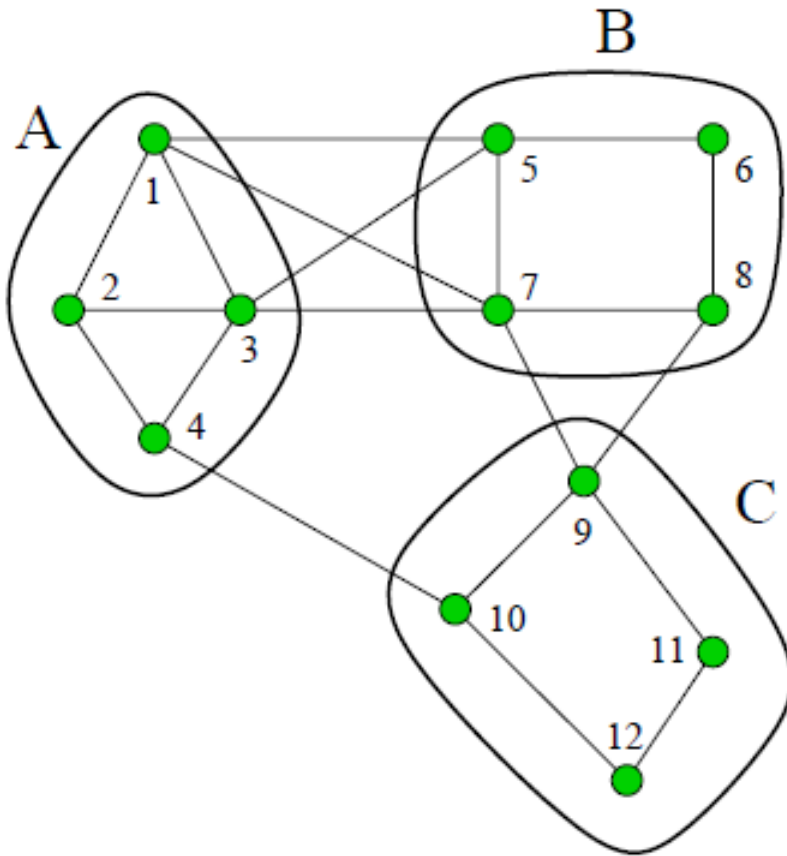


Graph Partitioning

Lecture 21

April 8, 2024

Graph partitioning



Three subdomains A, B, C

Edge cut = 7

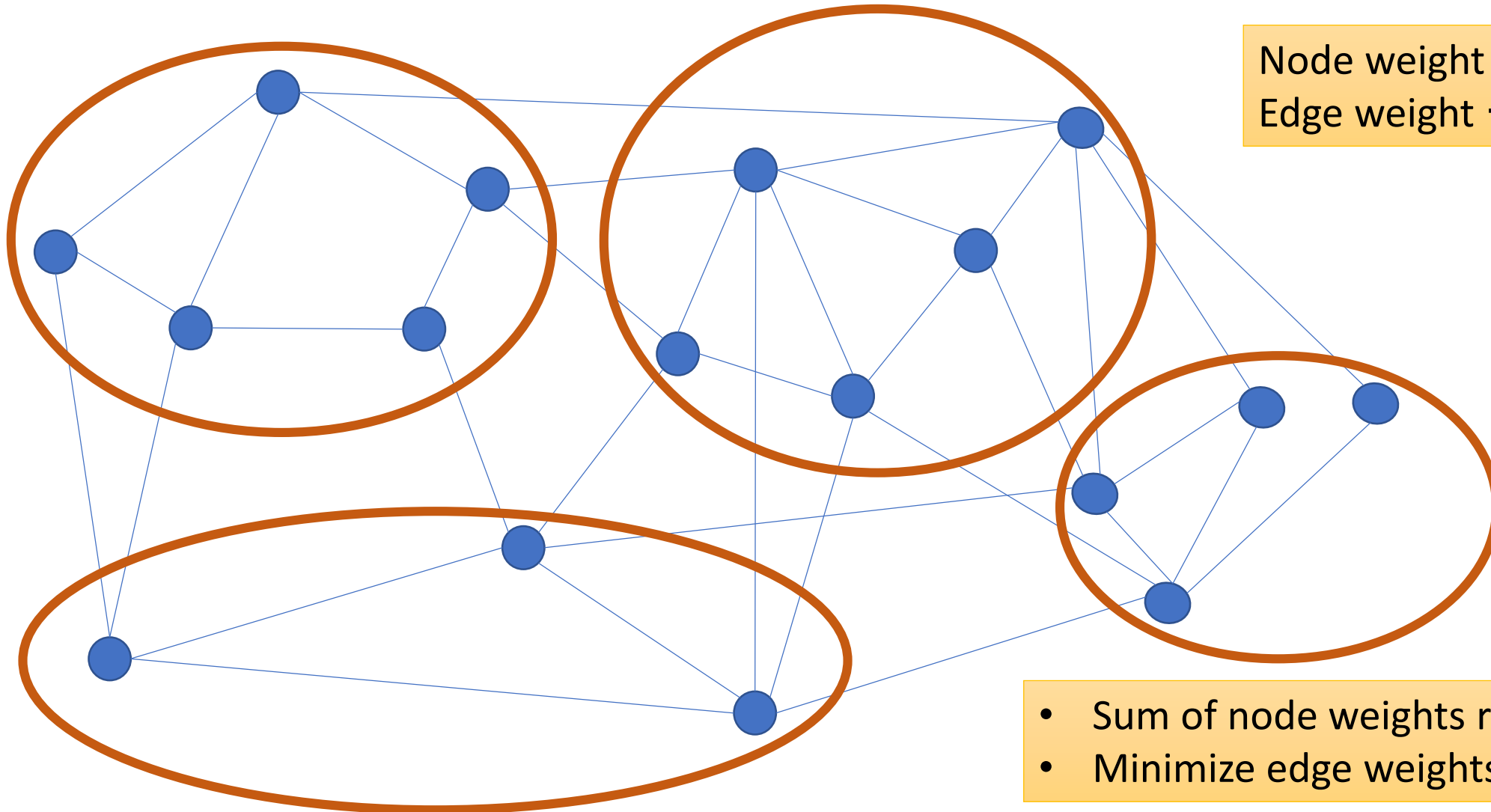
Communication cost \propto edge-cut

Communication volume?

