Models for Optimizing Multiple Communication Cost Metrics in Parallelizing Irregular Applications

Cevdet Aykanat

Bilkent University

SIAM Workshop on Combinatorial Scientific Computing

Seattle, WA, USA, 2020

Introduction & Motivation

- Our goal
  - Efficient parallelization of irregular applications for distributed-memory systems
  - Optimization of communication costs

- **Communication costs** = bandwidth cost + latency cost
  - **Bandwidth cost ≈ volume of data communicated**
    - **Total** communication volume
    - **Maximum** communication volume
  - **Latency cost ≈ number of messages**
    - **Total** number of messages
    - **Maximum** number of messages

Introduction & Motivation

- Communication time depends on:
  - latency t_s
  - volume t_w

$$t_{comm} = t_s + mt_w$$

- Around **2KB/4KB**
  - latency overhead equals to volume overhead
- ***Latency* is more important for small messages**
- Goal
  - Most existing approaches aim at reducing volume overhead
  - Aim at **latency overhead**
    - Key to scalability

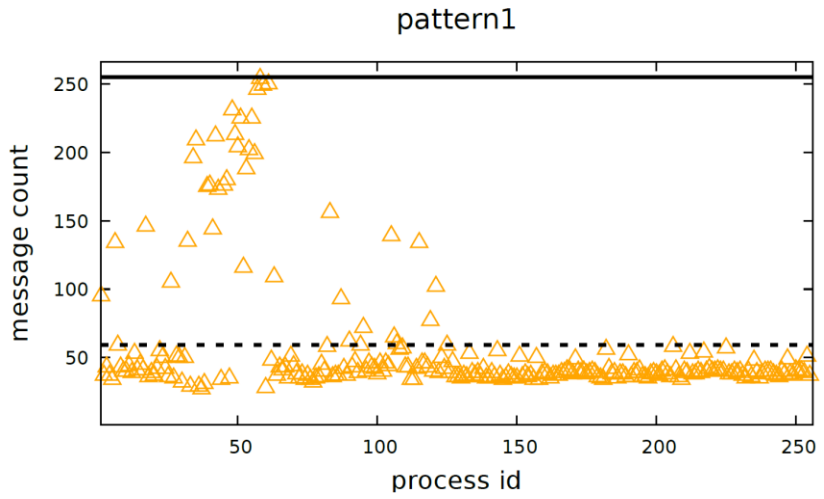
	Time (micro seconds)	
msg size	Cray XE6	Blue Gene/Q
4 B	1.9	5.3
8 B	1.9	5.3
16 B	1.9	5.3
32 B	1.9	5.5
64 B	1.8	5.5
128 B	2.6	7.8
256 B	1.9	8.0
512 B	2.2	9.5
1 KB	2.4	10.2
2 KB	2.7	10.6
4 KB	3.7	12.5
8 KB	9.5	14.3
16 KB	11.9	17.4
32 KB	16.9	22.0
64 KB	27.6	31.2
128 KB	29.3	49.7
256 KB	52.0	86.7
512 KB	108.3	159.7
1 MB	213.5	307.1
2 MB	413.5	602.8
4 MB	821.2	1191.8
8 MB	1636.6	2371.9

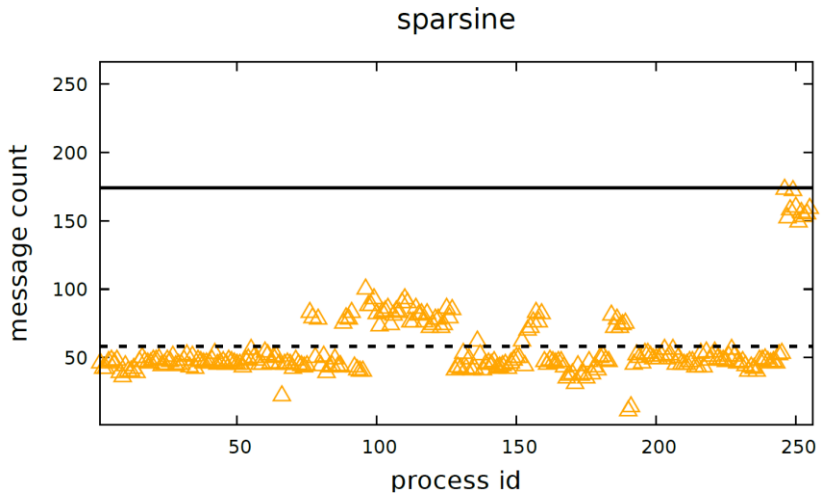
Latency dominates

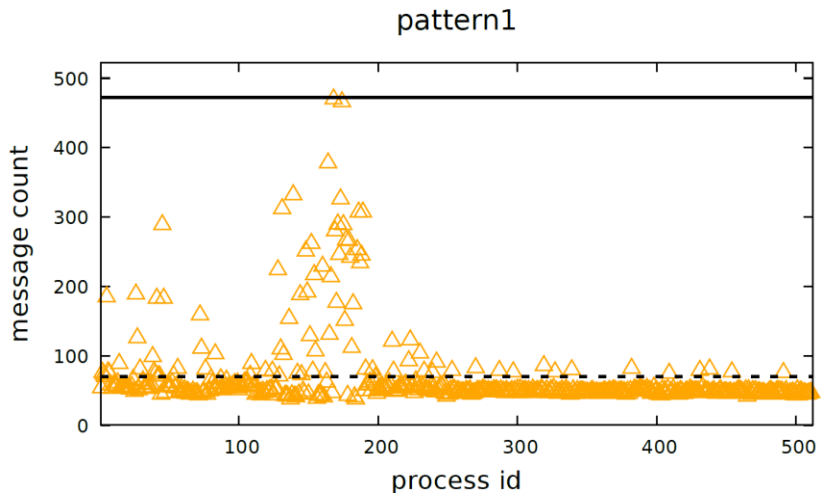
Volume dominates

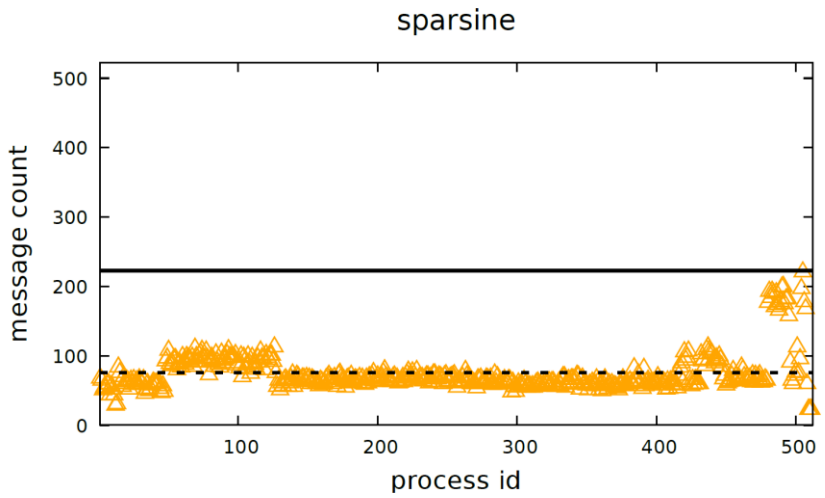
ping-pong experiments on two systems

Introduction & Motivation









Solid line: Maximum message count

Dashed line: Average message count

Top – 256 processes

Bottom – 512 processes

Outline

- Four different frameworks/models will be discussed
  - utilize existing partitioners in distinct ways
  - do not require development of novel partitioners

1. Communication hypergraph
   - minimize **total comm vol** and **total # of mssgs (two phase)**
2. Multi-stage hypergraph partitioning for Cartesian partitioning
   - minimize **total comm vol** and provide upper bound on **latency metrics**
3. Recursive hypergraph partitioning
   - Address **total comm vol** and **total # of mssgs (single phase)**
4. Regularization framework
   - a flexible medium to attain a trade-off between **bandwidth** and **latency** cost metrics **(two phase)**

1. Communication hypergraph

Metrics & Optimization

minimization of **total comm vol, max comm volume** and **total # of mssgs**

Methodology & Key Features

two phase

custom hypergraph models

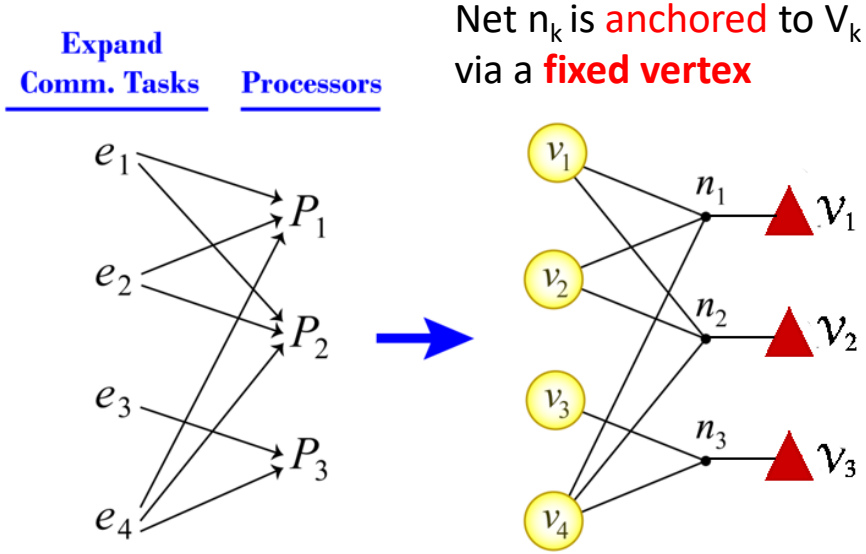
fixed vertices

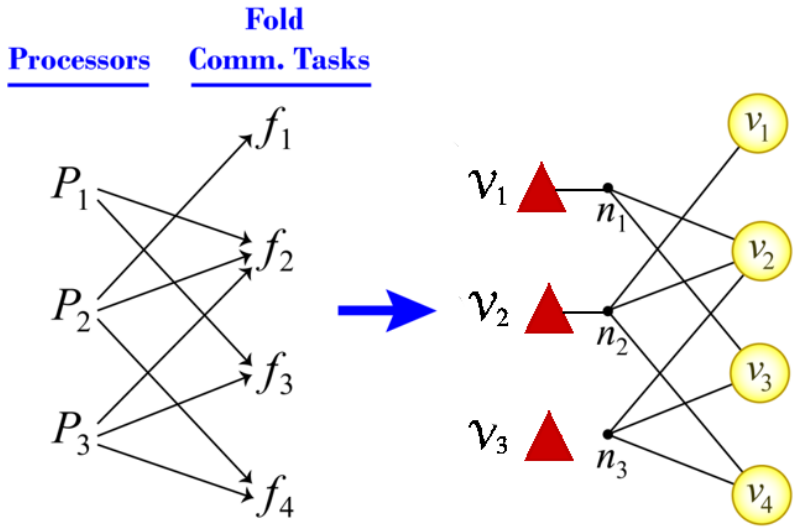
- **Two phase methodology**
  - *Phase 1: computational hypergraph/graph*
    - Minimize total comm volume
    - Balance on processors' computational loads
  - *Phase 2: communication hypergraph*
    - Minimize total # of mssgs
    - Balance on processors' volume loads

- ***Communication hypergraph model***
  - *vertices*: communication tasks
  - *nets*: processors

- ***Two types of communication tasks***
  - **Expand** communication task →**scatter-like**
  - **Fold** communication task → **reduce-like**

- A ***K-way partition*** induces (K: # of processors)
  - **communication task** to **processor** assignment

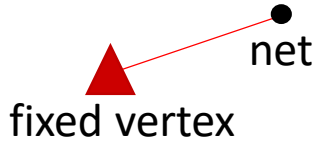




Partitioning communication hypergraph

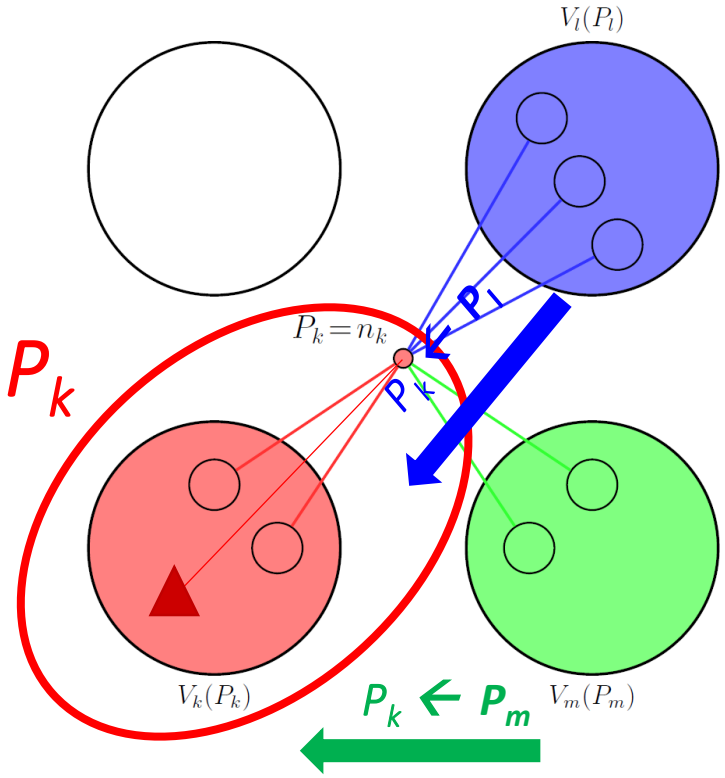
Partition into ***K*** to distribute communication operations among ***K*** processors

Net n_k is anchored to part V_k / P_k via a **fixed vertex**



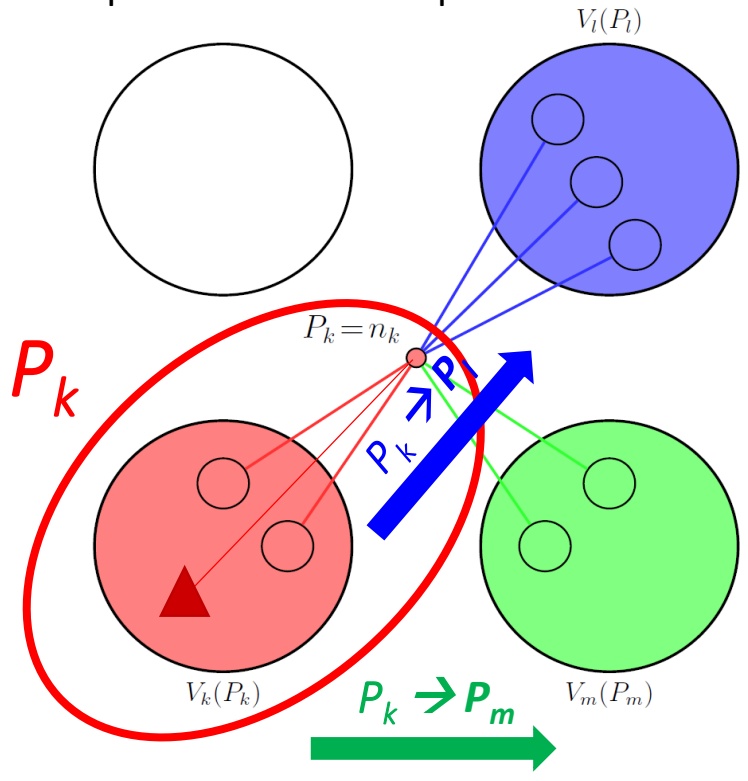
Expand tasks

Cut net → signifies messages that a processor will **receive** from other processors



Reduce tasks

Cut net → signifies messages that will be **sent** from a processor to other processors



- **Partitioning objective** minimizing cutsize ≈ minimizing **total # of mssgs**
- **Partitioning constraint** maintaining balance on part weights ≈ balances **comm volume loads** of processors

