Algorithm Design for High Performance CFD Solvers on Structured Grids

[11/19] • [SC20]

Algorithm Design for High Performance CFD Solvers on Structured Grids

Hengjie Wang, Aparna Chandramowlishwaran

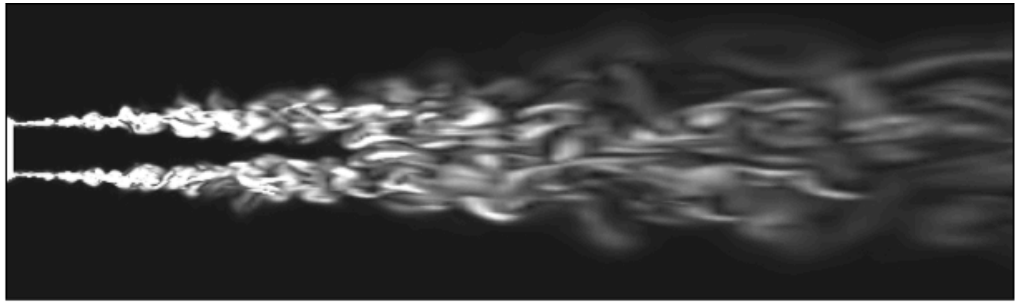
HPC Forge
University of California, Irvine

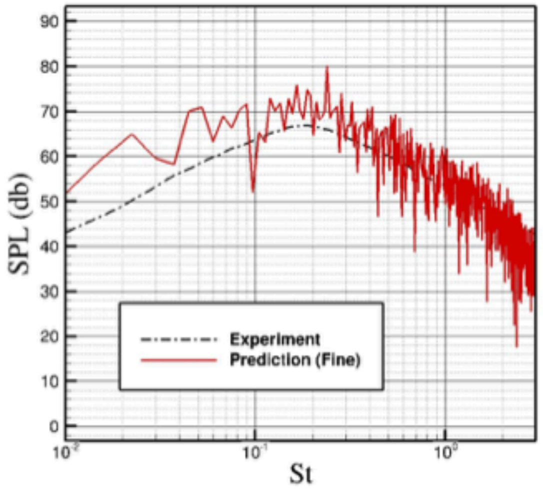
Outline

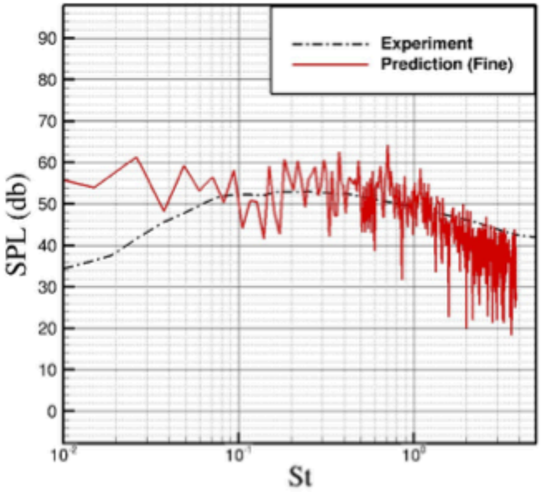
- **Motivation and Background**
- Grid Partitioner
- Pencil: Pipelined Distributed Stencil Computation
- Deep Learning + CFD
- Summary

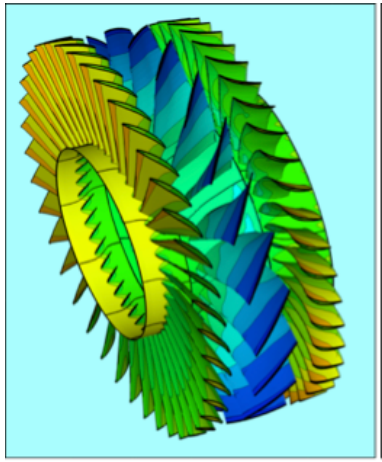
Motivation

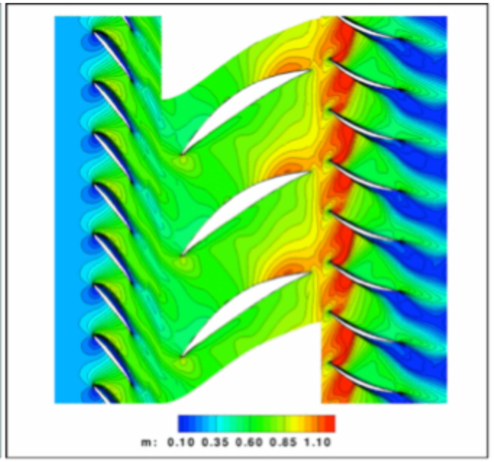
- Computational Fluid Dynamics (CFD)

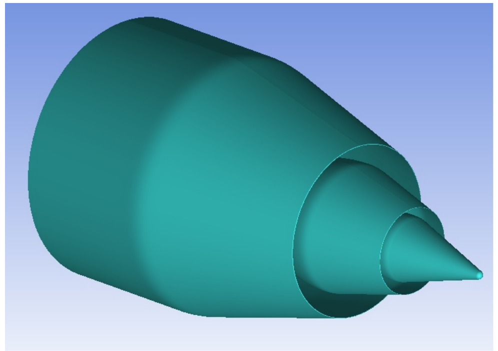


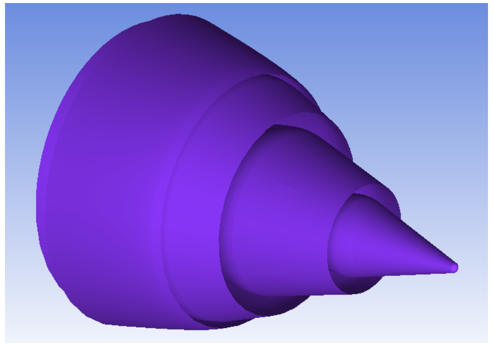








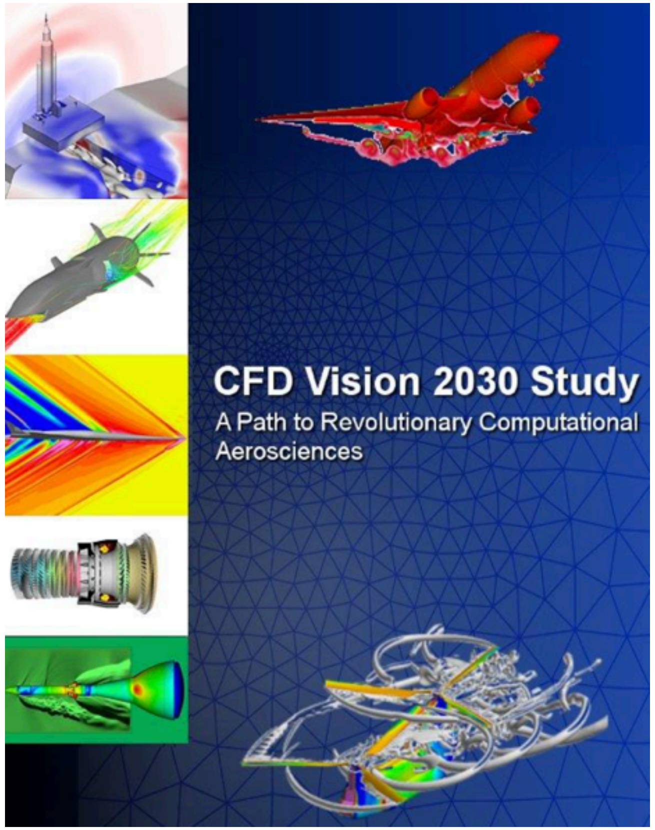




Application Drivers
(Context : HiPER)

Motivation

- NASA CFD Vision 2030 Study

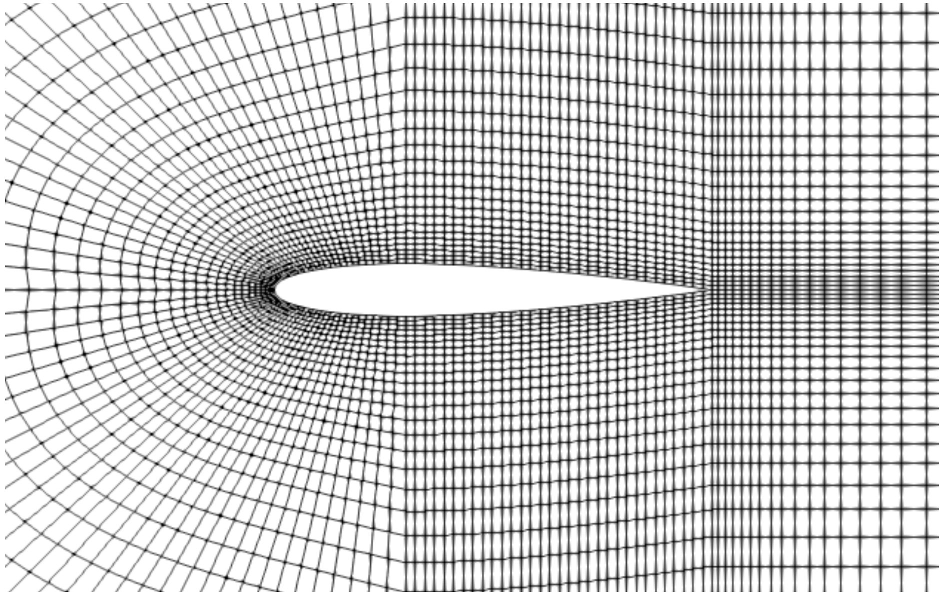


2. **HPC hardware is progressing rapidly and technologies that will prevail are difficult to predict.** However,

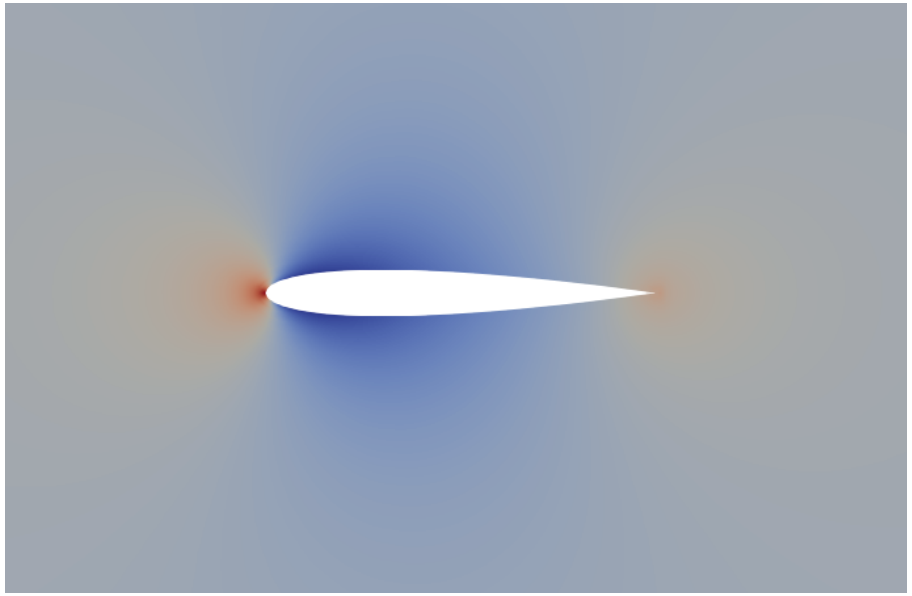
5. **Revolutionary algorithmic improvements will be required to enable future advances in simulation capability.** Traditionally, developments in improved discreti-

Computational Fluid Dynamics

- Discretize space with a grid
- Solve governing equations on the grid



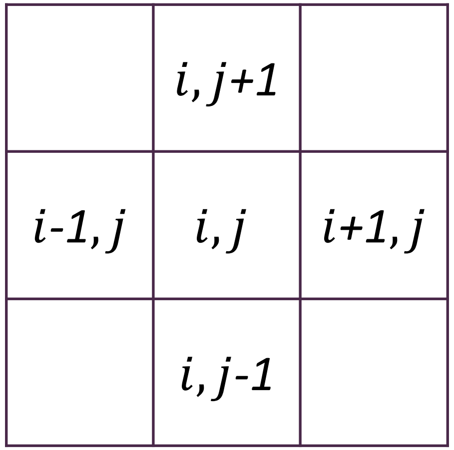
Space discretized by a gird



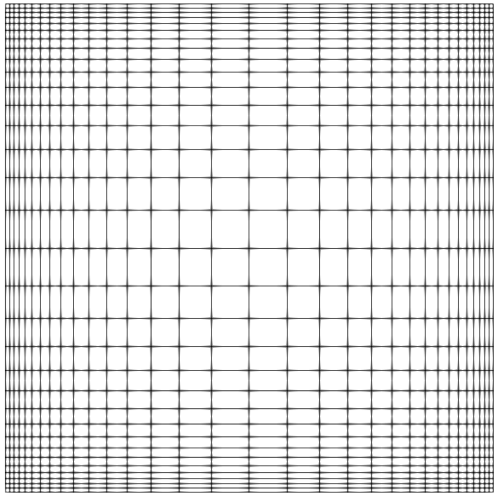
Variables (velocity, pressure) on the grid

Structured Grids

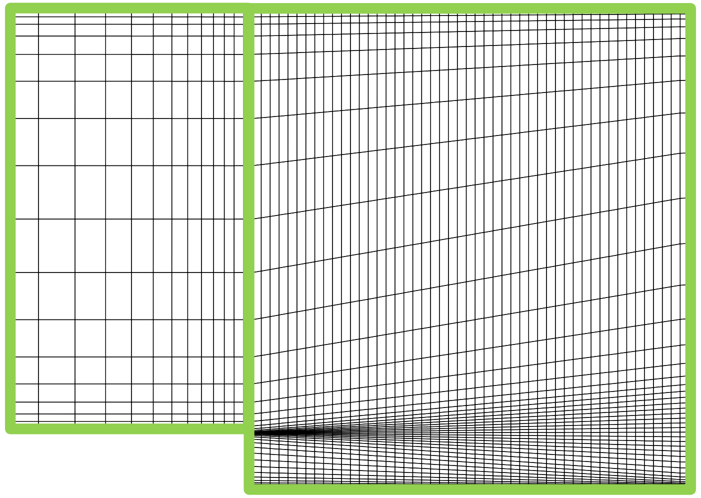
- Regular connectivity between grid cells
- Identical mapping between the grid's data and the memory layout
- Organized into rectangular blocks