“Nevertheless, there is a strong correlation between edge-cuts and communication costs” []*

*Hypergraph-Partitioning-Based Decomposition
for Parallel Sparse-Matrix Vector Multiplication

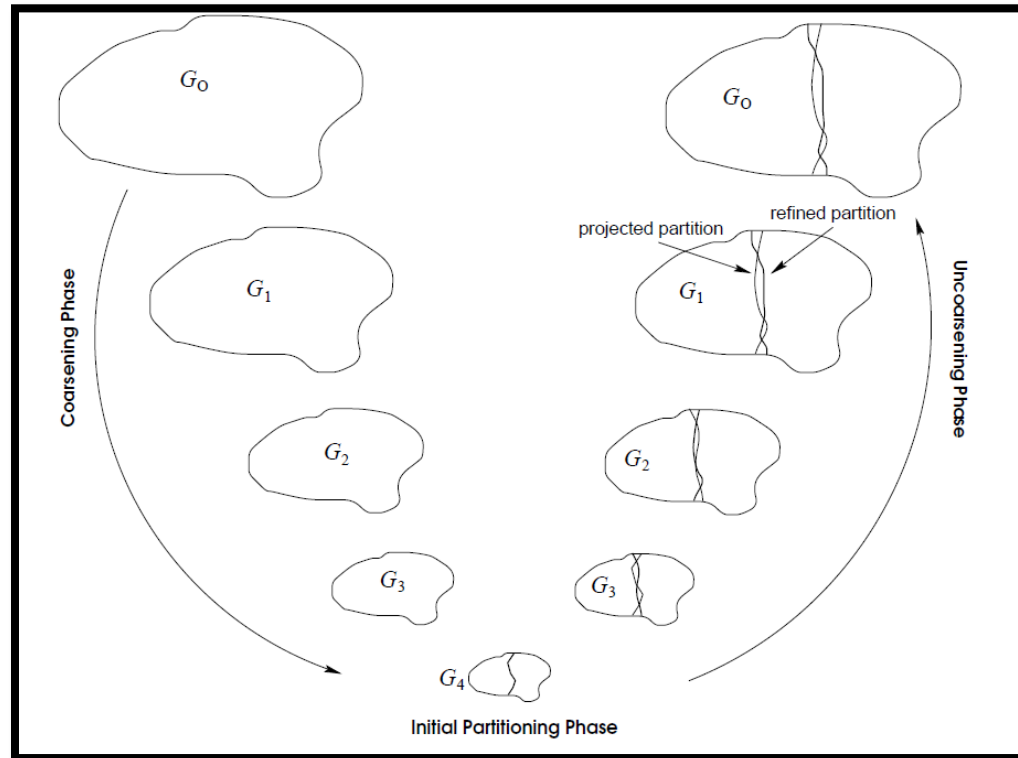
k-way partitioning



Node weight \rightarrow Computation
Edge weight \rightarrow Communication

- Sum of node weights roughly same
- Minimize edge weights across partitions

Multilevel Partitioning for Bisections



Coarsen

Uncoarsen



G_0 is coarsened to a few hundred vertices

A set of vertices of G_i is combined to form a single vertex of G_{i+1}

G_{i+1} is constructed from G_i by finding a matching of G_i

Maximal matching will ensure large number of edges

RM

HEM

KL

High-quality bisection (2-way partition) of smaller graph is computed

KL refinement

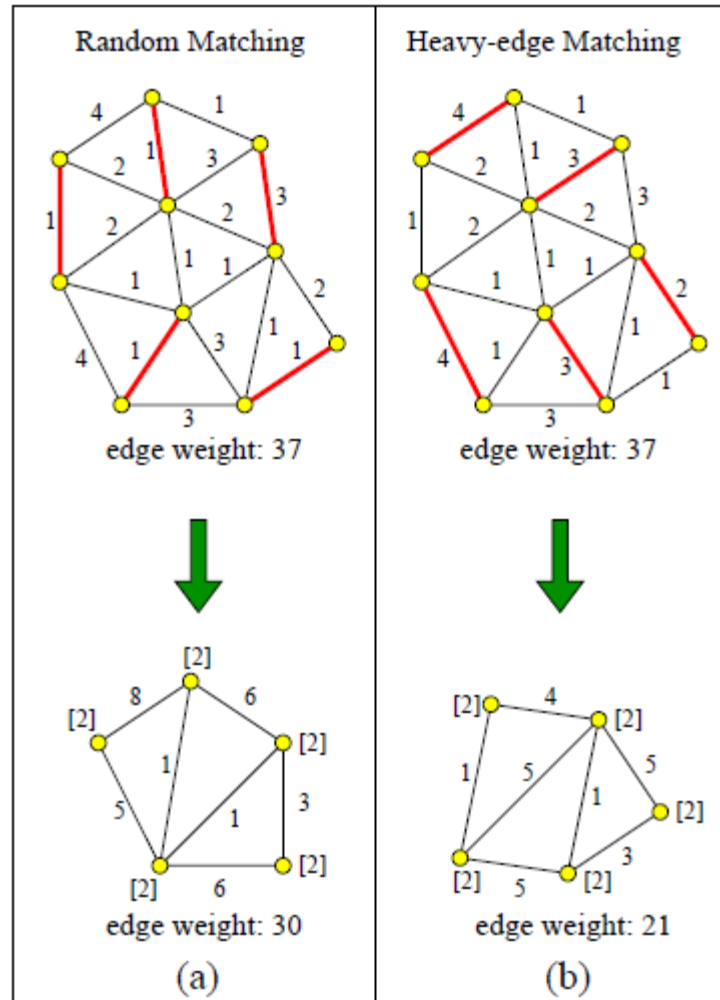
Smaller graph is uncoarsened – may lead to better edge cuts

REFERENCES:

