Multi-Criteria Partitioning of Multi-Block Structured Grids

Hengjie Wang   Aparna Chandramowlishwaran

HPC Forge
University of California, Irvine

Jun. 27, 2019

Outline

Background

Algorithms

Tests and Results

Conclusion

Outline

Background

Algorithms

Tests and Results

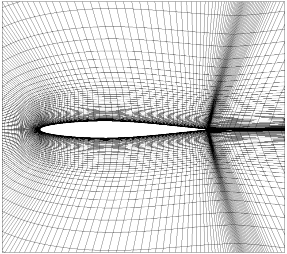
Conclusion

Structured Grid

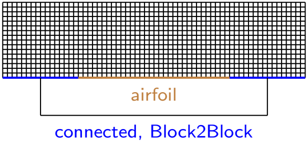
- Structured Grid: Regular connectivity between grid cells.

	i,j+1	
i-1,j	i,j	i+1,j
	i,j-1	

- Block: grid unit equivalent to a single rectangle.



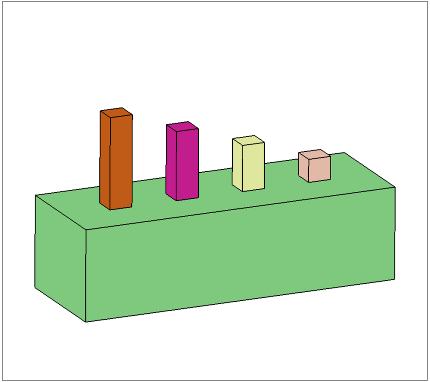
Airfoil Grid



Block

Structured Grid

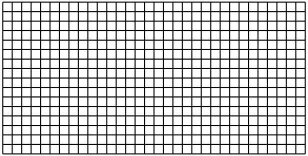
- Multi-Block Structured Grids

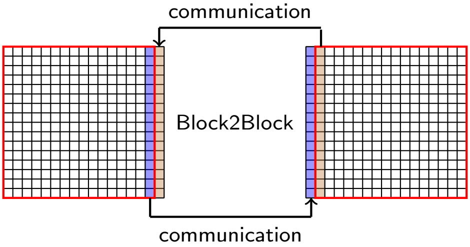


Bump3D, 5 blocks

Halo Exchange

Split a block into 2 partitions and assign each partition to a node:



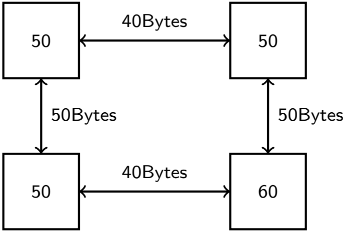


Hybird Programming Model

Hybrid programming model:

- 1 MPI process per node and spawn threads within a node.
- Assume shared memory copy takes no time.

Partition 4 blocks onto 2 nodes:



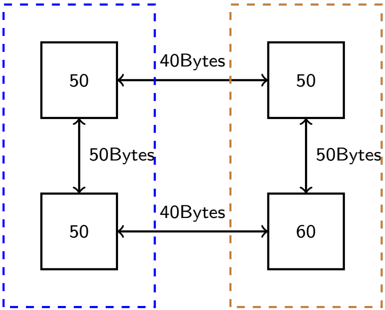
Average Workload $\overline{W}$ 105

Hybrid Programming Model

Hybrid programming model:

- 1 MPI process per node and spawn threads within a node.
- Assume shared memory copy takes no time.

Partition 4 blocks onto 2 nodes:



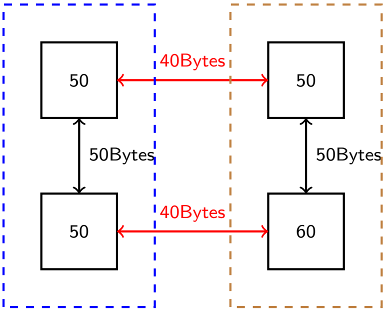
Average Workload $\overline{W}$	105
Imbalance	$5/105$

Hybird Programming Model

Hybrid programming model:

- 1 MPI process per node and spawn threads within a node.
- Assume shared memory copy takes no time.

Partition 4 blocks onto 2 nodes:



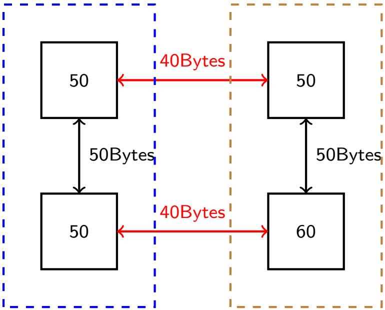
Average Workload $\overline{W}$	105
Imbalance	5/105
Edge Cuts	2
Communcation Volume	80 Bytes

Hybird Programming Model

Hybrid programming model:

- 1 MPI process per node and spawn threads within a node.
- Assume shared memory copy takes no time.

Partition 4 blocks onto 2 nodes:



Average Workload $\overline{W}$	105
Imbalance	5/105
Edge Cuts	2
Communcation Volume	80 Bytes
Shared Memory Copy	100 Btyes

Objectives

Given the number of partitions n_p, workload per partition $\overline{W}$, the partitioner should:

- Achieve load balance
- Minimize communication cost

Objectives

Given the number of partitions n_p, workload per partition $\overline{W}$, the partitioner should:

- Achieve load balance
  - Trade off load balance for communication cost
- Minimize communication cost

Objectives

Given the number of partitions n_p, workload per partition $\overline{W}$, the partitioner should:

- Achieve load balance
  - Trade off load balance for communication cost
- Minimize communication cost
  - Reduce the inter-node communication
  - Convert Block2Block communication to shared memory copy

Outline

Background

Algorithms

Tests and Results

Conclusion

State-of-the-art Methods

The state-of-the-art methods can be divided into two strategies:

- Top-down strategy:
  - Cut large blocks and assign sub-blocks to partitions.
  - Group small blocks to fill partitions.

  Examples:
  Greedy [Ytterström 97]
  Recursive Edge Bisection (REB) [Berger 87]
  Integer Factorization (IF)

- Bottom-up strategy:

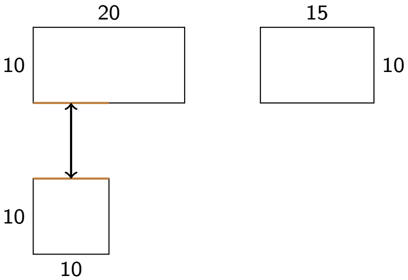
  Transform the problem to graph partitioning and use graph partitioner.

  Examples:
  Metis [Karypis 94], Scotch [Roman 96], Chaco [Leland 95]

Greedy Algorithm

Greedy Algorithm:

- Assign (part of) the largest block to the most underload partition.
- Cut at the longest edge of a block.

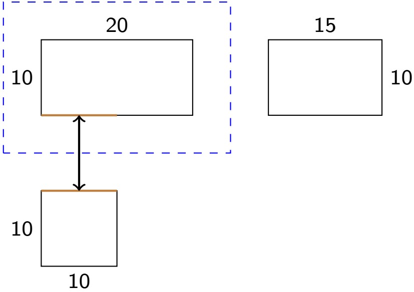


$\overline{W} = 300 \quad W_p = 0$

Greedy Algorithm

Greedy Algorithm:

- Assign (part of) the largest block to the most underload partition.
- Cut at the longest edge of a block.