Regular Connectivity



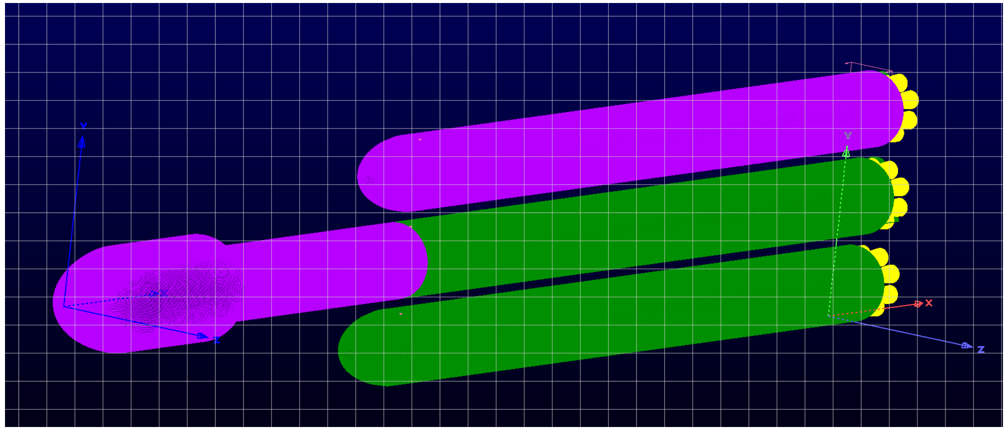
Single Block Grid



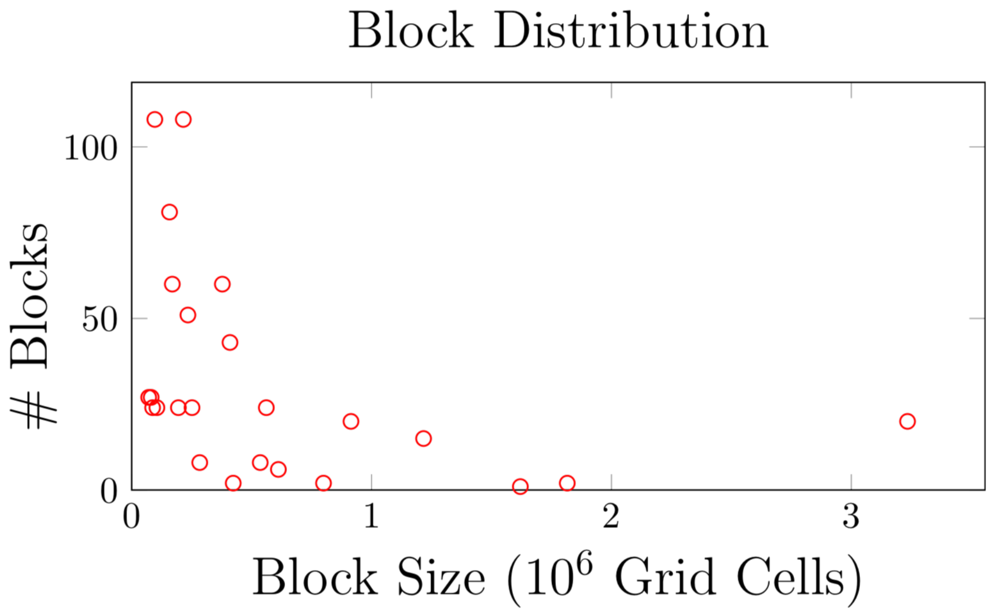
Backward Facing Step
Multi-Block Grid

Multi-Block Structured Grids

- Structured grids for realistic engineering applications consist of $10^2 \sim 10^3$ blocks
  - Rocket model created with SpaceX's released geometric specifics

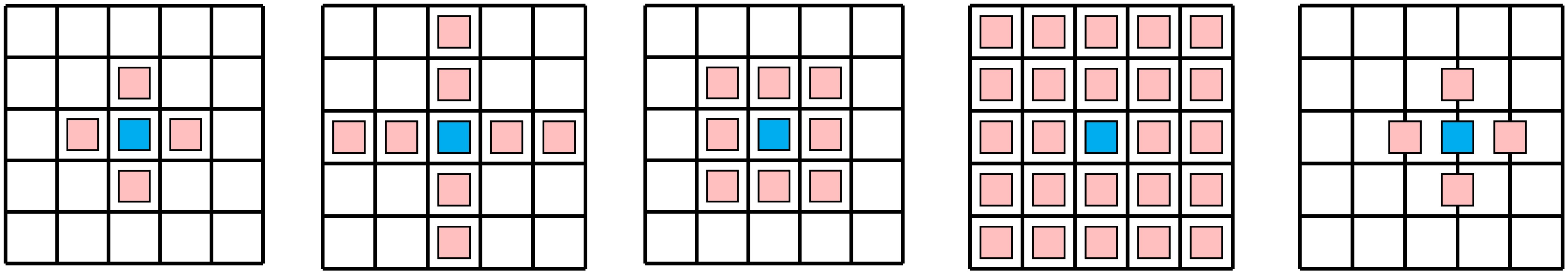


Multi-Block Grid, 769 blocks



Stencils

- The most common computational pattern in CFD using structured girds
  - Characterized by a regular shape
  - Different shapes and radius (*r*)

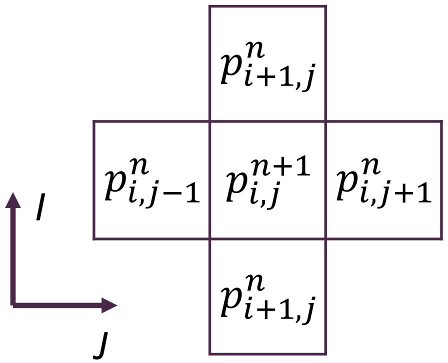


Star, *r* = 1 | Star, *r* = 2 | Box, *r* = 1 | Box, *r* = 2 | Staggered, *r* = 1

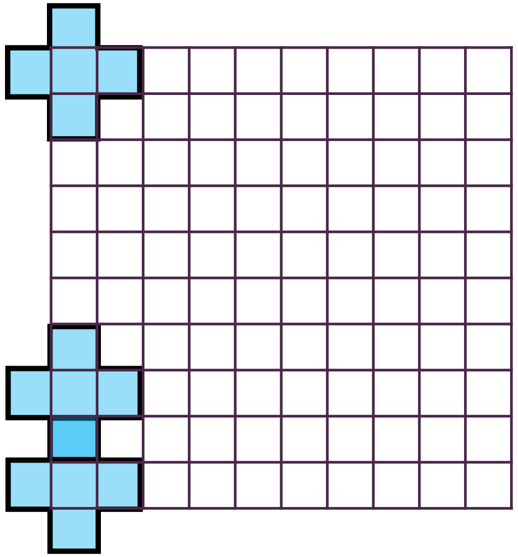
Stencil Computation

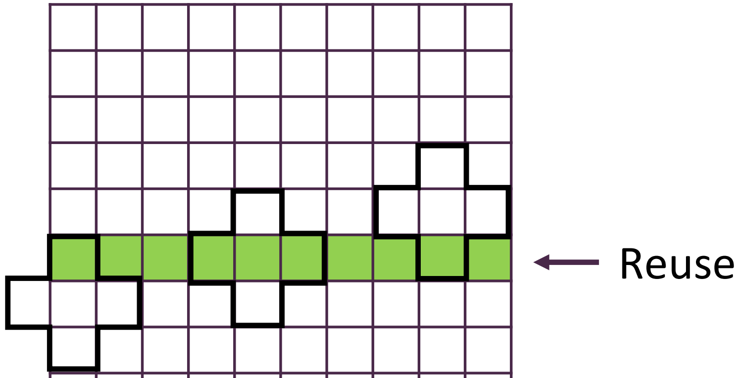
- Typically memory-bound
  - Flops, memory accesses $\sim$ grid size
- Significant data reuse
  - Solve 2D Poisson equation $\nabla^2 p = 0$ with finite difference

Cache Tiling



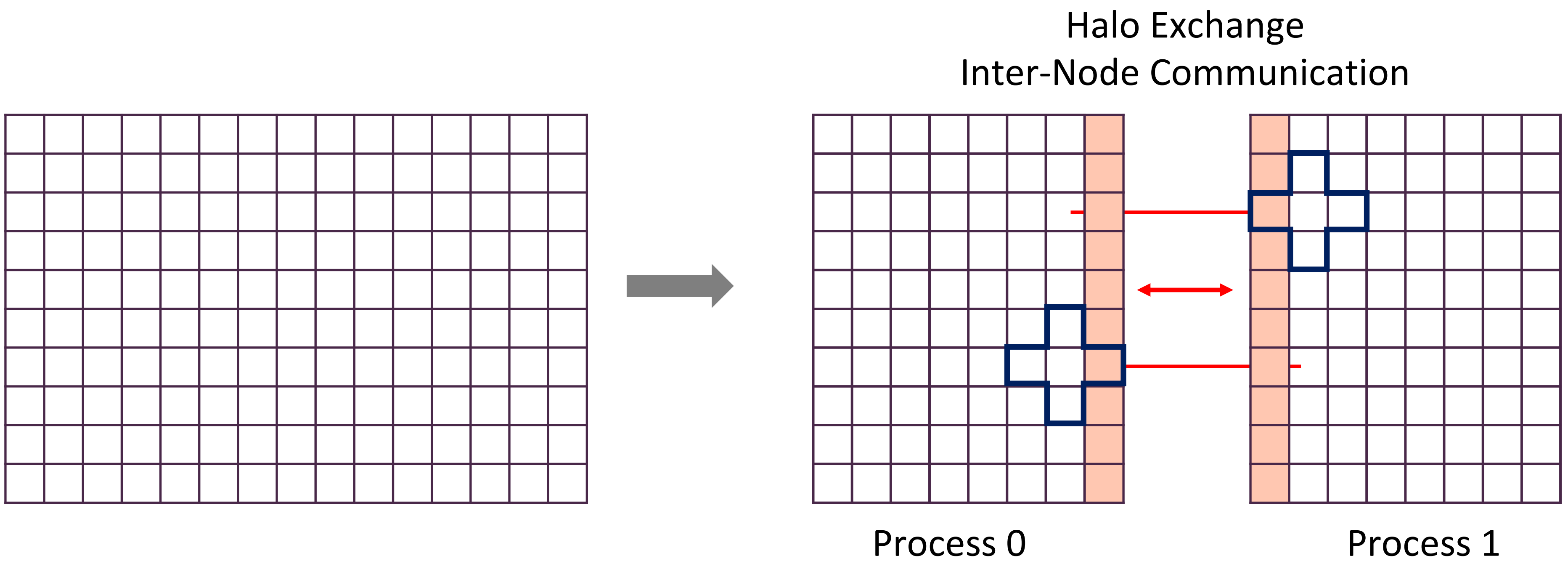
2D 5-Point Stencil





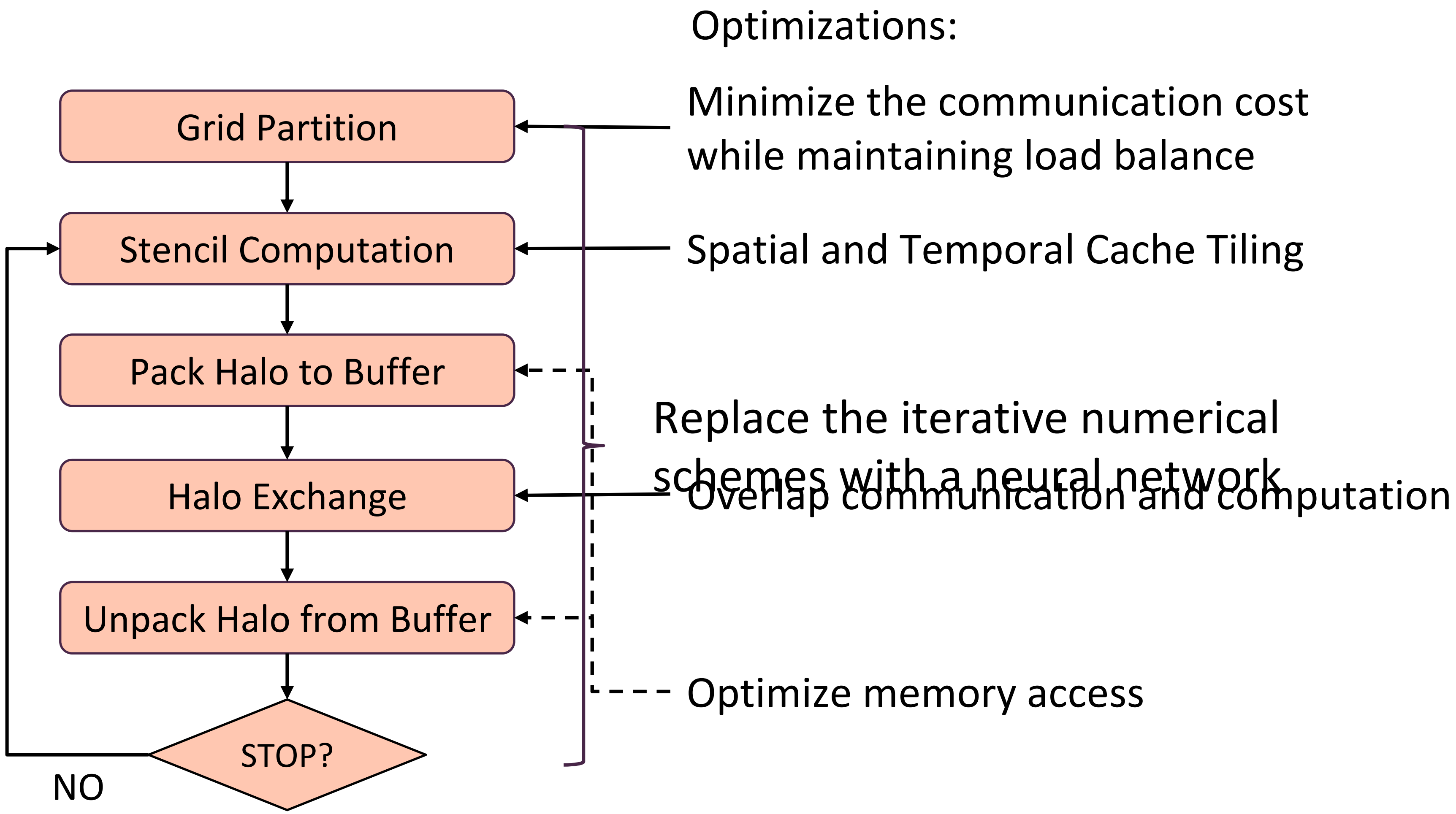
Distributed Stencil Computation

- Blocks are partitioned to sub-blocks and distributed across processes
- Processes communicate to exchange halo layers



