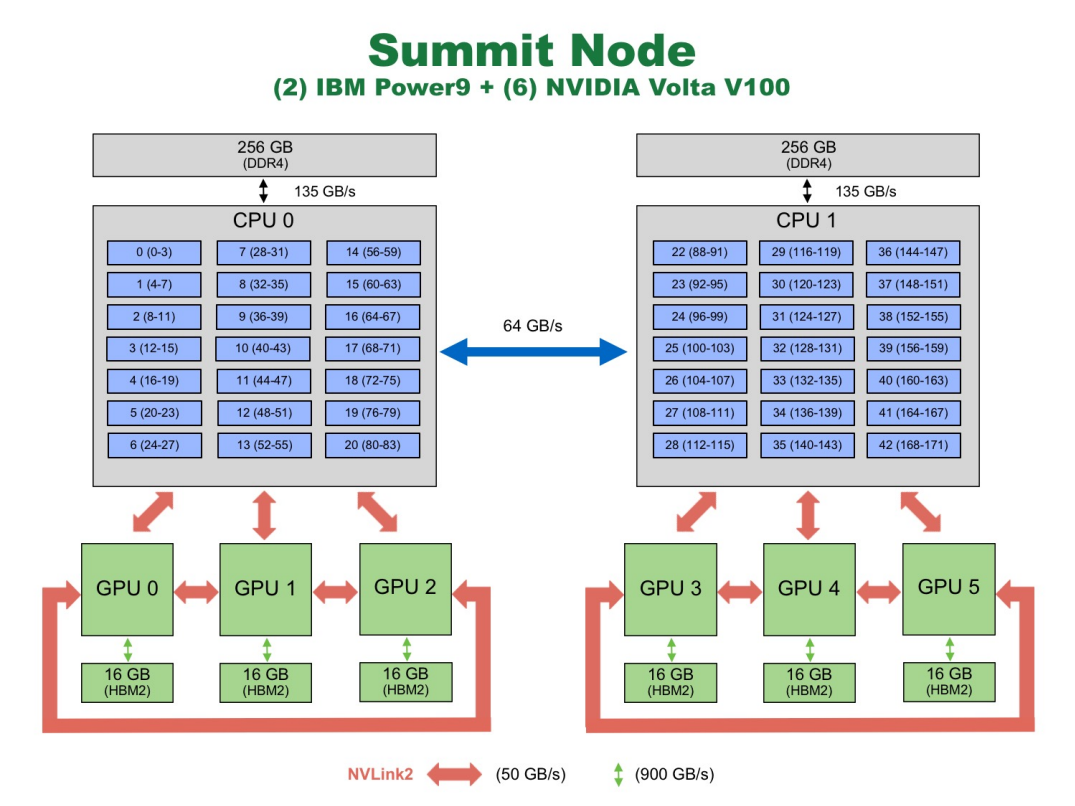
- First sort compute intensive tasks in increasing order of their data transfer time
- Then sort the communication intensive tasks in decreasing order of their computation time

Memory capacity is limited

- The problem is NP-Complete
  - Reduced 3PAR problem to this problem
- Proposed static and dynamic based approaches and evaluated them on traces of molecular simulations

Performance evaluation on Summit supercomputer

- TAMM: Tensor Algebra for Manybody Methods
- CCSD application, molecule, cc-pVDZ (737 basis functions, 220 nodes), aug-cc-pVDZ (1243 basis functions, 256 nodes)



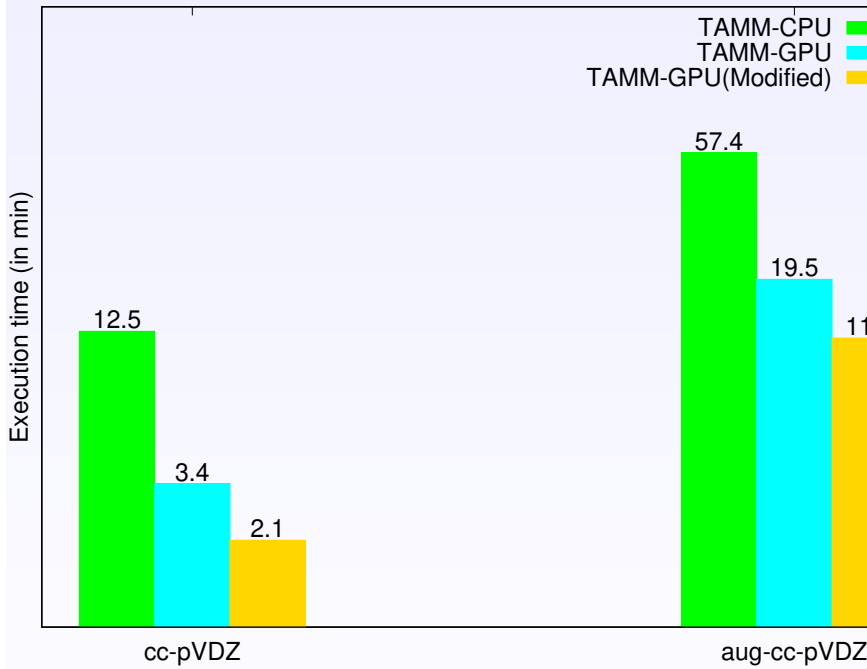


Fig source: `https://www.olcf.ornl.gov`

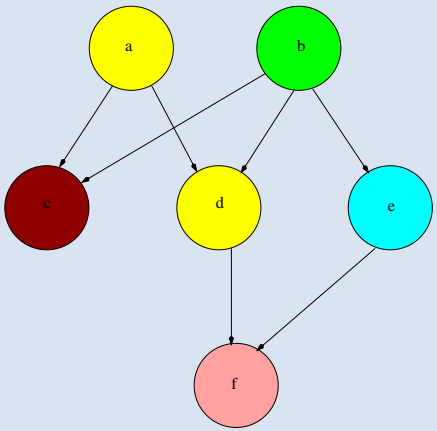
Table of Contents

Performance 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, XKaapi or PaRSEC