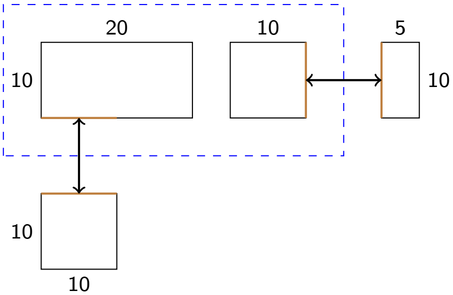


$\overline{W} = 300 \quad W_p = 200$

Greedy Algorithm

Greedy Algorithm:

- Assign (part of) the largest block to the most underload partition.
- Cut at the longest edge of a block.

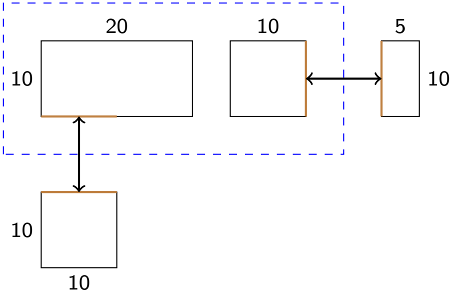


$\overline{W} = 300 \quad W_p = 300$

Greedy Algorithm

Greedy Algorithm:

- Assign (part of) the largest block to the most underload partition.
- Cut at the longest edge of a block.

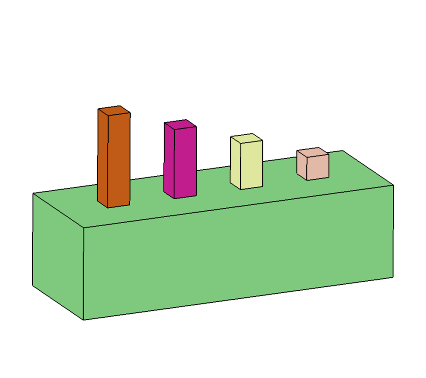


$\overline{W} = 300 \quad W_p = 300$

- ✗ Ignores the connectivity between blocks.
- ✗ Creates excessive small blocks when cutting a large block

Greedy Algorithm

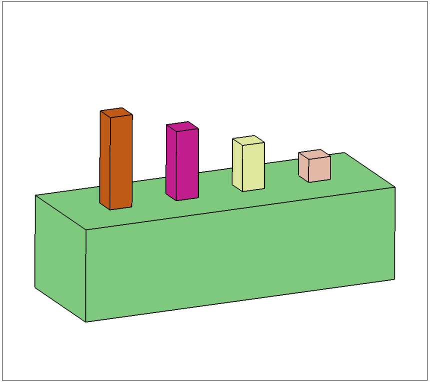
Bump3D grid: 5 blocks, the largest block is 27 times larger than the rest.



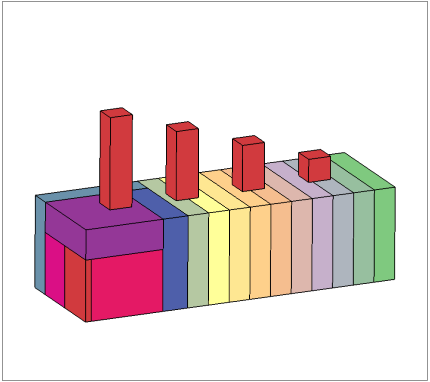
Bump3D blocks

Greedy Algorithm

Bump3D grid: 5 blocks, the largest block is 27 times larger than the rest.



Bump3D blocks



Greedy, 16 partitions

Bottom-up Strategy

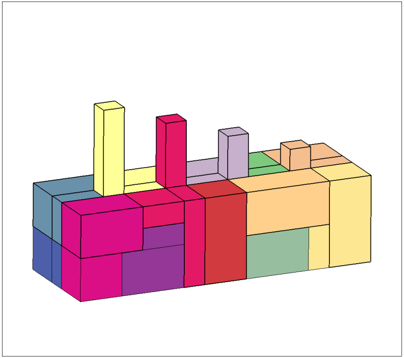
Bottom-up: Convert the structured grid partitioning to general graph partitioning.

For a graph partitioner to work well, it needs **large number of vertices per partition**.

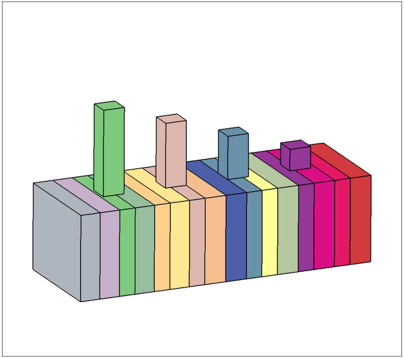
1. **Over-decompose** blocks, construct graph with blocks as vertices
2. Apply graph partitioner: Metis, Scotch, Chaco, etc
3. Merge blocks within one partition

Bottom-up Strategy

Use Metis as the graph partitioner to generate 16 partitions with different over-decomposition method.



Over-Decompose to elementary blocks



Over-Decompose with IF

Limitations of State-of-the-art Methods

Above methods share the limitations:

- Flat MPI, ignore the shared memory on the algorithm level.

Limitations of State-of-the-art Methods

Above methods share the limitations:

- Flat MPI, ignore the shared memory on the algorithm level.
- The communication performance does not distinguish the shared memory copy and inter-nodes data transfer.

Limitations of State-of-the-art Methods

Above methods share the limitations:

- Flat MPI, ignore the shared memory on the algorithm level.
- The communication performance does not distinguish the shared memory copy and inter-nodes data transfer.
- Primarily focus on reducing communication volume, ignore the effect of network's latency.

Our Partition Algorithms

Our contributions:

- Use $\alpha - \beta$ model to measure communication cost, which incorporates communication volume, edge cut, and network properties.

Our Partition Algorithms

Our contributions:

- Use $\alpha - \beta$ model to measure communication cost, which incorporates communication volume, edge cut, and network properties.
- Propose new partition algorithms following the top-down strategy.

Our Partition Algorithms

Our contributions:

- Use $\alpha - \beta$ model to measure communication cost, which incorporates communication volume, edge cut, and network properties.
- Propose new partition algorithms following the top-down strategy.
  - Modify Recursive Edge Bisection (REB) and Integer Factorization (IF) for cutting large blocks ($W > \overline{W}$).

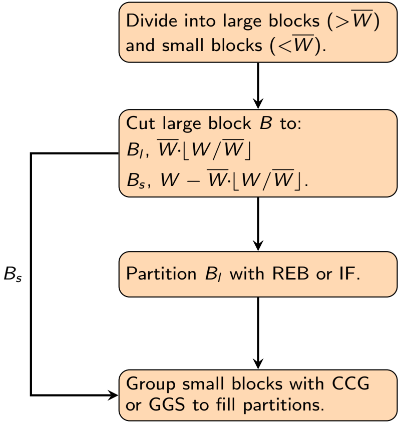
Our Partition Algorithms

Our contributions:

- Use $\alpha - \beta$ model to measure communication cost, which incorporates communication volume, edge cut, and network properties.
- Propose new partition algorithms following the top-down strategy.
  - Modify Recursive Edge Bisection (REB) and Integer Factorization (IF) for cutting large blocks ($W > \overline{W}$).
  - Propose Cut-Combine-Greedy (CCG) and Graph-Grow-Sweep (GGS) for grouping small blocks.