Distributed Stencil Computation

- General Algorithm:

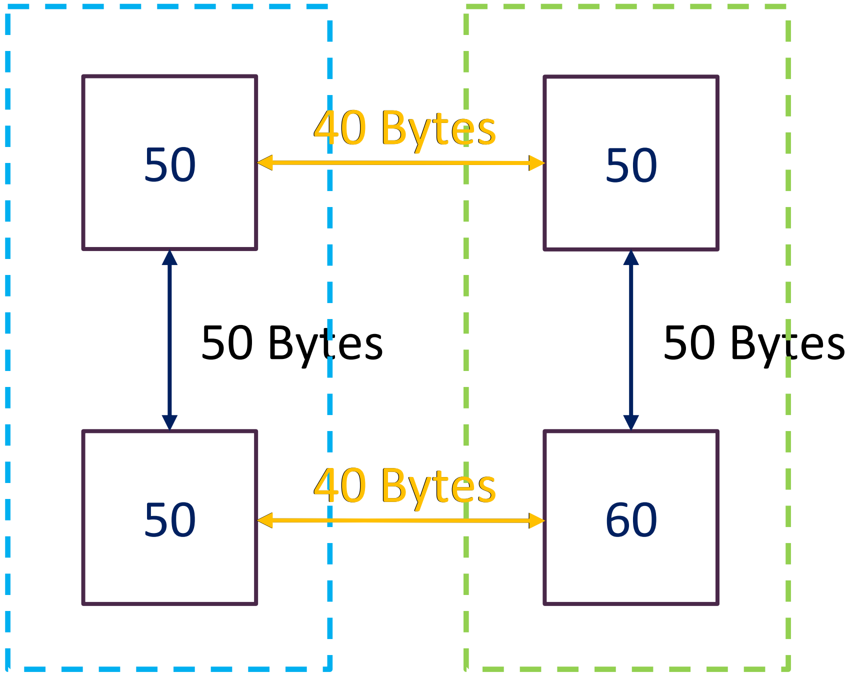


Outline

- Motivation and Background
- Grid Partitioner
  - **Introduction**
  - Algorithms
  - Experiments and Results
  - Summary
- Pencil: Pipelined Distributed Stencil Computation
- Deep Learning + CFD
- Summary

Assumptions and Basic Concepts

- Hybrid Programming Model:
  - One MPI process per node and spawn one thread per core
  - Conform to modern architecture
  - Assume shared memory copy takes no time
- Partition 4 blocks across 2 nodes:



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

Assumptions and Basic Concepts

- Given the number of partitions n_p, the partitioner should:
  - Achieve load balance
  - Minimize the inter-node communication

State-of-the-art

- Top-down:
  - Cut large blocks and assign sub-blocks to partitions
  - Group Small blocks to fill partitions

  Examples: Greedy, Recursive Edge Bisection, Integer Factorization
- Bottom-Up:
  - Transform the problem into graph partitioning via over-decomposition
  - Apply a graph partitioner

  Examples: Metis, Scotch, Chaco

Limitations of the State-of-the-art

- The algorithm does account for shared memory copy
- Use partitions with flat MPI

  The performance mixes shared memory copy and inter-node communication
- Primarily focus on reducing communication volume, ignore the effect of network's latency

Contributions

- New cost function, unifying the communication volume, edge cuts, and network specifics (bandwidth and latency)
- Novel partition algorithms
  - Modify Recursive Edge Bisection (REB) and Integer Factorization (IF) for cutting large blocks
  - Propose Cut-Combine-Greedy (CCG) and Graph-Grow-Sweep (GGS) for grouping small blocks

Outline

- Motivation and Background
- Grid Partitioner
  - Introduction
  - **Algorithms**
  - Experiments and Results
  - Summary
- Pencil: Pipelined Distributed Stencil Computation
- Deep Learning + CFD
- Summary

Cost Function

- $\alpha - \beta$ model: α latency (s), β bandwidth (Bytes/s), S message size (Bytes) :

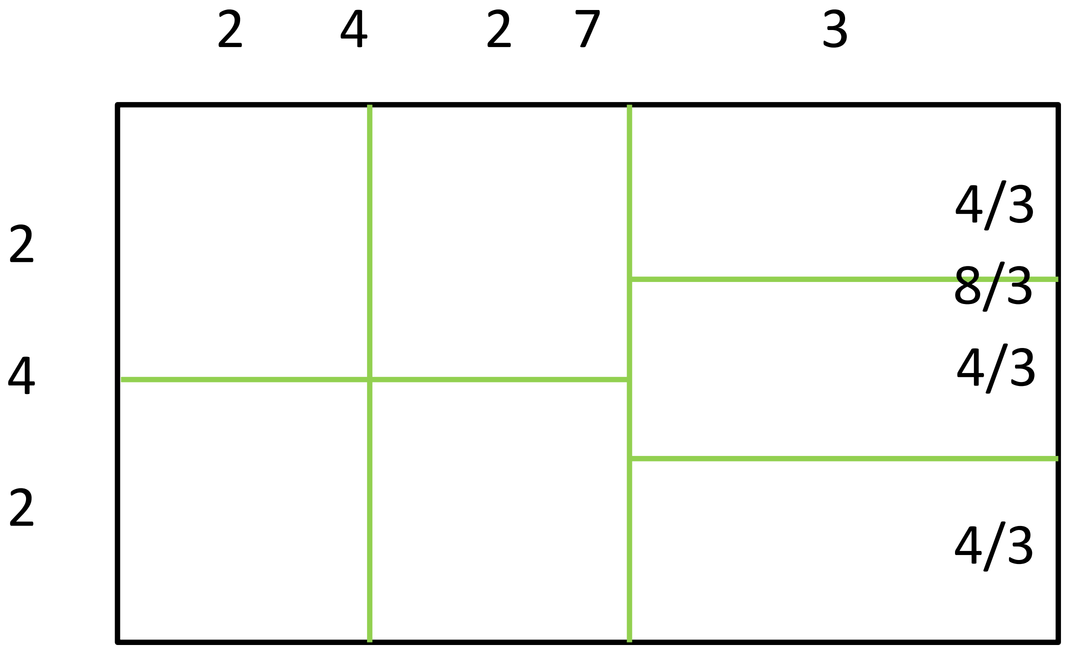
$$t_{msg} = \alpha + \frac{S}{\beta}$$

- Sum over all the inter-node messages:

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

Cutting Large Blocks

- Recursive Edge Bisection (REB) [Berger 1987]
  - Recursively choose the cut that introduces minimum communication cost



$n_p = 7$

Cutting Large Blocks

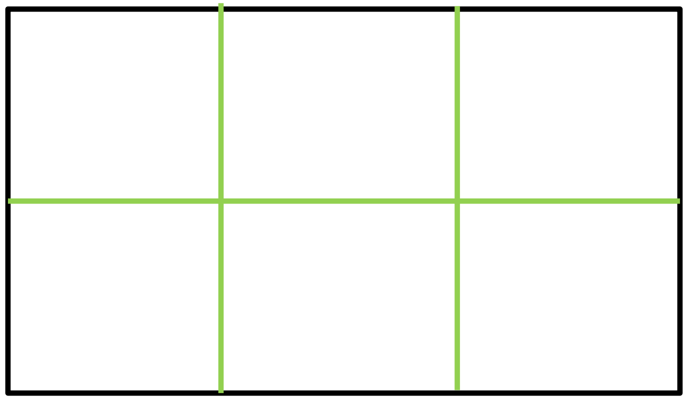
- Integer Factorization (IF)

$$n_p = n_i \cdot n_j \cdot n_k, \qquad \frac{n_i}{l_i} \approx \frac{n_j}{l_j} \approx \frac{n_k}{l_k}$$

  - Choose the $\{n_i, n_j, n_k\}$ that leads to the minimum communication cost
  - If n_p is prime, then cut off one partition and factorize the rest

7

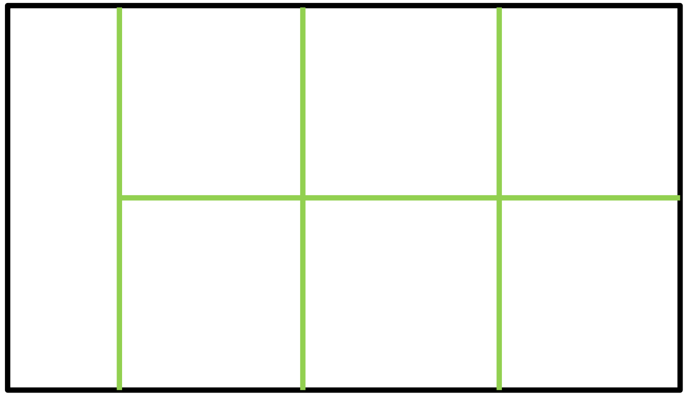
4



$l_i = 7, l_j = 4, n_p = 6$

7

4



$l_i = 7, l_j = 4, n_p = 7$

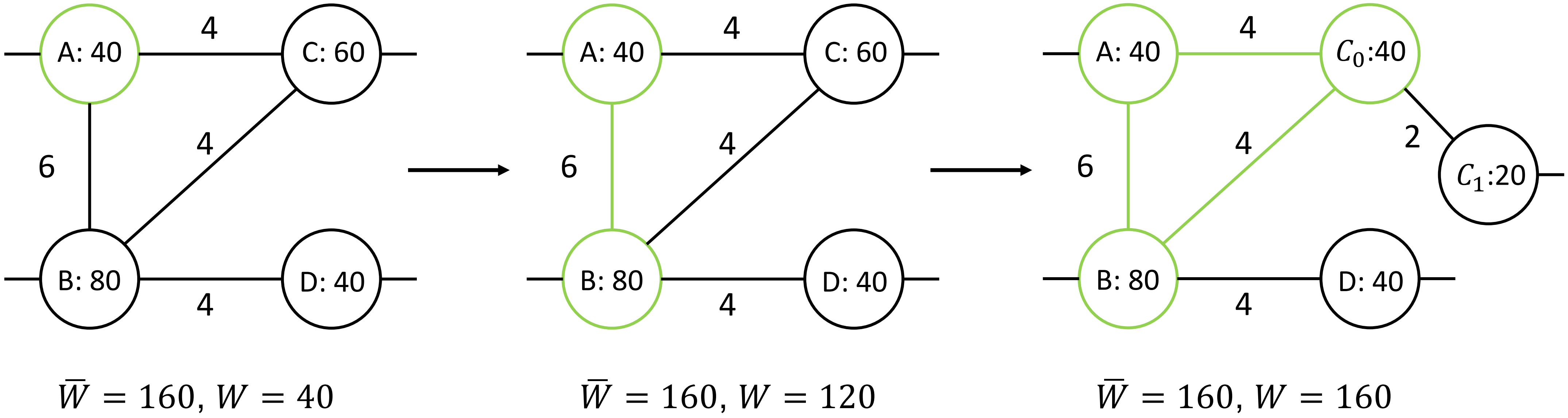
REB vs IF

- REB
  - ✓ Reduces communication volume
  - ✗ Introduces new edge cuts
- IF
  - ✓ Aligns block boundaries and avoids new edge cuts
  - ✗ May not be as good as REB in reducing communication volume

Grouping Small Blocks

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

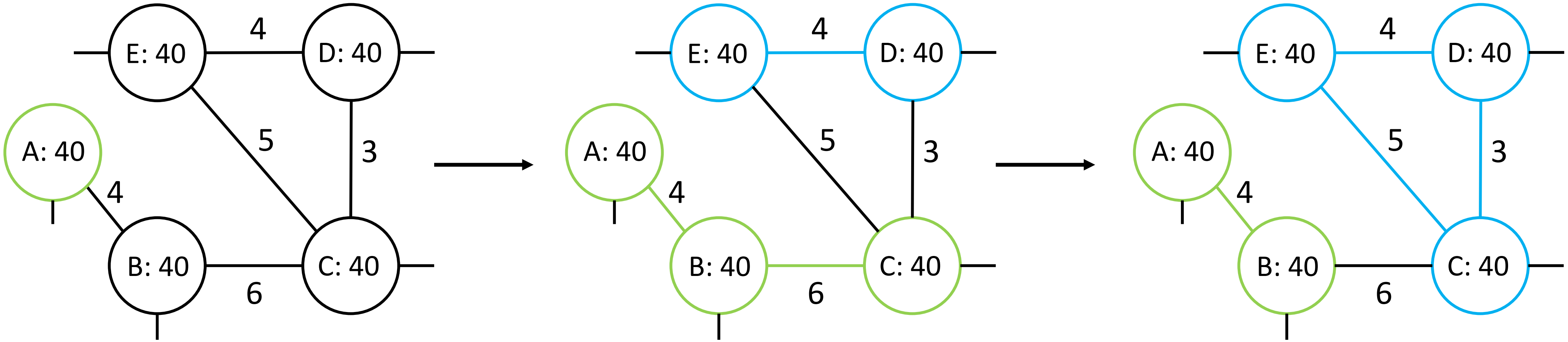
- Include (part of) the block that reduces max communication cost into the partition
- Convert inter-node communication to shared memory copy



Grouping Small Blocks

Graph-Growing-Sweep (GGS): repeatedly use graph-growing to group small blocks

- Avoid cutting blocks
- Convert inter-node communication to shared memory copy



$\overline{W} = 120, W_1 = 40, W_2 = 0$ $\overline{W} = 120, W_1 = 120, W_2 = 80$ $\overline{W} = 120, W_1 = 80, W_2 = 120$