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

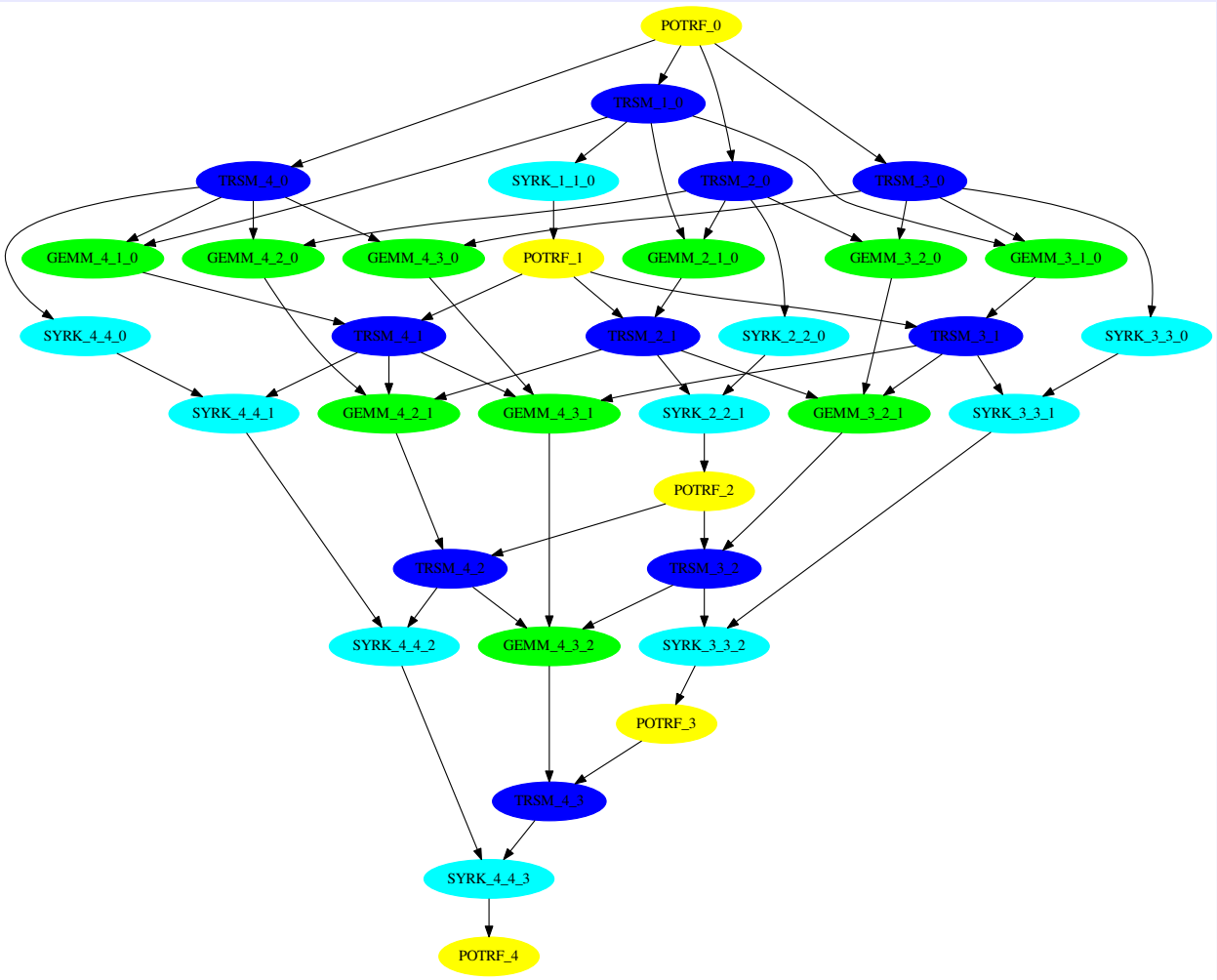
Tiled Cholesky Factorization

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

```
1:  for k = 0 to N − 1 do
2:      L[k][k] ← POTRF(A[k][k]);
3:      for i = k + 1 to N − 1 do
4:          L[i][k] ← TRSM(L[k][k], A[i][k]);
5:      end for
6:      for j = k + 1 to N − 1 do
7:          A[j][j] ← SYRK(L[j][k], A[j][j]);
8:          for i = j + 1 to N − 1 do
9:              A[i][j] ← GEMM(L[i][k], L[j][k], A[i][j]);
10:         end for
11:     end for
12: end for
```

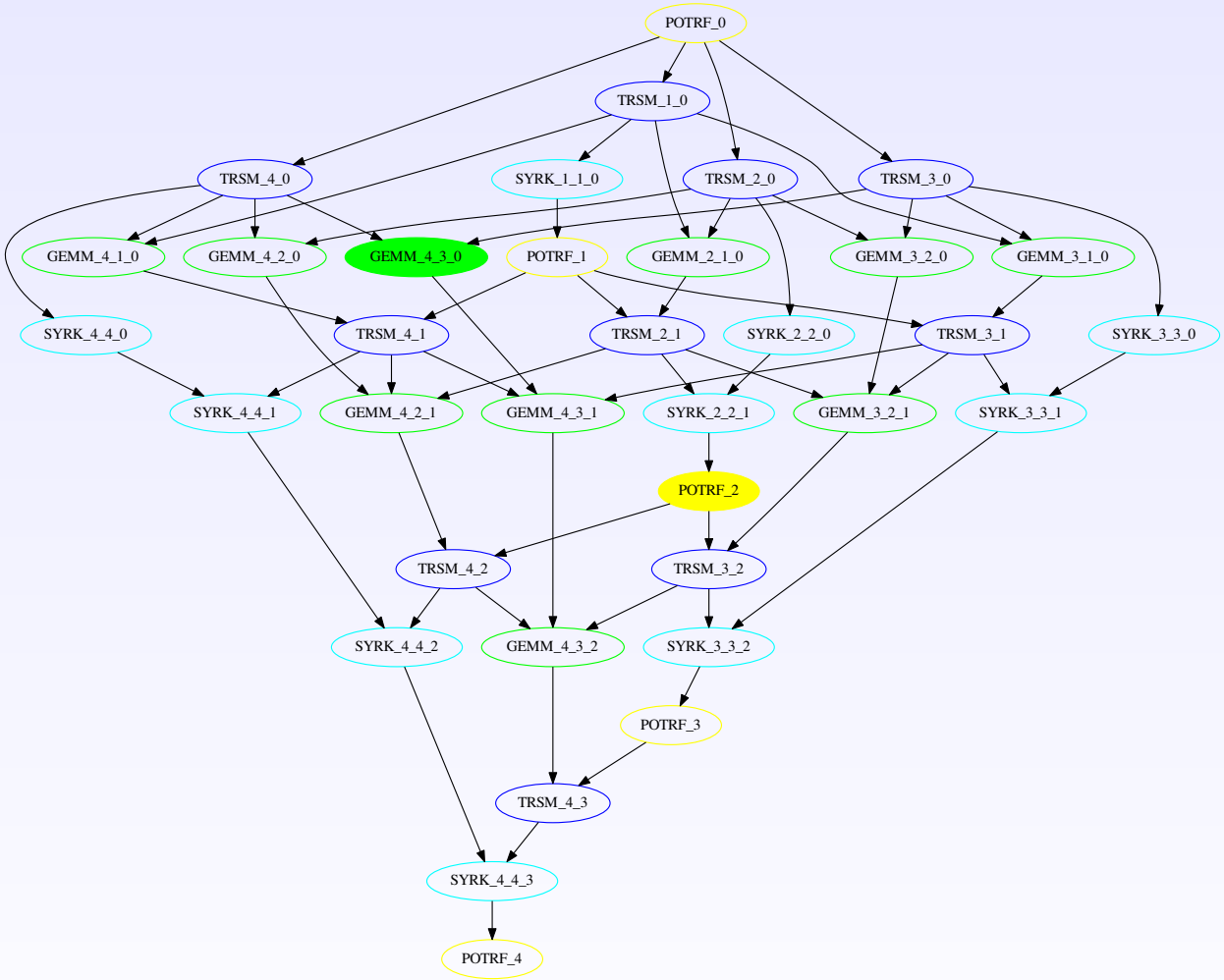
Algorithm 1: Pseudocode of the tiled Cholesky factorization

Tiled Cholesky Task Graph (N=5)



- Exhibits complex dependencies pattern

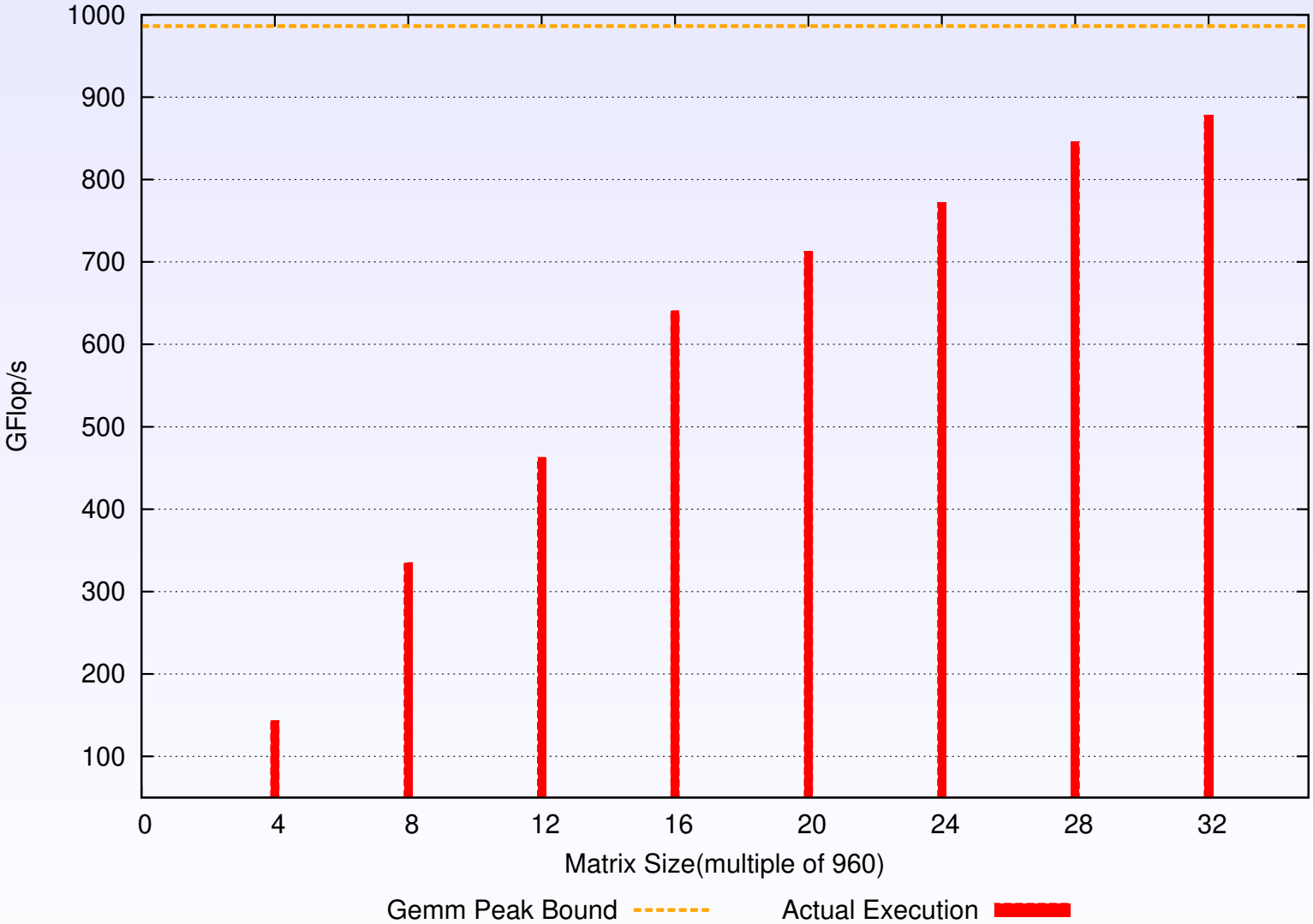
Tiled Cholesky Task Graph (N=5)