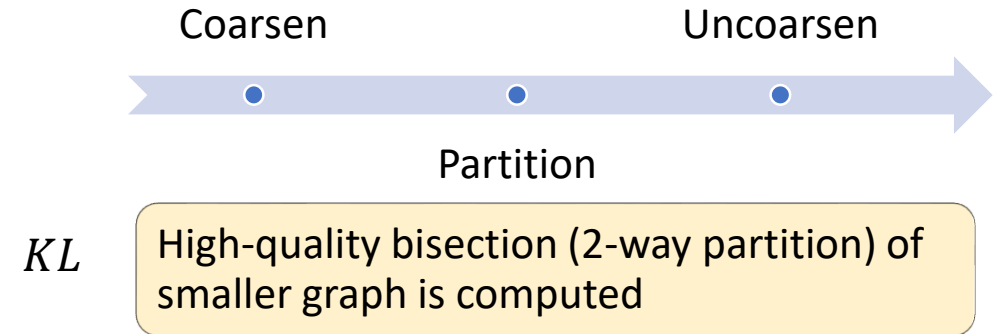
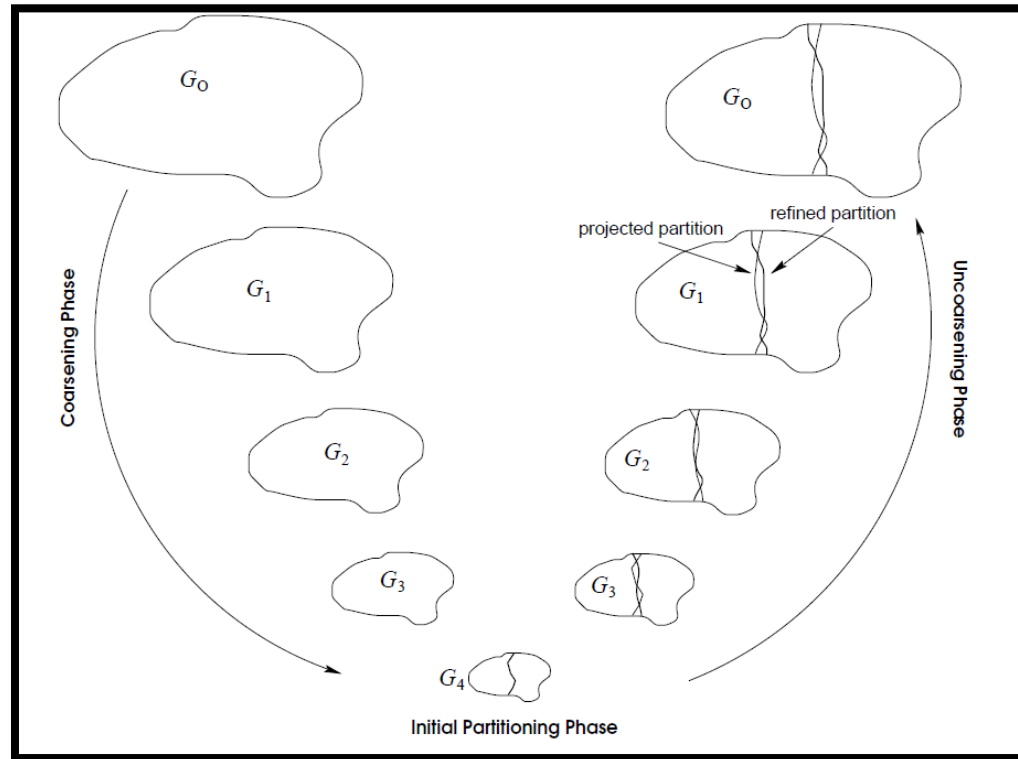
KARYPIS ET AL. A FAST AND HIGH QUALITY MULTILEVEL SCHEME FOR PARTITIONING IRREGULAR GRAPHS
SCHLOEGEL ET AL., GRAPH PARTITIONING FOR HIGH PERFORMANCE SCIENTIFIC SIMULATIONS

Random and Heavy-edge Matching

1. Visit vertex in any random order
2. If a vertex u is not yet matched, select and unmatched adjacent vertex v
3. Include edge (u, v) in the matching



Multilevel Partitioning

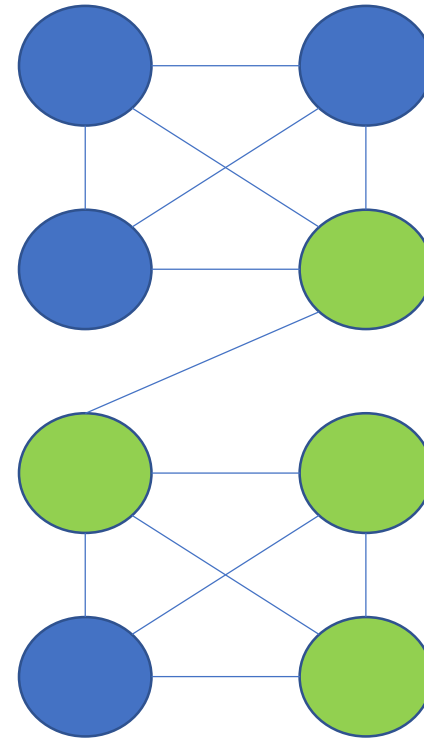
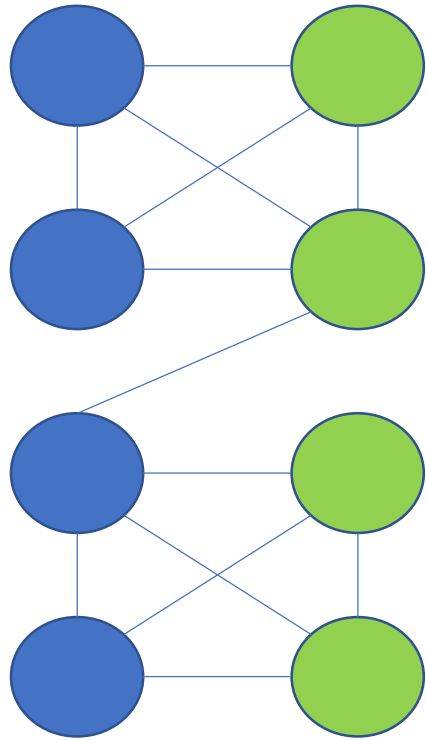


- Start with an initial arbitrary bipartition
- Repeat
 - Swap for a pair of vertices to maximize gain
- Repeat with different random initial partitions and select the best

REFERENCES:

KARYPIS ET AL. A FAST AND HIGH QUALITY MULTILEVEL SCHEME FOR PARTITIONING IRREGULAR GRAPHS
SCHLOEGEL ET AL., GRAPH PARTITIONING FOR HIGH PERFORMANCE SCIENTIFIC SIMULATIONS