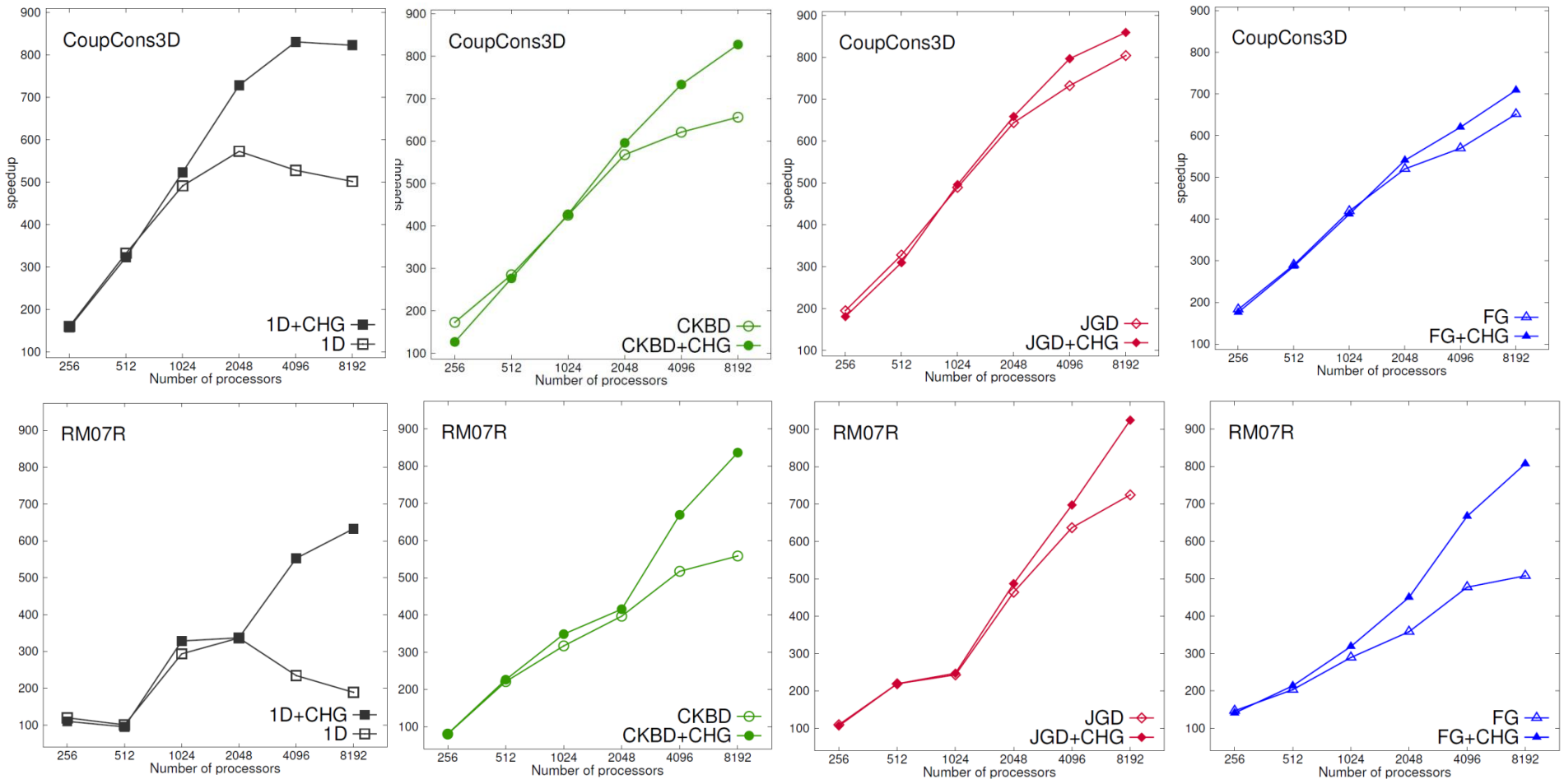
Distribution of communication tasks among four processors

Applications

- Propoposed for 1D row- and column-parallel SpMV [1]
- Extended and enhanced for 2D row-column-parallel SpMV
  - 2D Fine-grain partitioning [2], Jagged and Cartesian matrix partitioning [3]
- Extended and enhanced for 1D-parallel SpGEMM algorithms [4]
  - row-row-paralel
  - outer-product-paralel
  - inner-product-parallel

[1] Encapsulating Multiple Communication-Cost Metrics in Partitioning Sparse Rectangular Matrices for Matrix-Vector Multiplies, *Bora Ucar and Cevdet Aykanat*, **SIAM Journal on Scientific Computing**, 2004.
[2] Minimizing communication cost in fine-grain partitioning of sparse matrices, *Bora Ucar and Cevdet Aykanat, **ISCIS**, 2003.*
[3] Reducing latency cost in 2D sparse matrix partitioning models, *R. Oguz Selvitopi and Cevdet Aykanat*, **Parallel Computing**, 2016.
[4] Partitioning models for scaling parallel sparse matrix-matrix multiplication, *Kadir Akbudak, Oguz Selvitopi, Cevdet Aykanat*, **ACM Transactions on Parallel Computing (TOPC)**, 2018.

SpMV Speedup – Benefits of reducing latency



28 SpMV instances

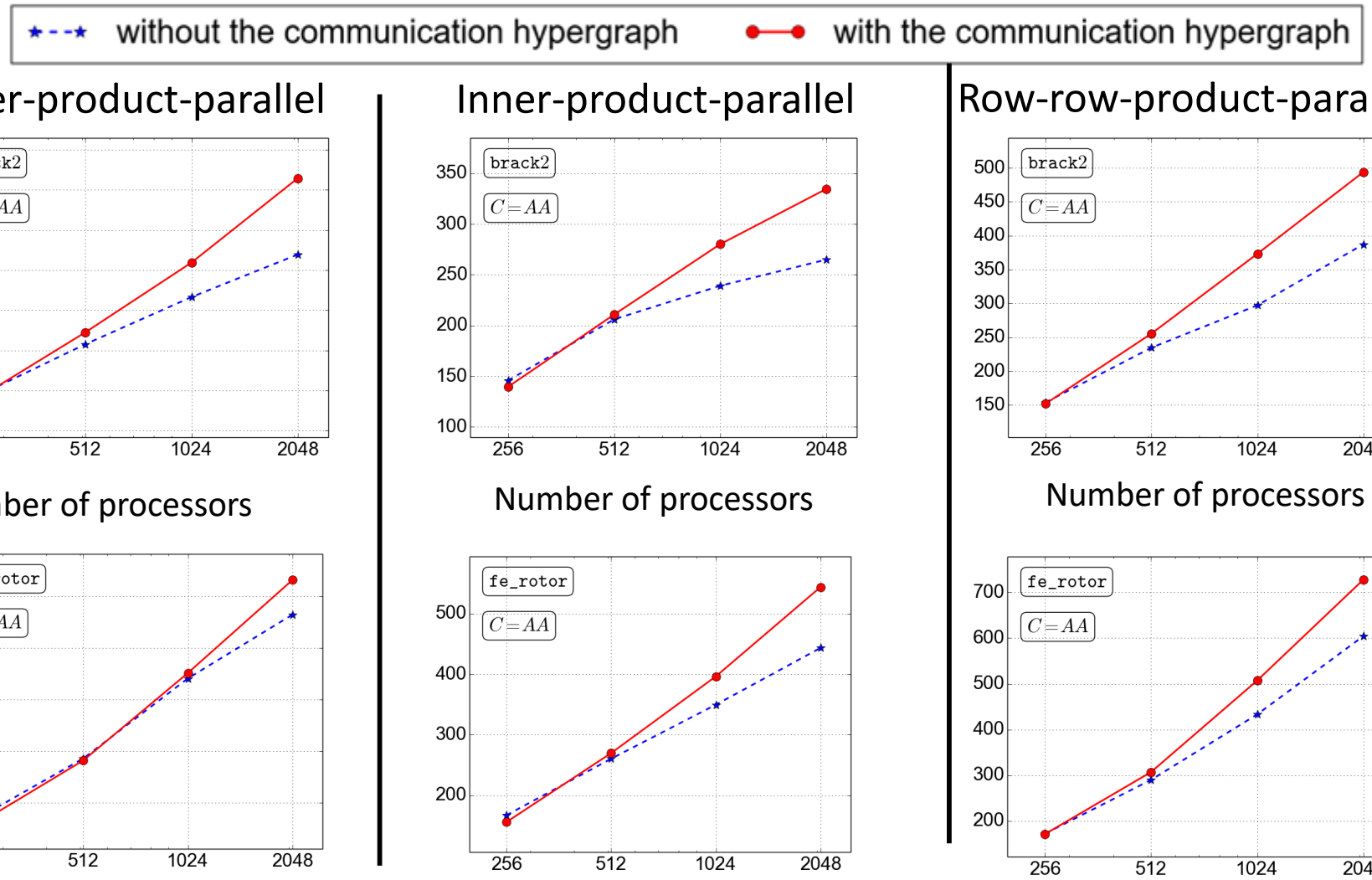
CHG enhanced 2D jagged model obtains the most promising results

CHG: communication hypergraph

CKBD: Checkerboard, JGD: Jagged, FG: Fine grain

Reducing latency cost in 2D sparse matrix partitioning models, *R. Oguz Selvitopi and Cevdet Aykanat*, **Parallel Computing**, 2016.

SpGEMM: Strong Scaling Experiments (Communication hypergraph models)

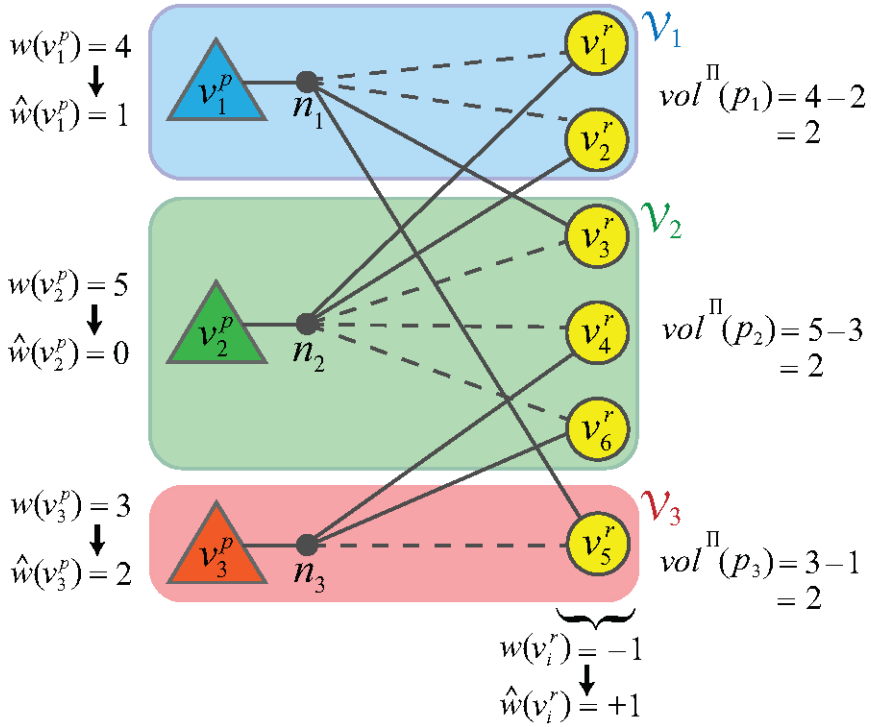


25 C=AA SpGEMM instances

CHG enhanced Row-row-parallel model obtains the most promising results

Partitioning models for scaling parallel sparse matrix-matrix multiplication, *Kadir Akbudak, Oguz Selvitopi, Cevdet Aykanat*, **ACM Transactions on Parallel Computing (TOPC)**, 2018.

Enhanced for encoding send-volume balancing in reduce operations



- Original model does not encapsulate minimizing **maximum volume loads** on irregular reduce tasks
- New vertex weights:
  - **Degree** for processor vertices
  - **-1** for comm task vertices
- Part weights correctly encapsulate send volume loads of processors
- Tools do not support **negative** weights
- Vertex reweighting scheme
  - **max degree - Degree** for processor vertices
  - **+1** for comm task vertices

Reduce Operations: Send Volume Balancing While Minimizing Latency, *M. Ozan Karsavuran, Seher Acer and C. Aykanat*, **IEEE Transactions on Parallel and Distributed Systems,** 2020.

2. Multi-stage hypergraph partitioning for Cartesian partitioning

Metrics & Optimization

minimization of **total communication volume**

upper bound on **latency metrics - total/maximum # of mssgs**

Methodology & Key Features

partitioning along each dimension

multi-constraint partitioning

HP models for Cartesian partitioning

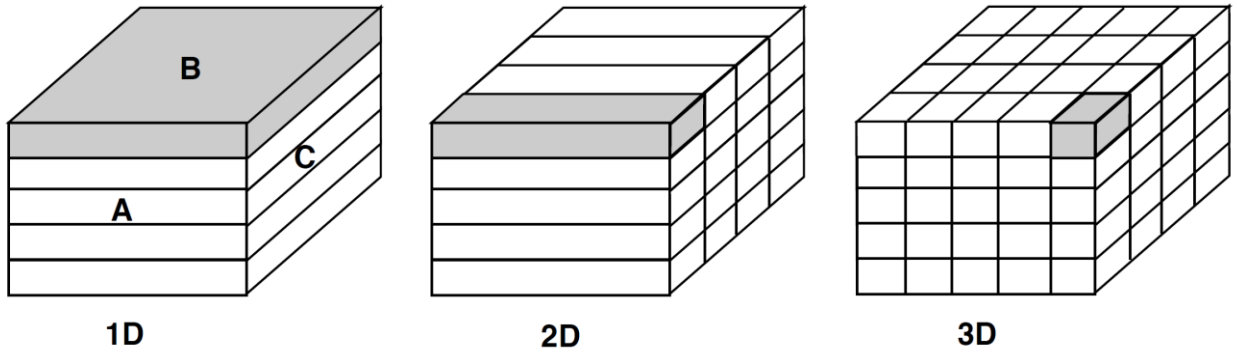
- **multi-dimensional** workload arrays
  - SpMV: **2D** sparse matrix
  - Tensor: **N-dimenional** sparse workload array for N-mode tensor
  - SpGEMM: **3D** sparse workcube

- Assumption: **multi-dimensional** virtual processor topology (VPT)

- Multi-dimensional hypergraph partitioning framework that matches VPT dimensions
  - Enforces upper bounds on **latency cost metrics**
- At each partitioning stage
  - Minimize the **total comm volume** along the respective dimension
- **Multi-constraint** vertex weight formulation
  - Encode **computational load balance**

Cartesian Partitioning Applications

- First proposed for **2D row-column-paralel SpMV** [1][2]
  - 2D paralell SpMV [3]: pre- and post-comm along cols and rows of VPT
- Extended to **N-dimensional sparse tensor decomposition** [4]
  - CPD-ALS
- Recently enhanced for **2D- and 3D-parallel SpGEMM** algorithms [5]
  - 2D: Sparse Summa [6]
  - 3D: Split-3D-SpGEMM [7]
  - 2D- and 3D-cartesian partitioning of a 3D task domain



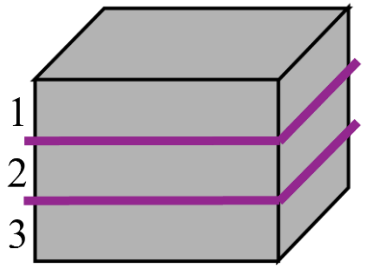
[1] A hypergraph-partitioning approach for coarse-grain decomposition, *Umit. V. Çatalyürek and Cevdet Aykanat,* ACM/IEEE SC2001.
[2] On Two-Dimensional Sparse-Matrix Partitioning: Models, Methods and a Recipe, *Umit V. Çatalyürek, Cevdet Aykanat and Bora Ucar*, SIAM Journal on Scientific Computing,, 2010.
[3] An efficientparallel algorithm for matrix-vector multiplication, *B. Hendrickson, R. Leland, and S. Plimpton*, Int. J.High Speed Computing, 1995
[4] Improving medium-grain partitioning for scalable sparse tensor decomposition, *Seher Acer, Tugba Torun, Cevdet Aykanat*, IEEE TPDS, 2018.
[5] Cartesian Partitioning Models for 2D and 3D Parallel SpGEMM Algorithms, *Gunduz Vehbi Demirci and Cevdet Aykanat,* IEEE TPDS (under review)
[6] Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments, *A. Buluc¸ and J. R. Gilbert*, SIAM Journal on Scientific Computing, 2012.
[7] Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication, *A. Azad, G. Ballard, A. Buluc, J. Demmel, L. Grigori, O. Schwartz, S. Toledo, and S. Williams*, SIAM Journal on Scientific Computing, 2016.