CCG vs GGS

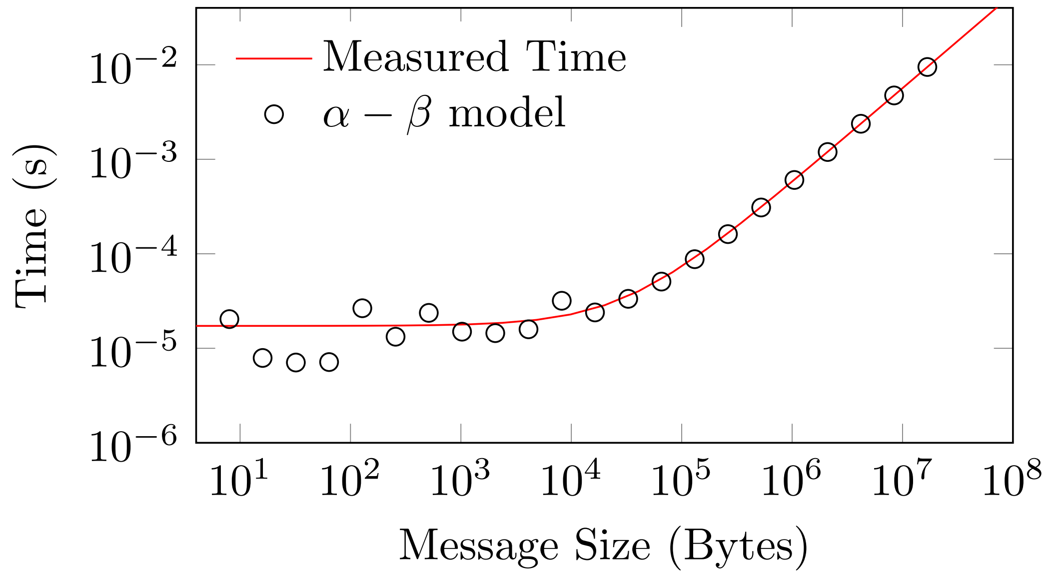
- CCG
  - ✓ Converts more inter-node communication to shared memory copy
  - ✗ Creates more edge cuts and introduces new messages
- GGS
  - ✓ Converts less communication to shared memory copy
  - ✗ Avoids cutting blocks and introduces less new messages

Outline

- Motivation and Background
- Grid Partitioner
  - Introduction
  - Algorithms
  - **Experiments and Results**
  - Summary
- Pencil: Pipelined Distributed Stencil Computation
- Deep Learning + CFD
- Summary

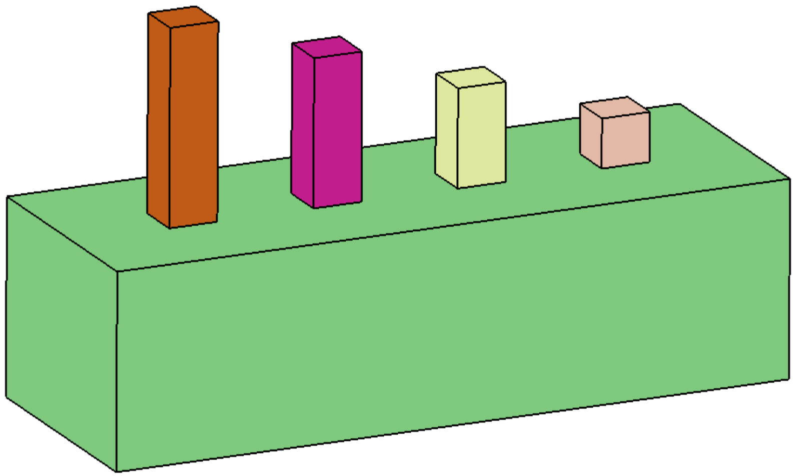
Test Setup

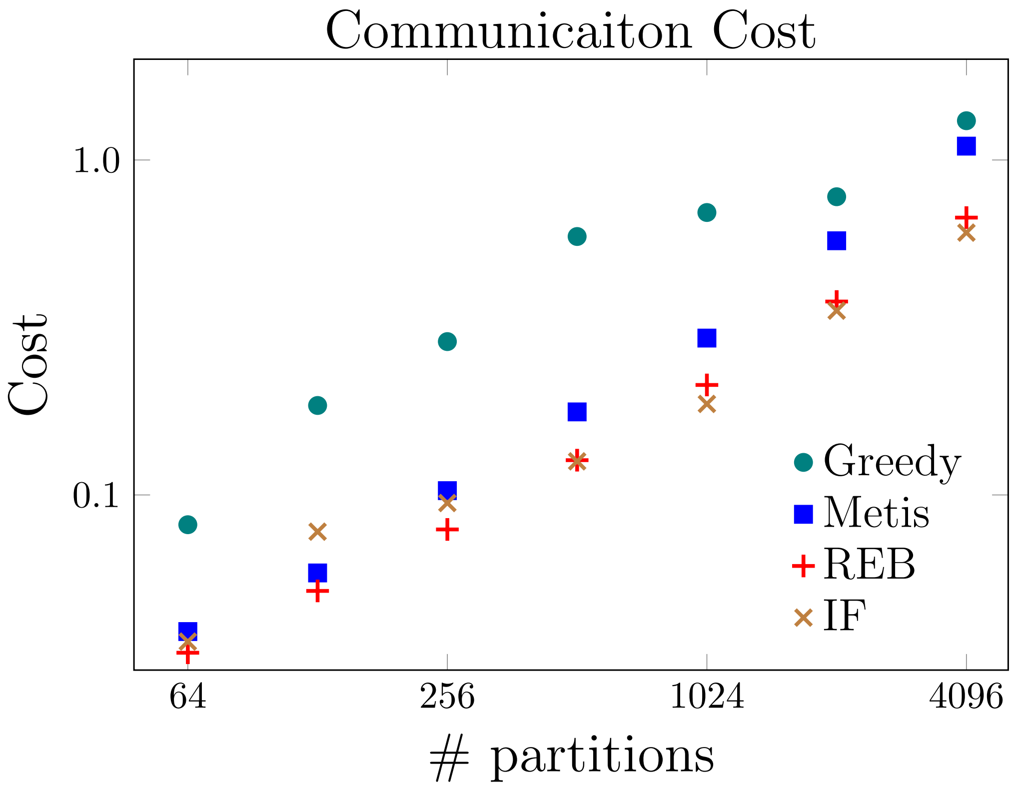
- Evaluated with a MPI + OpenMP based Jacobi Solver
- Compare to Greedy and Metis + Over-Decomposition
- Mira Supercomputer: IBM BlueGene/Q nodes, 16 cores per node
- Network latency $\alpha = 1.73\times10^{-5}$ s and bandwidth $\beta = 1.77\times10^{9}$ bytes/s



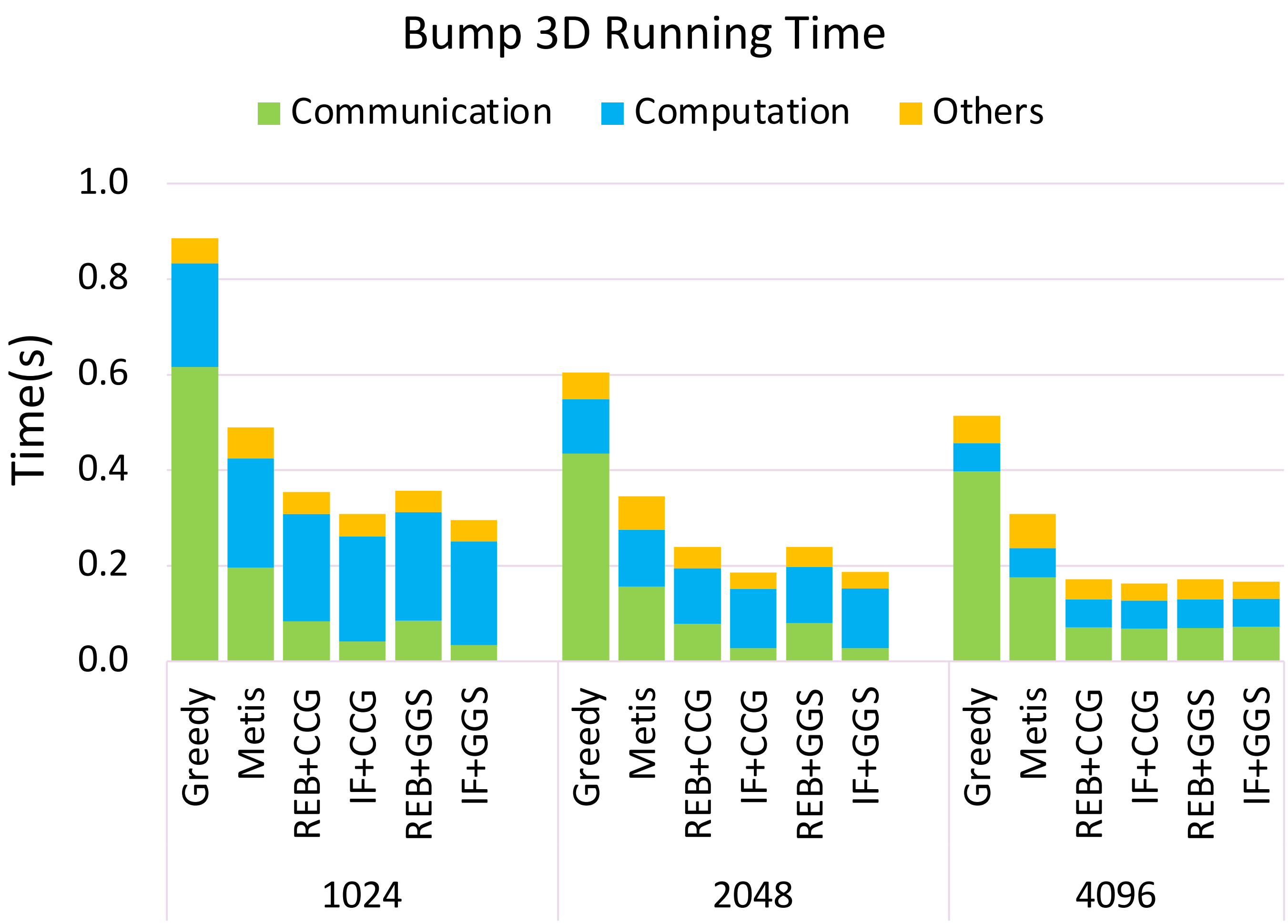
Bump 3D

- Bump3D: 5 blocks and 8.3×10^{7} cells in total
- Beyond 512 partitions, estimate cost: Greedy > Metis > REB > IF



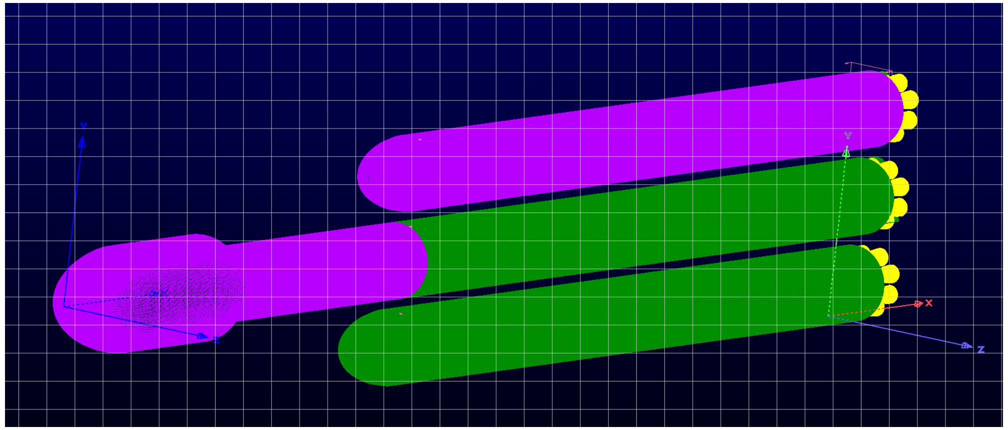


Bump 3D

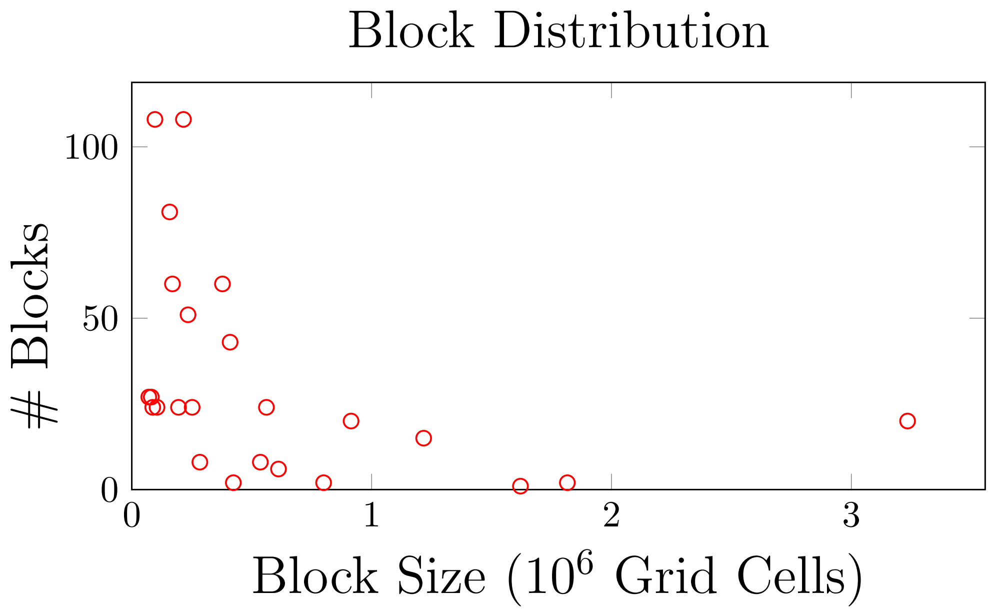


- Consistent with cost model
- At 4096 nodes, IF outperforms Greedy by 5.80x and Metis 2.56x in communication
- Latency has more effect