- Does not need synchronization at each outer iteration

Performance vs Bounds of Cholesky Factorization

With StarPU *dmdas* (HEFT based) scheduler on a single node



Gap between performance and bound is significant

Our Contributions – Single Node Perspective

- Better bounds on performance
- A resource centric HeteroPrio scheduler for task graphs on two types of resources
- Performance guarantee of HeteroPrio for a set of independent tasks on two types of resources

Machine Information & Library used

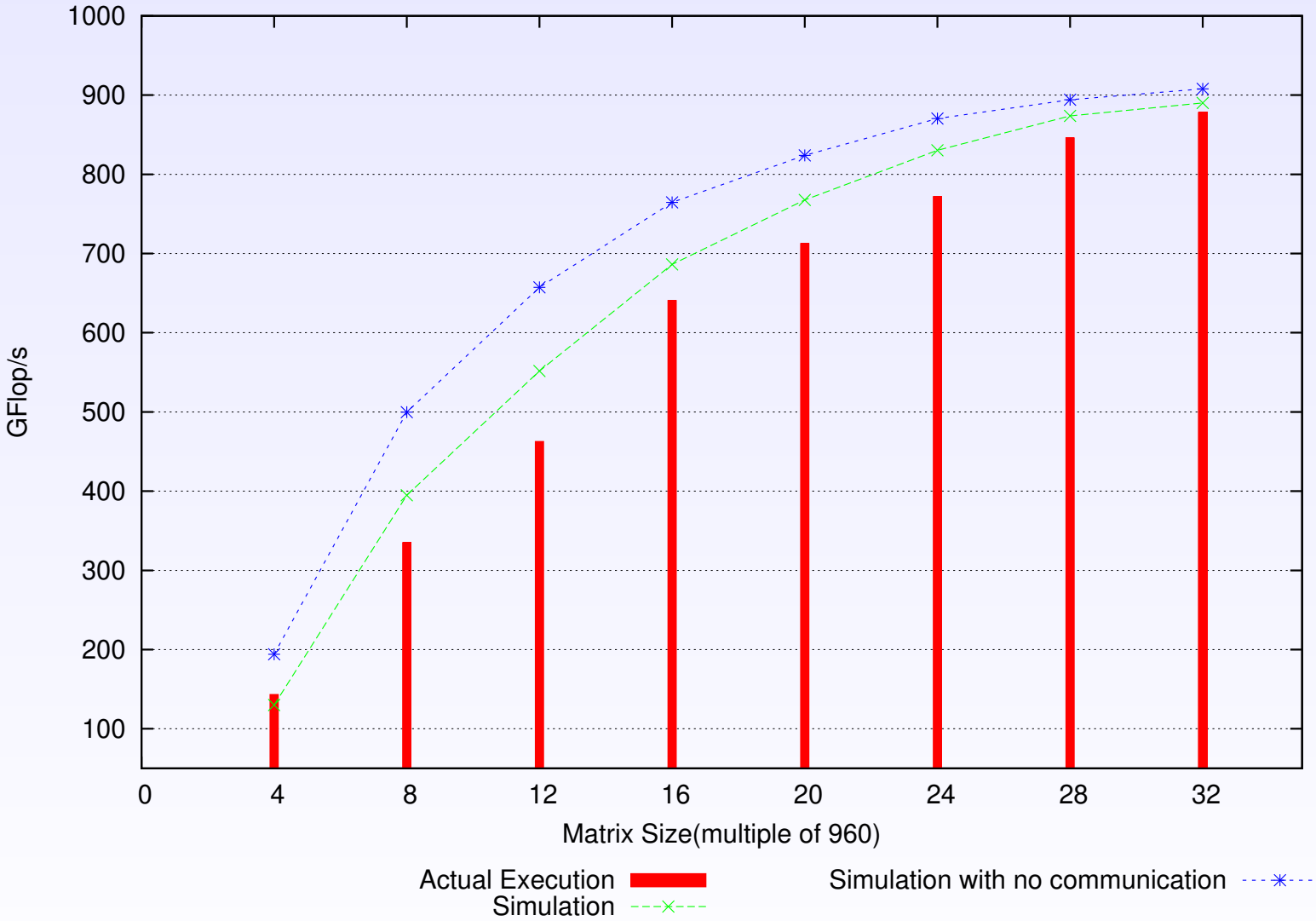
- Cholesky kernel of Chameleon library with StarPU runtime
- We use TileSize (BlockSize) =960
- Heterogeneous platform description (Single Node)
  - 12 CPU cores (**9 computational cores**) of Intel R Xeon R X5650 processors
  - **3** Nvidia Tesla M2070 **GPUs**
  - Theoretical peak performance = 1641 GFlop/s
  - GEMM peak performance = 981 GFlop/s

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

Table: GPUs relative performance

Impact of communication on Cholesky Factorization

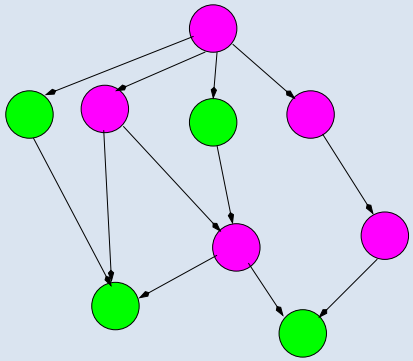
With StarPU *dmdas* (HEFT based) scheduler



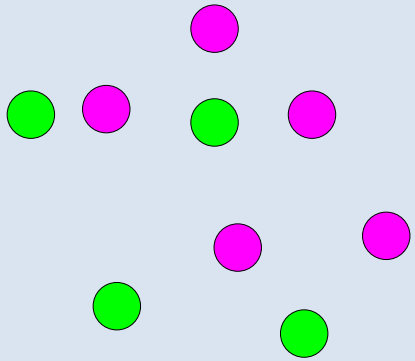
Impact of communication on performance is very less for larger matrices