Hypergraph Model for Cartesian Partitioning of 3D Tensor

- **3-stage** hypergraph partitioning model with *total cutsize = total communication volume*
  - **Vertices** represent slices and nets represent (sub)slices
  - **Net** n connects vertex v if intersection of their slices is a nonzero fiber
  - Direct extension of 2D **checkerboard** does not encode comm volume correctly
    - Previos partitions incur **subnets** in the later HP models

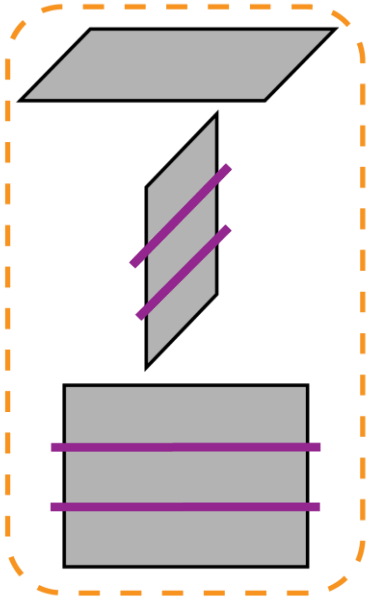
QxRxS

3x4x2 processor mesh



Stage 1

- Hypergraph $\mathcal{H}^A = (\mathcal{V}^A, \mathcal{N}^B \cup \mathcal{N}^C)$
  - one vertex for each horizontal slice
  - one net for each lateral slice, one net for each frontal slice
- Q-way partition of $\mathcal{H}^A$
  - Q chunks of horizontal slices
  - dividing slices along other modes into Q subslices

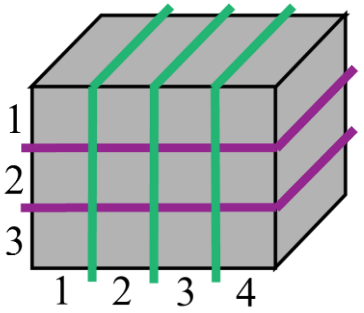


Hypergraph Model for Cartesian Partitioning of 3D Tensor

- **3-stage** hypergraph partitioning model with *total cutsize = total communication volume*
  - **Vertices** represent slices and nets represent (sub)slices
  - **Net** n connects vertex v if intersection of their slices is a nonzero fiber
  - Direct extension of 2D **checkerboard** does not encode comm volume correctly
    - Previos partitions incur **subnets** in the later HP models

QxRxS

3x4x2 processor mesh

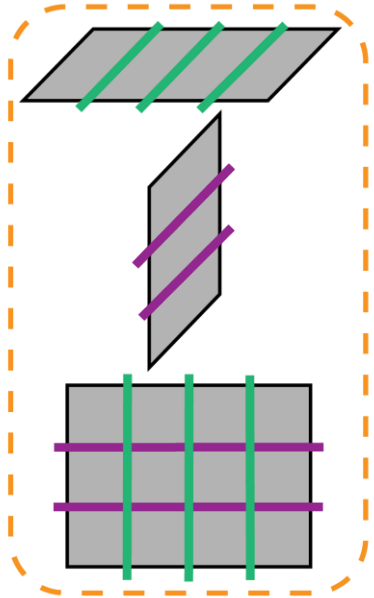


Stage 1

- Hypergraph $\mathcal{H}^A = (\mathcal{V}^A, \mathcal{N}^B \cup \mathcal{N}^C)$
  - one vertex for each horizontal slice
  - one net for each lateral slice, one net for each frontal slice
- Q-way partition of $\mathcal{H}^A$
  - Q chunks of horizontal slices
  - dividing slices along other modes into Q subslices

Stage 2

- Hypergraph $\mathcal{H}^B = (\mathcal{V}^B, \mathcal{N}^A \cup \mathcal{N}^C)$
  - one vertex for each lateral slice
  - one net for each horizontal slice, one net for each frontal ***subslice***
- R-way partition of $\mathcal{H}^B$
  - R chunks of lateral slices
  - dividing (sub)slices along other modes into R subslices

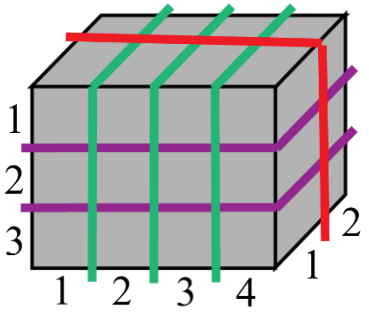


Hypergraph Model for Cartesian Partitioning of 3D Tensor

- **3-stage** hypergraph partitioning model with *total cutsize = total communication volume*
  - **Vertices** represent slices and nets represent (sub)slices
  - **Net** n connects vertex v if intersection of their slices is a nonzero fiber
  - Direct extension of 2D **checkerboard** does not encode comm volume correctly
    - Previos partitions incur **subnets** in the later HP models

QxRxS

3x4x2 processor mesh

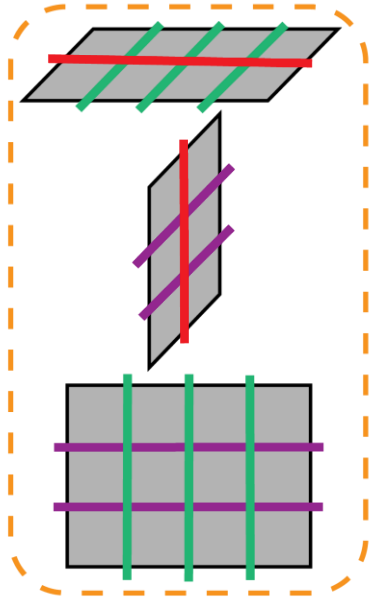


Stage 2

- Hypergraph $\mathcal{H}^B = (\mathcal{V}^B, \mathcal{N}^A \cup \mathcal{N}^C)$
  - one vertex for each lateral slice
  - one net for each horizontal slice, one net for each frontal ***subslice***
- R-way partition of $\mathcal{H}^B$
  - R chunks of lateral slices
  - dividing (sub)slices along other modes into R subslices

Stage 3

- Hypergraph $\mathcal{H}^C = (\mathcal{V}^C, \mathcal{N}^A \cup \mathcal{N}^B)$
  - one vertex for each frontal slice
  - one net for each horizontal ***subslice***, one net for each lateral ***subslice***
- S-way partition of $\mathcal{H}^C$
  - S chunks of frontal slices
  - dividing subslices along other modes into S subsubslices



Experiments

- Partitioning: PaToH with 4% imbalance at each stage
- Baseline: DMS [1] (random permutation)
- Parallel CPD-ALS code: implemented in C (MPI, Intel MKL)
- System: Cray XC40 (two 12-core Intel Haswell Xeon, 128 GB)
- Decomposition rank: $F = 16$
- Number of processors: 64, 128, 256, 512, 1024
- 12 sparse tensors: nonzeros between 466K and 140M; nine 3-mode and three 4-mode

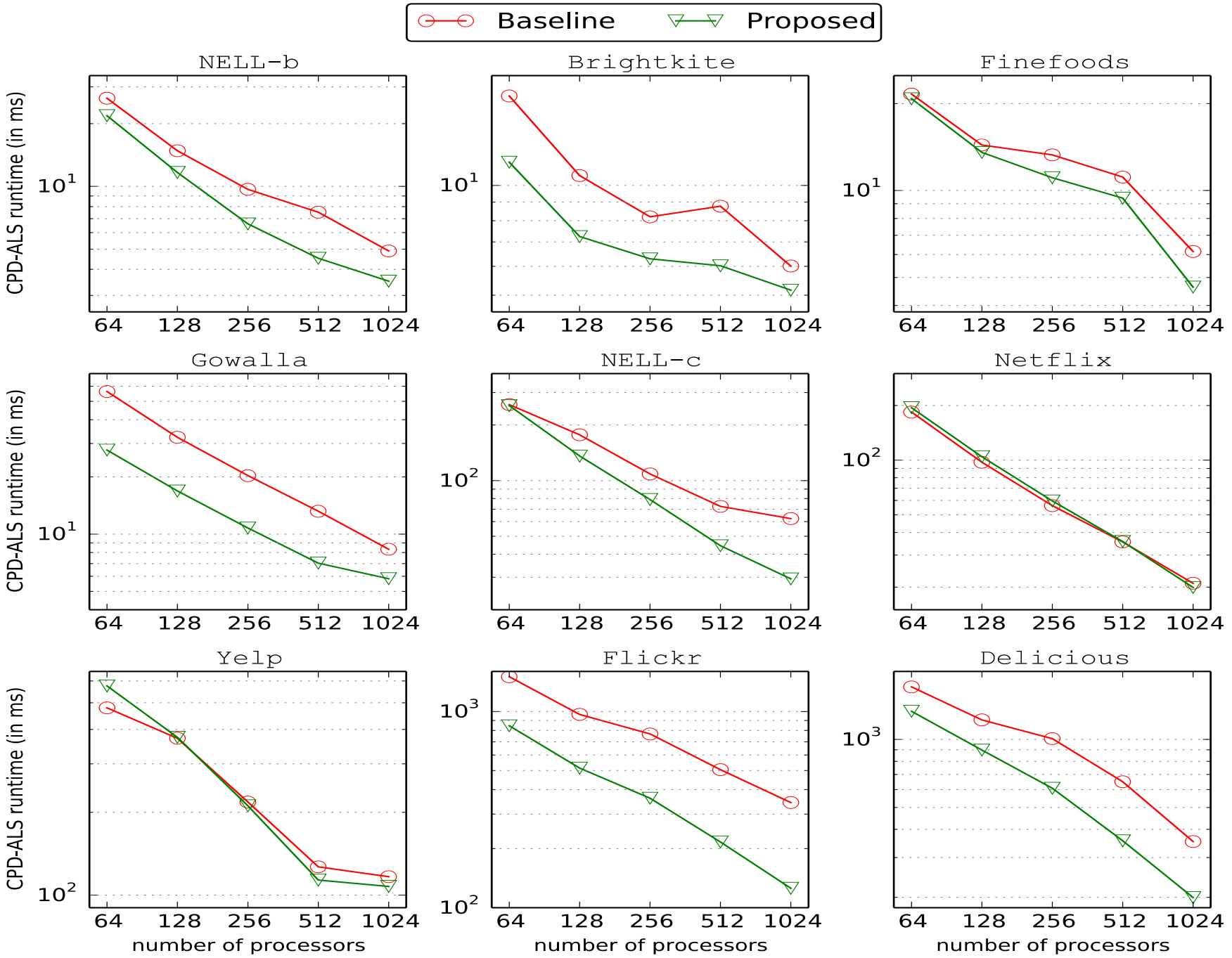
Average results of proposed model normalized w.r.t. those of baseline model

number of processors	imbalance	number of messages		communication volume		parallel CPD-ALS runtime	
		max	avg	max	avg	communication	total
64	1.01	0.97	0.93	0.61	0.42	0.50	0.82 18%
128	1.01	0.97	0.93	0.60	0.45	0.56	0.78 22%
256	1.05	0.97	0.91	0.60	0.49	0.59	0.74 26%
512	1.05	0.98	0.90	0.53	0.51	0.61	0.72 28%
1024	1.05	0.97	0.85	0.53	0.53	0.61	0.72 28%
overall	**1.03**	**0.97**	**0.90**	**0.57**	**0.48**	**0.57**	**0.76**
reduction rates:	**-3%**	**3%**	**10%**	**43%**	**52%**	**43%**	**24%**

[1] S. Smith and G. Karypis, *A medium-grained algorithm for distributed sparse tensor factorization*, IPDPS'16.

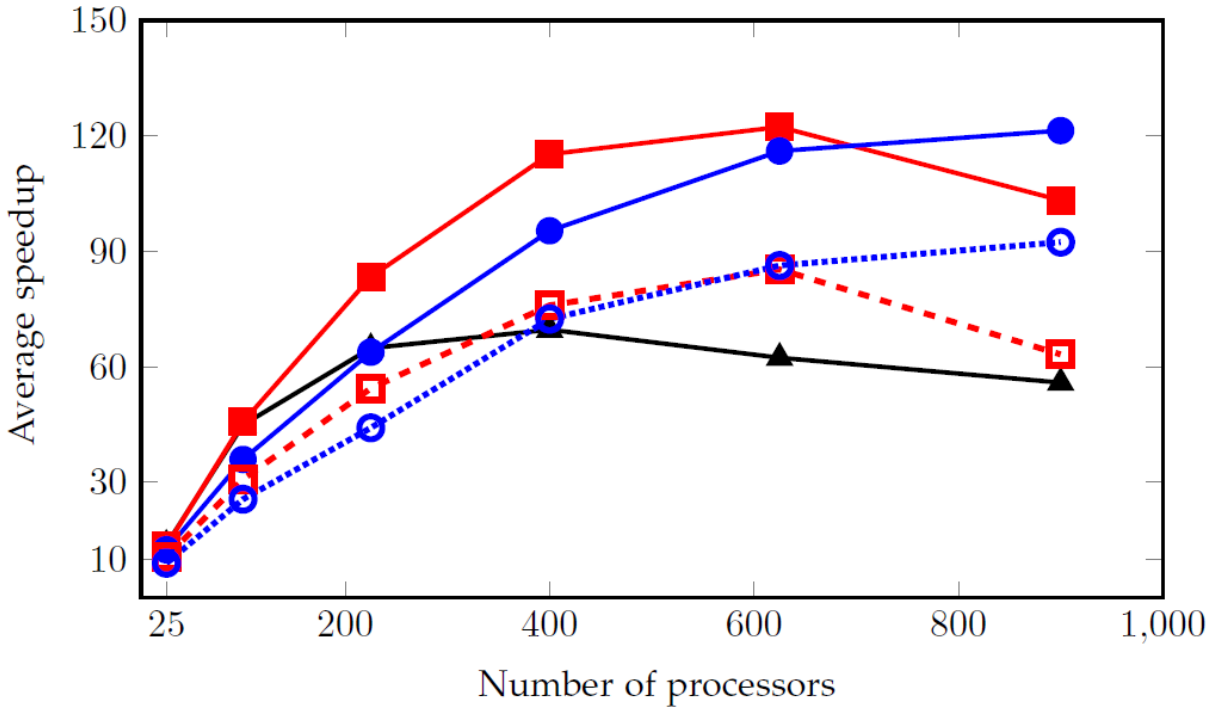
Experiments

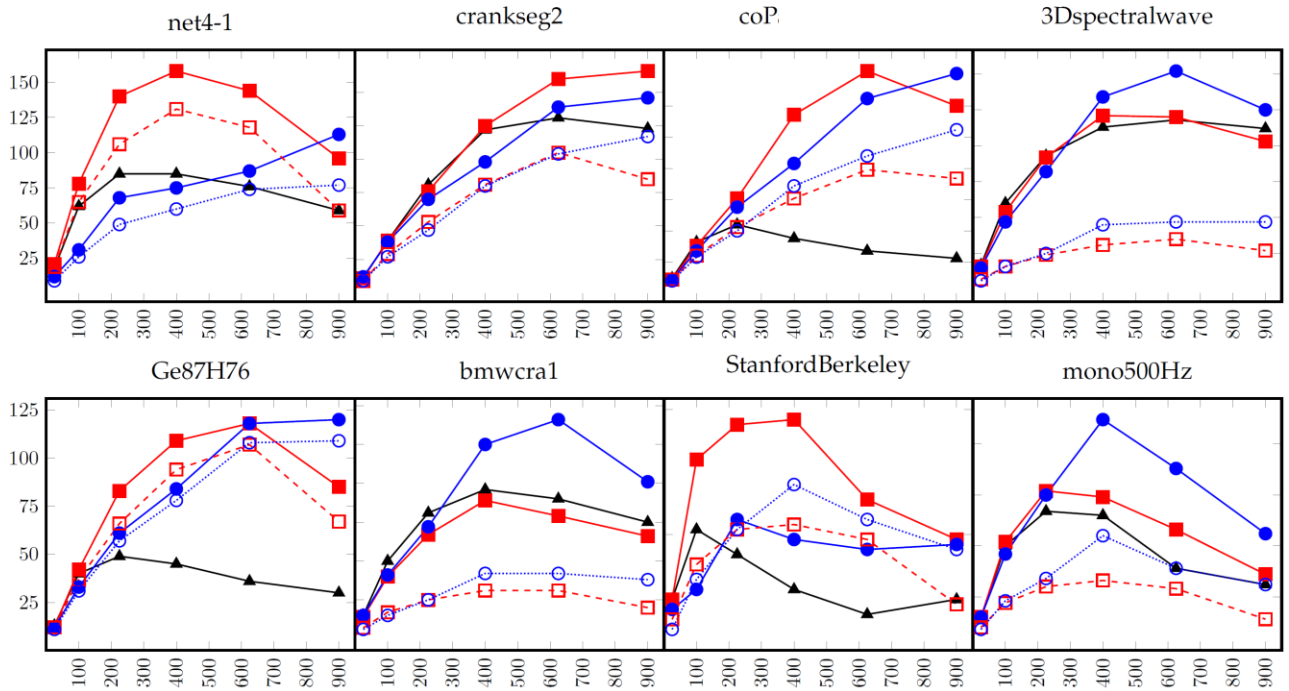
- Strong scaling results
- CPD-ALS runtime
  - Baseline: random cartesian partitioning



Experiments for 2D and 3D SpGEMM

H:Hypergrah
R:Random





Average speedup curves for 20 C=AA instances

- With increasing number of processors
  - performance gap between random and hypergraph partitioning increases significantly
  - 3D begins to perform better than 2D

3. Recursive hypergraph partitioning

Metrics & Optimization

minimization of **total comm vol** and **total # of mssgs**

Methodology & Key Features

single phase

message net augmentation

Recursive HP framework

- Addresses **total comm volume** and **total # of mssgs**
- **Single phase** framework

- Standard HP models: **nets** encapsulate **total comm vol**
- Augment standard HP model with **message nets**
  - encapsulate **total # of messages**

- First proposed for scaling **1D row-parallel and column parallel SpMV** [1]
- Extended for **2D row-column-parallel SpMV** [2]
  - based on **fine-grain** and **medium-grain** [3] partitioning

[1] A Recursive Hypergraph Bipartitioning Framework for Reducing Bandwidth and Latency Costs Simultaneously, *Oguz Selvitopi, Seher Acer and Cevdet Aykanat*, IEEE TPDS, 2017.
[2] Optimizing nonzero-based sparse matrix partitioning models via reducing latency, *Seher Acer, Oguz Selvitopi, Cevdet Aykanat*, Journal of Parallel and Distributed Computing, 2018.
[3] A Medium-grain method for fast 2D bipartitioning of sparse matrices, *Daniel Pelt and Rob Bisseling,* **IPDPS** 2014.