Rocket Model

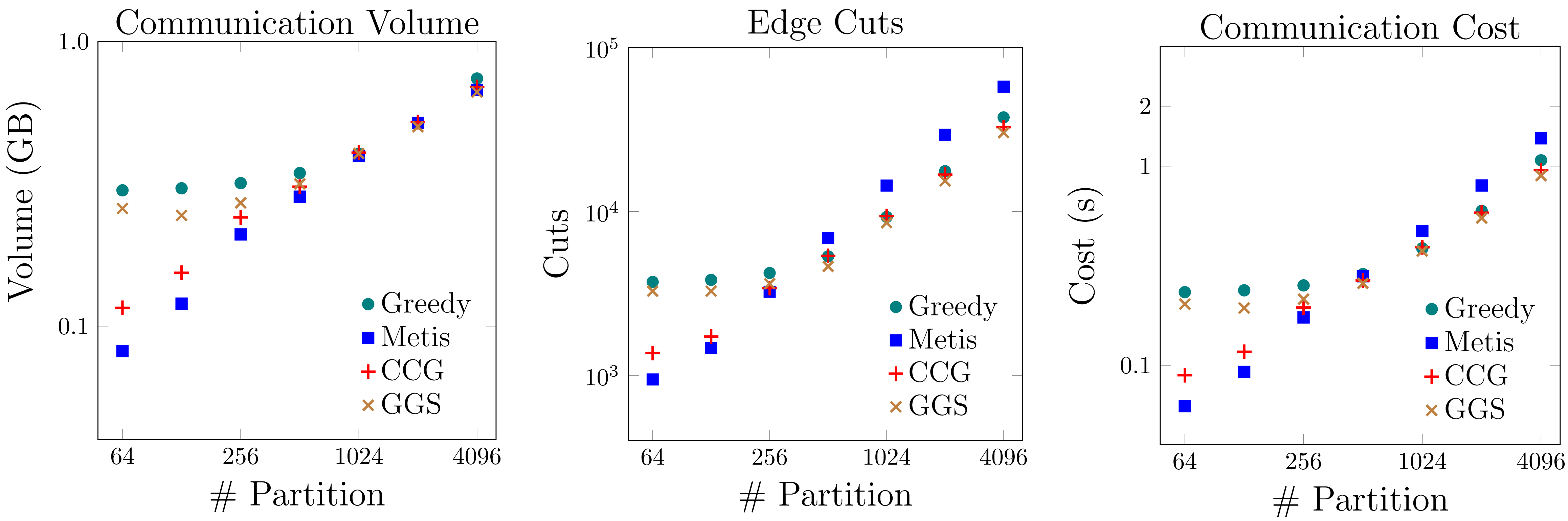


Rocket model created with SpaceX's released geometry specifics, 769 blocks

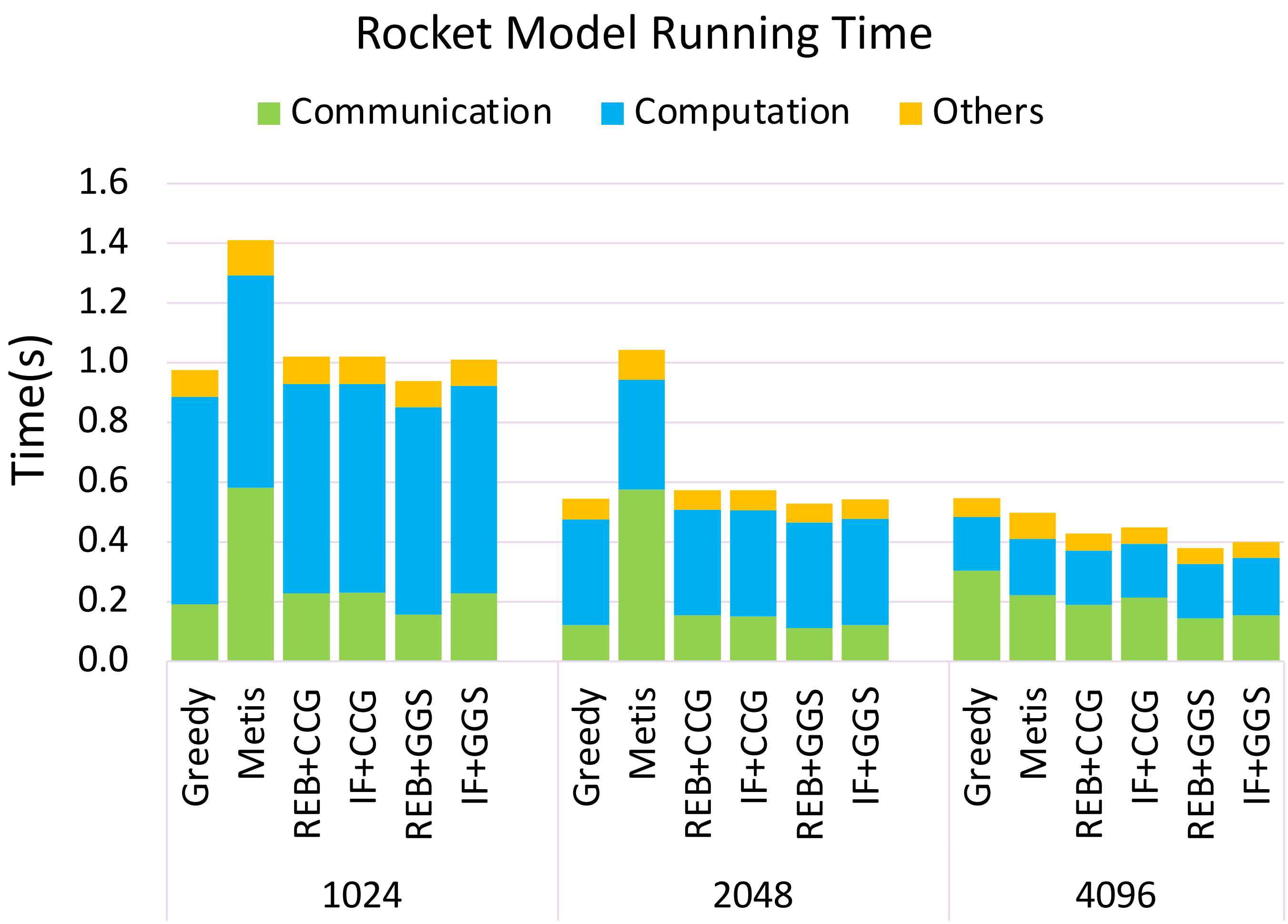


Rocket Model

- Metis performs the worst for 1024-4096 partitions for its large cut edges
- Greedy achieves similar performance compared to REB + CCG and IF + GGS



Rocket Model



- Metis shows good performance at 4096 partitions.
- Greedy shows good performance at 1024, 2048 partitions.
- At 4096 nodes, IF outperforms Greedy by 2.11x and Metis 1.54x in communication

SC20

Summary

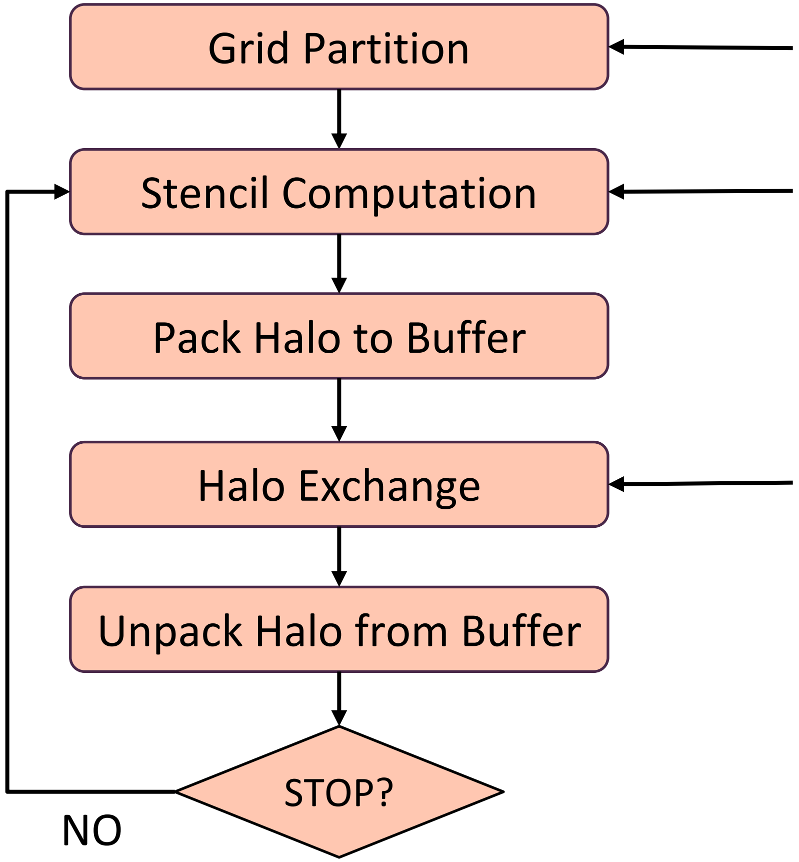
- Use $\alpha - \beta$ model to define a new cost function, unifying the communication volume, edge cuts and network latency and bandwidth
- Propose modified REB, IF for cutting large blocks and novel algorithms CCG, GGS for grouping small blocks
- Evaluated with an MPI + OpenMP based Jacobi solver on up to 4096 nodes, our partitioner achieves significant speed up in communication:
  - 5.80x over Greedy, 2.57x over Metis on Bump 3D
  - 2.11x over Greedy, 1.54x over Metis on Rocket Model

Outline

- Motivation and Background
- Grid Partitioner
- Pencil: Pipelined Distributed Stencils
  - **Introduction**
  - Algorithm
  - Tests and Results
  - Summary
- Deep Learning + CFD
- Summary

Distributed Stencil Computation

- General Algorithm:



Optimizations:

Minimize the communication cost while maintaining load balance

Spatial and Temporal Cache Tiling

Overlap communication and computation

Limitations of the State-of-the-art and Challenges

- Most temporal tiling methods are designed for shared memory systems
  - Find the optimal combination of MPI, OpenMP and temporal tiling
- Temporal tiling is not directly applicable to multi-block grids
  - Most temporal tiling methods are designed for a single block

Temporal Tiling is not Directly Applicable to Multi-Block Grids

- Temporal tiling works for perfectly nested loop – single block

Single Block

```
// time loop
for (int t=1; t<NT; ++t)
  // space loops
  for (int i=0; i<NI; ++i)
    for (int j=0; j<NJ; ++j)
      for (int k=0; k<NK; ++k)
        compute_stencil(i, j, k);
```

- Introduces data dependencies between blocks per time iteration
- Prevents tiling the time loop

Multi-Block

```
for (int t=1; t<NT; ++t) {
  for (int block=0; block<NB; ++block) {
    get_block_size(block, sizes);
    for (int i=0; i<sizes[0]; ++i)
      for (int j=0; j<sizes[1]; ++j)
        for (int k=0; k<sizes[2]; ++k)
          compute_stencil(i, j, k);
  }
  for (int block=0; block<NB; ++block)
    exchange_boundary(block, nHalo);
}
```

Limitations of the State-of-the-art and Challenges

- Most temporal tiling methods are designed for shared memory systems
  - Find the optimal combination of MPI, Threads(OpenMP) and temporal tiling
- Temporal tiling is not directly applicable to multi-block grids
  - Most temporal tiling methods are designed for a single block
- How to hide the communication cost efficiently with temporal tiling?
  - Non-blocking communication does not necessarily overlap
  - Data dependency

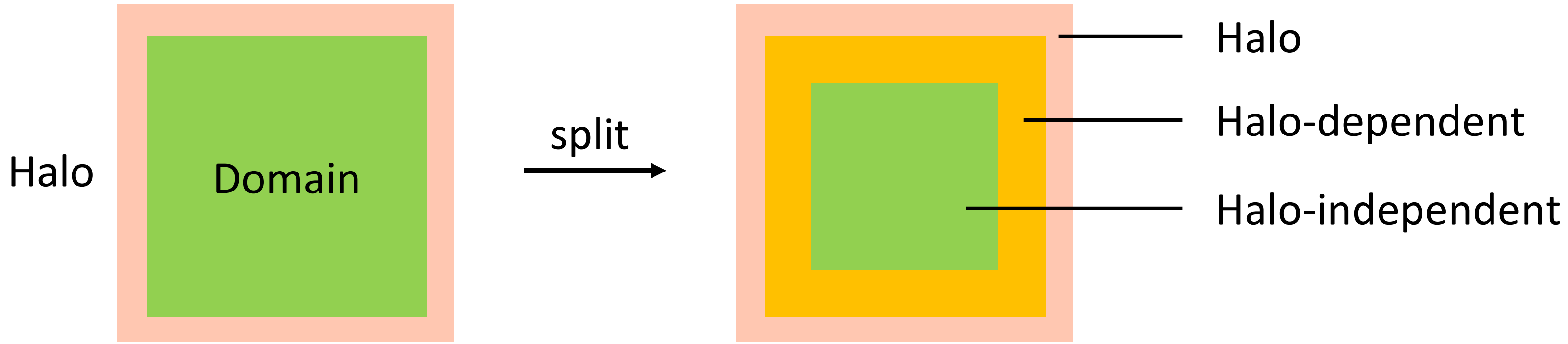
Overlap Communication and Computation

- Non-Blocking communication
  - Communication does not necessarily proceed outside MPI routines

```
MPI_Isend(...);
MPI_Irecv(...);
compute_stencil();
MPI_Waitall();
```

Most of the communication ends up serialized with computation

- Data dependency between stencil computation and halo



Contributions

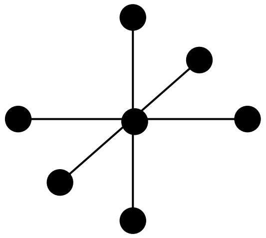
- **Pencil**: A Pipelined Algorithm for Distributed Stencil Computation
  - Find an optimal combination of MPI, OpenMP, and temporal tiling
  - Extend temporal tiling to multi-block grids
  - Pipeline computation and communication to achieve overlap

Outline

- Motivation and Background
- Grid Partitioner
- Pencil: Pipelined Distributed Stencils
    - Introduction
    - **Algorithms**
    - Experiment and Results
    - Summary
- Deep Learning + CFD
- Summary

Flat MPI vs OpenMP – Memory Arrangement

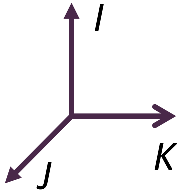
- Solve $\nabla^2 p = b$ on a block of size N^3 on 8 cores
  - 3D 7-point Stencil
  - Streaming access: $K \rightarrow J \rightarrow I$
  - Without cache tiling, OpenMP is more likely to spill the cache