Our Partition Algorithms

Cut large blocks $\rightarrow$ Group small blocks:



Measure of Communication cost

$\alpha - \beta$ model: α latency (s), β bandwidth (bytes/s)

$$\text{Cost(s)} = \alpha + \frac{\text{sizeof}(message)}{\beta}$$

For Block2Block message:

$$t_{b2b} = \alpha + \frac{\#Halo \cdot FaceArea \cdot \text{sizeof}(cell)}{\beta}$$

Sum over all Block2Block messages:

$$\sum t_{b2b} = \alpha \cdot \sum \text{Edge Cuts} + \frac{\text{Communication Volume}}{\beta}$$

Cut Large Block: find a cut

Input: Block B, workload W_{cut}, tolerance ϵ, partition P (optional)

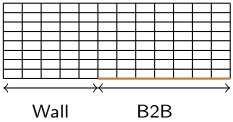
Output: a cut satisfies:

(1) the cut-off sub-block fits in $[W_{cut}(1-\epsilon),\ W_{cut}(1+\epsilon)]$

(2) introduces minimum communication cost δt

Find a cut

for i = x, y, z **do**
  Get area of i's norm face A_i
  f = floor($W_{cut}(1-\epsilon)/A_i$)
  c = ceil($W_{cut}(1+\epsilon)/A_i$)
  for pos∈[posFloor, posCeil] **do**
    $\delta t_{cut} = \sum_{b2bcut} \alpha + t_{b2b}(A_i) - \sum_{B_j \in p} t_{b2b}(\mathrm{cut}, B_j)$
    if $\delta t_{cut} < \delta t_{min}$ **then**
      $\delta t_{min} = \delta t_{cut}$
      cut.pos = pos



Cut Large Block: find a cut

Input: Block B, workload W_{cut}, tolerance ϵ, partition P (optional)

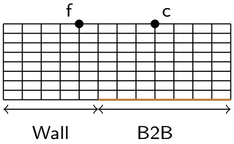
Output: a cut satisfies:

(1) the cut-off sub-block fits in $[W_{cut}(1-\epsilon), W_{cut}(1+\epsilon)]$

(2) introduces minimum communication cost δt

Find a cut

for i = x, y, z **do**
  Get area of i's norm face A_i
  $f = \text{floor}(W_{cut}(1-\epsilon)/A_i)$
  $c = \text{ceil}(W_{cut}(1+\epsilon)/A_i)$
  for pos∈[posFloor, posCeil] **do**
    $\delta t_{cut} = \sum_{b2bcut} \alpha + t_{b2b}(A_i) - \sum_{B_j \in p} t_{b2b}(\text{cut}, B_j)$
    if $\delta t_{cut} < \delta t_{min}$ **then**
      $\delta t_{min} = \delta t_{cut}$
      cut.pos = pos



Cut Large Block: find a cut

Input: Block B, workload W_{cut}, tolerance ϵ, partition P (optional)

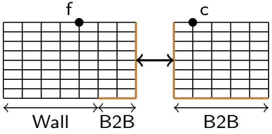
Output: a cut satisfies:

(1) the cut-off sub-block fits in $[W_{cut}(1-\epsilon), W_{cut}(1+\epsilon)]$

(2) introduces minimum communication cost δt

Find a cut

for i = x, y, z **do**
  Get area of i's norm face A_i
  f = floor$(W_{cut}(1-\epsilon)/A_i)$
  c = ceil$(W_{cut}(1+\epsilon)/A_i)$
  for pos∈[posFloor, posCeil] **do**
    $\delta t_{cut} = \sum_{b2bcut} \alpha + t_{b2b}(A_i) - \sum_{B_j \in p} t_{b2b}(\mathrm{cut}, B_j)$
    if $\delta t_{cut} < \delta t_{min}$ **then**
      $\delta t_{min} = \delta t_{cut}$
      cut.pos = pos



Cut Large Block: find a cut

Input: Block B, workload W_{cut}, tolerance ϵ, partition P (optional)

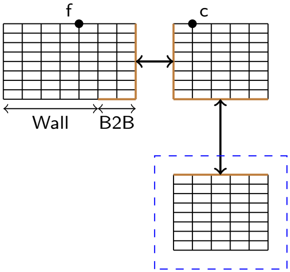
Output: a cut satisfies:

(1) the cut-off sub-block fits in $[W_{cut}(1-\epsilon), W_{cut}(1+\epsilon)]$

(2) introduces minimum communication cost δt

Find a cut

1: **for** i = x, y, z **do**
2: Get area of i's norm face A_i
3: f = floor($W_{cut}(1-\epsilon)/A_i$)
4: c = ceil($W_{cut}(1+\epsilon)/A_i$)
5: **for** pos∈[posFloor, posCeil] **do**
6: $\delta t_{cut} = \sum_{b2bcut} \alpha + t_{b2b}(A_i) - \sum_{B_j \in p} t_{b2b}(\mathrm{cut}, B_j)$
7: **if** $\delta t_{cut} < \delta t_{min}$ **then**
8: $\delta t_{min} = \delta t_{cut}$
9: cut.pos = pos



Cut Large Block: find a cut

Input: Block B, workload W_{cut}, tolerance ϵ, partition P (optional)

Output: a cut satisfies:

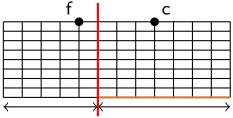
(1) the cut-off sub-block fits in $[W_{cut}(1-\epsilon), W_{cut}(1+\epsilon)]$

(2) introduces minimum communication cost δt

Find a cut

```
for i = x, y, z do
  Get area of i's norm face A_i
  f = floor(W_cut(1 − ε)/A_i)
  c = ceil(W_cut(1 + ε)/A_i)
  for pos∈[posFloor, posCeil] do
    δt_cut = Σ_{b2bcut} α + t_b2b(A_i) − Σ_{B_j∈p} t_b2b(cut, B_j)
    if δt_cut < δt_min then
      δt_min = δt_cut
      cut.pos = pos
```

$$\delta t_{cut} = \sum_{b2bcut} \alpha + t_{b2b}(A_i) - \sum_{B_j \in p} t_{b2b}(\mathrm{cut}, B_j)$$



Cut Large Block: REB

Recursive Edge Bisection (REB):

Algorithm: REB

```
function REB_BLOCK(B, n_p)        ▷ Block B fits in n_p partitions.
    if n_p == 1 then
        return
    W = B's workload
    W_l = W · ⌊n_p/2⌋ / n_p, W_r = W − W_l
    find_min_cut(B, W_l, ε, cut)
    cut B into B_l of W_l and B_r of W_r
    reb_block(B_l, ⌊n_p/2⌋)
    reb_block(B_r, ⌈n_p/2⌉)
```