RB Framework and Message Nets

Basics

- We augment standard hypergraph model with *message nets*
  - The nets in the standard models: *volume nets*
- Our model relies on *recursive hypergraph bipartitioning*

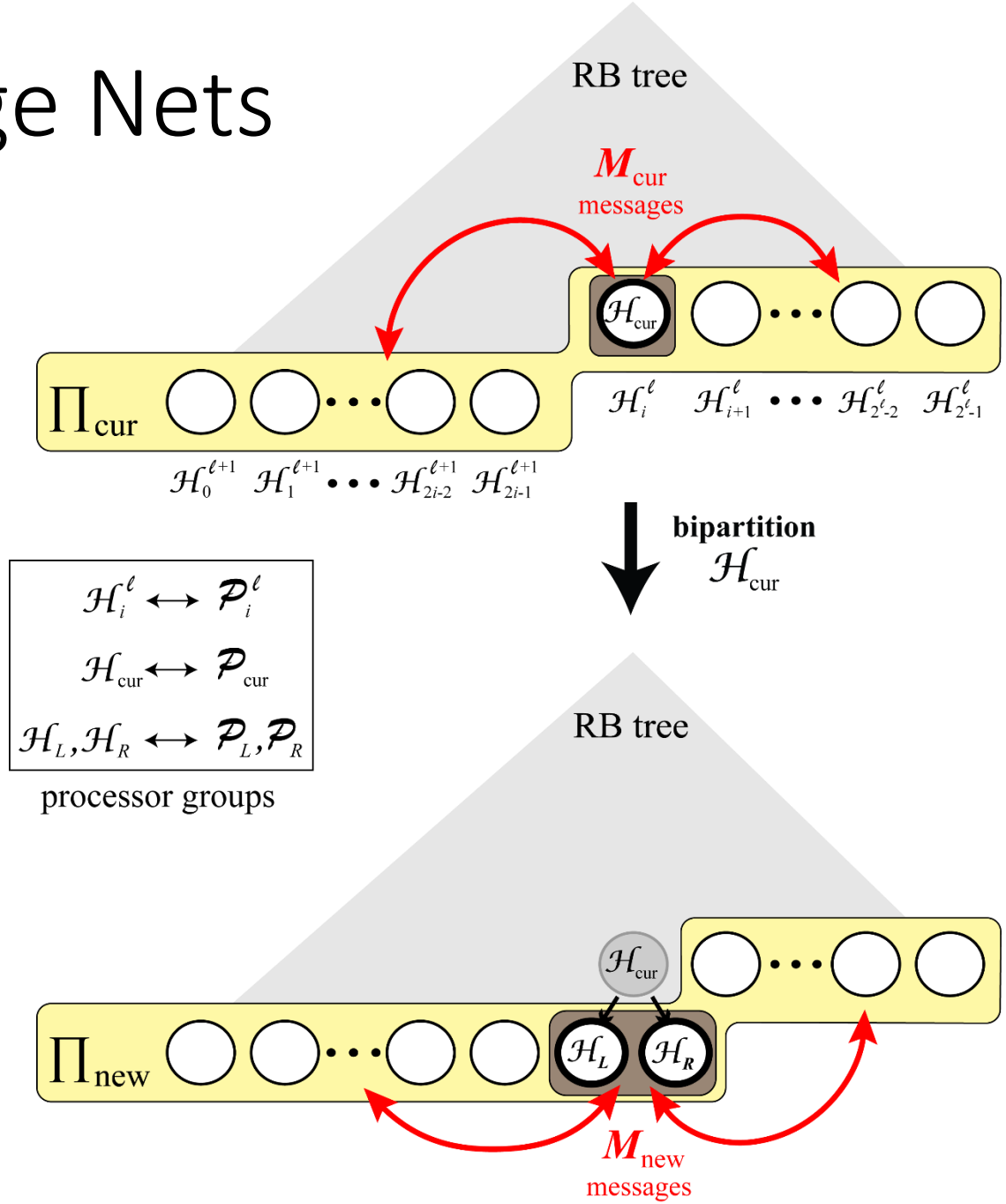
Nets

- Volume nets: maintained via net-splitting
- Message nets: added to the current hypergraph to be bipartitioned

Having **both** net types in bipartitions ⟷ **Simultaneous** reduction of bandwidth and latency costs

Message nets

- A message net connects vertices representing items/tasks that necessitate a message together
  - Such items/tasks are encouraged to be together either in P_L or P_R
- A send net s_k added for each P_k which P_{cur} sends a message to
  - Connects vertices representing input items sent to P_k
- A receive net r_k added for each P_k which P_{cur} receives a message from
  - Connects vertices representing tasks that need input items received from P_k



RB-based Partitioning

Number of cut message nets = $M_{new} - M_{cur}$

Increase in number of messages that P_{cur} communicates with others

Correctness

Message nets and volume nets with respective costs of t_s and t_w

- Minimizing cutsize ≈ minimizing the increase in communication cost
- Provides a more **accurate** communication cost **representation**

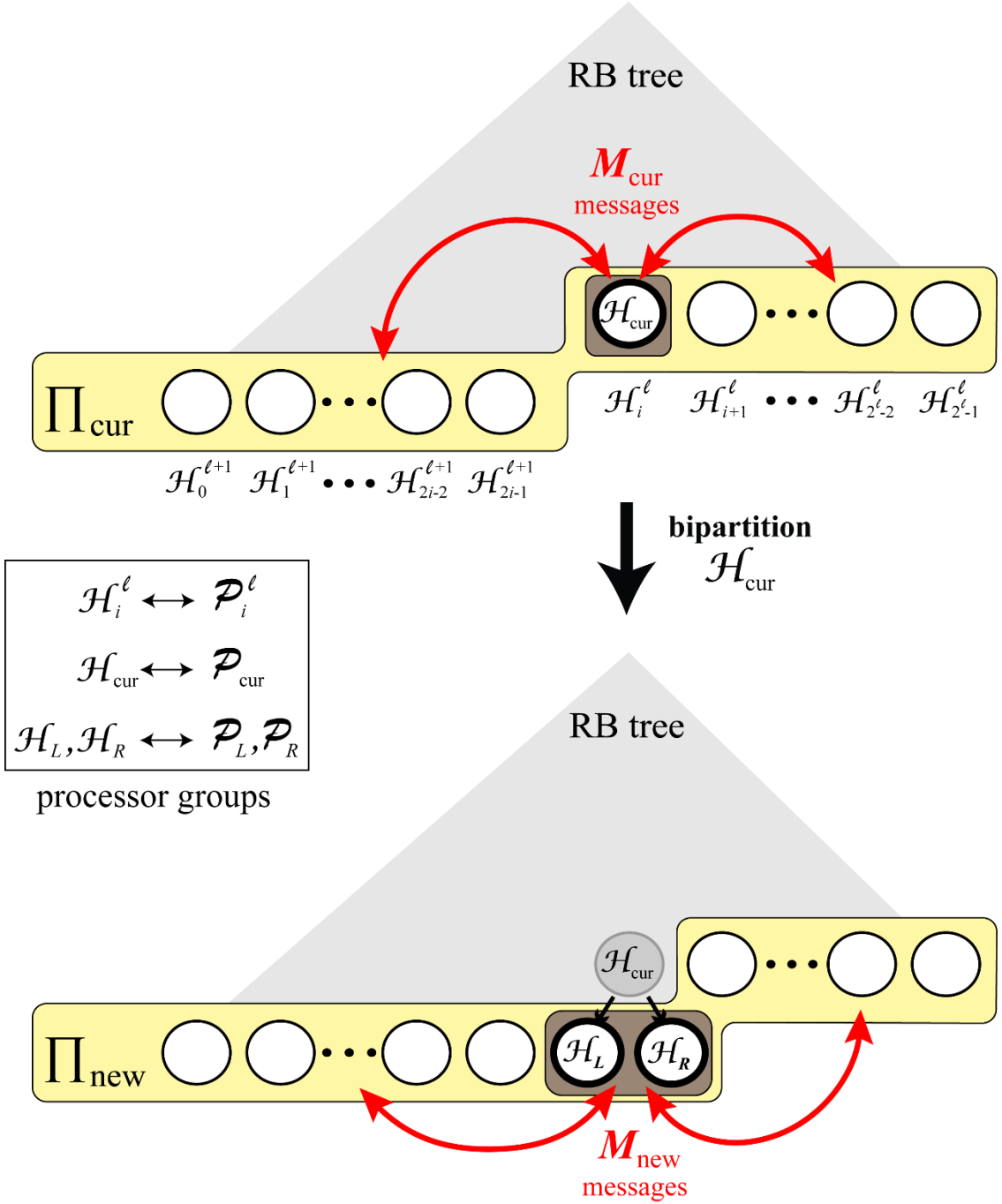
Flexible

Can be realized by using any hypergraph partitioning tool

Cheap

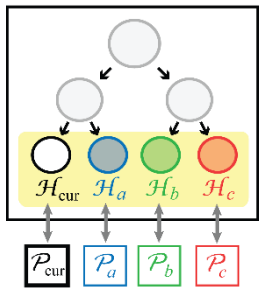
$Cost$(our model) = $Cost$(standard model) + $O(p \log_2 K)$

- Our model traverses each pin once for each RB tree level

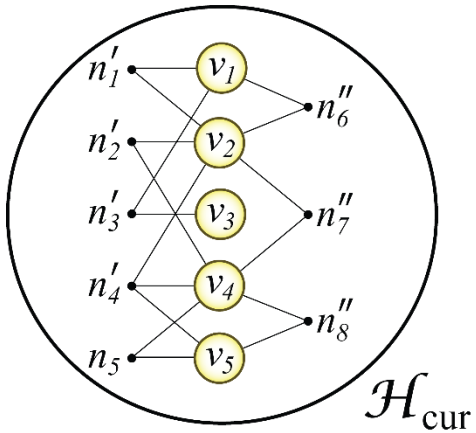


Hypergraphs in the Leaf Nodes of the Recursive Bipartitioning Tree

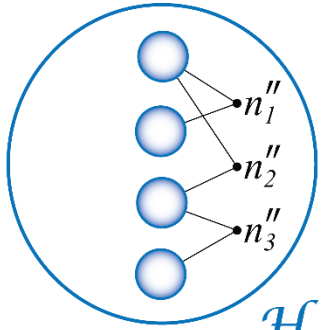
Recursive Bipartitioning Tree Status



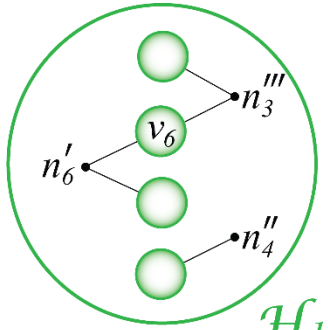
associated processor groups



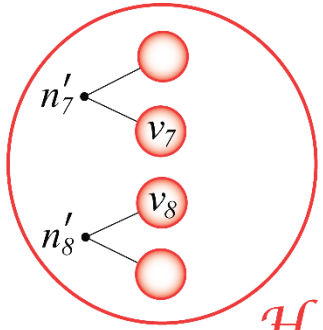
$\mathcal{H}_{\text{cur}}$



$\mathcal{H}_a$



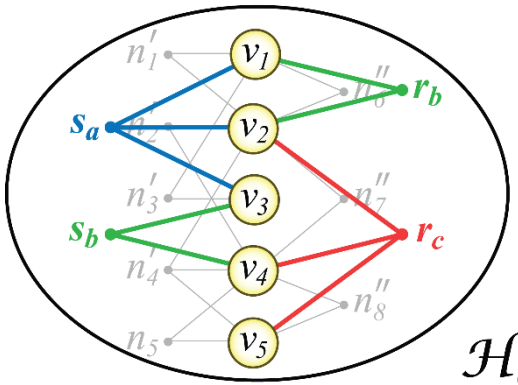
$\mathcal{H}_b$



$\mathcal{H}_c$

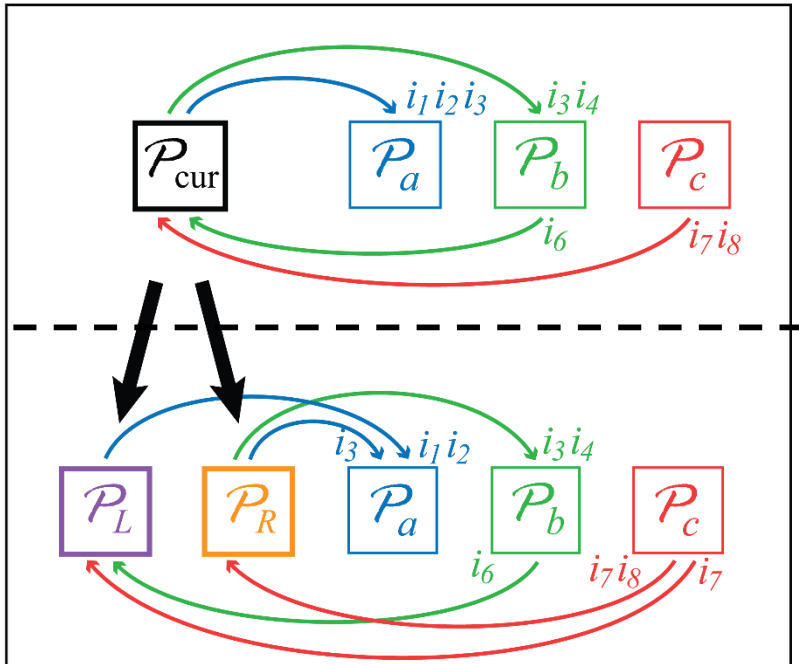
addition of message nets

s_a, s_b: send nets
r_b, r_c: receive nets

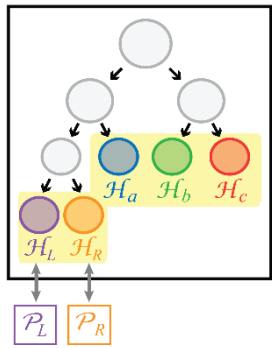


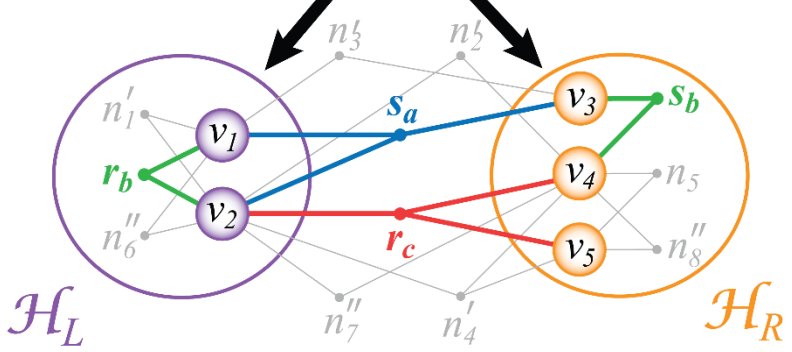
$\mathcal{H}^M_{\text{cur}}$

messages communicated by $\mathcal{P}_{\text{cur}}$



bipartitioning $\mathcal{H}^M_{\text{cur}}$





$\mathcal{H}_L$ $\mathcal{H}_R$

Experiments

For message net cost of 50 with unit volume net cost:

- Total number of messages: $\mathbf{35\% - 44\%}$ improvement
- Maximum number of messages: $20\% - 31\%$ improvement
- Total volume: $17\% - 48\%$ degradation
- Maximum volume: $25\% - 85\%$ degradation
- Partitioning time: $8\% - 33\%$ degradation
- Parallel SpMV time: $\mathbf{8\% - 29\%}$ improvement

↑ message net cost ⟷ ↑ improvements in latency metrics

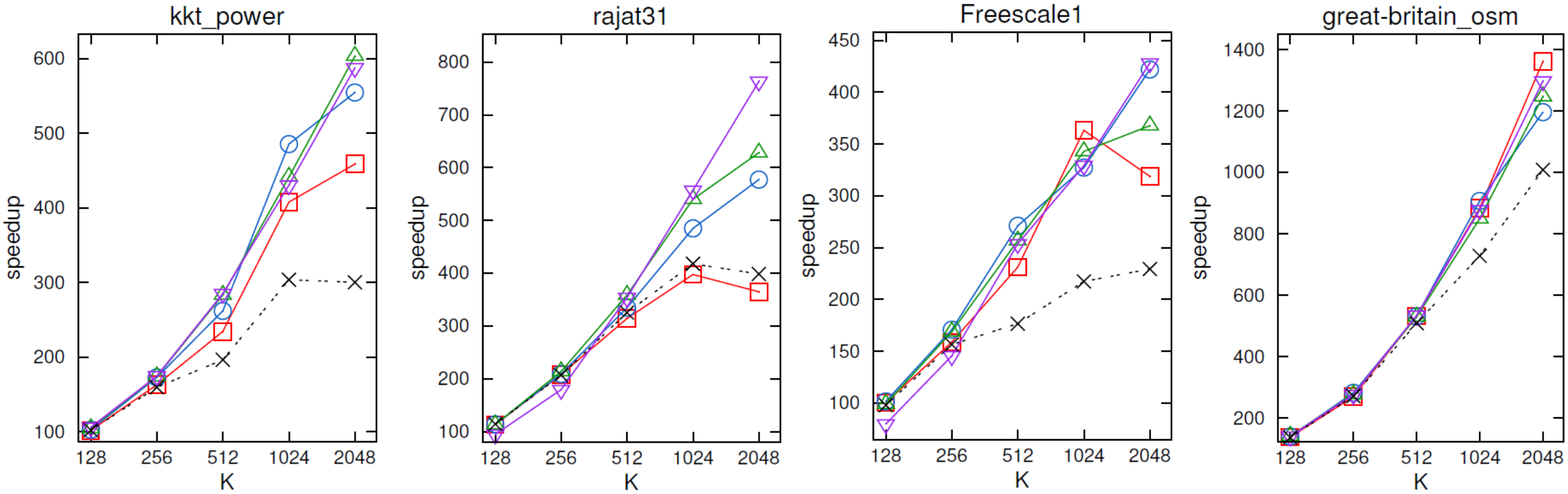
↑ improvement rate in latency metrics ⟷ ↑ degradation rates in bandwidth metrics

↑ number of processors ⟷ ↑ improvements in parallel SpMV time

Average results for 30 SpMV instances normalized with respect to standard model

message net cost	K	number of messages		volume		partitioning time	parallel SpMV time
		tot	max	tot	max		
50	128	0.65	0.76	1.17	1.25	1.08	**0.924**
	256	0.59	0.70	1.25	1.44	1.14	**0.846**
	512	0.56	0.69	1.33	1.57	1.21	**0.760**
	1024	0.57	0.74	1.41	1.69	1.24	**0.715**
	2048	0.59	0.80	1.48	1.85	1.33	**0.708**

1D parallel SpMV Experiments



HP-L-50: refers to message net cost of 50 with unit volume net cost

4. Regularization framework

Metrics & Optimization

trade-off between **bandwidth** and **latency** costs

Methodology & Key Features

two phase

many small-sized messages

Regularization Framework

- **Regularize** irregular **P2P** messages
  - impose regular communication pattern(s)
  - **VPT (V**irtual **P**rocess **T**opology**)**

- Exploit **VPT** to attain a trade-off between **bandwidth** and **latency** cost metrics
- Using different **dimensions** in the formation of the **VPT**
  - low-dimensional VPT → favors bandwidth costs over latency costs
  - high-dimensional VPT → favors latency costs over bandwidth costs.

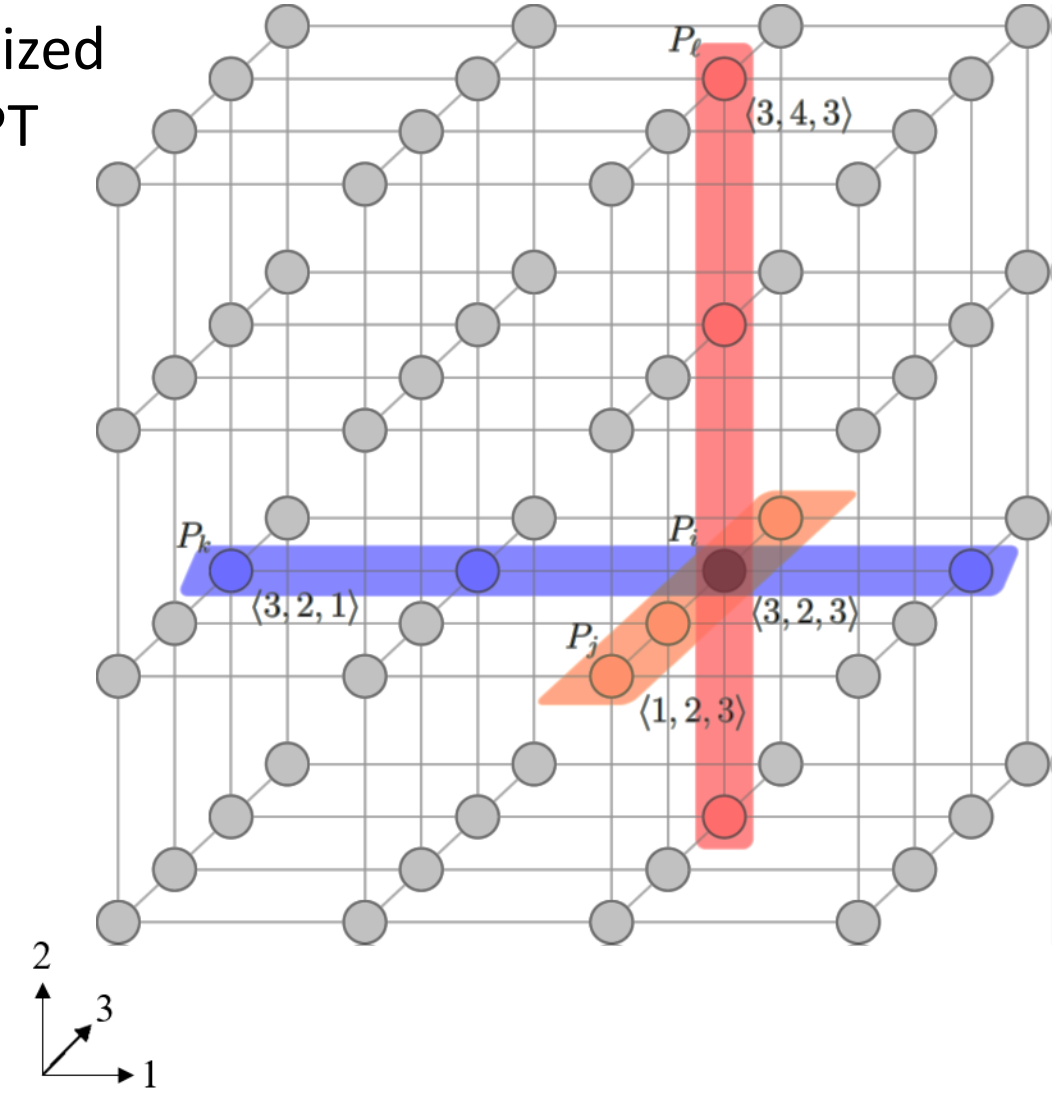
- Especially tailored for latency bound applications where
  - messages are **small** or **medium sized**
  - there is **high variance** in the **message counts** of processors

Oguz Selvitopi and Cevdet Aykanat. "Regularizing Irregularly Sparse Point-to-point Communications." The International Conference for High Performance Computing, Networking, Storage, and Analysis, SC19

Virtual Process Topology (VPT): Basics

- Characterized by
  - **Dimension**
  - **Dimension sizes**
  - **Process neighborhood**

K=64 processes organized into 3-dimensional VPT $T_3(4,4,4)$

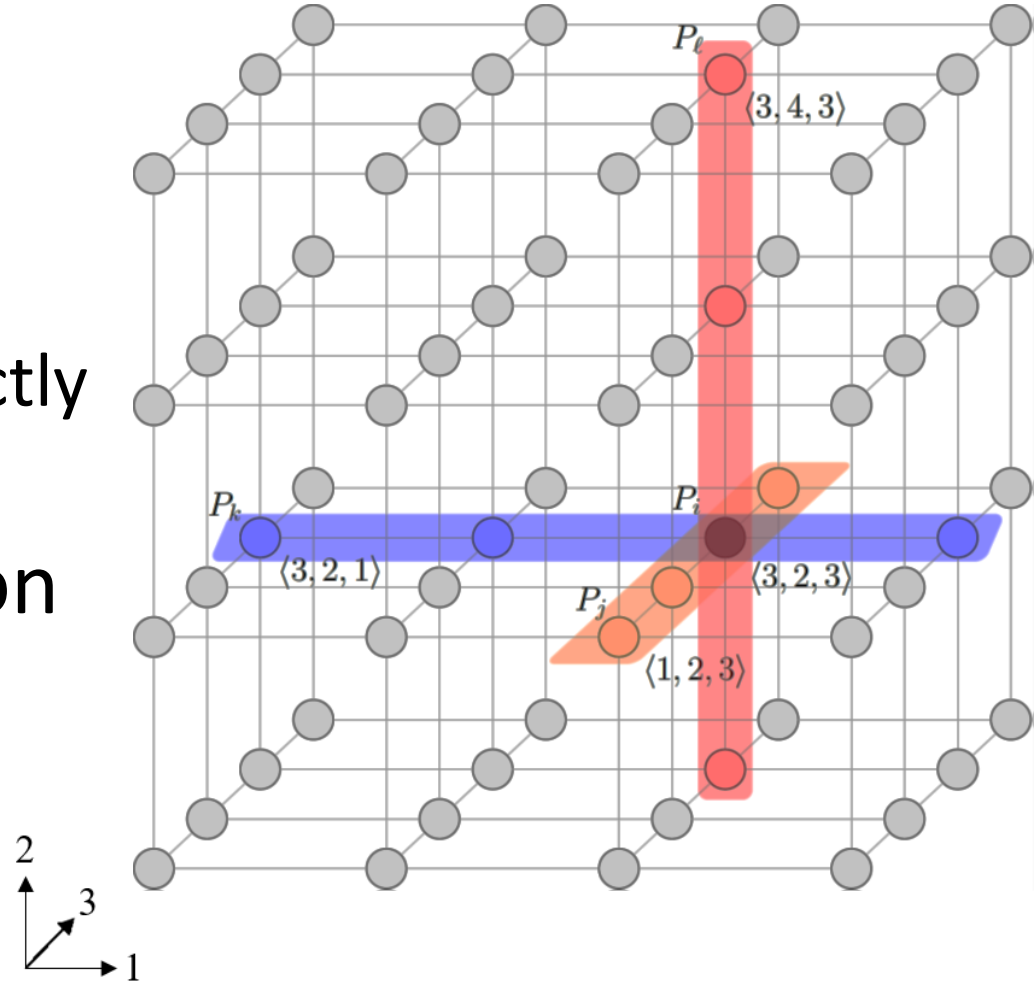


- K processors $P_1, P_2, \ldots, P_K$
- n-dimensional VPT $T_n(k_1, k_2, \ldots, k_n)$
  - k_d: size of dimension $1 \leq d \leq n$
  - $K = k_1 \times k_2 \times \cdots \times k_n$

- Denote each processor P_i as a vector of n coordinates
  - $\langle P_i^n, P_i^{n-1}, \ldots, P_i^1 \rangle$, where $P_i^d \in \{1, 2, \ldots, k_d\}$

VPT: Process neighborhood

- Two processors are **neighbors** if
  - They differ in a single coordinate
  - $N(P_i, d)$ = neighbors of P_i at dimension d
  - Only processes in the same group can directly communicate with each other
- vs. neighborhood definitions in common regularly structured applications



Blue: Neighbors of P_i along 1st dimension

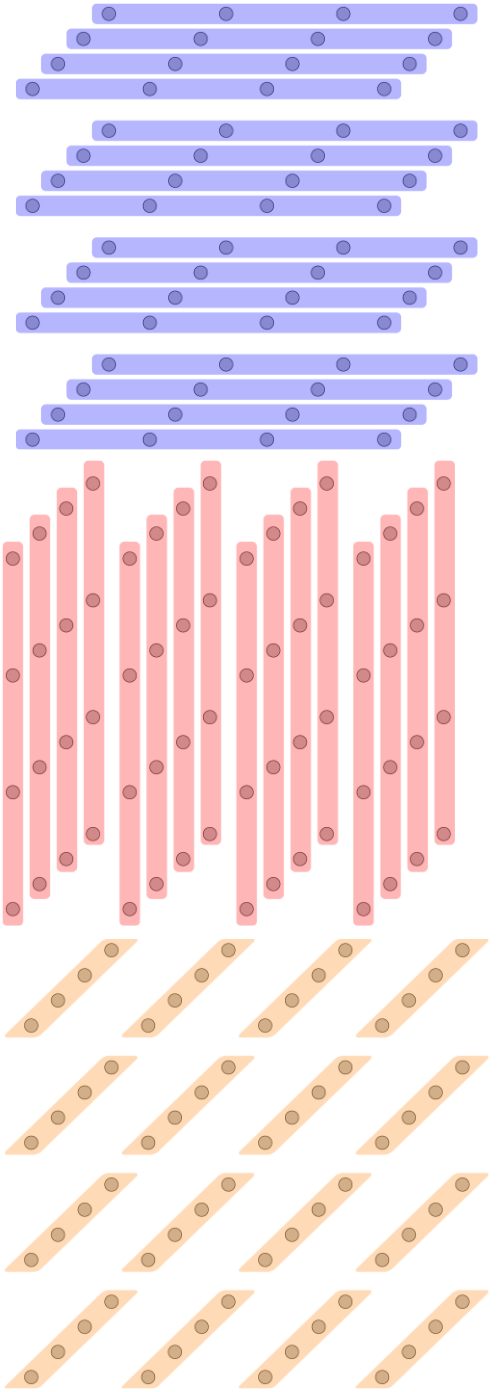
Red: Neighbors of P_i along 2nd dimension

Orange: Neighbors of P_i along 3rd dimension

VPT: Process neighborhood

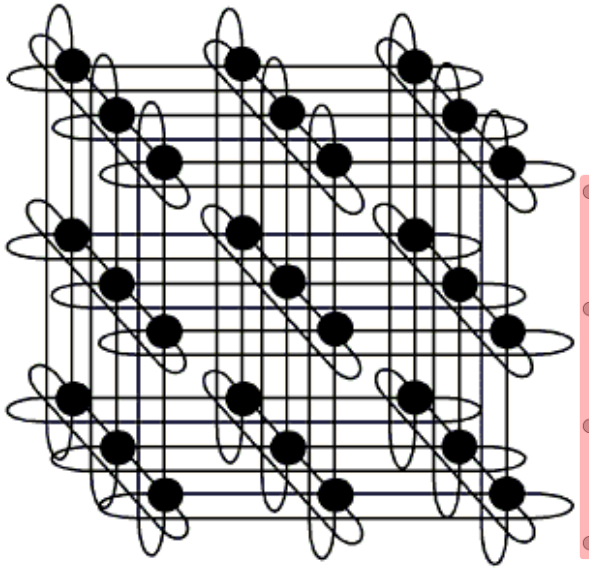
- Two processors are **neighbors** if
  - They differ in a single coordinate
- In dimension *d*, there are K/k_d processor groups, each containing k_d processors
  - Only processes in the same group can directly communicate with each other
- vs. neighborhood definitions in common regularly structured applications