Area Bound

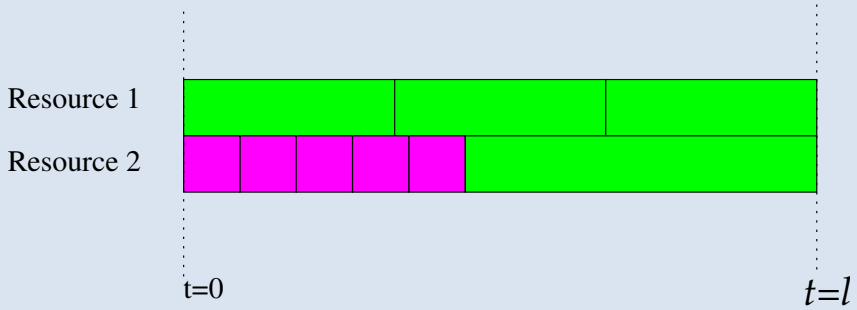
DAG



No Dependencies

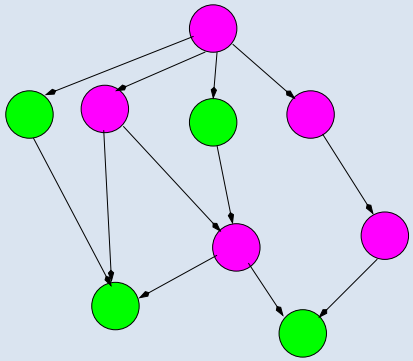


Minimum execution time (minimize *l*)

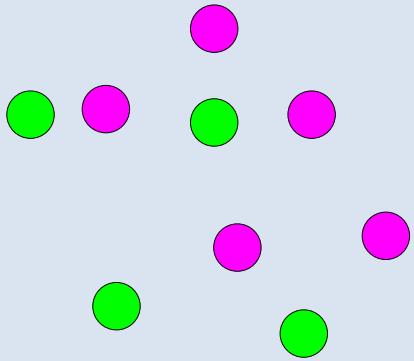


Iterative Bound

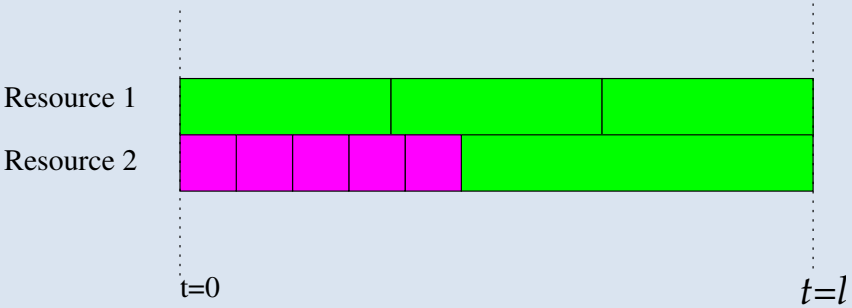
DAG



No Dependencies

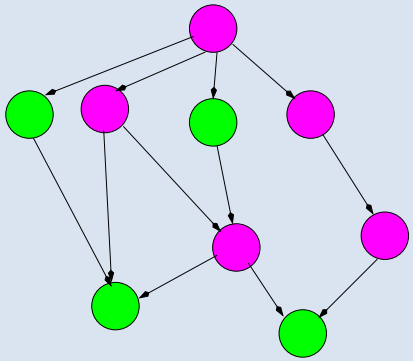


Minimum execution time (minimize *l*)

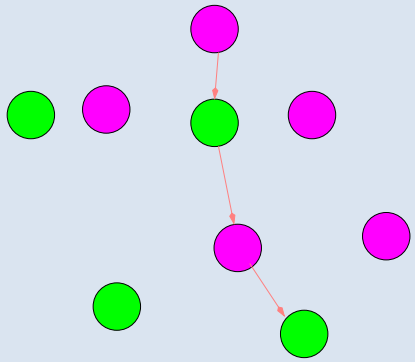


Iterative Bound

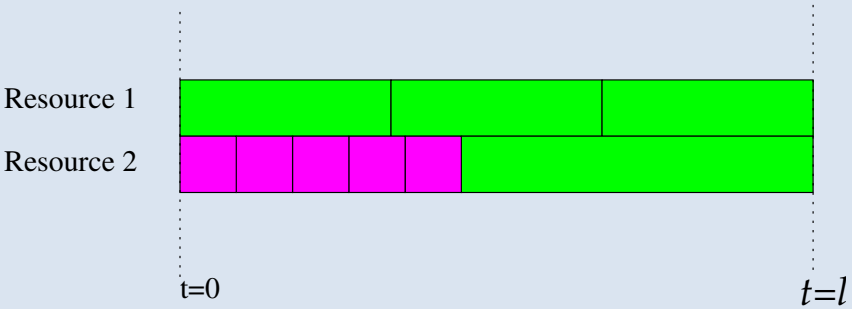
DAG



Some Dependencies



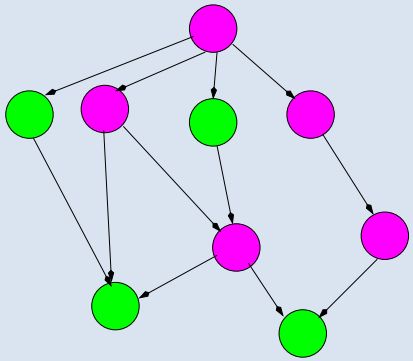
Minimum execution time (minimize *l*)



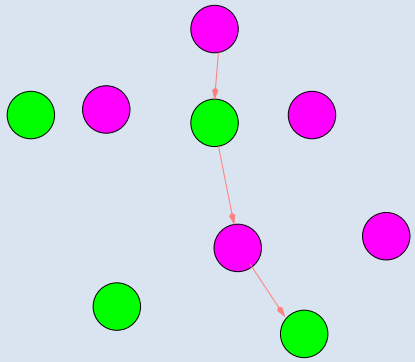
- ⋆ If any path in DAG is larger than *l*