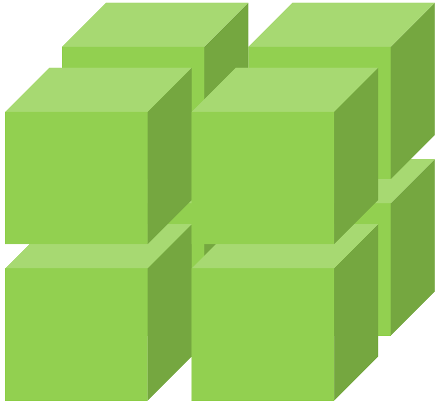


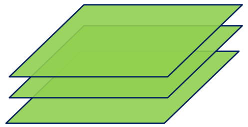
3D 7-point

Each process allocates $(N/2)^3$ cells

One process allocates N^3 cells

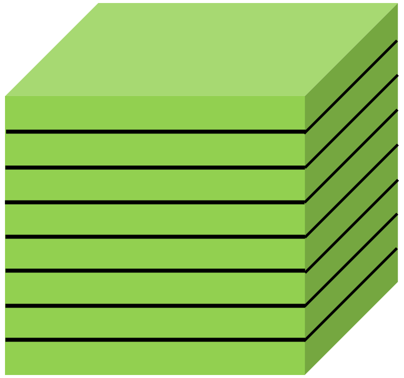


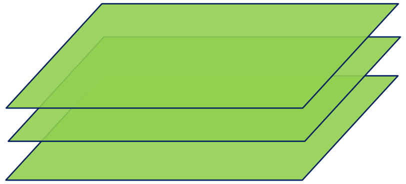




Flat MPI *J*-*K* plane area $(N/2)^2$

Flat MPI $2 \times 2 \times 2$



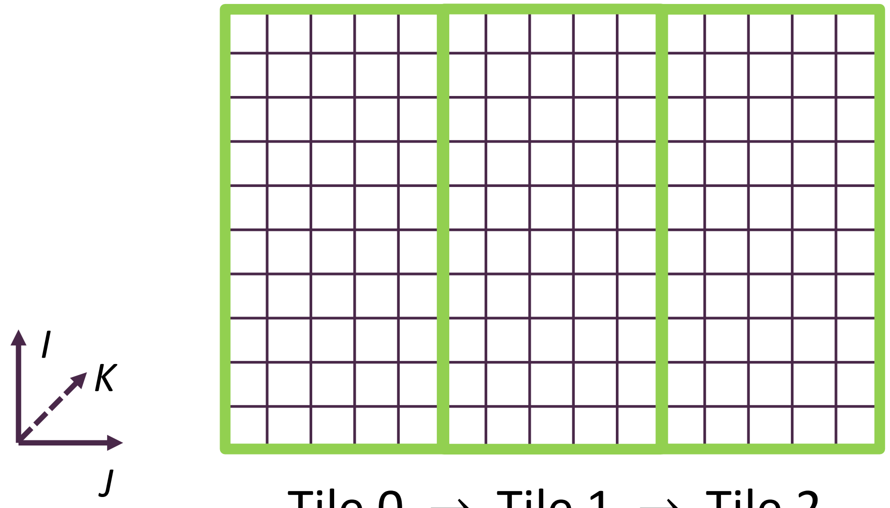


OpenMP *J*-*K* plane area N^2

OpenMP $8 \times 1 \times 1$

Spatial Tiling

- Keep *K* unsplit for SIMD and pre-fetching
- Split *J* to reduce *J*-*K* plane area



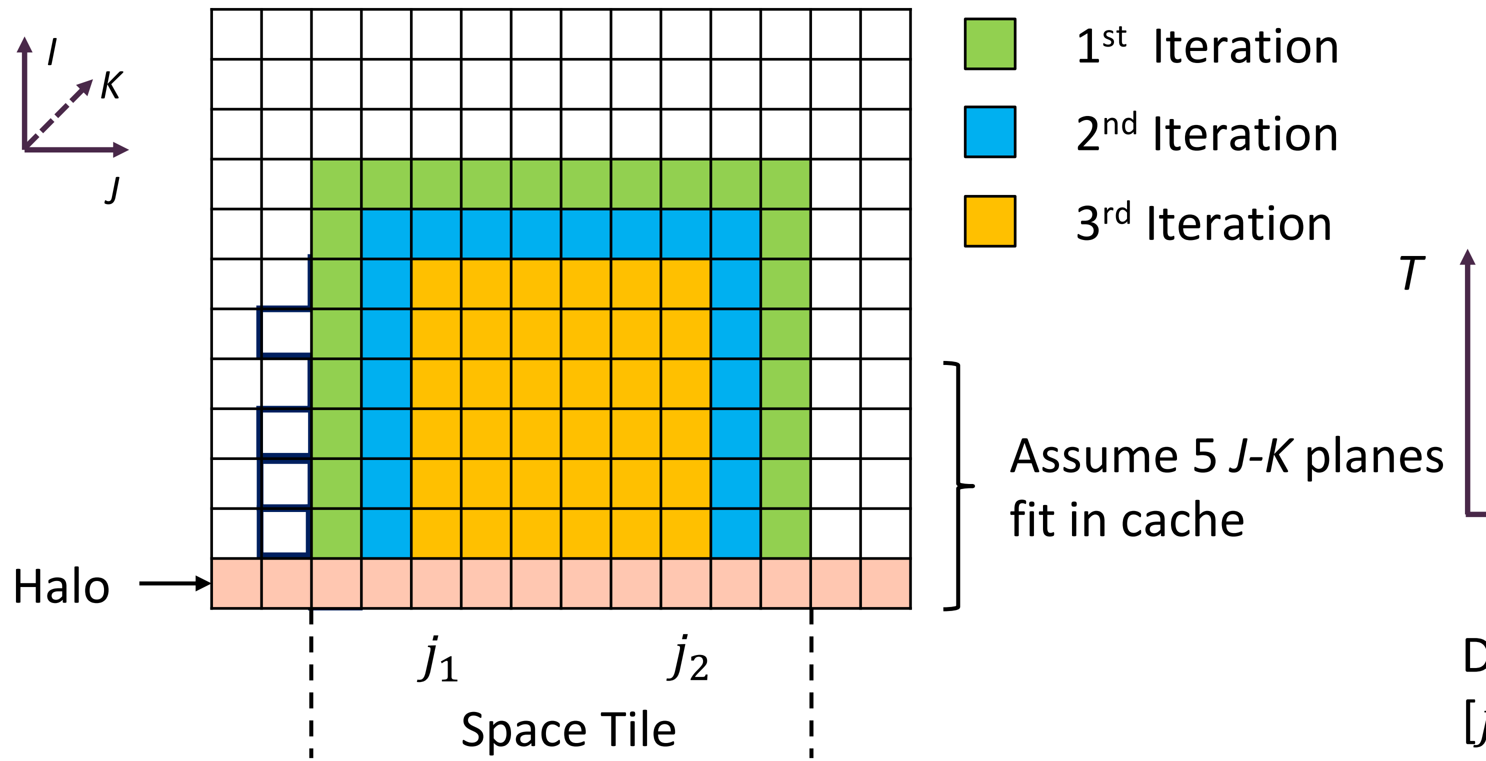
Tile 0 → Tile 1 → Tile 2

- Each *J*-*K* plane is read and written once per iteration
- LLC can hold multiple planes

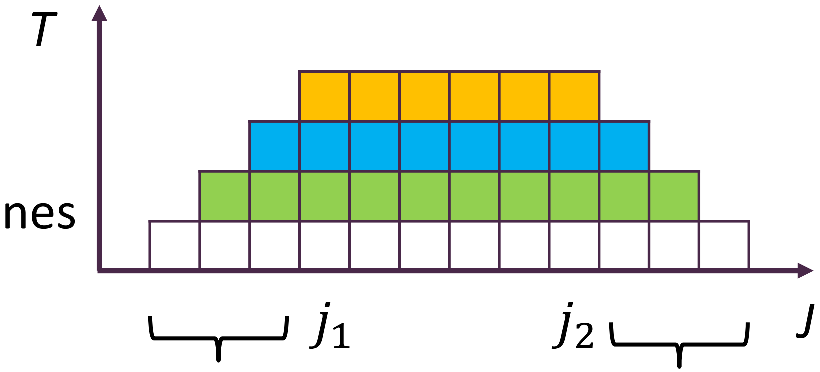
  Fuse iterations → Temporal Tiling

Temporal Tiling

- Fuse iterations in time



Time-Space Tile
Fuse 3 iterations on $[j_1, j_2]$



Dependent data from adjacent ranges $[j_0, j_1)$ and $(j_2, j_3]$

Optimal Combination of MPI, OpenMP, and Temporal Tiling

- Hybrid MPI + OpenMP Tiling:

1. Decompose *K* with MPI processes
   - Based on Cache and Domain sizes
   - ✓ Results in small *J-K* planes and reduces required cache quota
2. Decompose *J* with OpenMP threads
   - ✓ Streaming access in *K* for SIMD and pre-fetching
3. March in *I* with temporal tiling (and pipeline)

Temporal Tiling for Multi-Block Grids

- DeepHalo [Sawdey 1998, Ding 2001, Kjolstad 2010]

Multi-Block

```
for (int t=1; t<NT; ++t) {
  for (int block=0; block<NB; ++block) {
    get_block_size(block, sizes);
    for (int i=0; i<sizes[0]; ++i)
      for (int j=0; j<sizes[1]; ++j)
        for (int k=0; k<sizes[2]; ++k)
          compute_stencil(i, j, k);
  }
  // blocks' connections
  for (int block=0; block<NB; ++block)
    exchange_boundary(block, nHalo);
}
```

Multi-Block with DeepHalo

```
for (int t=1; t<NT; ++t) {
  // fused iteration
  for (int tt=0; tt<tFused; ++tt) {
    for (int block=0; block<NB; ++block) {
      get_block_size(block, sizes);
      for (int i=0; i<sizes[0]; ++i)
        for (int j=0; j<sizes[1]; ++j)
          for (int k=0; k<sizes[2]; ++k)
            compute_stencil(i, j, k);
    }
  }
  // blocks' connections
  for (int block=0; block<NB; ++block)
    exchange_boundary(block, tFused*nHalo);
}
```

- Fuse time iterations for each block
- Fewer data transfers, larger volume per transfer

Overlap Communication and Computation

- Enforce the concurrency of computation and communication

Naive Implementation

```
MPI_Isend(...);
MPI_Irecv(...);
compute_stencil();
MPI_Waitall();
```

✗ No overlap

Dedicated Core (DC)

```
if (thread == 0) {
  MPI_Isend(...);
  MPI_Irecv(...);
  MPI_Waitall();
} else {
  compute_stencil();
}
```

✓ Robust

✗ Use 1 less core for computation

Repeated Poll (RP)

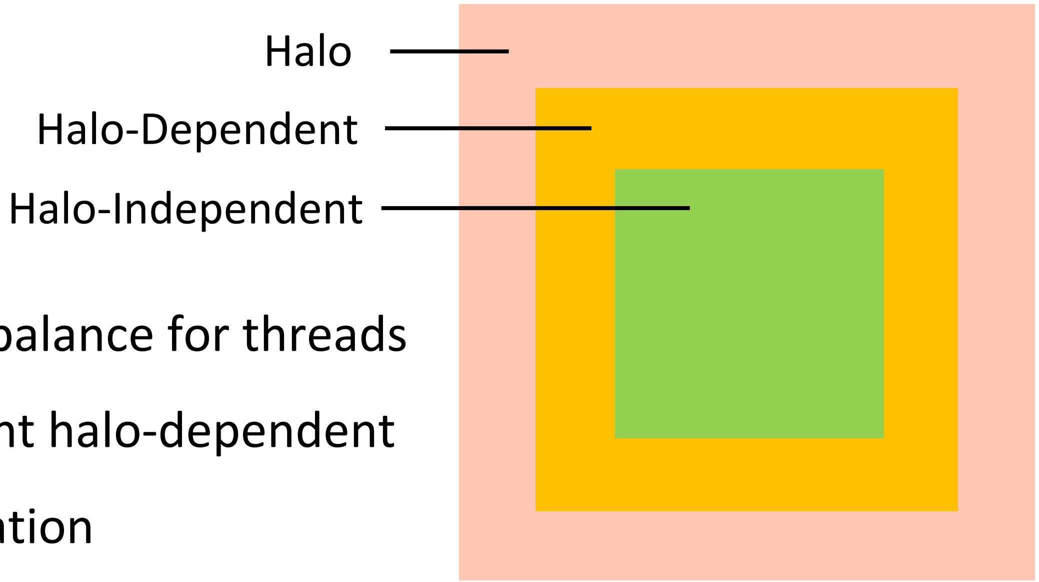
```
MPI_Isend(...);
MPI_Irecv(...);
for (int i=0; i<NI; ++i) {
  for (int j=0; j<NJ; ++j)
    for (int k=0; k<NK; ++k)
      compute_stencil(i, j, k);
   MPI_Test();
}
MPI_Waitall();
```

✓ Use all cores for computation

✗ Network-specific Behavior

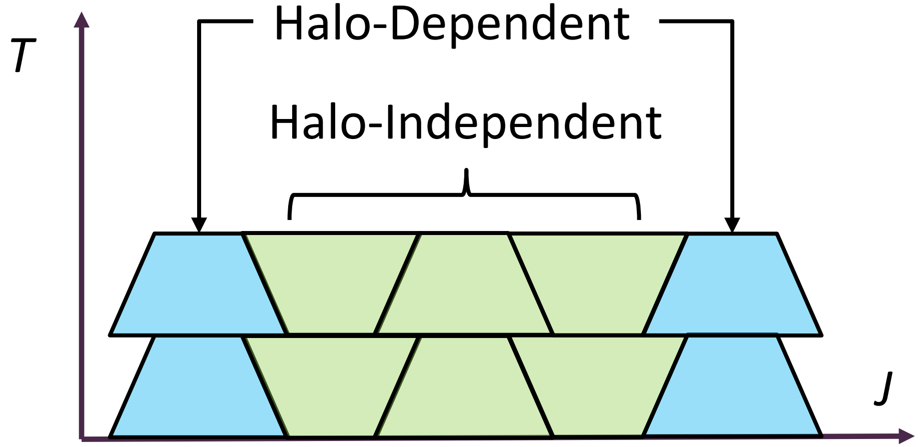
State-of-the-Art for Overlapping Computation and Communication

- Split domain to resolve data dependency
- Divide tiles based on halo dependency



✗ Load imbalance for threads

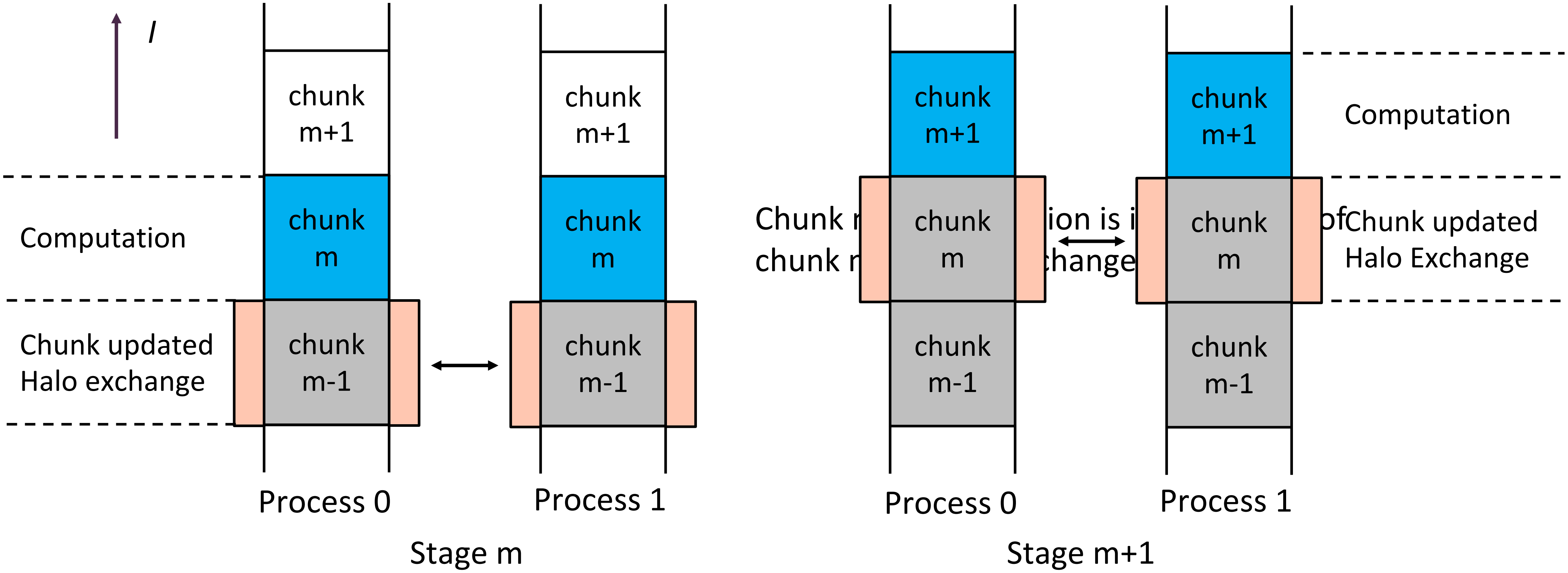
✗ Inefficient halo-dependent computation