Blue: Neighbor process groups along 1st dimension
Red: Neighbor process groups along 2nd dimension
Orange: Neighbor process groups along 3rd dimension



VPT versus k-ary n-cube networks

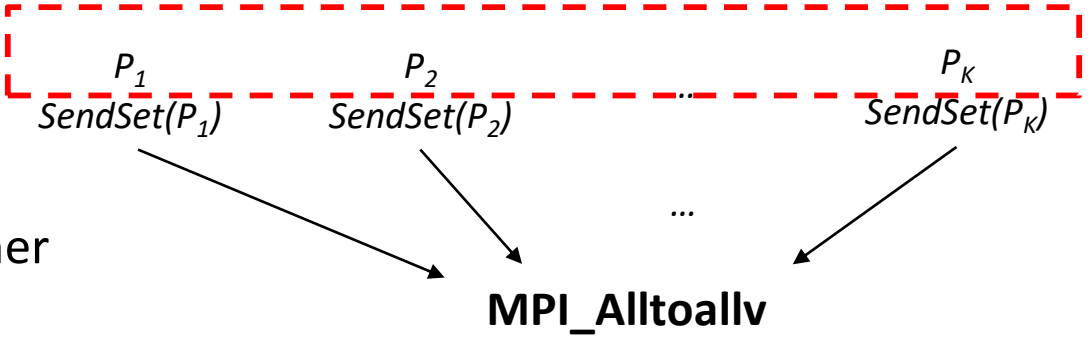
- k-ary n-cube networks
  - Generalization of hypercubes
  - k = # of nodes at each dimension
  - n = # of dimensions
  - nodes in a dimension are connected as a 1D torus

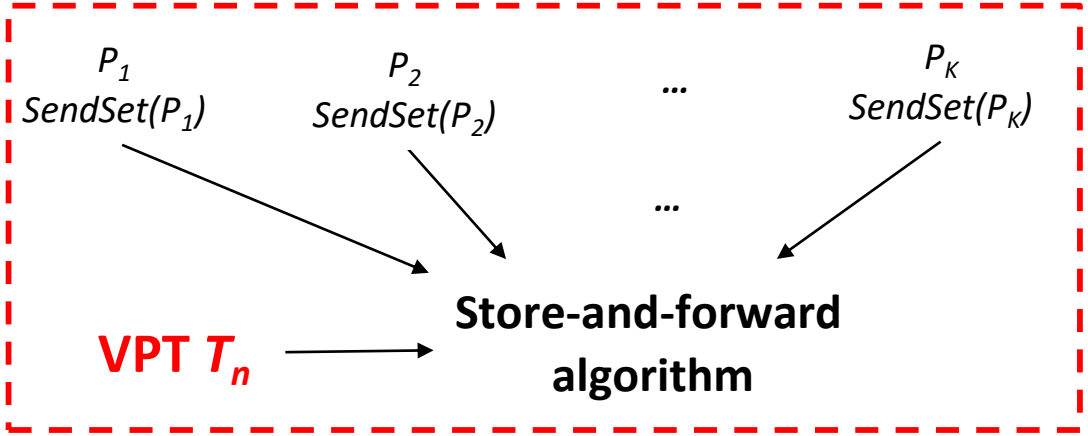


- **Two basic differences**
  - Context: software vs. hardware
  - Neighborhood definition

Store-and-forward Algorithm: Multi-stage Communication

- A few more definitions
  - ***SendSet(P_i)***: subset of processors P_i needs to send a mssg to
  - ***m_{ij}***: the message to be sent from P_i to P_j
- Straightforward approach (***T_1***)
  - No structure: each process may communicate with each other
- **Use VPT to perform communication**
- A VPT ***T_n*** with *n > 1*
  - **disables direct comm** between nonneighbor processes
  - **store-and-forward** scheme to handle nonneighbor mssgs
  - **staged execution:** communication proceeds in *n* stages
  - communication in a stage restricted to neighbor processes along the respective dimension





Communicating a Single Message on VPT

- Message m_{ij}
  - Source P_i, destination P_j
  - At each stage Store or Forward

▷ How many times a message get forwarded?
- Hamming distance
- E-cube routing for hypercubes
- Dimension-ordered deterministic routing

Stage d

m_{ij} with destination
$$P_j = \langle P_j^n, \dots, P_j^{d+1}, \boldsymbol{P_j^d}, P_j^{d-1}, \dots, P_j^1 \rangle$$

Arrives at intermediate process P_k
Decide whether to forward m_{ij} by comparing dth coordinate

$P_j^d = P_k^d$ ➔ Store m_{ij} at this stage

$P_j^d \neq P_k^d$ ➔ Forward m_{ij} to
$\langle P_j^n, \dots, P_j^{d+1}, \boldsymbol{P_k^d}, P_j^{d-1}, \dots, P_j^1 \rangle$

Communicating a Single Message on a VPT

Stage 1:

source $\langle 1,1,\mathbf{2}\rangle$

destination $\langle 3,3,\mathbf{4}\rangle$

to $\langle 1,1,\mathbf{4}\rangle$

Stage 2:

source $\langle 1,\mathbf{1},4\rangle$

destination $\langle 3,\mathbf{3},4\rangle$

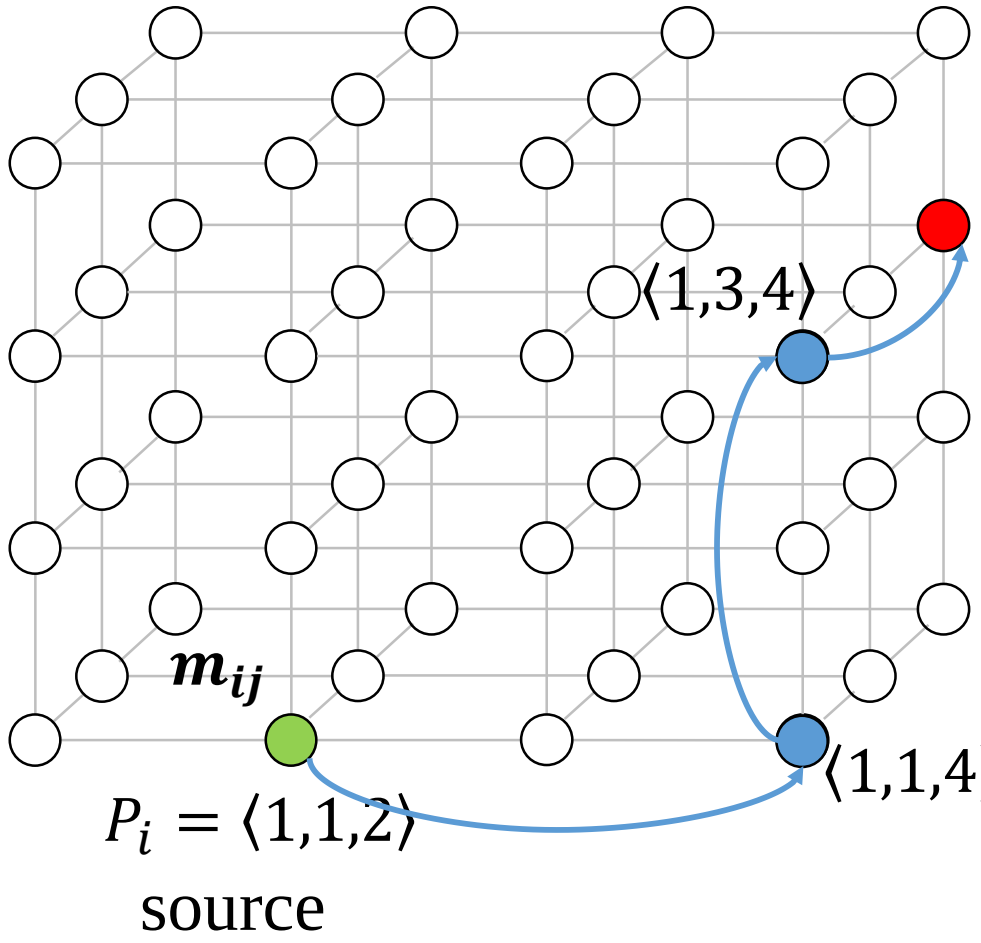
to $\langle 1,\mathbf{3},4\rangle$

Stage 3:

source $\langle \mathbf{1},3,4\rangle$

destination $\langle \mathbf{3},3,4\rangle$

to $\langle \mathbf{3},3,4\rangle$



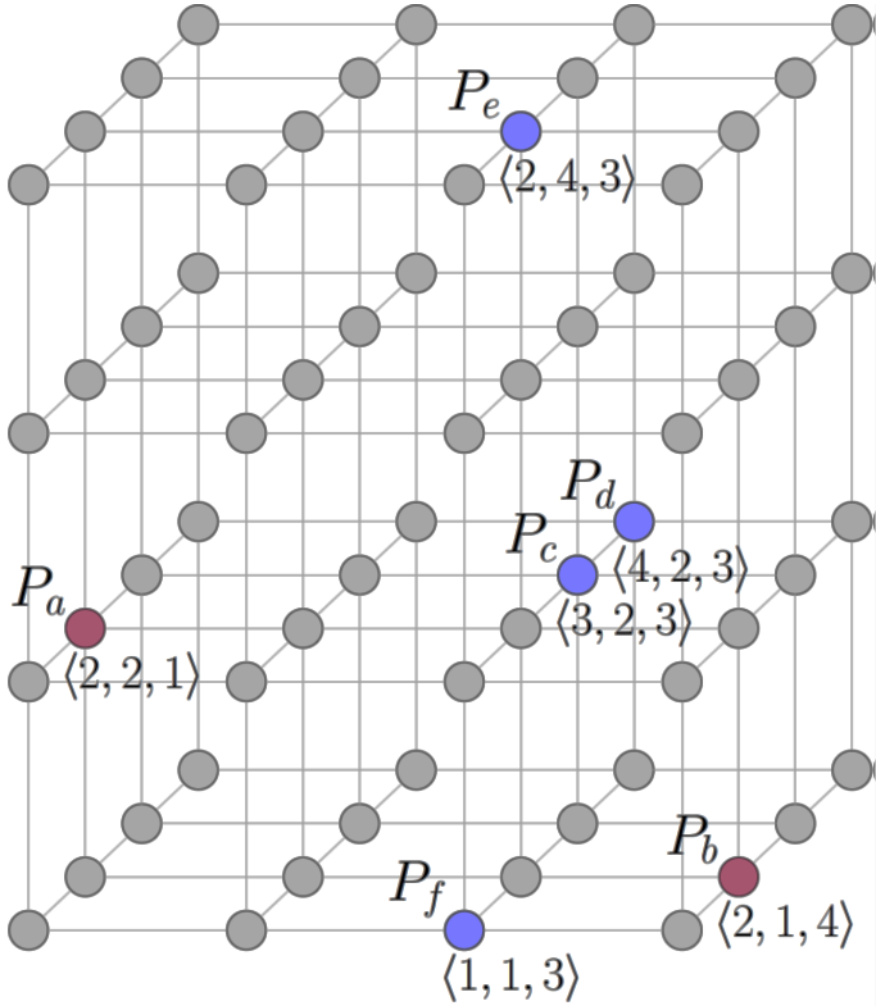
Multiple messages on VPT

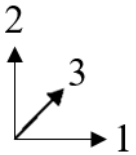
- The communication between P_i and its neighbor P_j
  - A single message containing a number of submessages

- **Submessage**
  - two-tuple: (destination processor, message content)

- Direct message between P_i and P_j → $\boldsymbol{M_{ij}}$
- Submessage with source P_i and destination P_h → $\boldsymbol{(P_h, m_{ih})}$

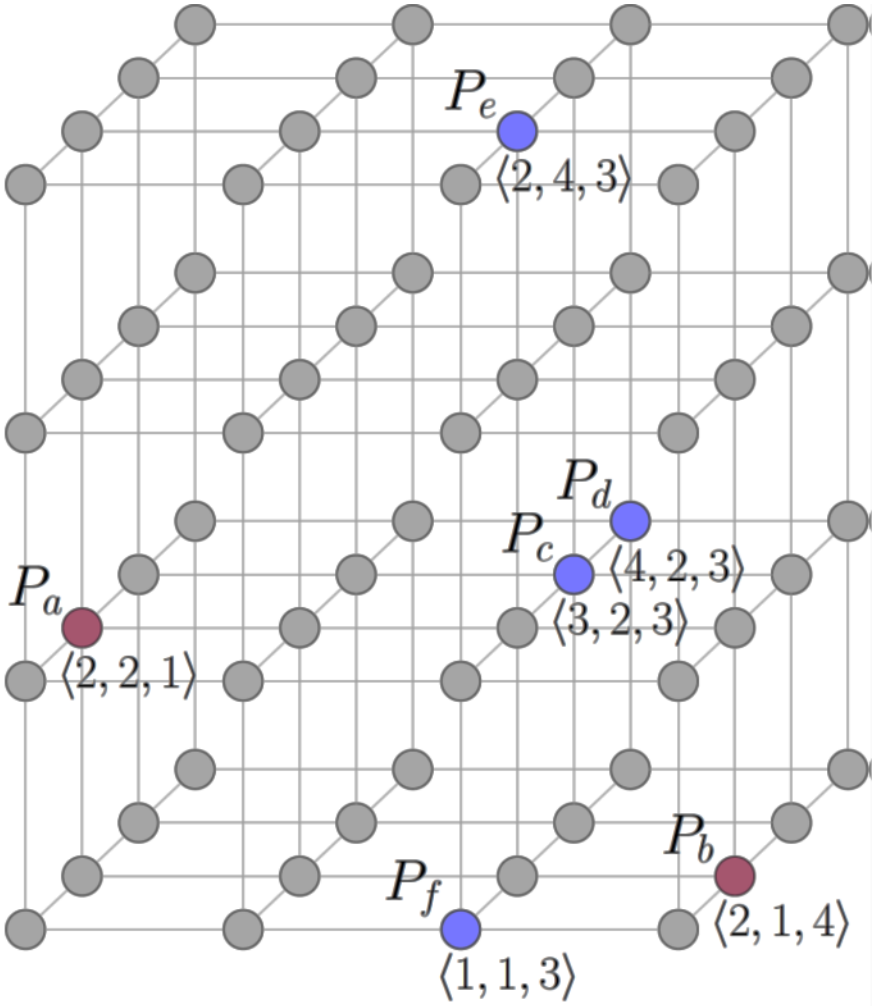
Multiple messages on VPT - Example

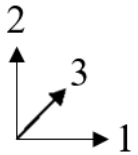




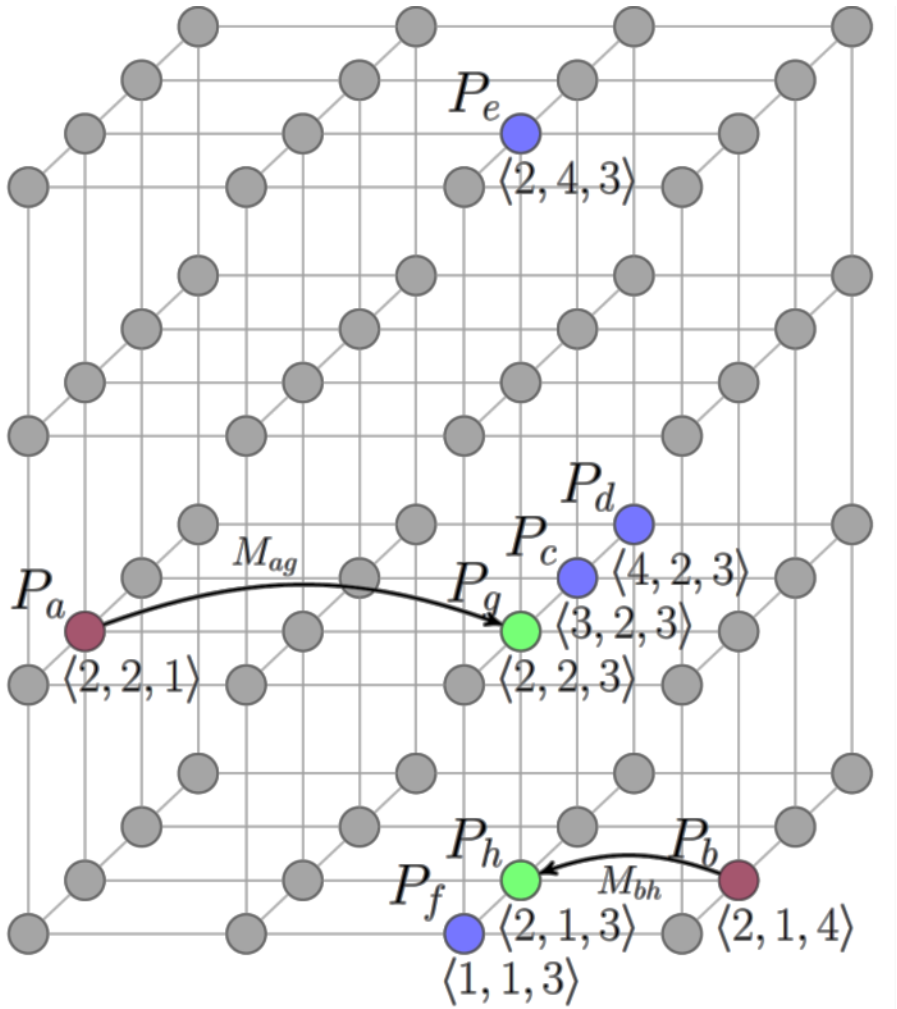
$$SendSet(P_a) = \{P_c, P_d, P_e\}$$
$$SendSet(P_b) = \{P_c, P_d, P_f\}$$