KL Algorithm - Example



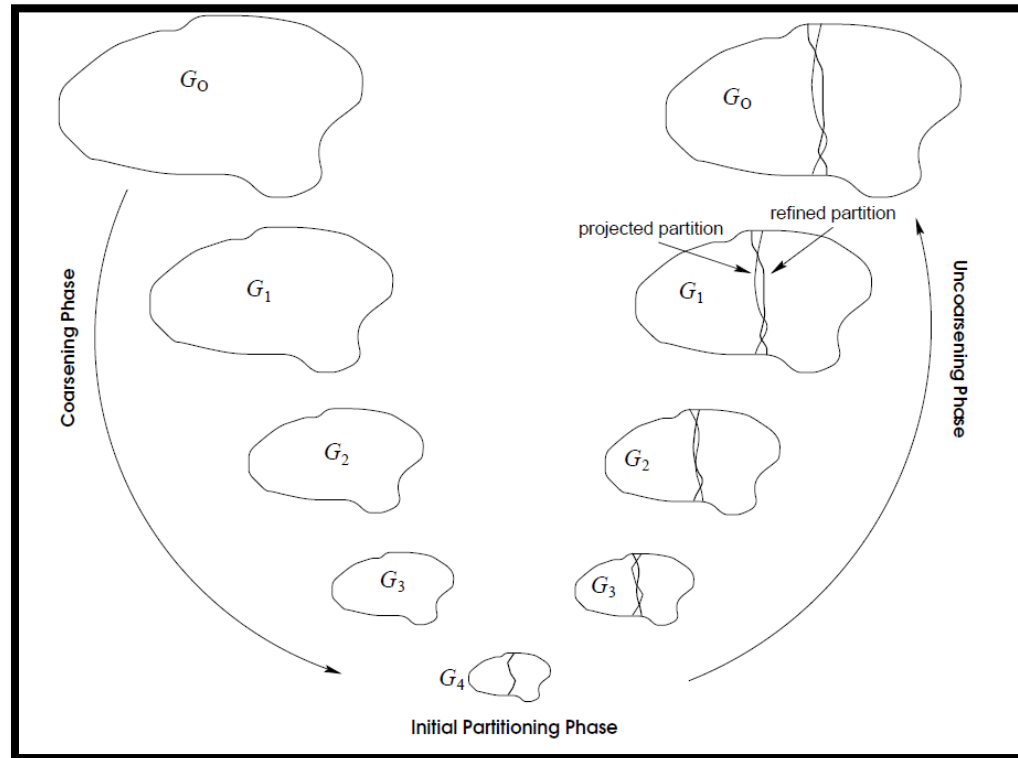
KL Algorithm

Given a partition (A,B) of the cells, the main idea of the algorithm is to move a cell at a time from one block of the partition to the other in an attempt to minimize the cutset of the final parti-

B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," Bell System Technical Journal, Vol. 49, Feb. 1970, pp. 291-307

A Linear-time Heuristic for Improving Network Partitions, Fiduccia and Mattheyses, DAC, 1982

Multilevel Partitioning for Bisections



KL refinement

Smaller graph is uncoarsened – may lead to better edge cuts

- Partitions are projected back to the original graph
- Local refinement may improve the projected partition
 - Swap for a subset of vertices if edge-cut is reduced

REFERENCES:

KARYPIS ET AL. A FAST AND HIGH QUALITY MULTILEVEL SCHEME FOR PARTITIONING IRREGULAR GRAPHS
SCHLOEGEL ET AL., GRAPH PARTITIONING FOR HIGH PERFORMANCE SCIENTIFIC SIMULATIONS

K-way partition

- Coarsen the graph using maximal matching
- Recursive bisection of coarse graph
- Subdivide each part using 2-way partitions
- KL refinement algorithm
 - Iterative algorithm (5 – 10 on average, in practice)
 - Move subsets of vertices if it leads to decrease in edge cuts
- $\log k$ phases
- Extensively due to its simplicity

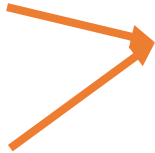
Some Results for 256-way Partitioning

Matrix	MSB	MSB-KL	Chaco-ML	Our multilevel
144	607.27	650.76	95.59	48.14
4ELT	24.95	26.56	7.01	3.13
598A	420.12	450.93	67.27	35.05
ADD32	18.72	21.88	4.23	1.63
AUTO	2214.24	2361.03	322.31	179.15
BCSSTK30	426.45	430.43	51.41	22.08
BCSSTK31	309.06	268.09	39.68	15.21
BCSSTK32	474.64	540.60	53.10	22.50
BBMAT	474.23	504.68	55.51	25.51
BRACK2	218.36	222.92	31.61	16.52
CANT	978.48	1167.87	108.38	47.70
COPTER2	185.39	194.71	31.92	16.11
CYLINDER93	671.33	697.85	91.41	39.10
FINAN512	311.01	340.01	31.00	17.98
FLAP	279.67	331.37	35.96	16.50
INPRO1	341.88	352.11	56.05	24.60
KEN-11	121.94	137.73	13.69	4.09

Performance on 200 MHz MIPS R4400

	RM			HEM			KL(1)	
	32EC	CTime	UTime	32EC	CTime	UTime	32EC	RTime
BCSSTK31	44810	5.93	2.46	45991	6.25	1.95	45267	1.05
BCSSTK32	71416	9.21	2.91	69361	10.06	2.34	66336	1.39
BRACK2	20693	6.06	3.41	21152	6.54	3.33	22451	2.04
CANT	323.0K	19.70	8.99	323.0K	20.77	5.74	323.4K	3.30
COPTER2	32330	5.77	2.95	30938	6.15	2.68	31338	2.24
CYLINDER93	198.0K	16.49	5.25	198.0K	18.65	3.22	201.0K	1.95
4ELT	1826	0.77	0.76	1894	0.80	0.78	1834	0.44
INPRO1	78375	9.50	2.90	75203	10.39	2.30	75676	1.28
ROTOR	38723	11.94	5.60	36512	12.11	4.90	38214	4.98
SHELL93	84523	36.18	10.24	81756	37.59	8.94	91723	9.27
TROLL	317.4K	62.22	14.16	307.0K	64.84	10.38	317.5K	9.55
WAVE	73364	18.51	8.24	72034	19.47	7.24	74486	8.72