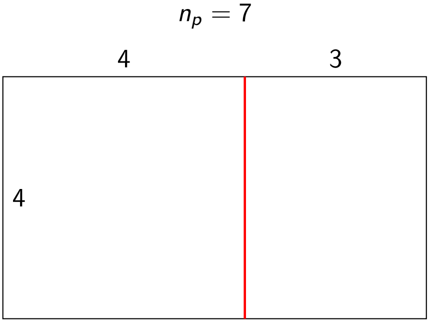
$n_p = 7$

Cut Large Block: REB

Recursive Edge Bisection (REB):

Algorithm: REB

function REB_BLOCK(B, n_p)
  ▷ Block B fits in n_p partitions.
  if $n_p == 1$ **then**
    return
  $W = B$'s workload
  $W_l = W \cdot \frac{\lfloor n_p/2 \rfloor}{n_p}, W_r = W - W_l$
  find_min_cut(B, W_l, ϵ, cut)
  cut B into B_l of W_l and B_r of W_r
  reb_block(B_l, $\lfloor n_p/2 \rfloor$)
  reb_block(B_r, $\lceil n_p/2 \rceil$)

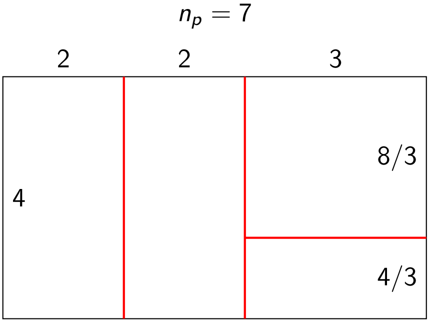


Cut Large Block: REB

Recursive Edge Bisection (REB):

Algorithm: REB

```
function REB_BLOCK(B, n_p)
    ▷ Block B fits in n_p partitions.
  if n_p == 1 then
    return
  W = B's workload
  W_l = W · ⌊n_p/2⌋ / n_p, W_r = W − W_l
  find_min_cut(B, W_l, ϵ, cut)
  cut B into B_l of W_l and B_r of W_r
  reb_block(B_l, ⌊n_p/2⌋)
  reb_block(B_r, ⌈n_p/2⌉)
```

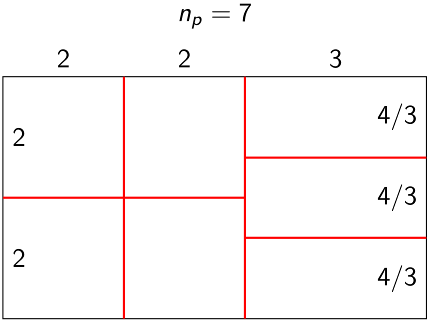


Cut Large Block: REB

Recursive Edge Bisection (REB):

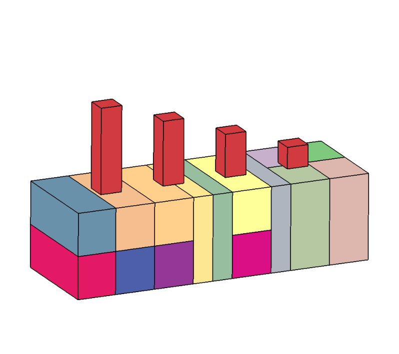
Algorithm: REB

```
function REB_BLOCK(B, n_p)
    ▷ Block B fits in n_p partitions.
  if n_p == 1 then
    return
  W = B's workload
  W_l = W · ⌊n_p/2⌋ / n_p , W_r = W − W_l
  find_min_cut(B, W_l, ε, cut)
  cut B into B_l of W_l and B_r of W_r
  reb_block(B_l, ⌊n_p/2⌋)
  reb_block(B_r, ⌈n_p/2⌉)
```

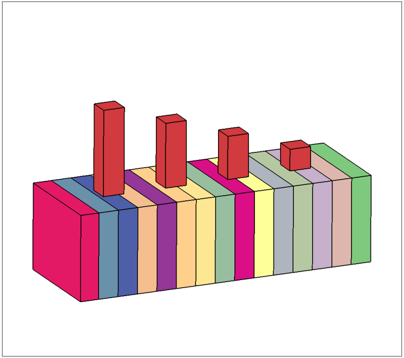


Cut Large Block: REB

Using REB to split Bump3D into 16 partitions with different α, β values:



$\alpha = 10^{-5}$, $\beta = 10^{9}$



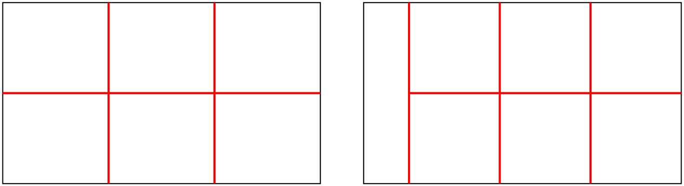
$\alpha = 10^{-4}$, $\beta = 10^{9}$

Cut Large Block: IF

Integer Factorization (IF):

$$n_p = n_x \cdot n_y \cdot n_z, \quad \frac{n_x}{l_x} \approx \frac{n_y}{l_y} \approx \frac{n_z}{l_z}$$

If n_p is prime, cut off one partition and factorize the rest.

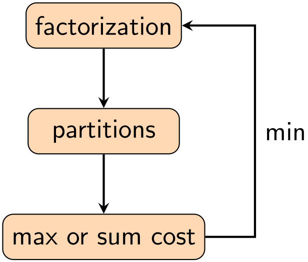


$l_x = 7, l_y = 4, n_p = 6$ $l_x = 7, l_y = 4, n_p = 7$

Cut Large Block: IF

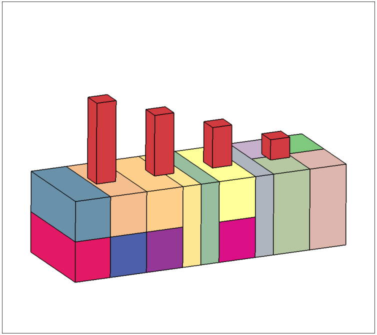
Generalize IF by using $\alpha - \beta$ cost function:

- Compare $n_p = n_x n_y n_z$ and $n_p = 1 + n_x n_y n_z$ for every case.
- Choose the factorization based on max or sum $\alpha - \beta$ cost.

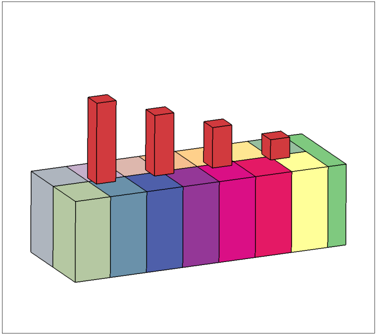


Cut Large Block: Compare REB and IF

Compare REB and IF on Bump3D, $\alpha = 10^{-5}$, $\beta = 10^{9}$, $n_p = 16$.
In general, IF is better at reducing edge cuts and REB better at reducing communication volume.



