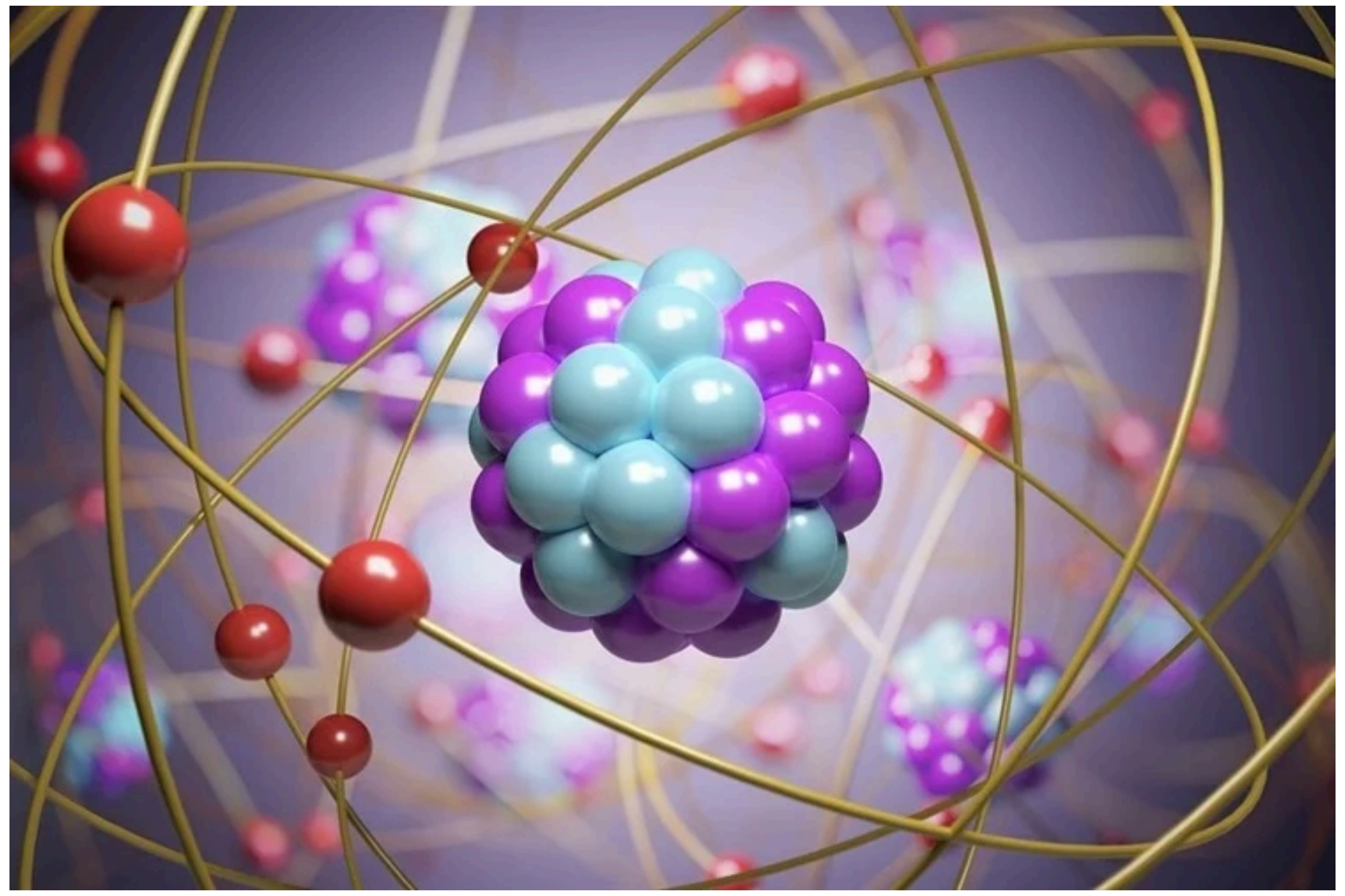
DISTAL: Compiling Tensor Algebra to Distributed Machines