Parallel Graph Partitioning

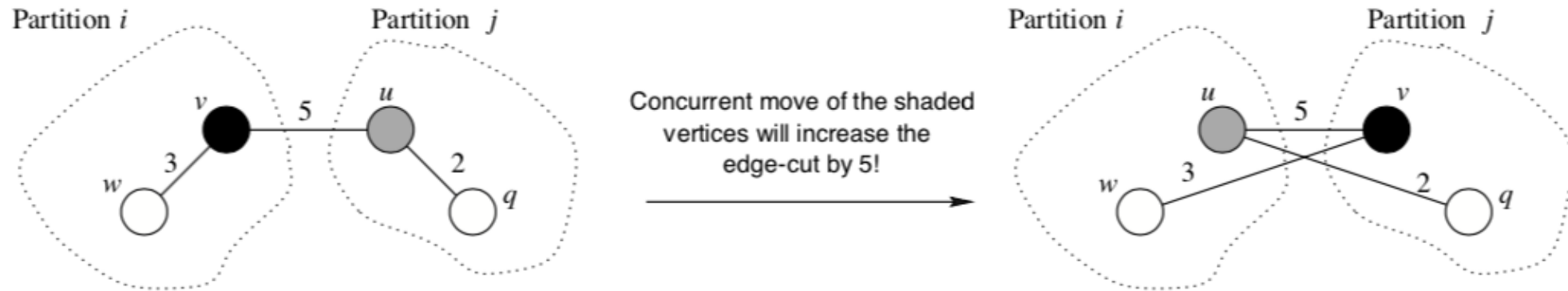
- Coarsening
 - Partitioning
 - Uncoarsening
- 
- 95% of computation
- The diagram shows three bullet points: 'Coarsening', 'Partitioning', and 'Uncoarsening'. From the right side of 'Coarsening' and 'Partitioning', two orange arrows originate. These arrows converge towards the text '95% of computation' on the right. A third arrow also originates from the right side of 'Uncoarsening' and points towards the same area, though it is less prominent than the others.

Parallel Graph Partitioning – Challenges

Coarsening phase

- Divide N vertices among P processes
- Each process computes matchings between vertices stored locally
- Works well if
 - High average degree
 - Each process stores well connected subgraphs
 - Not practical!
- Matching across different processes (better matchings)
 - Inconsistencies have to be resolved (lot of communications)

Refinement and Uncoarsening – Challenges



- Concurrently both sets of vertices moved by P_i and P_j – may not lead to decrease in edge-cut
- Communication required

Topology-aware task mapping for reducing communication contention on large parallel machines, Tarun Agarwal, Amit Sharma, Laxmikant V. Kale, IPDPS 2006

A process mapping strategy that minimizes the impact of topology by heuristically minimizing the total number of hop-bytes communicated

Process Mapping

- Topology graph
- Task graph
 - $c_{ab} \rightarrow$ amount of communication (bytes) on e_{ab} between v_a and v_b
- Map: $v_t \in V_t$ placed on $v_p \in V_p$

$$G_p = (V_p, E_p)$$

$$G_t = (V_t, E_t)$$

$$P : V_t \longrightarrow V_p$$

Metric 1: Hop-bytes

Total size of inter-processor communication in bytes weighted by distance between the respective end-processors.

The overall hop-bytes is the sum of hop-bytes due to individual nodes in the task graph.

$$HB(G_t, G_p, P) = \frac{1}{2} \sum_{v_a \in V_t} HB(v_a)$$

$$\text{where } HB(v_a) = \sum_{e_{ab} \in E_t} HB(e_{ab})$$

$$\text{where } HB(e_{ab}) = c_{ab} \times d_p(P(v_a), P(v_b))$$

Metric 2: Hops per byte

Average number of network links a byte has to travel under a task mapping

$$\text{Hops per Byte} = \frac{HB(G_t, G_p, P)}{\sum_{e_{ab} \in E_t} c_{ab}}$$

$$\text{Hops per Byte} = \frac{\sum_{e_{ab} \in E_t} c_{ab} \times d_p(P(v_a), P(v_b))}{\sum_{e_{ab} \in E_t} c_{ab}}$$

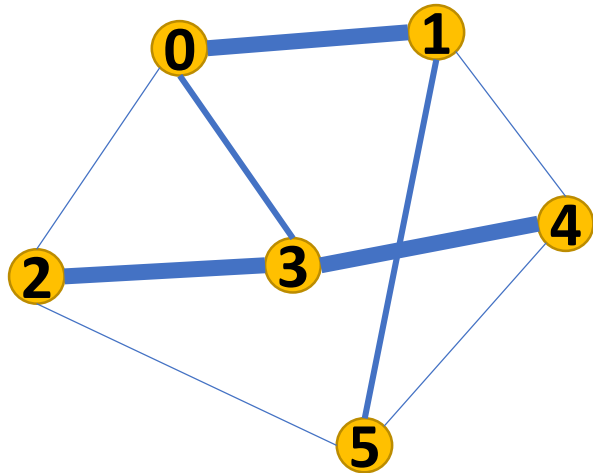
Problem

Assume we have n compute objects and p processors.

The problem of balancing compute load involves partitioning the n compute objects into p groups such that the total compute load of objects in each group is roughly the same.