REB, edge cuts 66, communication volume 1.57×10^{6}

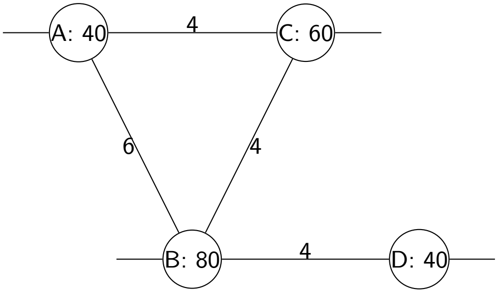


IF, edge cuts 66, communication volume 1.61×10^{6}

Group Small Blocks: CCG

Cut-Combine-Greedy (CCG): cut and combine small blocks in a greedy fashion.

- Include (part of) the block reducing max communication cost to the partition.
- Convert Block2Block communication to shared memory copy.

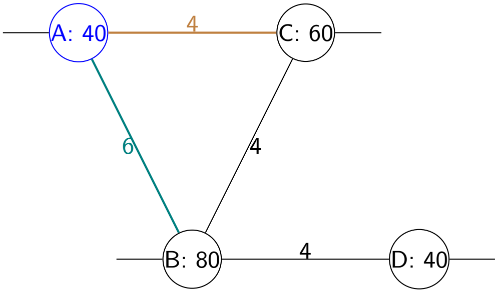


$\overline{W} = 160 \quad W_p = 0$

Group Small Blocks: CCG

Cut-Combine-Greedy (CCG): cut and combine small blocks in a greedy fashion.

- Include (part of) the block reducing max communication cost to the partition.
- Convert Block2Block communication to shared memory copy.



$\overline{W} = 160 \quad W_p = 40$

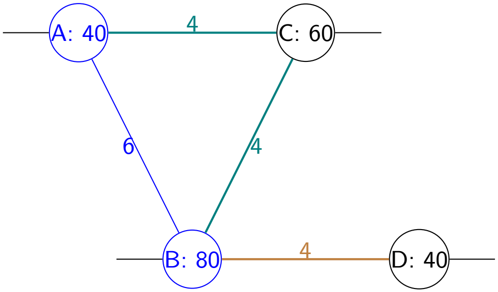
B cost: -6

C cost: -4

Group Small Blocks: CCG

Cut-Combine-Greedy (CCG): cut and combine small blocks in a greedy fashion.

- Include (part of) the block reducing max communication cost to the partition.
- Convert Block2Block communication to shared memory copy.



$\overline{W} = 160 \quad W_p = 120$

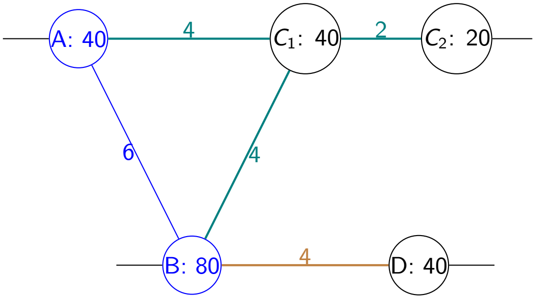
C overload

D cost: -4

Group Small Blocks: CCG

Cut-Combine-Greedy (CCG): cut and combine small blocks in a greedy fashion.

- Include (part of) the block reducing max communication cost to the partition.
- Convert Block2Block communication to shared memory copy.



$\overline{W} = 160 \quad W_p = 120$

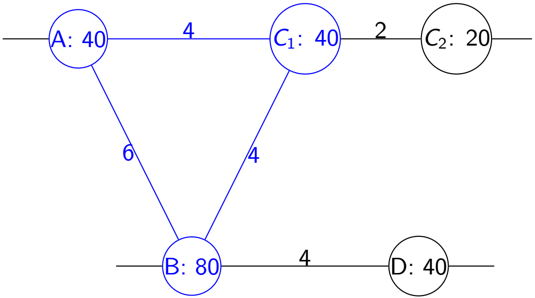
C_1 cost: -4-4+2 = -6

D cost: -4

Group Small Blocks: CCG

Cut-Combine-Greedy (CCG): cut and combine small blocks in a greedy fashion.

- Include (part of) the block reducing max communication cost to the partition.
- Convert Block2Block communication to shared memory copy.



$\overline{W} = 160 \quad W_p = 160$

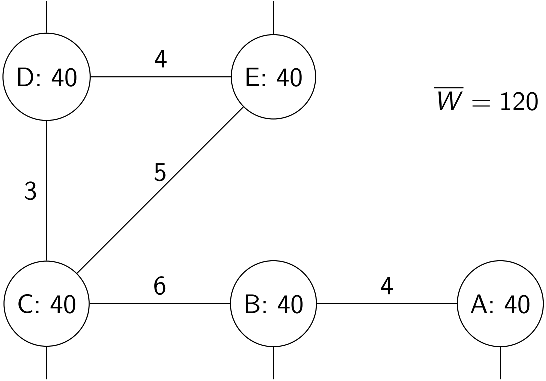
Convert cost: 14

Introduce new cost: 2

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.

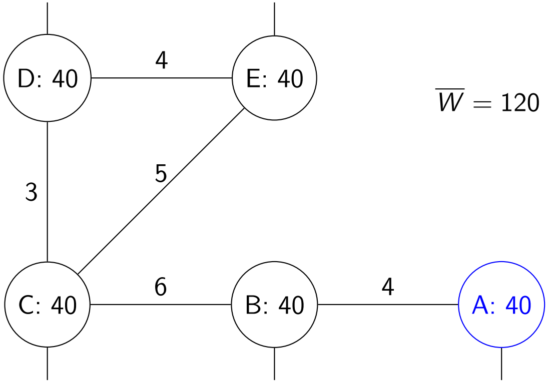


$\overline{W} = 120 \quad W_1 = 0 \quad W_2 = 0$

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.

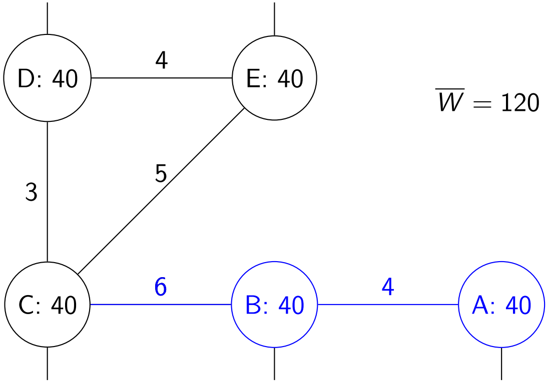


$\overline{W} = 120 \quad W_1 = 40 \quad W_2 = 0$

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.

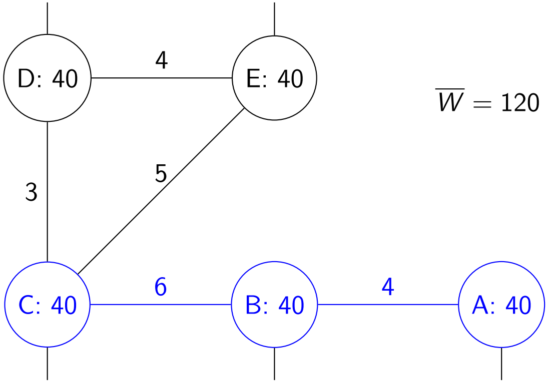


$$\overline{W} = 120 \quad W_1 = 80 \quad W_2 = 0$$

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.