Iterative Bound

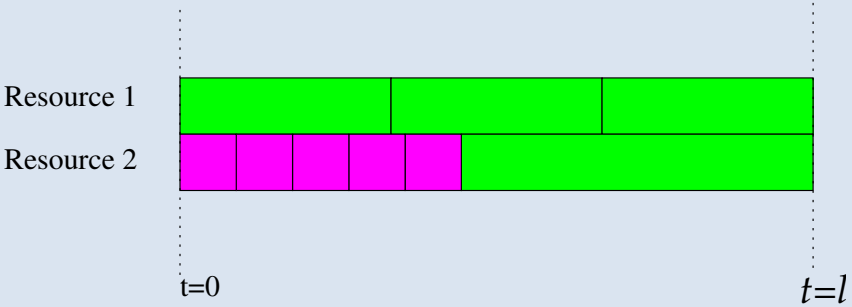
DAG



Some Dependencies

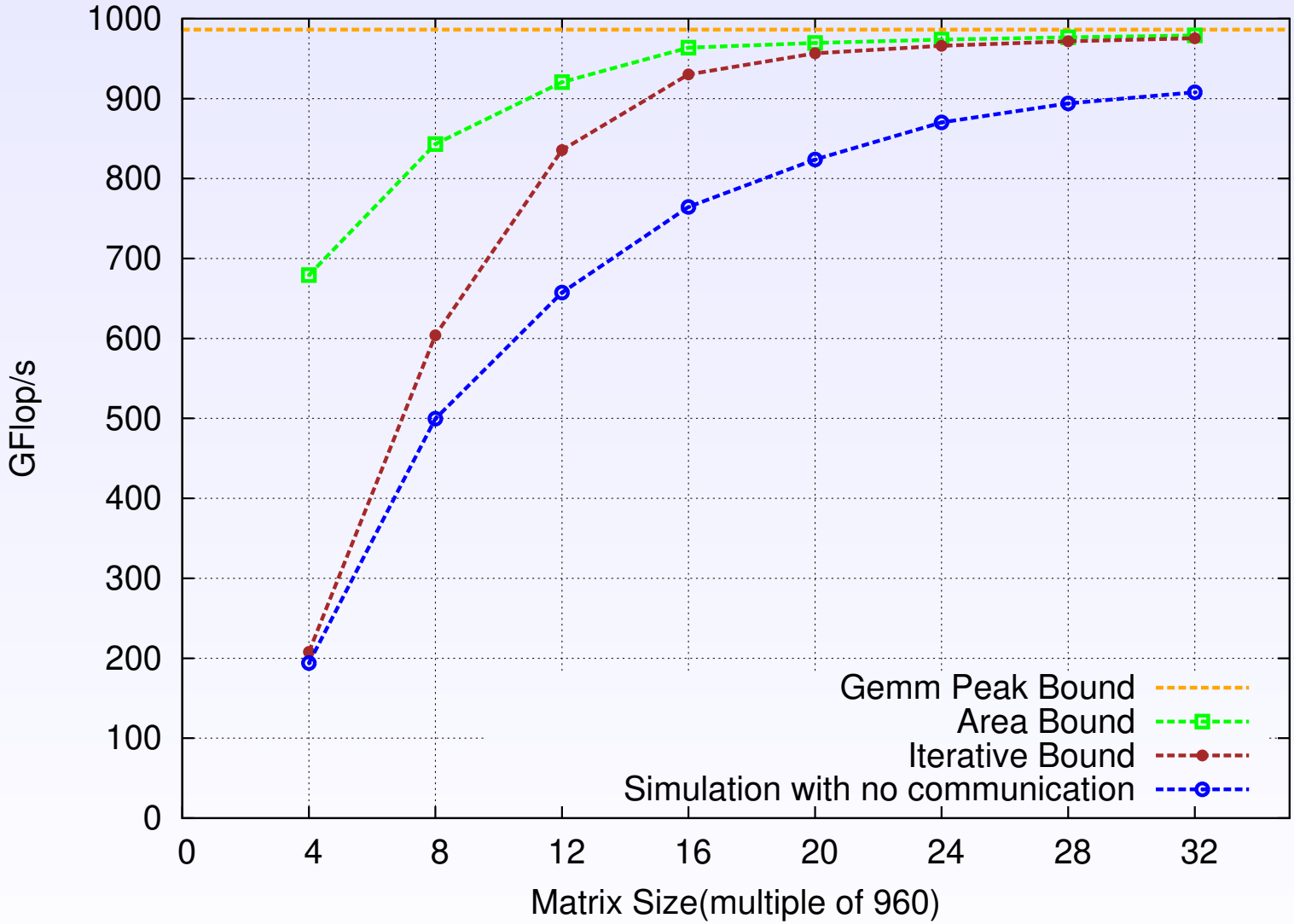


Minimum execution time (minimize *l*)



- ⋆ 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



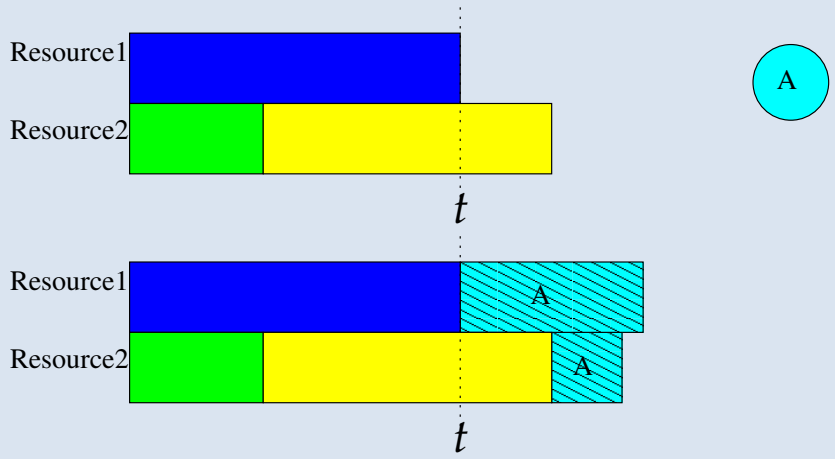
Dynamic Scheduling

- Decide based on state of resources and estimation of execution/transfer time
- Possibly uses some priorities among the ready tasks
- Pros: Fault tolerance, Load balancing
- Cons: may be less efficient due to local views

heft scheduler

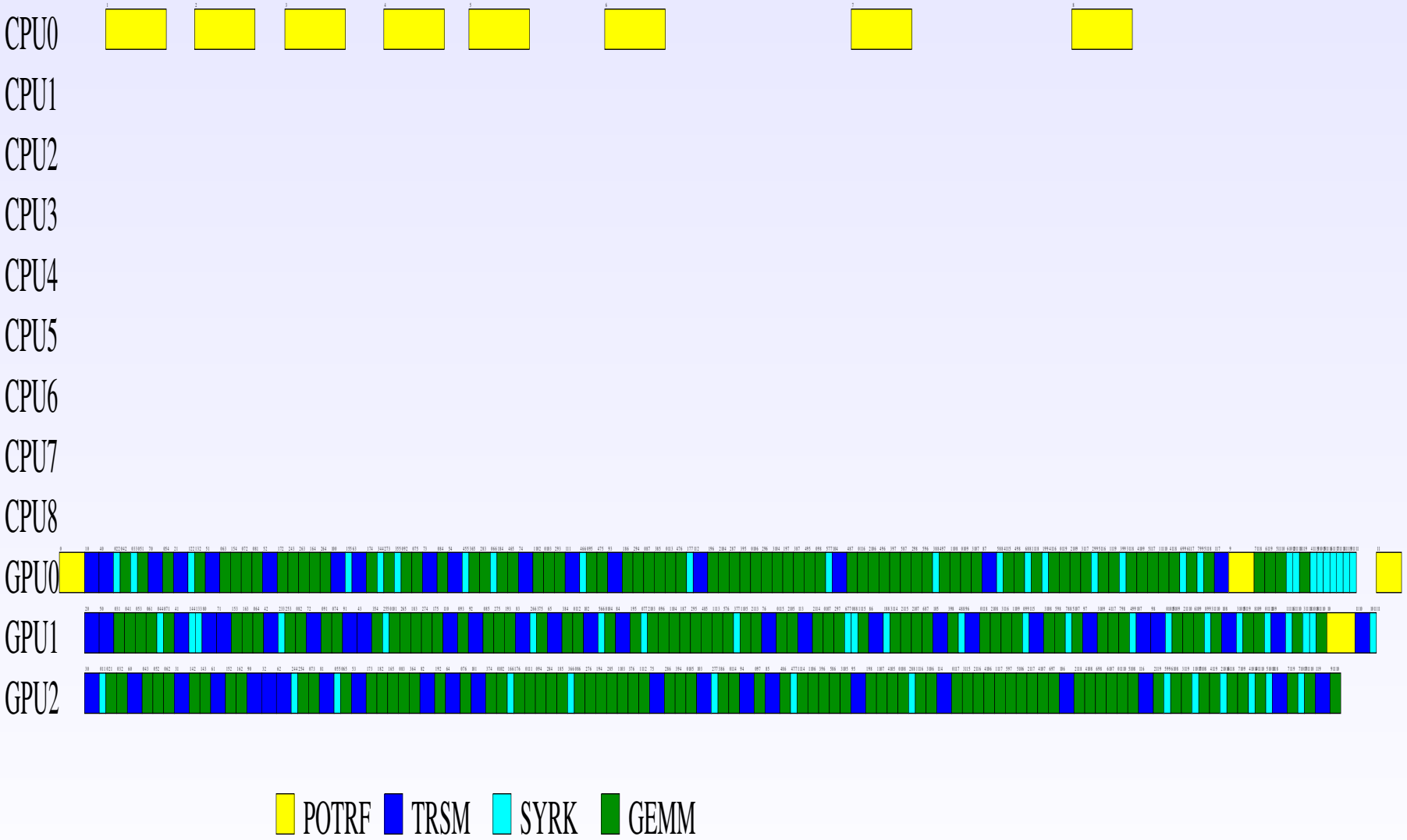
- Task centric scheduler
- Based on Heterogeneous Early Finish time (HEFT) heuristic (proposed by *Topcuoglu et al.* in 2002)