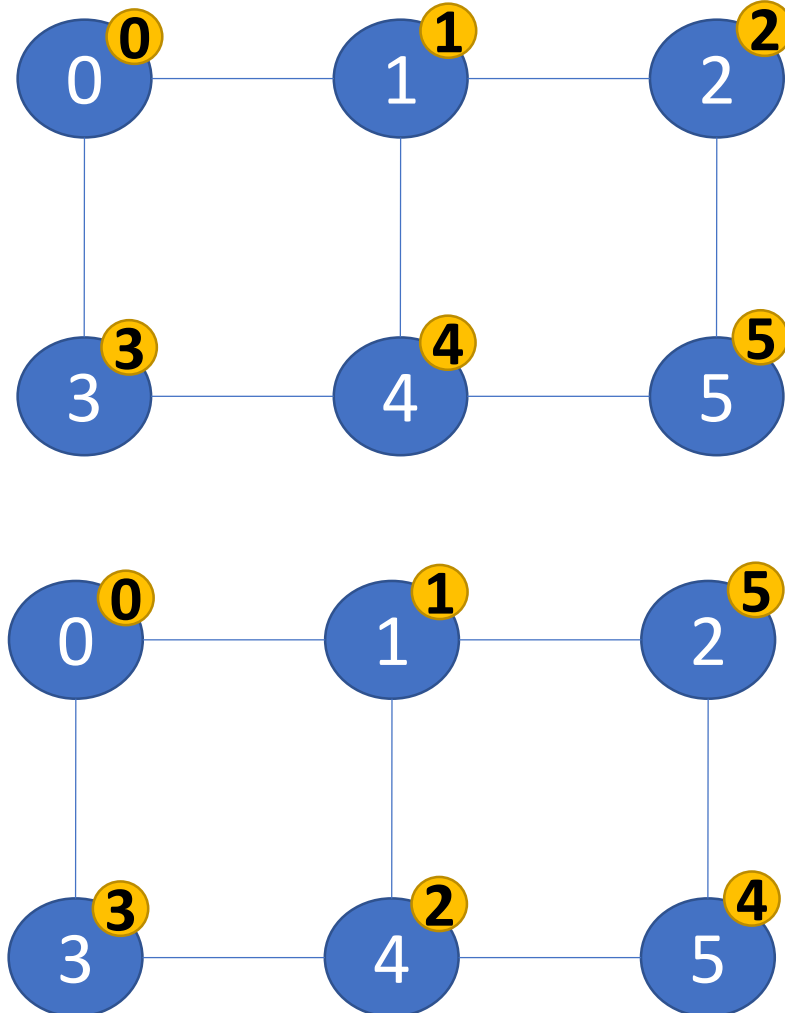
The second problem, that of reducing network contention, involves placing these groups onto the p processors such that more heavily communicating groups are placed on nearby processors. This would make each message travel over a smaller number of links leading to a reduction in the average data transferred across individual links.

Communication Graph Mapping – Recap



	512	64	256		
512				64	256
64			512		64
256		512		512	
	64		512		64
	256	64		64	

Linear mapping



Linear

01: 512*1
 02: 64*2
 03: 256*1
 14: 64*1
 15: 256*2
 23: 512*3
 25: 64*1
 34: 512*1
 45: 64*1

= 3648

Towards optimal

01: 512*1
 02: 64*2
 03: 256*1
 14: 64*2
 15: 256*1
 23: 512*1
 25: 64*2
 34: 512*2
 45: 64*1

= 3008

Estimation Function

- $f_{est}(t, p, M)$ = Cost of placing a task t onto processor p under current task mapping M
- Estimate how critical it is to place a task in the current cycle, select the task with maximum criticality
- T_k is the set of tasks yet to be placed, \bar{T}_k is the set of tasks already placed
- P_k is the set of processors that are available, \bar{P}_k is the set of processors already allocated

$$\begin{array}{l} T_k \cap \bar{T}_k = \emptyset \\ P_k \cap \bar{P}_k = \emptyset \end{array}$$

Estimation Function

- $f_{est}(t, p, M)$ = Cost of estimating the placement of a task t onto processor p under current task mapping M
- First order approximation w.r.t placed tasks

- Hop-bytes

$$f_{est}(t_i, p, M) = \sum_{t_j \in \bar{T}_k} c_{ij} d_p(p, M(t_j))$$

- Second order approximation w.r.t all tasks

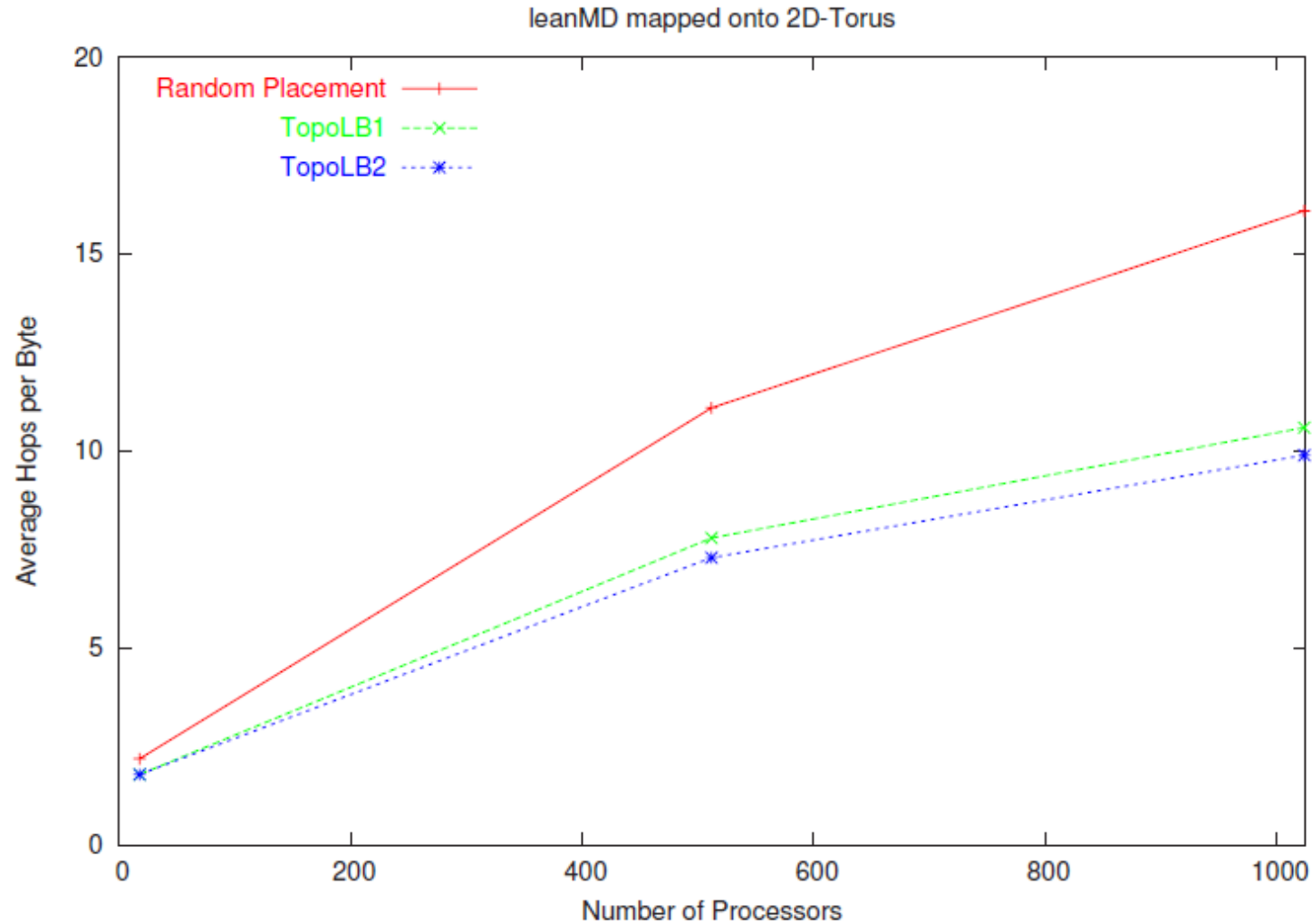
$$d_p(p, M(t_j)) \approx \frac{\sum_{p_j \in V_p} d_p(p, p_j)}{|V_p|}$$