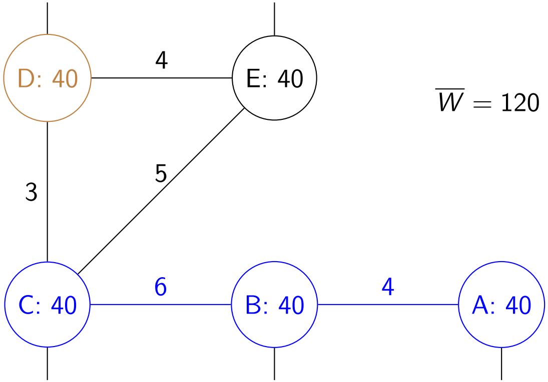


$\overline{W} = 120 \quad W_1 = 120 \quad W_2 = 0$

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.

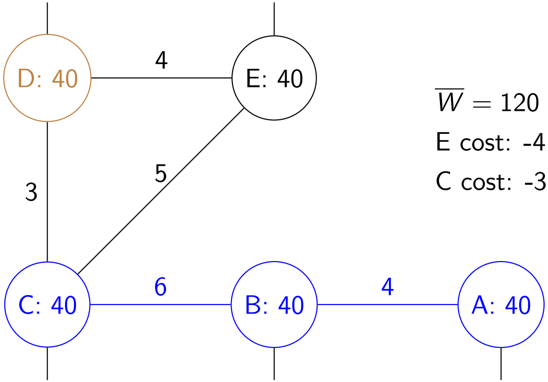


$\overline{W} = 120 \quad W_1 = 120 \quad W_2 = 40$

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.



$\overline{W} = 120 \quad W_1 = 120 \quad W_2 = 40$

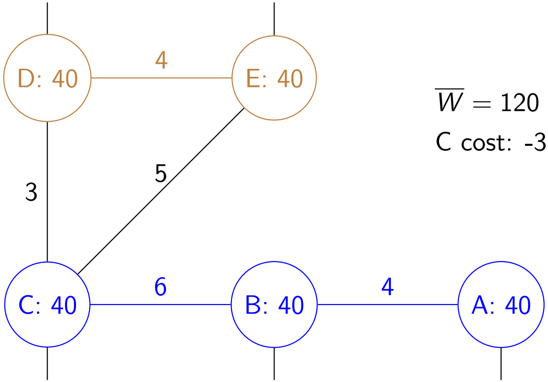
E cost: -4

C cost: -3 + 6 = 3

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.



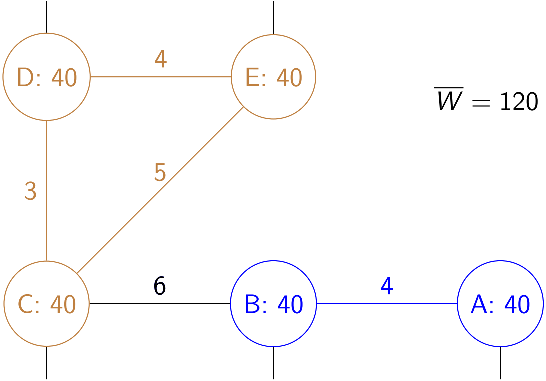
$\overline{W} = 120 \quad W_1 = 120 \quad W_2 = 80$

C cost: -3 - 5 + 6 = -2

Group Small Blocks: GGS

Graph-Growth-Sweep (GGS): Repeatedly use graph growing to group small blocks.

- Convert Block2Block communication to shared memory copy.
- Avoid cutting blocks.



$\overline{W} = 120 \quad W_1 = 80 \quad W_2 = 120$

Group Small Blocks: Compare CCG and GGS

To compare CCG and GGS:

When the #blocks is large, CCG

- ✔ converts more communication to shared memory copy.
- ✘ creates more cuts and new Block2Block communications.

GGS

- ✘ converts less communication to shared memory copy.
- ✔ avoids cutting blocks and introduces less new Block2Block communications.

Outline

Background

Algorithms

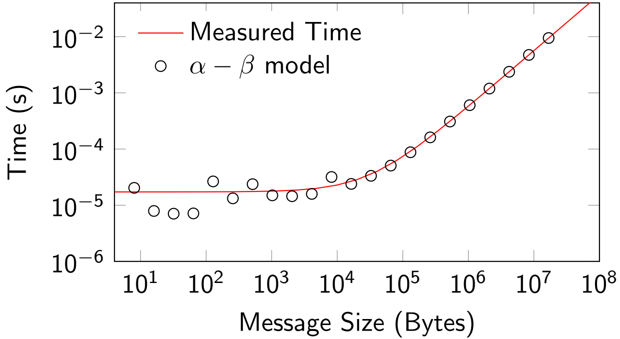
Tests and Results

Conclusion

Hardware and Network Specs

- Our experiments are performed on Mira.

  IBM BlueGene/Q cluster, 16 cores per node
- The latency $\alpha = 1.73 \times 10^{-5}s$ and bandwidth $\beta = 1.77 \times 10^{9}$ bytes/s are measured by a pingpong test.



Numerical Experiment

Implement a Jacobi type solver with MPI and OpenMP.

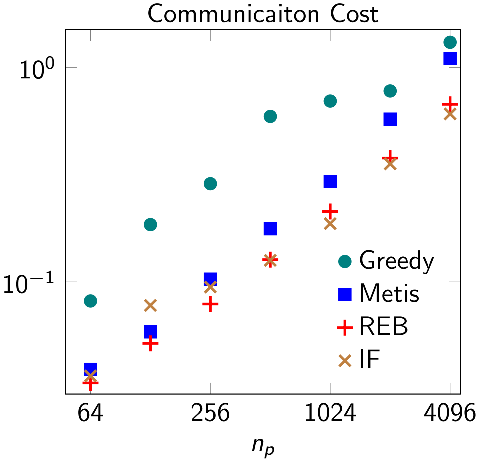
- Assign each MPI process to a node and spawn one OpenMP thread per core.
- Master thread calls MPI non-blocking routines which overlaps with shared memory copy.

Experimental Jacobi Solver

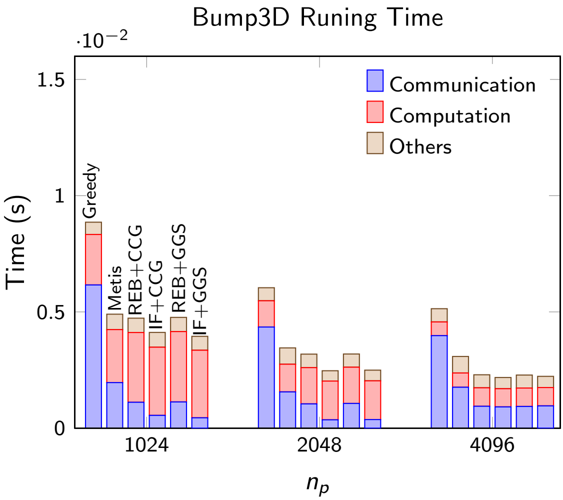
```
for i = 1 → NSTEP do
  ▷ #pragma omp for
    Copy halo data to sending buffer
  ▷ #pragma omp barrier
  ▷ #pragma omp master
    Update halo using non-blocking p2p communication
  ▷ #pragma omp for
    Copy halo data via shared memory within node
  ▷ #pragma omp barrier
  ▷ #pragma omp for
    Copy data from receiving buffer to halo region
  ▷ #pragma omp barrier
  ▷ split blocks evenly among threads
    Computation
  ▷ #pragma omp barrier
```