✗ Incompatible with domain decomposition for multi-block grids

Pipelining Communication and Computation

- Split the domain into chunks along the *l* dimension
- Domain decomposition can happen in any dimension



Outline

- Motivation and Background
- Grid Partitioner
- Pencil: Pipelined Distributed Stencils
  - Introduction
  - Algorithms
  - **Experiments and Results**
  - Summary
- Deep Learning + CFD
- Summary

Test Platforms

- Pencil is evaluated on two platforms

	Bebop (Argonne)	HPC3 (UCI)
Architecture	Intel Broadwell (Xeon E5-2695v4)	Intel Gold (Xeon 6248)
Sockets	2	2
Cores/Socket	18	20
GFlops/s (DP)	1200	2207
L2 Cache	32 KB	1024 KB
L3 Cache	90 MB	55 MB
Bandwidth	120.3GB/s	194.4GB/s
Network	Omni-Path	InfiniBand
Compiler	Inter 2017	GCC 8.4.0

Test Cases

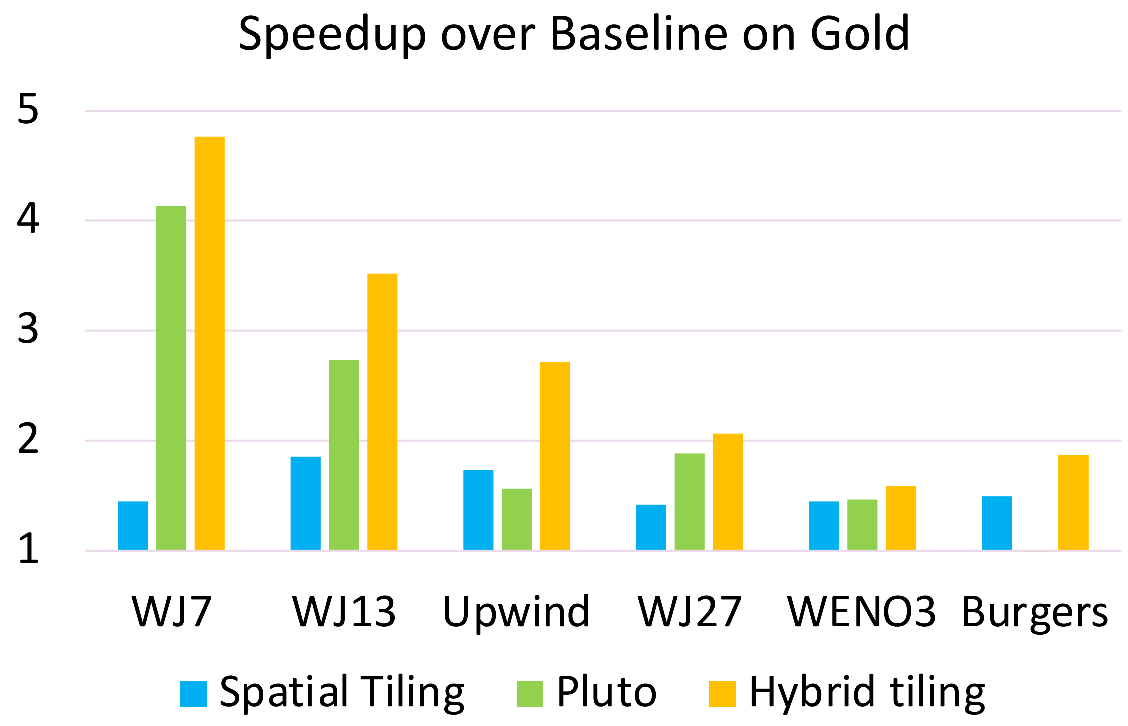
- Pencil is evaluated with six schemes on four stencils

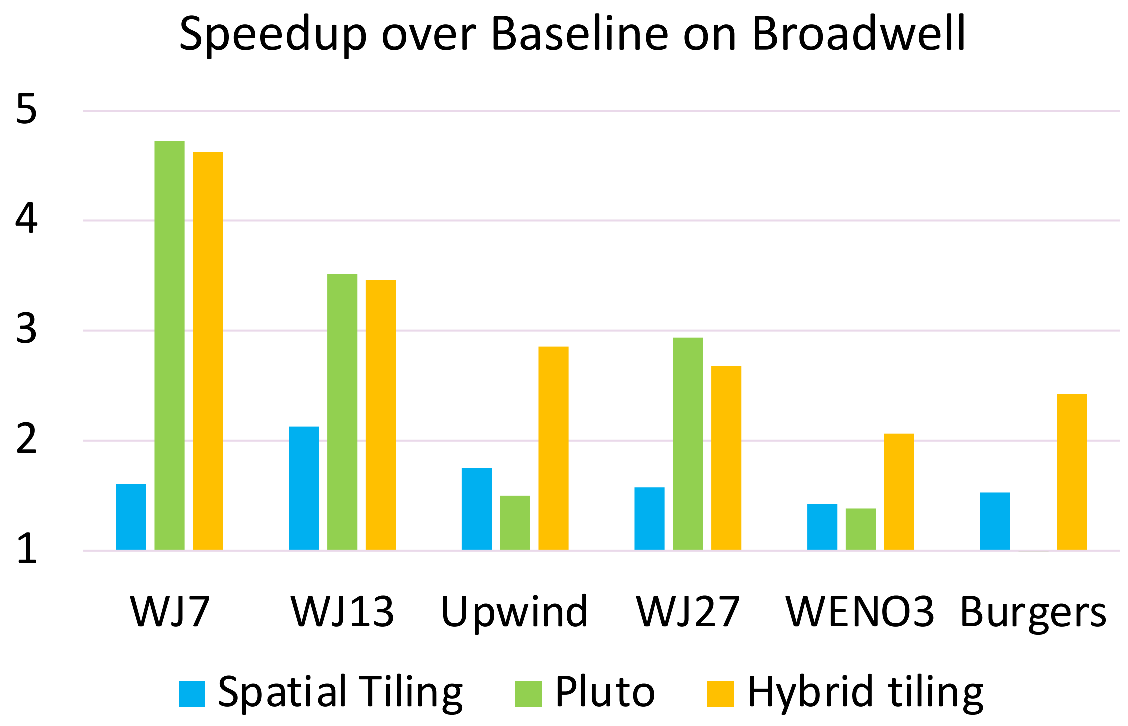
Equation	Schemes	Shape	Radius	AI	#In	#Out
$\nabla^2 p = b$	WJ 7pt	Star	1	0.31	2	1
	WJ 13pt	Star	2	0.5	2	1
	WJ 27pt	Box	1	0.94	2	1
$\partial_t \phi + \vec{u} \cdot \nabla \phi = 0$	Upwind	Star	2	0.71	4	1
	WENO3	Star	2	1.64	4	1
$\partial_t \vec{u} + \nabla \cdot (\vec{u}\,\vec{u}) = \nu \Delta \vec{u}$	Burgers-CD	Staggered	1	1.67	3	3

WJ: Weighted Jacobi; CD: Central Difference

Single Node Performance

- Domain size 480^3, use OpenMP without cache tiling as the baseline
- Up to 3.29x over spatial tiling and 1.9x over Pluto for complex schemes
- Compare with spatial tiling, Pluto (diamond tiling) [Bondhugula 2008, Bondhugula 2014]
- Temporal tiling should be evaluated against spatial tiling rather than baseline



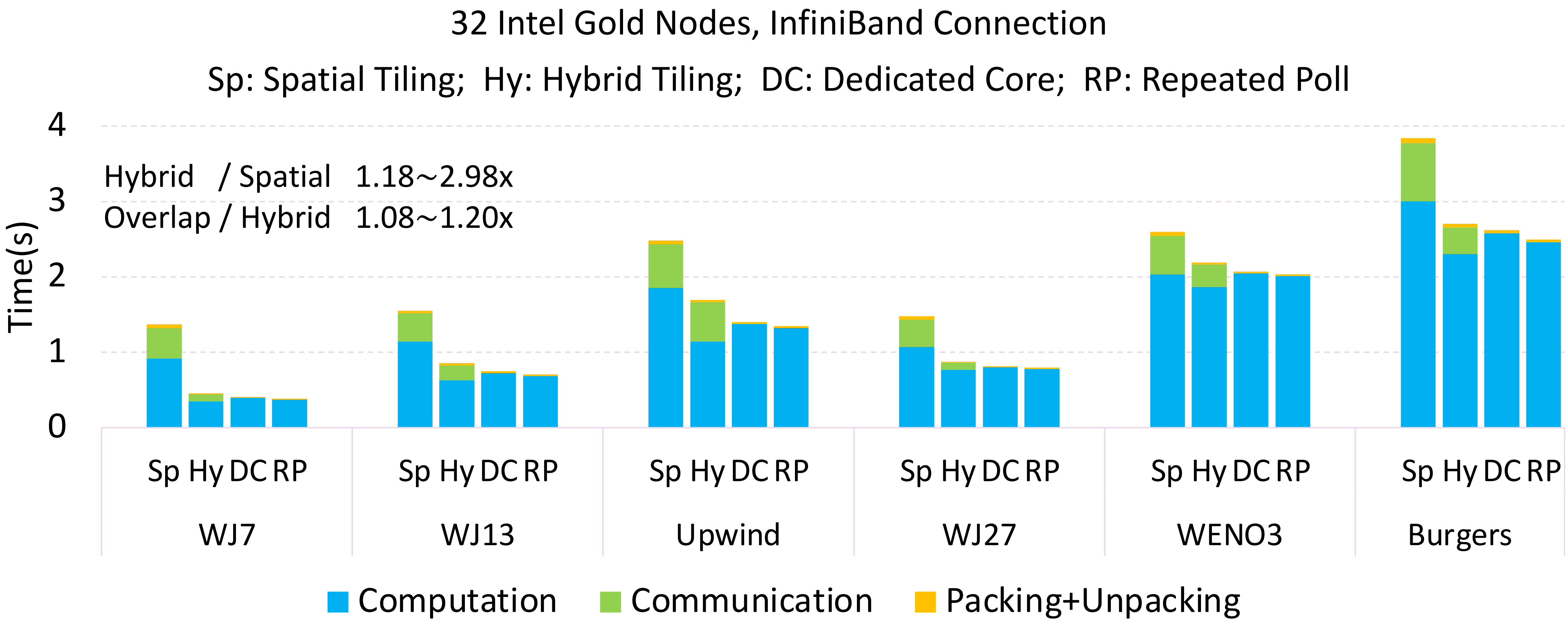


Test Setup for Pipelining Communication and Computation

- Pencil is evaluated on 32 nodes connected by InfiniBand (HPC3) or Omni-Path (Bebop)
- DeepHalo + Hybrid Tiling
- Load balance for computation and communication:
  - One block of size 480^3 per node
  - Periodic boundary conditions for all blocks

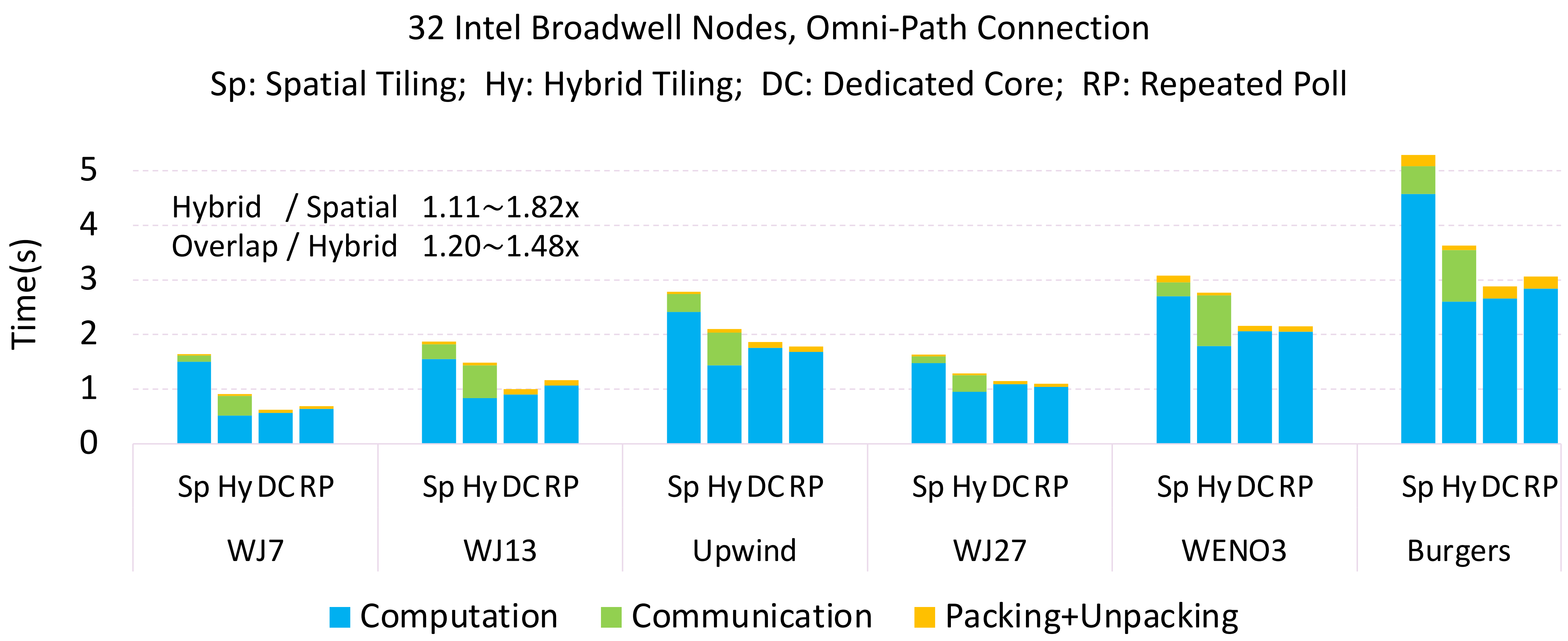
Pipelining Communication and Computation