$$f_{est}(t_i, p, M) = \sum_{t_j \in \bar{T}_k} c_{ij} d_p(p, M(t_j)) + \sum_{t_j \in T_k} c_{ij} \frac{\sum_{p_j \in V_p} d_p(p, p_j)}{|V_p|}$$

Mapping Heuristic

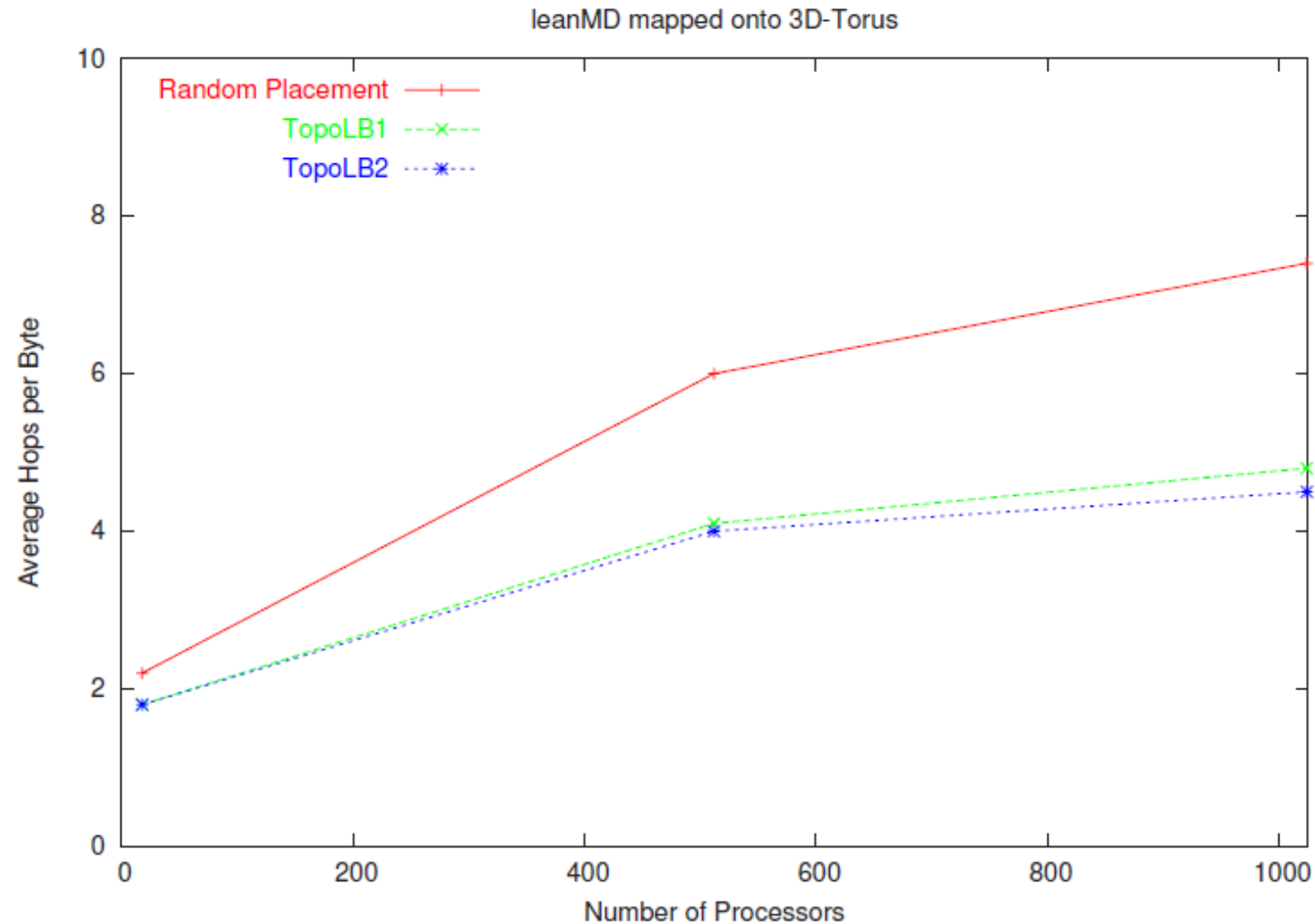
- The estimation function f (using hop-bytes) is maintained in a matrix $p \times p$.
- Rows: Task nodes, Columns: Processors
- Cell entry: current value of estimation function
- Maintain $f_{avg}[t]$ and $f_{min}[t]$.
- Select the task that maximizes $f_{avg}[t] - f_{min}[t]$ (most critical task).
- Find the processor p_i that minimizes f .
- Update T_k and P_k and the matrix (only dependant entries)