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



heft Scheduler

heft trace for 12 X 12 blocks of Cholesky Factorization

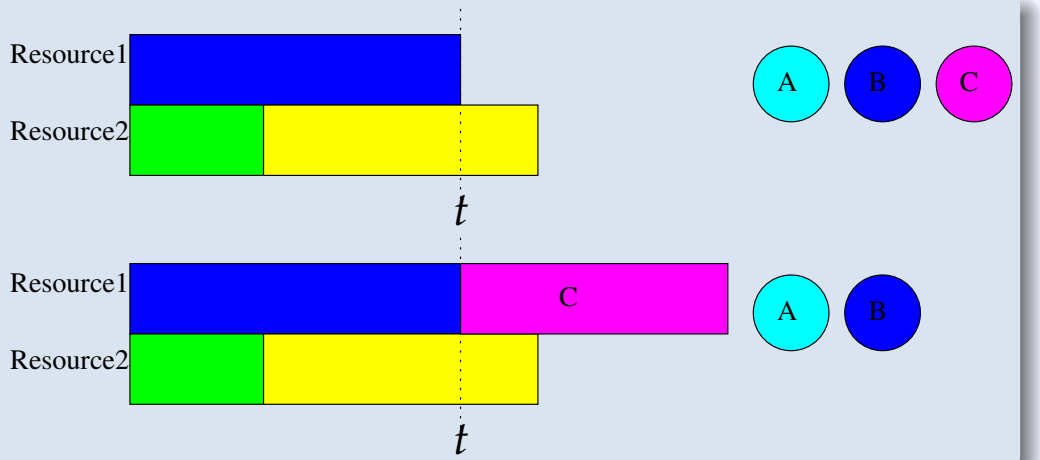


- Most of the CPU resources are not utilized (686 GFlop/s)

HeteroPrio (HP) Scheduler

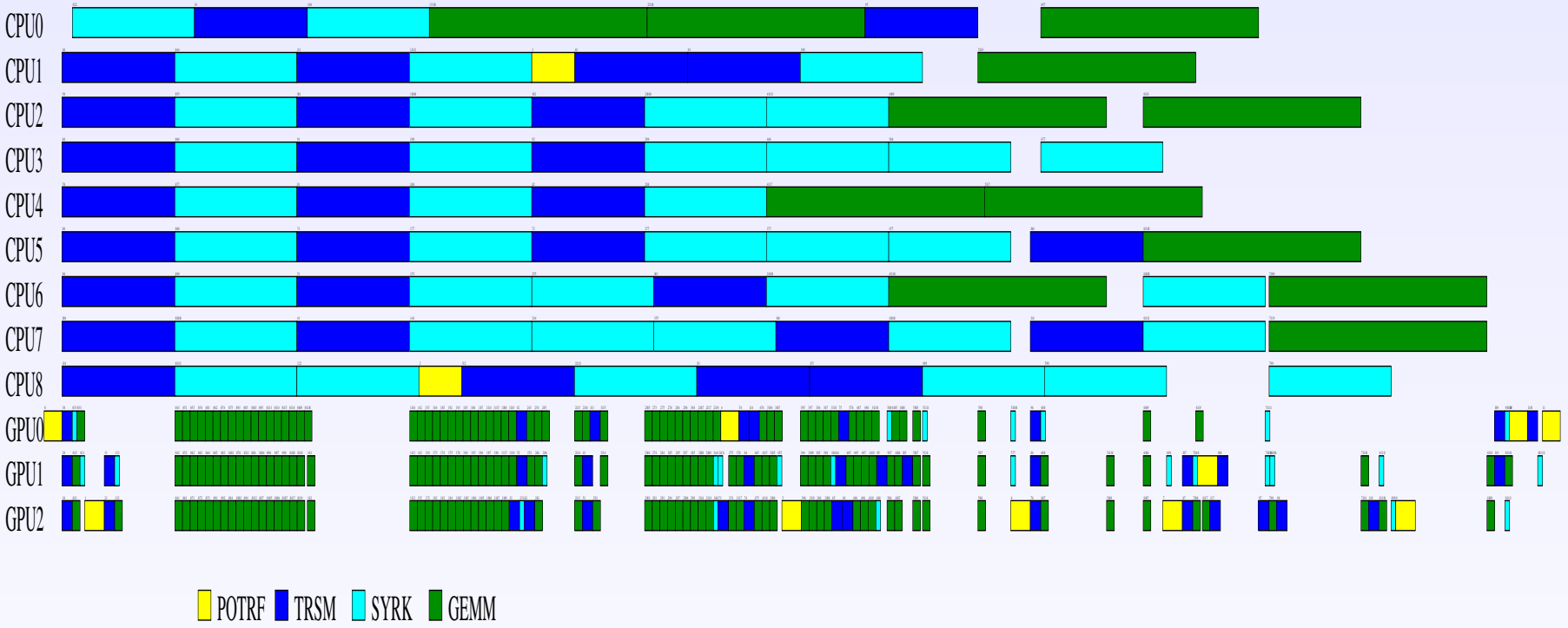
- Resource centric scheduler
- Based on tasks heterogeneity factors
- Proposed by *Agullo et al.* in 2016 for a large set of small independent tasks

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



HP scheduler

HP trace for 12 X 12 blocks of Cholesky Factorization



- Not much emphasis on progress (431 GFlop/s)

HeteroPrio (HP) with spoliation (SP)

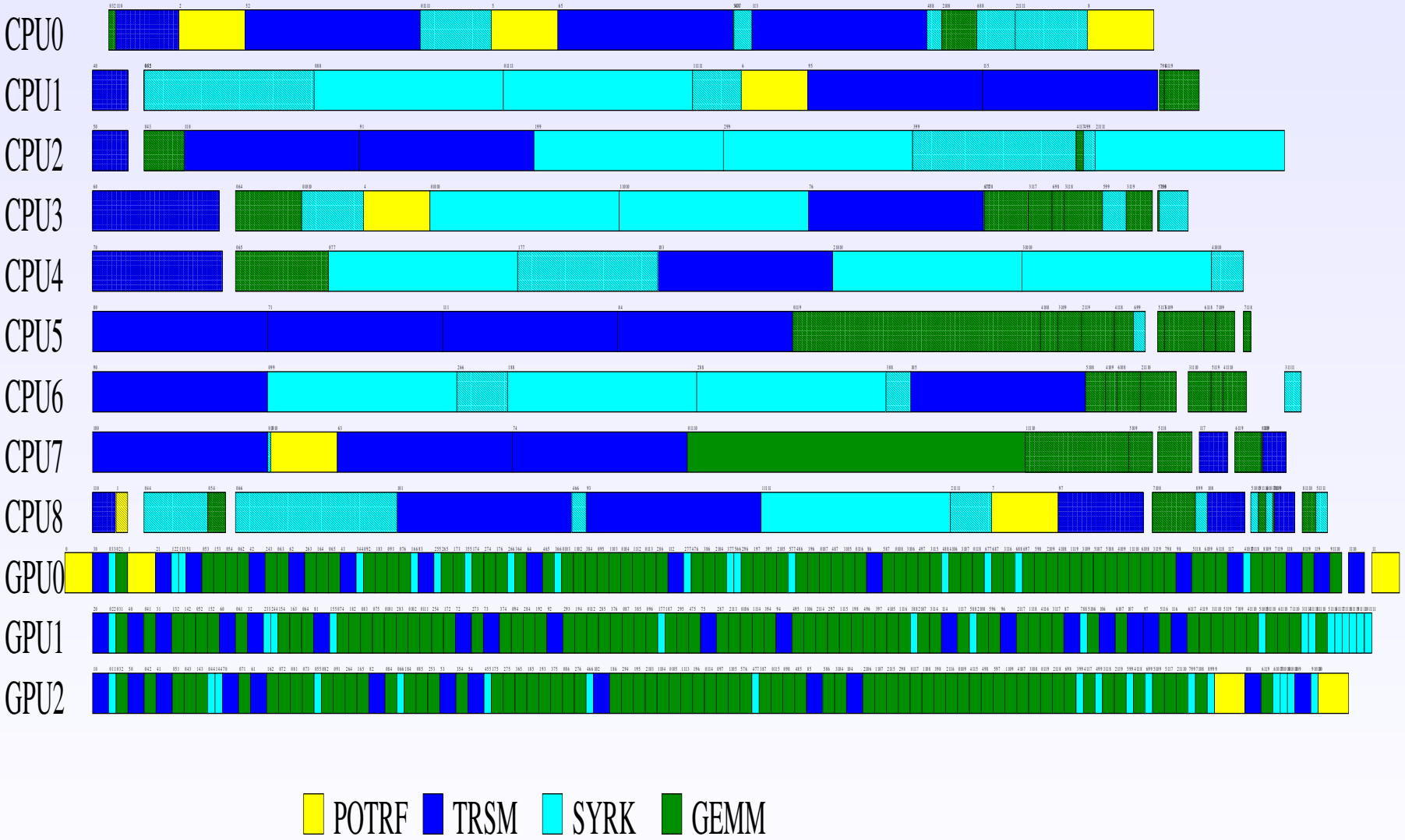
- HP scheduler
  - Good CPU Utilization
  - Ignorance of critical path
  - Significant idle time on GPUs
- HP with spoliation (HP+SP)
  - An idle GPU restarts highest priority CPU task if it finishes earlier