Multiple messages on VPT - Example



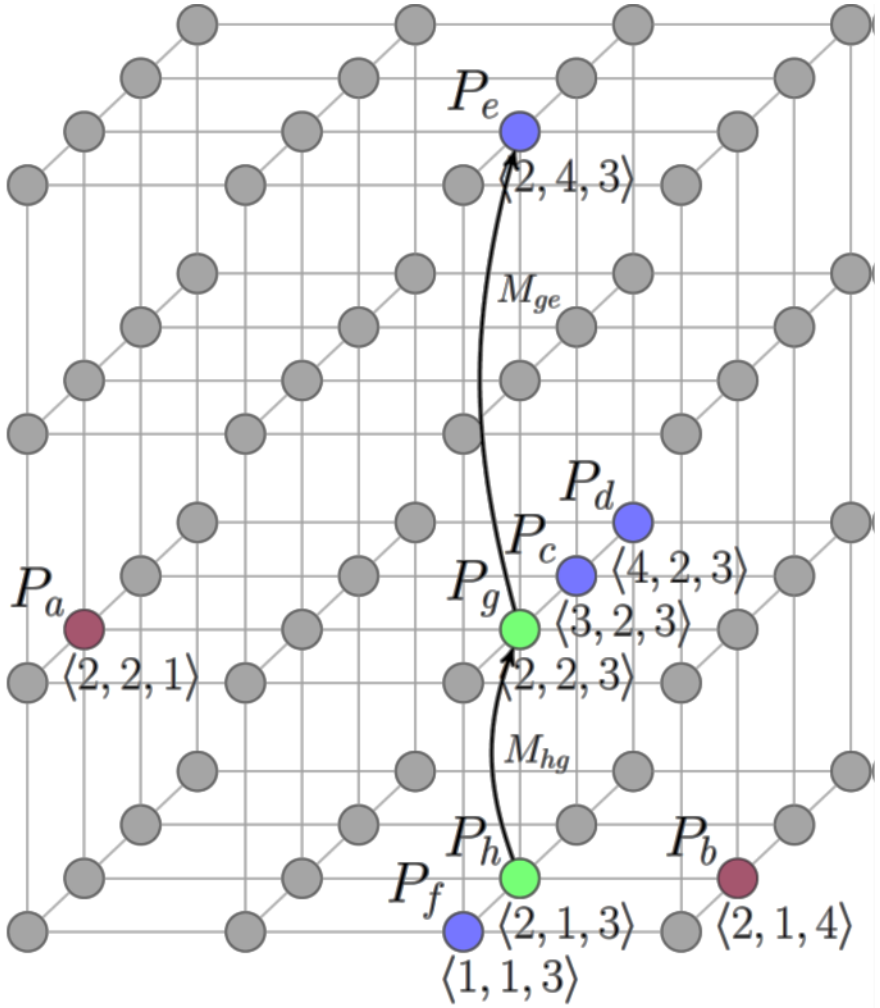


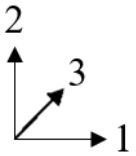
$$SendSet(P_a) = \{P_c, P_d, P_e\}$$
$$SendSet(P_b) = \{P_c, P_d, P_f\}$$



$$M_{ag} = \{(P_c, m_{ac}), (P_d, m_{ad}), (P_e, m_{ae})\}$$
$$M_{bh} = \{(P_c, m_{bc}), (P_d, m_{bd}), (P_f, m_{bf})\}$$

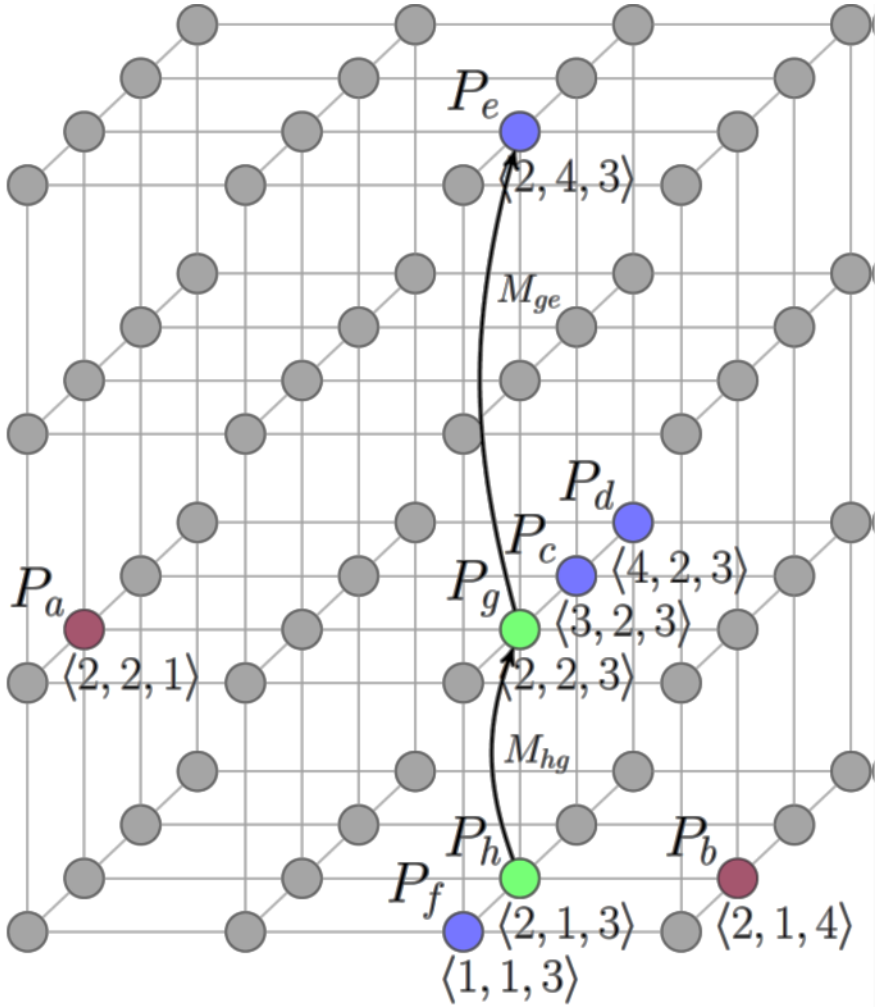
Multiple messages on VPT - Example

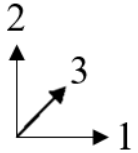




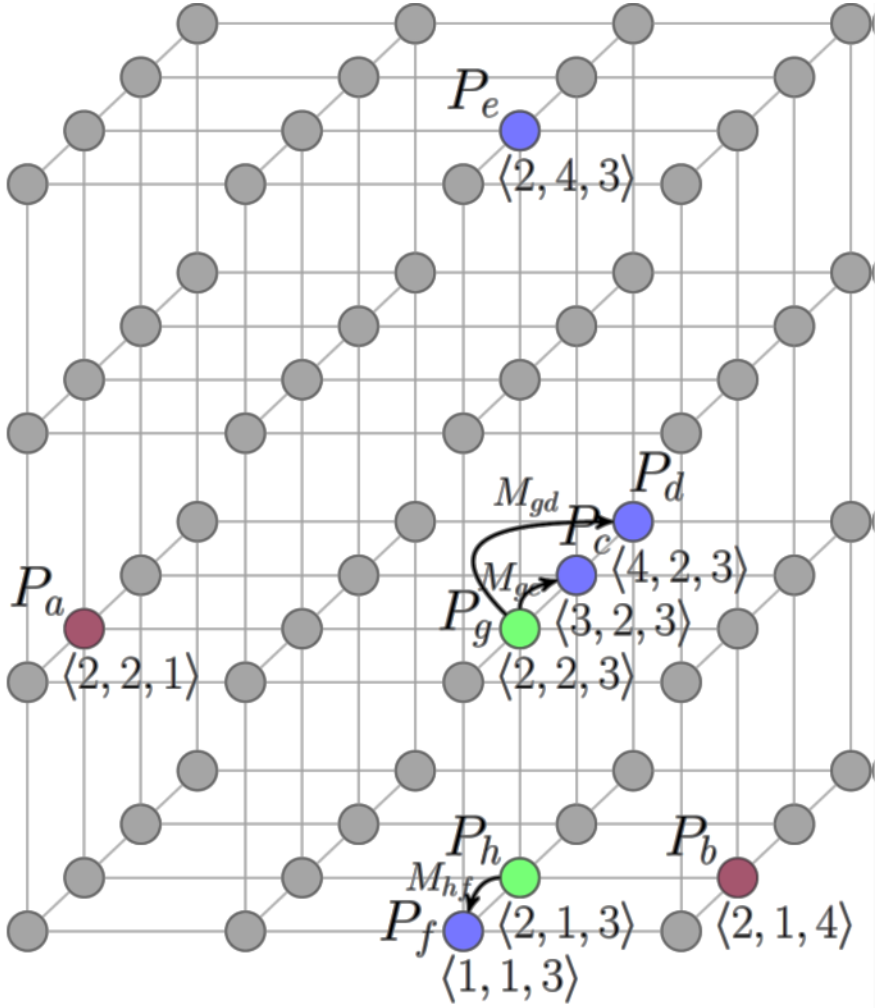
$$M_{ge} = \{(P_e, m_{ae})\}$$
$$M_{hg} = \{(P_c, m_{bc}), (P_d, m_{bd})\}$$

Multiple messages on VPT - Example





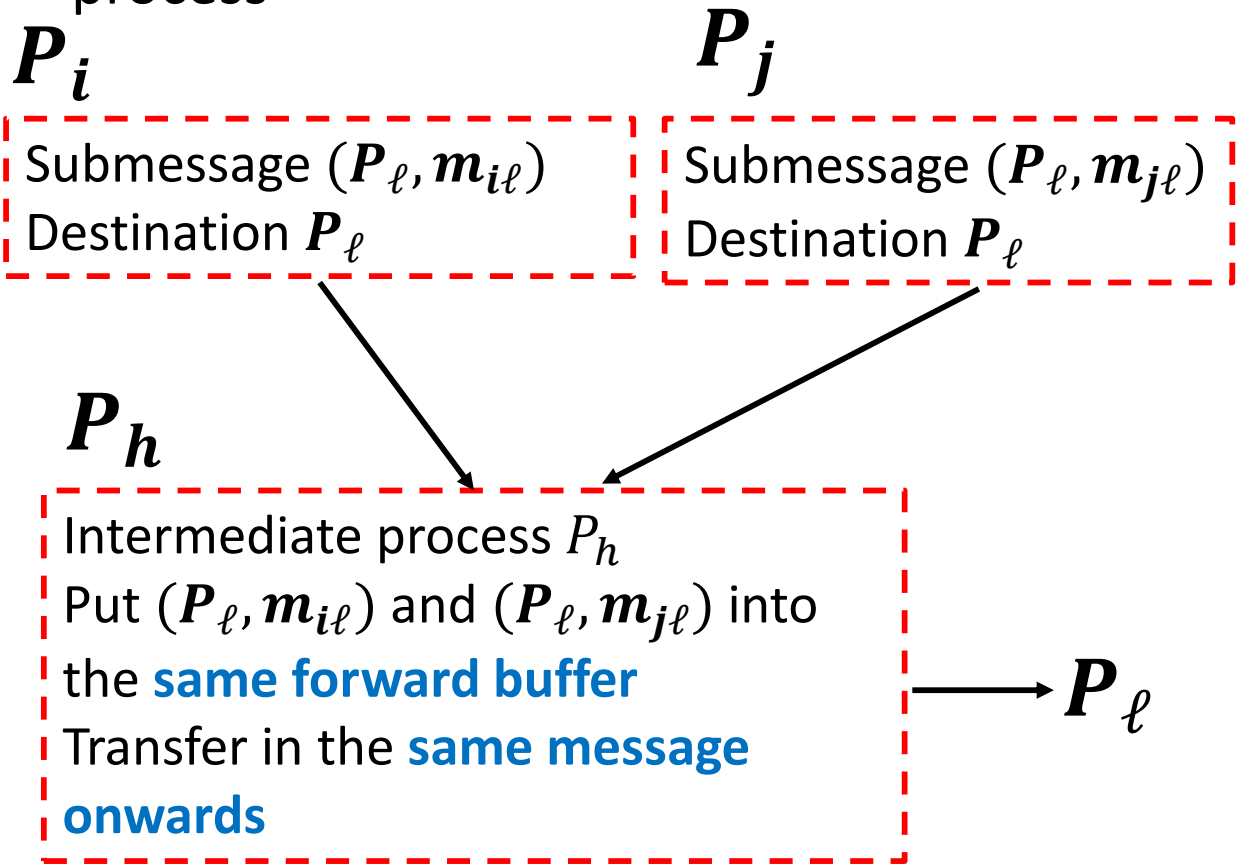
$$M_{ge} = \{(P_e, m_{ae})\}$$
$$M_{hg} = \{(P_c, m_{bc}), (P_d, m_{bd})\}$$



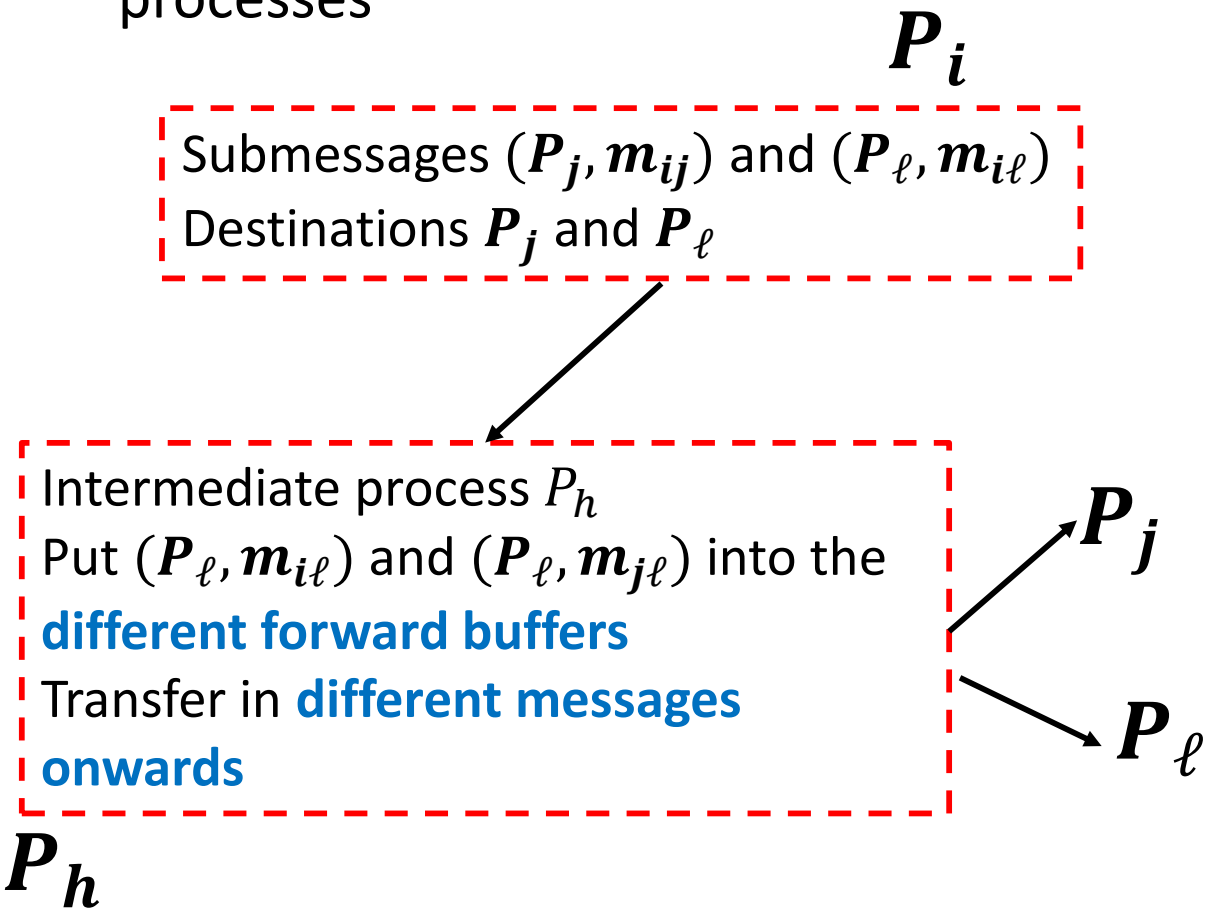
$$M_{gc} = \{(P_c, m_{ac}), (P_c, m_{bc})\}$$
$$M_{gd} = \{(P_d, m_{ad}), (P_d, m_{bd})\}$$
$$M_{hf} = \{(P_f, m_{bf})\}$$