Results – LeanMD on 2D Torus



34% reduction
in hops-per-byte

Results – LeanMD on 3D Torus

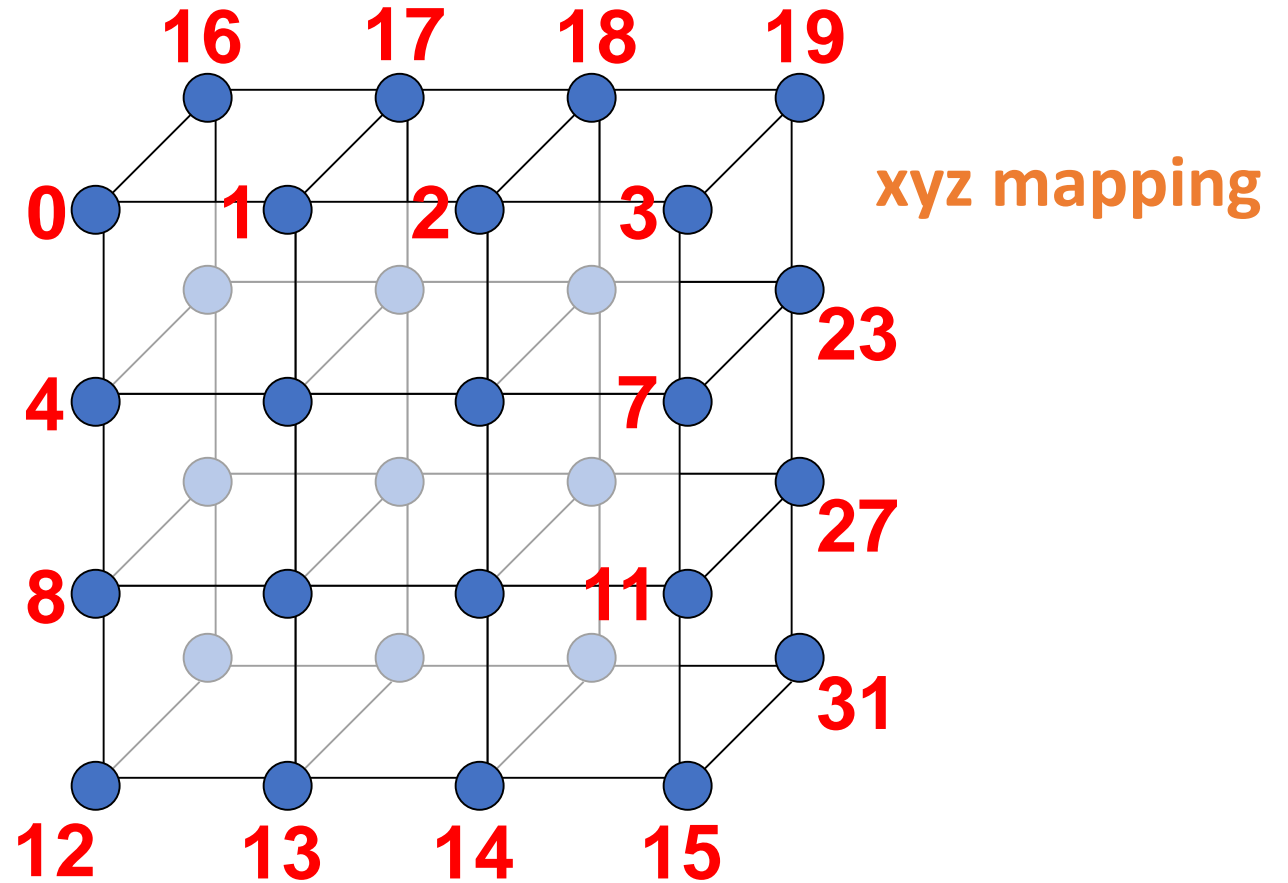


40% reduction
in hops-per-byte

Mapping for Regular Communication Pattern

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

8 x 4 2D virtual process topology

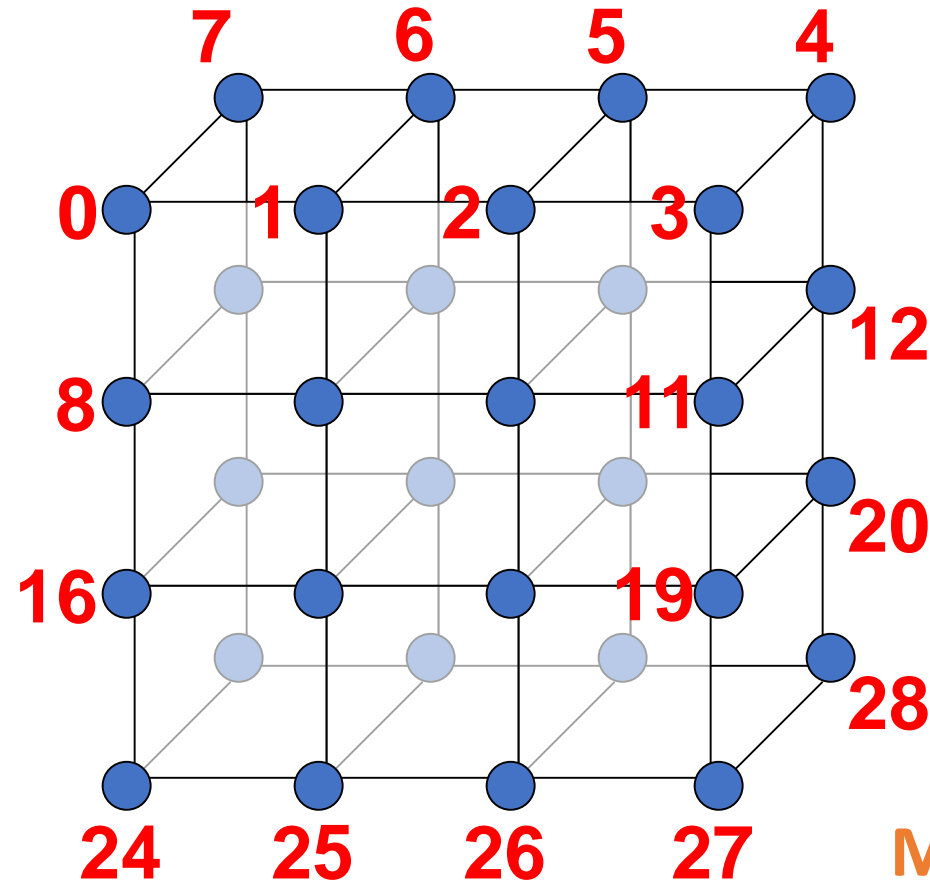


4 x 4 x 2 3D torus

Mapping for Regular Communication Pattern

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

8 x 4 2D virtual process topology



4 x 4 x 2 3D torus

Modified
xzy mapping