Successive correction to HP+SP with static information

- CPU selects lowest priority task among all tasks of same acceleration factor – **NonBlocking**
- HP+CGV: GPU selects highest priority task among all tasks of similar acceleration factors – **Progress**
- HP+PP: POTRF preempted on CPU (less accelerated on GPU) – **Best Affinity**
- HP+PCEPT: GPU restarts highest priority GEMM/SYRK task if it finishes earlier – **NonBlocking**

HP+PCEPT scheduler

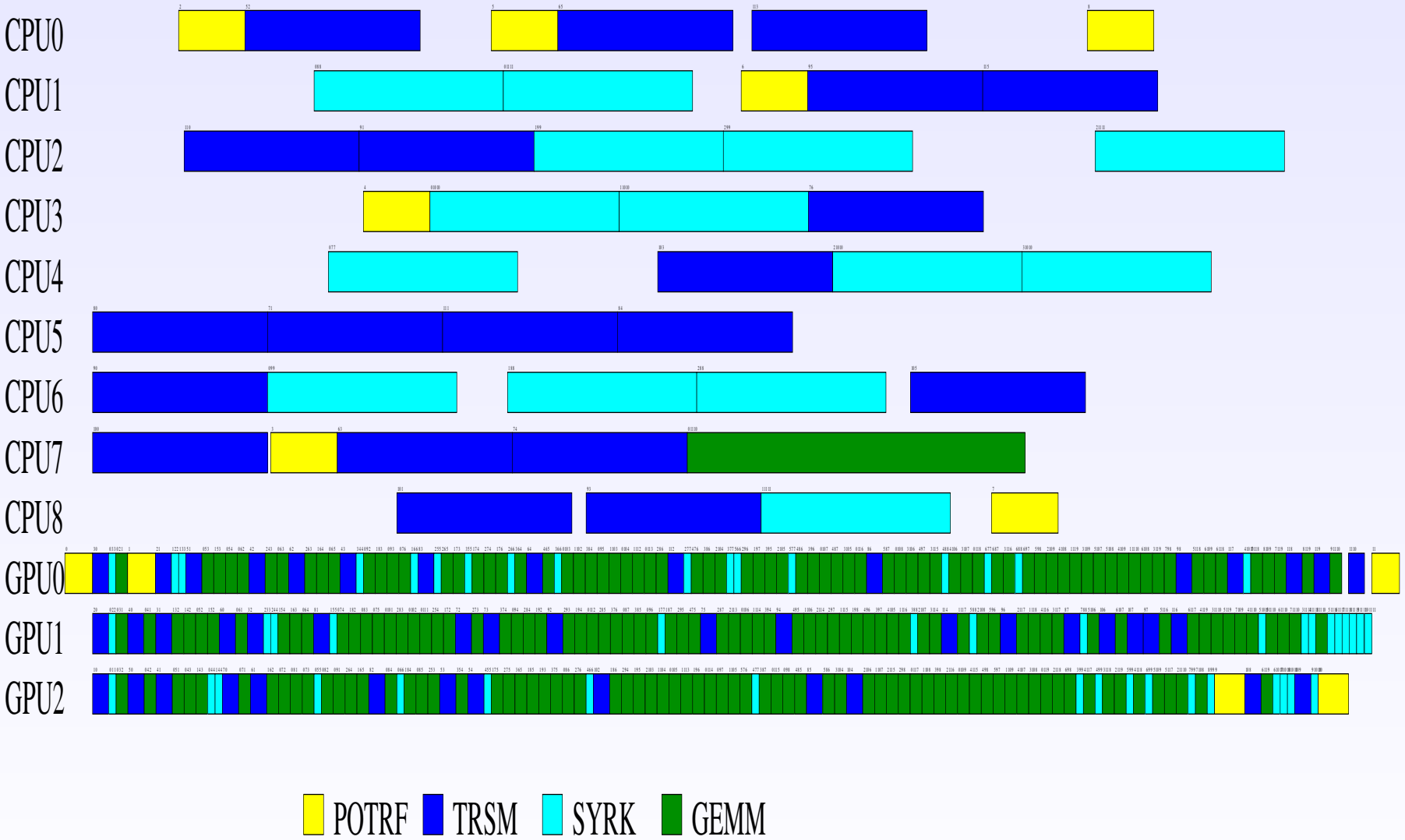
HP+PCEPT trace for 12 X 12 blocks of Cholesky Factorization



Achieved performance = 760 GFLop/s

HP+PCEPT scheduler

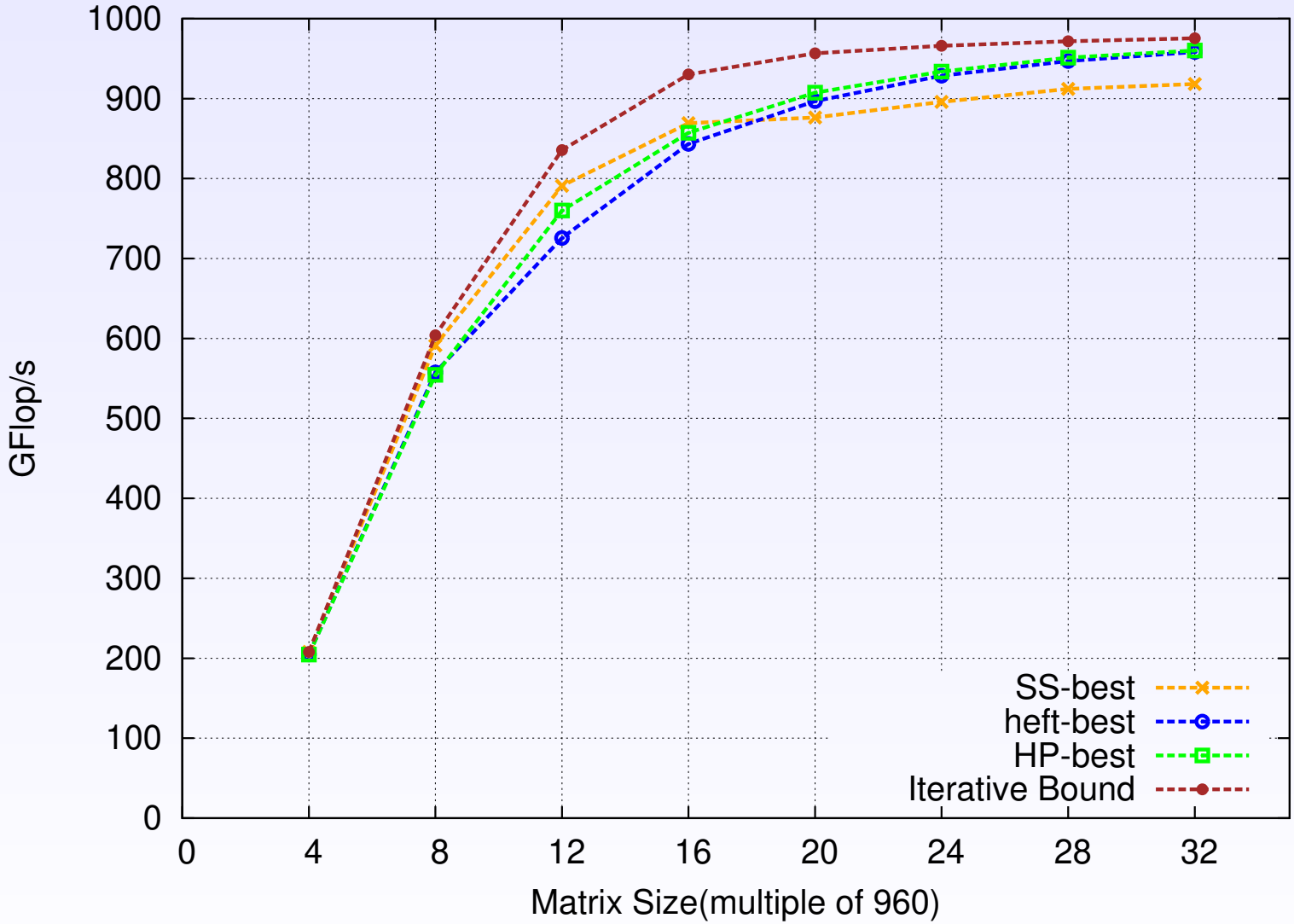
HP+PCEPT trace for 12 X 12 blocks of Cholesky Factorization