Bump3D Metrics: Communication Cost

Refine Bump3D 4 times in each direciton, 8.3×10^7 cells in total.
Beyond 512 partitions: Greedy $>$ Metis $>$ REB $>\approx$ IF

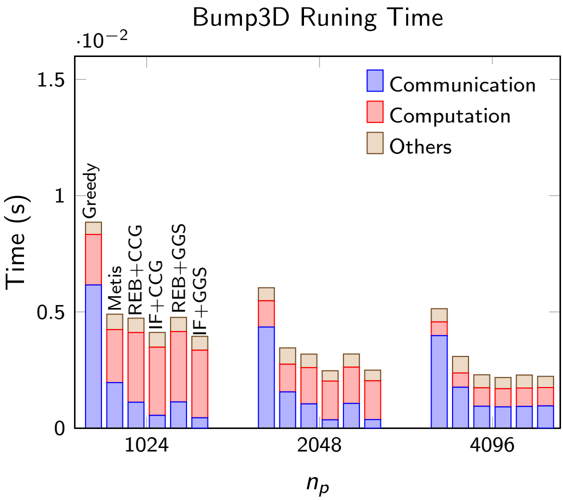


Bump3D Running Time



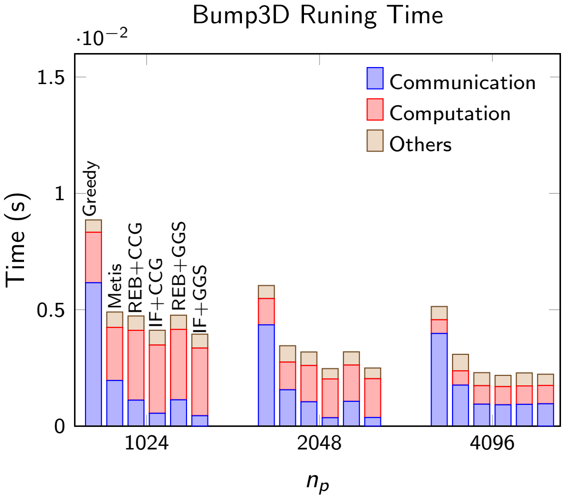
- Consistent with $\alpha - \beta$ cost.

Bump3D Running Time



- Consistent with $\alpha - \beta$ cost.
- Latency has more effect.

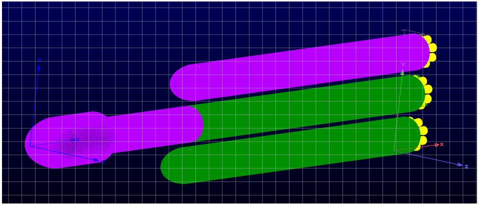
Bump3D Running Time

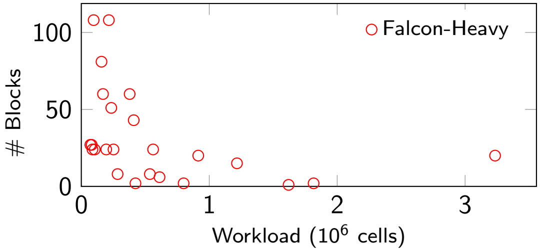


- Consistent with $\alpha - \beta$ cost.
- Latency has more effect.
- At 4096 partitions, IF achieves 5.80x over Greedy and 2.56x over Metis in communication.

Space X's Falcon-Heavy Grid

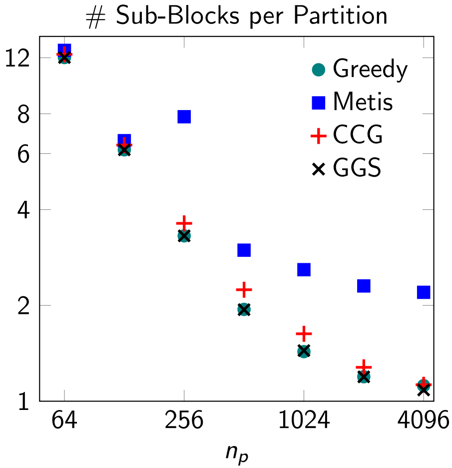
The Space X's Falcon-Heavy grid and its blocks' distribution:





Falcon-Heavy Metrics: # Sub-Blocks

Choose REB+GGS and IF+CCG to present GGS and CCG respectively.



Sub-Blocks:

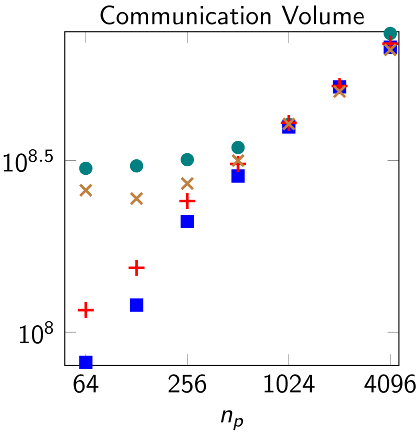
Metis > CCG > Greedy ≥ GGS

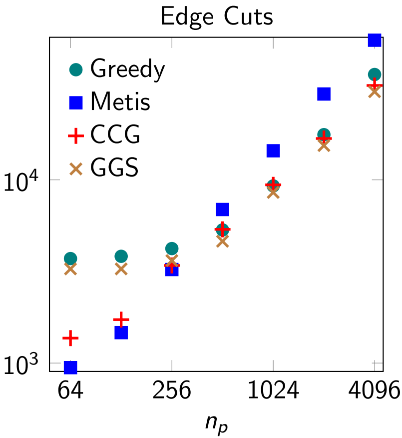
Dominating Pattern:

- 64-256 partitions group small blocks
- 1024-4096 partitions both

Falcon-Heavy Metrics: Volume and Edge Cuts

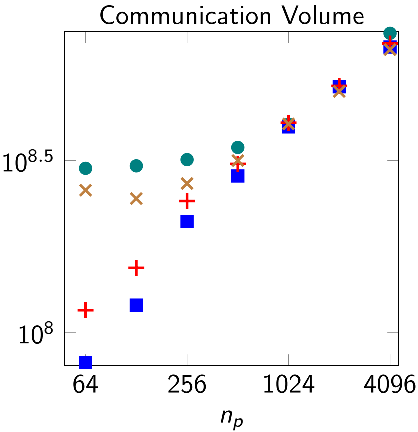
- Greedy produces the max communication volume and edge cuts for 64-256 partitions

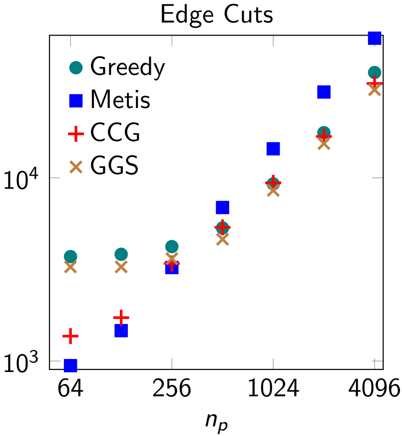




Falcon-Heavy Metrics: Volume and Edge Cuts

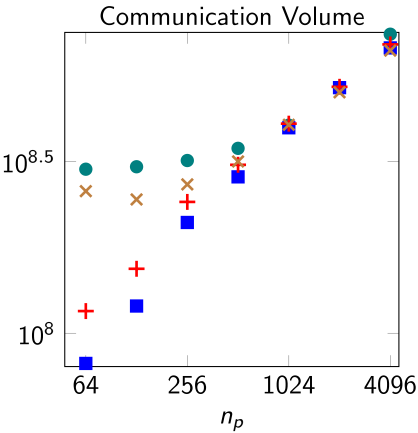
- Metis produces the min communication volume and edge cuts for 64-256 partitions, CCG 2nd.