Algorithm: Observations

- **Scenario 1**: two submessages originate from different processes but destined for the same process

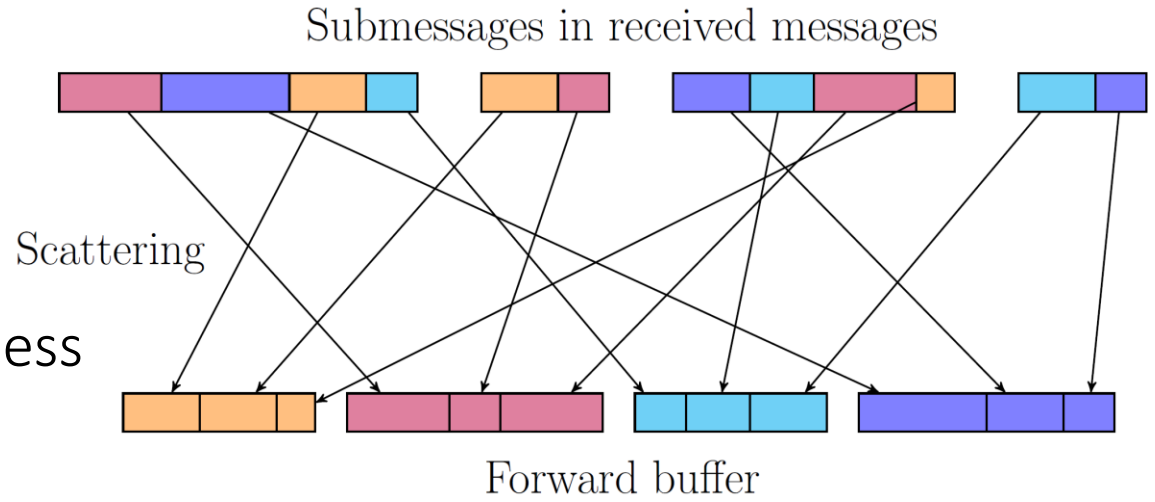


- **Scenario 2**: two submessages originate from the same process but destined for different processes



Algorithm: Observations

- Processing of submessages in received messages
  - Scatter submessages across multiple forward buffers
- A buffer that will be used in communication stage *d* may be filled with submessages received at any stage < *d*



- Extreme cases
  - T_1 → Each process may talk to every other process
    - Single stage communication
  - $T_{\log_2 K}$ → Each process may talk to a single proces at each stage
    - $\log_2 K$ stages of communication (hypercube topology)
    - K = number of processors

Forming VPT

- Given # of processes *K*
  - How to organize them?
  - How to choose VPT parameters?

- **Two important aspects**
  - The dimension of the VPT (=*n*)
  - The organization of the processes
    - *K=64, n=2* → 32x2, 16x4, 8x8

- $K = k_1 \times k_2 \times \ldots \times k_n$
- **Motivation**: $k_1, k_2, \ldots, k_n$ should be close to each other to keep max mssg count small

▷ **VPT formation scheme**

- First $\log_2 K \mod n$ dimensions → size $2^{\lfloor \log_2 K/n \rfloor + 1}$
- The remaining $n - (\log_2 K \mod n)$ → size $2^{\lfloor \log_2 K/n \rfloor}$
- Ensures no two dimension sizes differ by more than a factor of 2
- Lowest upper bound on max message count bound

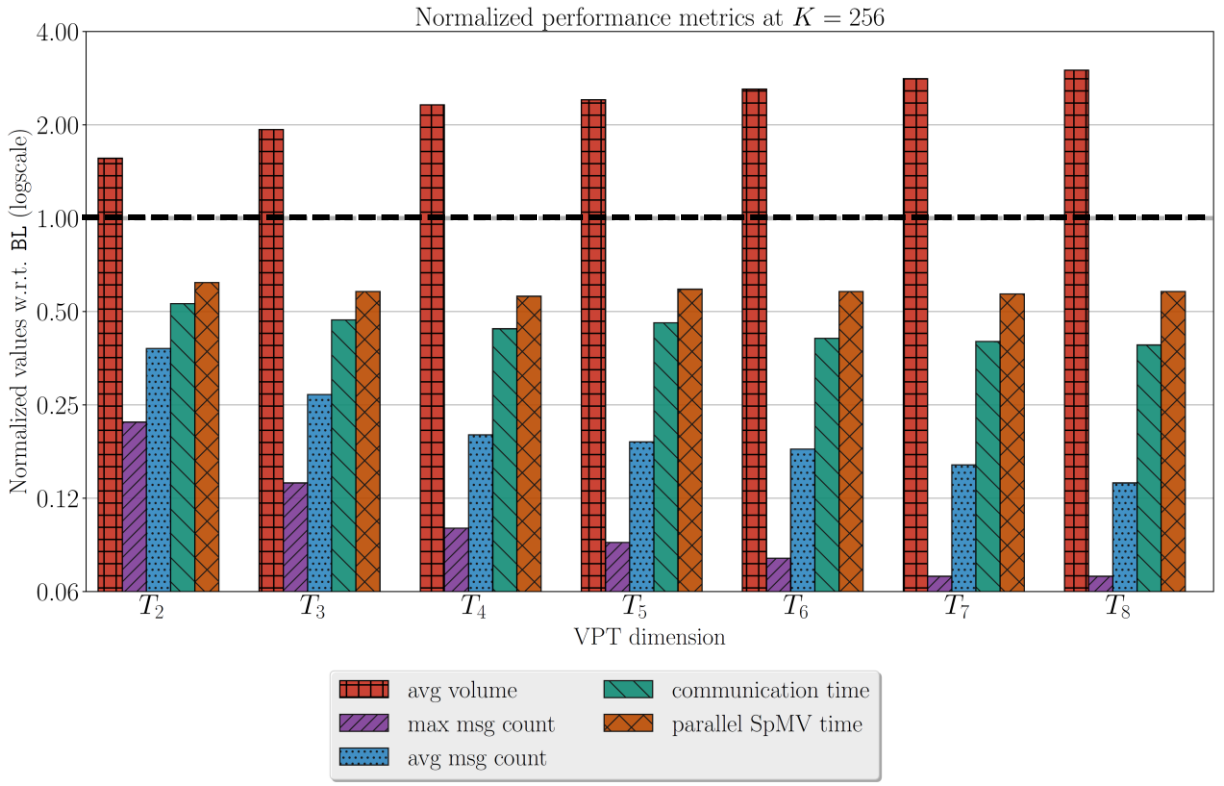
▷ Caveat: May not be always desirable!

1D row parallel SpMV Experiments: Setup

- **Schemes**
  - **BL** → Baseline (=STFW1)
  - **STFW** → **STFW2**, **STFW3**, etc. (STFWn: VPT with dimension n)
- **Systems**
  - BlueGene/Q, Cray XC40, Cray XK7
- **Processes**
  - 32 to 512 (small scale)
  - 4K to 16K (large scale)
- **Dataset**
  - 22 latency-bound instances
  - Sparsity pattern: dense rows/cols and high coefficient of variation on row/col degrees

Evaluation: Performance metrics

			Time (usec)			
Scheme	mmax	mavg	vavg	comm	SpMV	buffer size (KB)
BL	120.5	50.2	1181	825	1091	20.1
STFW2	26.5	18.8	1844	439	681	38.8
STFW3	16.5	13.4	2279	386	631	38.3
STFW4	11.9	10.1	2736	359	608	37.9
STFW5	11	9.5	2848	383	649	36.1
STFW6	10	8.8	3082	334	632	33.6
STFW7	9	7.9	3336	329	622	34.5
STFW8	8	7.2	3544	322	636	32.3



Volume metric

vavg = average volume **1.6x-3.3x** more volume than BL

Latency metrics

mmax = max msg count **4.6x-15.1x** improvement

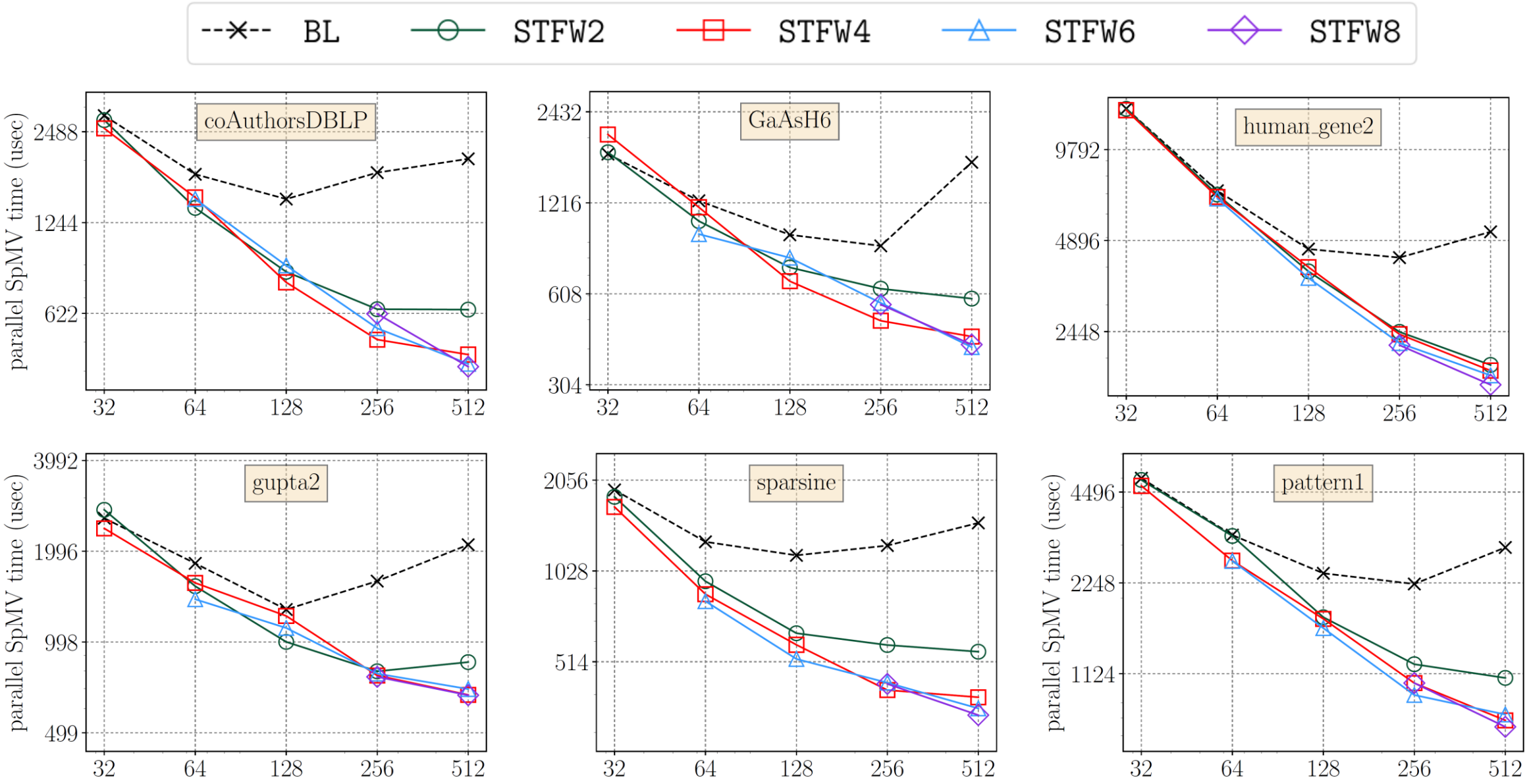
mavg = avg msg count **3x-8x** improvement

Up to **61%** improvement in **communication time**

Up to **44%** improvement in **overall SpMV time**

The **buffer usage < 2x** the buffer usage of BL!

Effect on Scalability



Communication Performance (large scale)

Torus network – 16K processes

Scheme	mmax	mavg	vavg	Comm time
BL	1054.9	137.6	425	8220
STFW2	160.6	61.0	683	1109
STFW3	65.1	34.4	887	498
STFW4	41.8	24.3	1064	391
STFW8	20	13.9	1568	510
STFW9	19	13.9	1601	491
STFW13	15	11.4	1917	694
STFW14	14	10.5	2017	696

Dragonfly network – 4K processes

Scheme	mmax	mavg	vavg	Comm time
BL	486.6	105.5	819	1419
STFW2	86.1	43.7	1271	294
STFW3	39.0	25.0	1682	221
STFW4	26.1	18.1	2024	238
STFW7	17.0	13.0	2594	199
STFW8	16.0	12.8	2663	270
STFW11	13.0	10.9	3098	289
STFW12	12.0	9.8	3312	387

mmax max message count
mavg average message count
vavg average volume (words)

95% improvement in communication time

86% improvement in communication time

2D Cartesian vs. STFW2 for SpMV

- 2D row-column-parallel SpMV utilizing 2D **Cartesian Partitioning**
  - **Pre-comm**: expand x-vector entries along *cols* of 2D **VPT**
  - Local SpMV computations
  - **Post-comm:** reduce on partial y-vector results along rows of 2D **VPT**
- 1D row-parallel SpMV using **STFW2**
  - **Pre-comm (STFW2)**
    - Send x-vector entries along columns of 2D **VPT**
    - Send and forward x-vector entries along rows of 2D **VPT**
  - Local SpMV computations

COMPARISON

- **2D Cartesian**
  - For large number of processors it suffers from large number of constraints
  - may limit the solution space
- **STFW2**
  - 1D partitioning
  - Two stages of communication
  - Latency cost can be further reduced by using 3D and higher dimensional VPTs

Conclusions

- Need better partitioning models for parallelizing irregular applications
  - More accurate encapsulation of communication costs on HPC systems
  - Communication is the bottleneck
    - Latency “lags” behind the bandwidth
    - Models should focus more on latency aspects
  - Communication costs are even more complex!
    - Network topology, node hierarchy, routing protocol, contention, etc.

- Directions taken in this talk
  - More capable partitioning models and/or frameworks
    - Enhance existing models / New partitioning models
  - Novel communication algorithms to offer trade-offs

- Challenges
  - Which method suits the given application?
  - Parallel and scalable partitioners
  - How to enhance models to take into account more complex cost functions?

Conclusions

Method	Bandwidth		Latency		Input/Output Conformability	# of Phases
	total	max	total	max		
Communication Hypergraph	✓1	✓	✓		✗2	two
Cartesian Partitioning	✓		✓3	✓3	✓	multi-stage
Message-net augmentation within RB HP	✓		✓		✓	one
STFW	✓1		✓3	✓3	✓	two

1: may increase in the second phase
2: 1D non-conformal, 2D fine-grain conformal
3: provides upper bound

Collaborators





Ümit V. Çatalyürek, Bora Uçar, B. Barla Cambazoğlu Enver Kayaaslan, Kadir Akbudak





R. Oğuz Selvitopi, Seher Acer, Gündüz V. Demirci, M. Ozan Karsavuran, Tuğba Torun