- DeepHalo reduces communication time



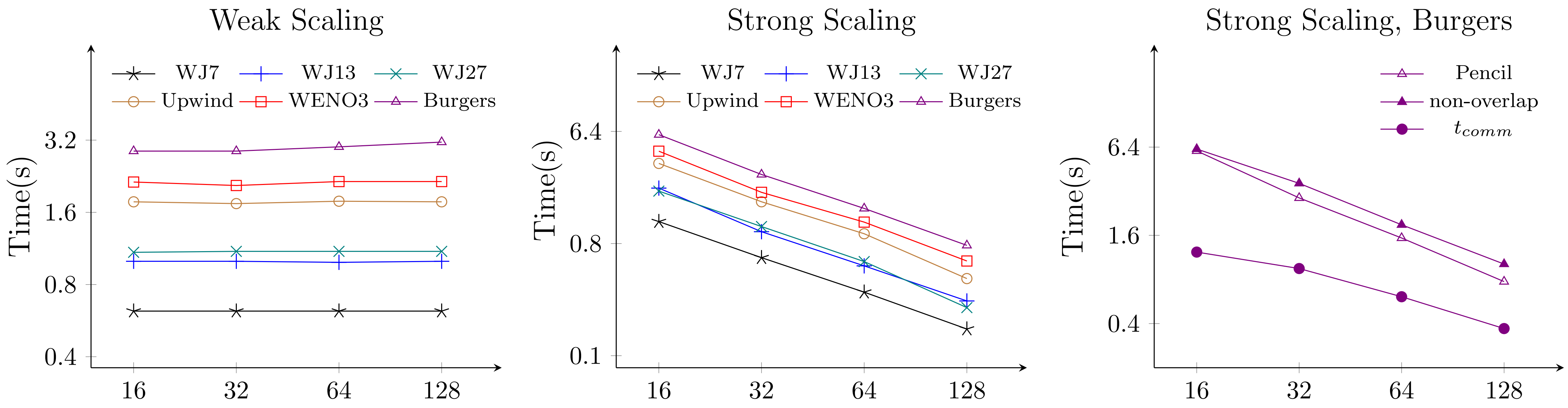
Pipelining Communication and Computation

- Overlap significantly improves performance when the communication is slow
- DeepHalo's effect on communication is network-specific



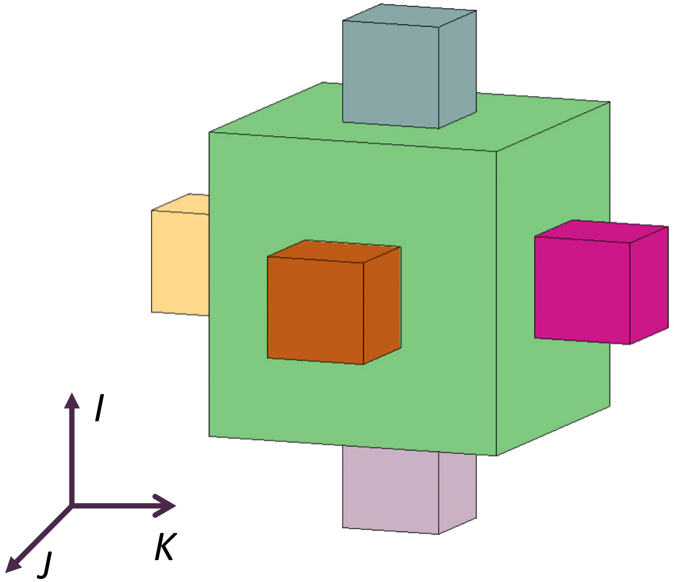
Scalability Test on Bebop 16 ~ 128 nodes

- Weak Scalability: 480^3 per node, periodic boundary conditions
- Strong Scalability: $1440 \times 1440 \times 960$ cells in total, periodic boundary conditions
- Overlap improves the strong scalability by hiding the communication cost

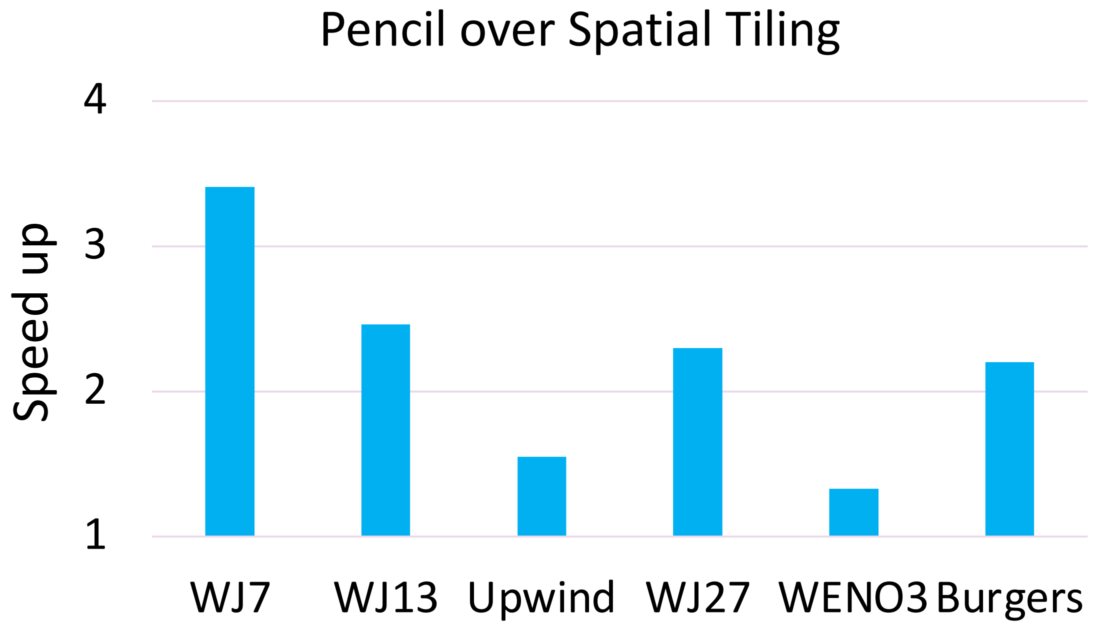


Multi-Block Grid Test

- Pencil is evaluated with a 6-block grid on 32 Broadwell nodes
- 1.33~3.41x over MPI + OpenMP with spatial tiling



3.5×10^9 cells in total



Summary

Find the optimal combination of MPI, OpenMP and temporal tiling

- Decompose K with MPI processes
- Decompose J with OpenMP threads
- March in I with temporal tiling and pipelining
- Evaluated by 6 schemes on 4 stencils, up to 1.9x over Pluto for complex schemes

Apply temporal tiling to multi-block grids via DeepHalo

- DeepHalo's effect on communication cost is network-specific.

Pipeline communication and computation to achieve overlap

- 1.3~3.4x over spatial tiling with flat MPI and MPI + OpenMP
- Improve strong scalability by hiding the communication cost

Outline

- Motivation and Background
- Grid Partitioner
- Pencil: Pipelined Distributed Stencils
- **Deep Learning + CFD**

  In collaboration with Prof. Ramin Bostanabad at UCI
- Summary

Introduction

- Neural Networks vs Conventional CFD methods

$$\frac{\partial}{\partial_t}\int_{\Omega} \vec{W}\, d\Omega + \oint_{\partial\Omega}(\vec{F_c} - \vec{F_v})\, dS = \int_{\Omega} \vec{Q}\, d\Omega$$

Boundary condition
Initial condition → Finite Volume, Finite Difference
Runge-Kutta, Multigird, ... → Flow Solution

Hours ~ Days

≪ Hour

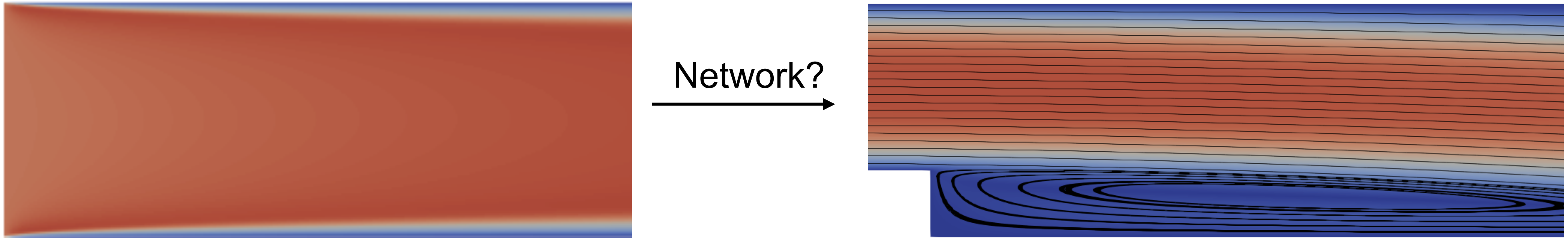
Flow Snapshots → Neural Networks
FF, CNN, RNN, ... → Flow Solution

Training Hours ~ Days

Advantages and Disadvantages of Using Neural Networks

- Neural Networks vs Conventional CFD methods
  - ✓ Improve performance when the networks are being re-used
  - ✗ Problem-Specific, i.e., unable to predict flows unseen in training

⟶ Generalize the network to geometries unseen in training



Channel Flow

Backward Facing Step

Preliminary Results

- Solve a 2D Homogeneous Poisson Equation $\nabla^2 p = 0$
- Train the network:
  - Finite Difference + Multigrid $\rightarrow$ sample solutions p^*
  - Regularizing the PDE error in loss function:

$$loss = \frac{1}{N}\left(\sum (p - p*)^2 + \alpha \sum (\nabla^2 p)^2\right)$$

Preliminary Results

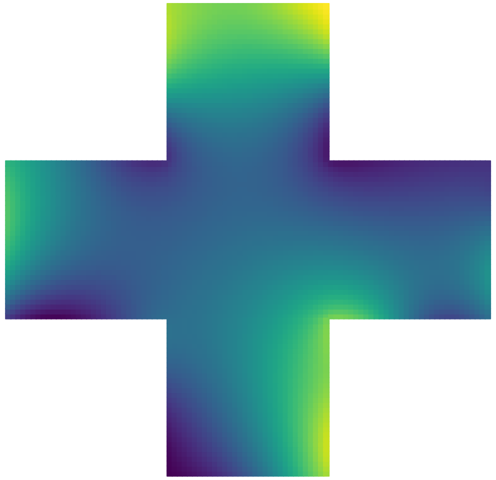
- Predict for domains of different shapes

Geometries in Training

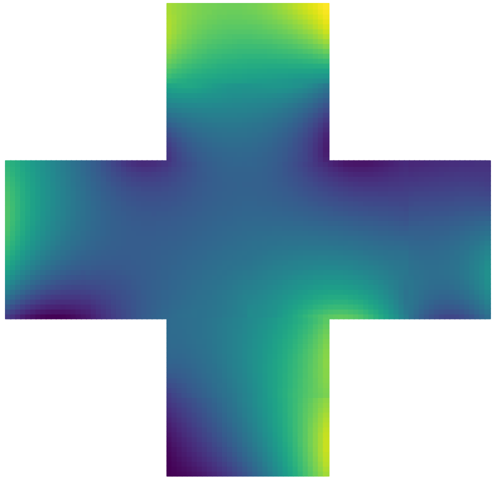
Ground Truth

Prediction
Relative MAE ~4%

Geometries Unseen in Training



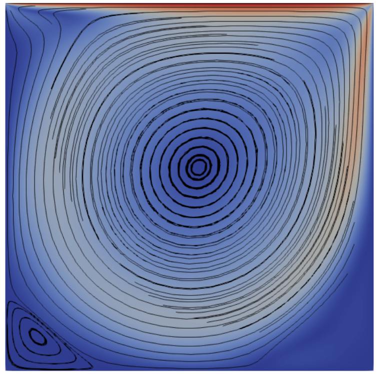
Ground Truth



Prediction
Relative MAE ~6%

Next Steps Towards Solving Naiver-Stokes Equation

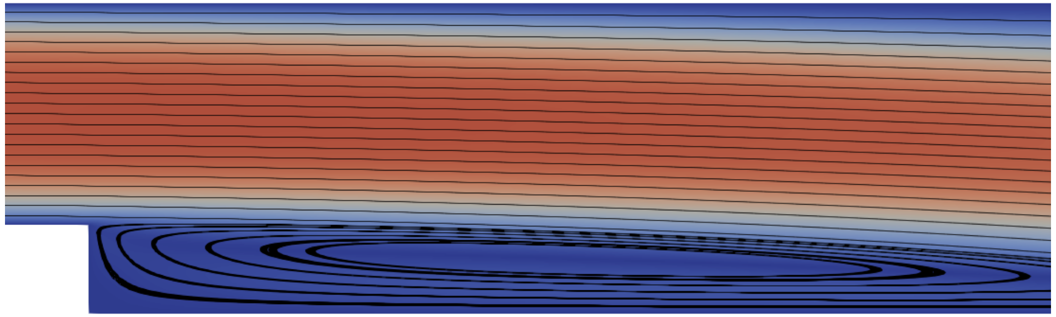
- Train networks with simple cases and predict for more complex flows



Lid-Driven Cavity

+

Channel Flow



Backward Facing Step

Outline

- Motivation and Background
- Grid Partitioner
- Pencil: Pipelined Distributed Stencils
- Deep Learning + CFD
- **Summary**

Summary

- Structured Gird Partitioner (*ICS 19*)
  - New cost function unifying algorithm factors and networks specifics
  - Novel partition algorithms
- Pencil: A Pipelined Algorithm for Distributed Stencils (*SC20*)
  - Identify the optimal combination of MPI, OpenMP, and Temporal tiling
  - Applicable to Multi-Block grid
  - Pipeline communication and computation to achieve overlap
- Deep Learning + CFD (*ongoing*)
  - Generalize the network to geometries unseen in training