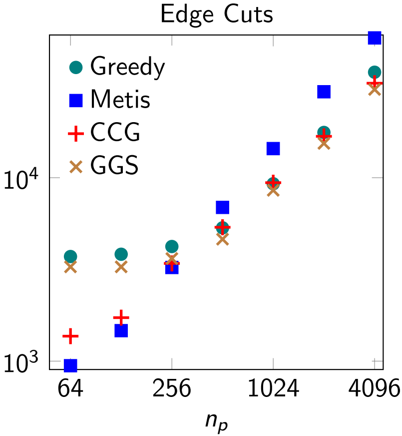




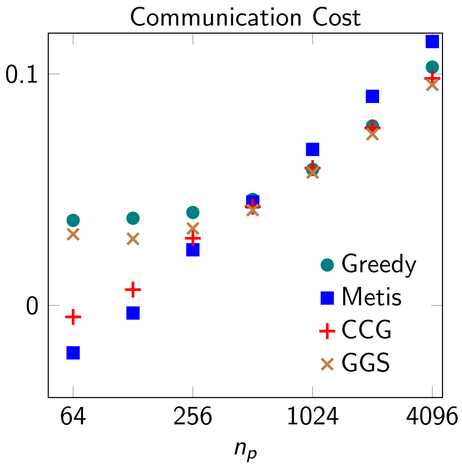
Falcon-Heavy Metrics: Volume and Edge Cuts

- Greedy, CCG, GGS produce close results for 1024–4096 partitions; GGS min.





Falcon-Heavy Metrics: Communication Cost



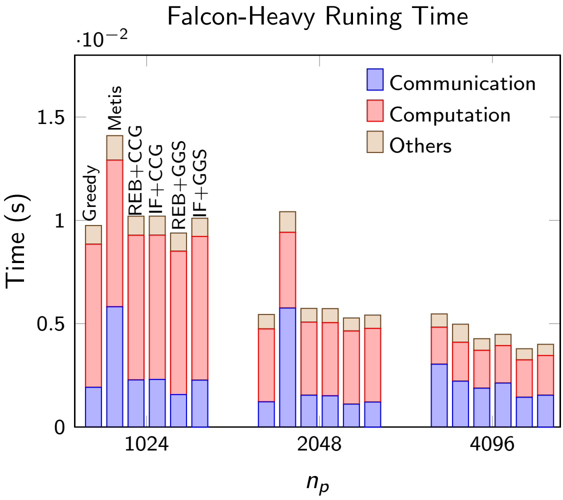
Consistent Pattern with metrics:

- 64-256 partitions

  Metis best, CCG 2nd
- 1024-4096 partitions

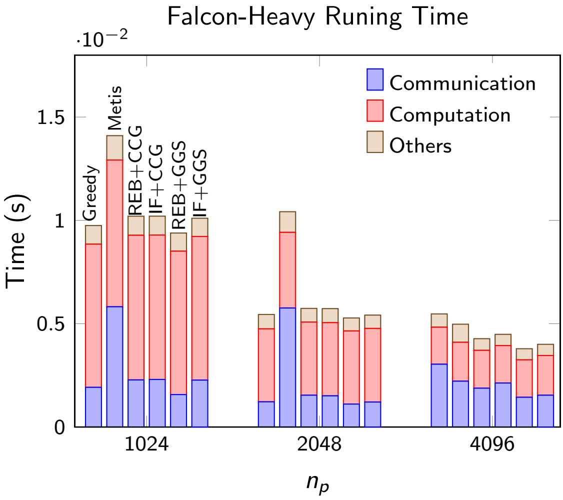
  Metis worst, GGS best

Falcon-Heavy Running Time



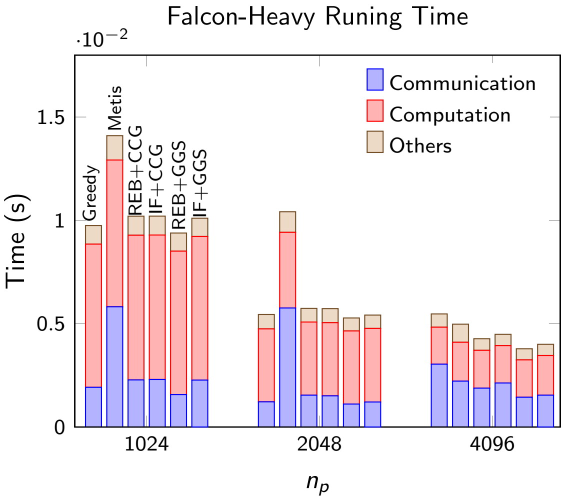
- Metis produces good result at 4096 partitions

Falcon-Heavy Running Time



- Metis produces good result at 4096 partitions
- Greedy creates good result at 1024, 2048 partitions

Falcon-Heavy Running Time



- Metis produces good result at 4096 partitions
- Greedy creates good result at 1024, 2048 partitions
- At 4096 partitions, REB+GGS achieves 2.11x over Greedy and 1.54x over Metis in communication.

Outline

Background

Algorithms

Tests and Results

Conclusion

Conclusion

- Use the $\alpha - \beta$ model to construct a cost function incorporating the edge cuts, communication volume and network specifics.

Conclusion

- Use the $\alpha - \beta$ model to construct a cost function incorporating the edge cuts, communication volume and network specifics.
- Propose modified REB, IF for cutting large blocks and CCG, GGS for grouping small blocks.

Conclusion

- Use the $\alpha - \beta$ model to construct a cost function incorporating the edge cuts, communication volume and network specifics.
- Propose modified REB, IF for cutting large blocks and CCG, GGS for grouping small blocks.
- Test our partitioner with a hybrid MPI+OpenMP based Jacobi solver on up to 4096 nodes.

Conclusion

- Use the $\alpha - \beta$ model to construct a cost function incorporating the edge cuts, communication volume and network specifics.
- Propose modified REB, IF for cutting large blocks and CCG, GGS for grouping small blocks.
- Test our partitioner with a hybrid MPI+OpenMP based Jacobi solver on up to 4096 nodes.
- Achieve significant speedup in communication on both Bump3D and Falcon-Heavy.