Achieved performance = 760 GFLop/s

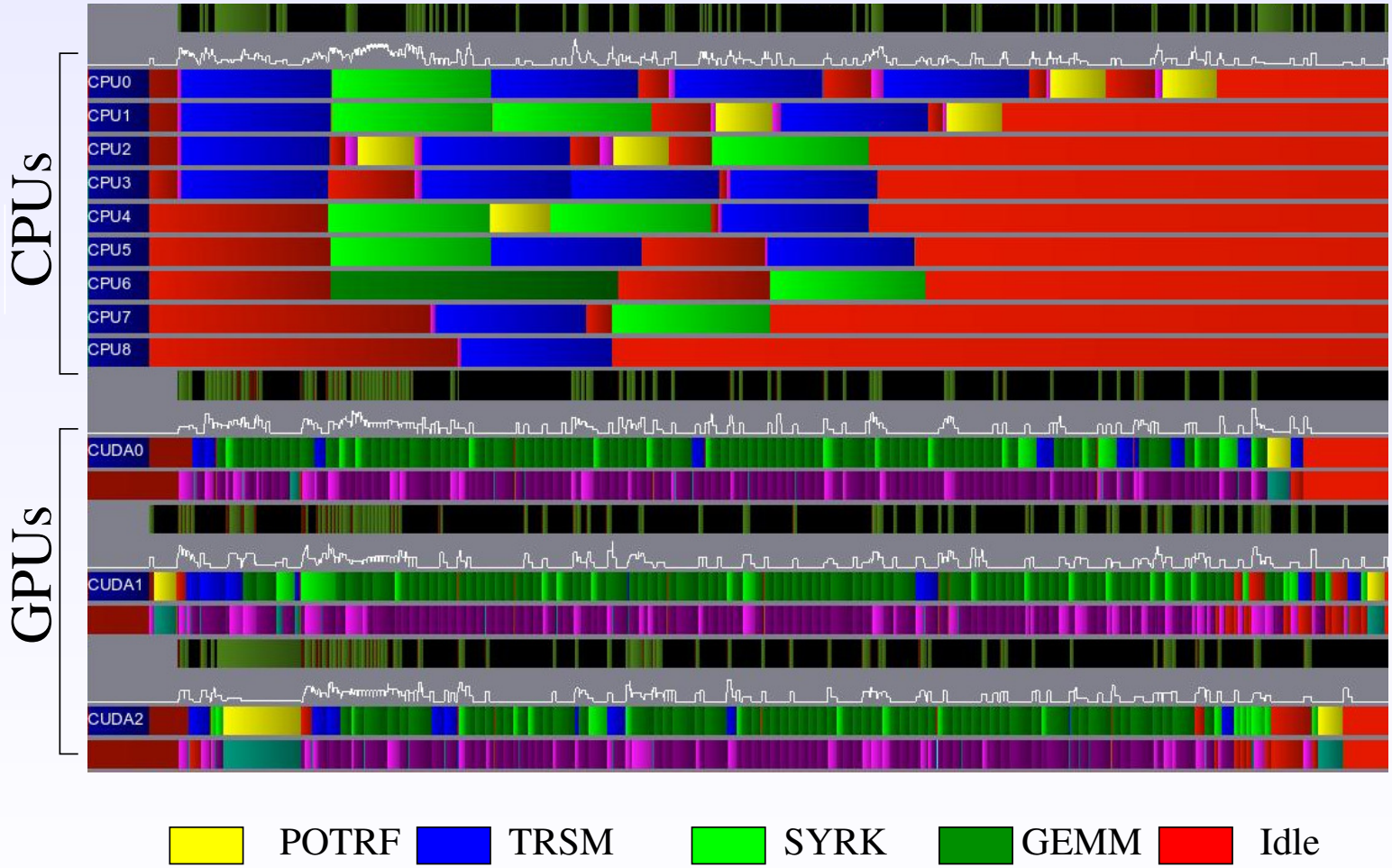
heft best performance = 726 GFlop/s

SS-Vs-heft-Vs-HP-Vs-Bound



Actual execution trace for 12 X 12 blocks of Cholesky Factorization

HeteroPrio schedule with 5% increase in CPU execution timings



HeteroPrio Approximation Ratios and Worst Case Example Ratios

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

1 CPU 1 GPU HeteroPrio Worst Case Example

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

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

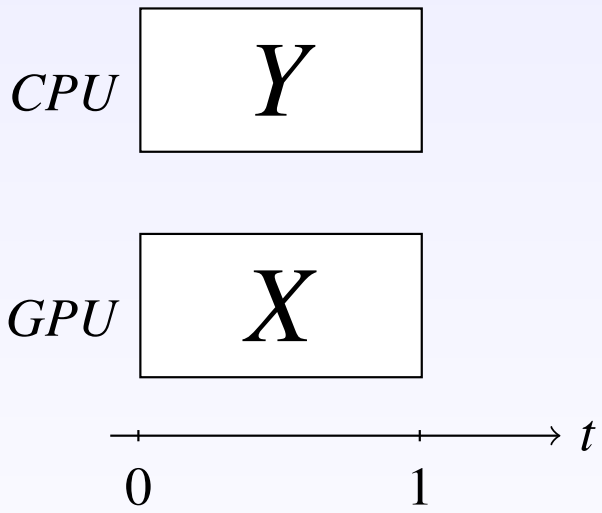


Figure: Optimal schedule

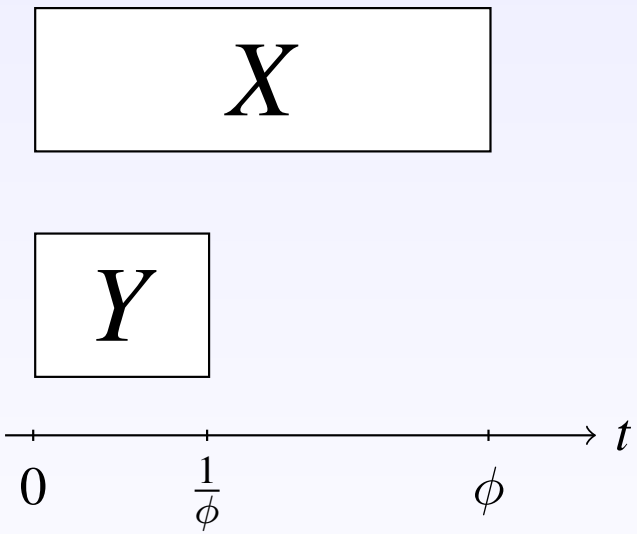


Figure: HeteroPrio schedule

Part II

Part2

Overview

Task Graph Bound[1]
Dynamic Scheduling Strategies[2]

1. Modify Existing Algorithms
   - Extension of the Existing Approaches
   - Exploratory Topics

2. Integration in the Team

Modify Existing Algorithms

- Understand the existing Algorithms
- Derive communication lower bounds
- Design new algorithms & implement them
- Homogeneous Systems
- Extend the same principle for Heterogeneous Systems

Tensor Contraction & Strassen's Matrix Multiplication

content...

Hierarchical Matrix concepts to Tensors

content...

Roofline model with Dependencies

content...

New Tensor Representations

content...

Architecture Aware Algorithms

content...

Randomization in Tensor Computations

content...

Integration in the Team

content...