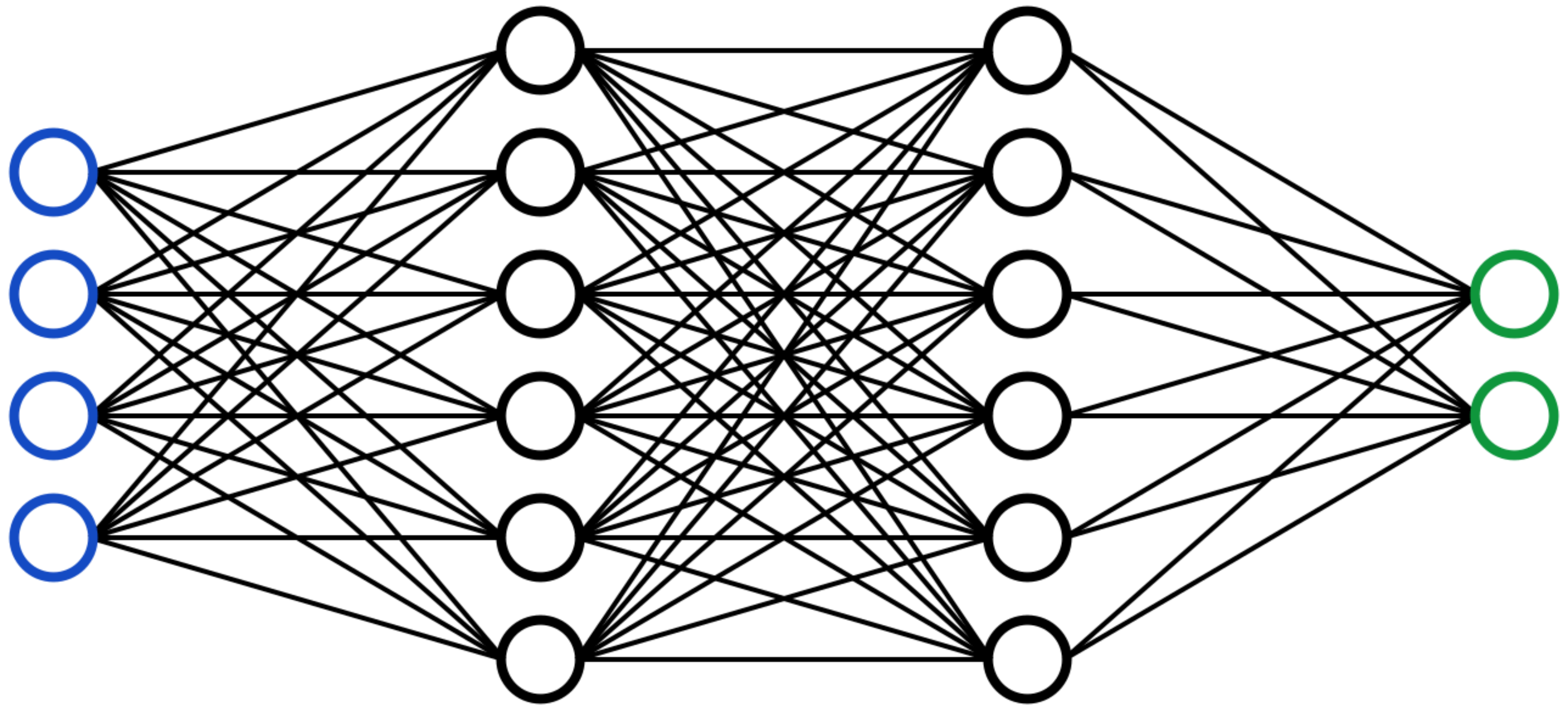
Rohan Yadav, Fred Kjolstad, Alex Aiken

$$A(i, j) = B(i, j, k) \cdot c(k) \longrightarrow$$

Tensor computations are ubiquitous



Scientific Computing



Machine Learning

Data Analytics

Key Challenge:

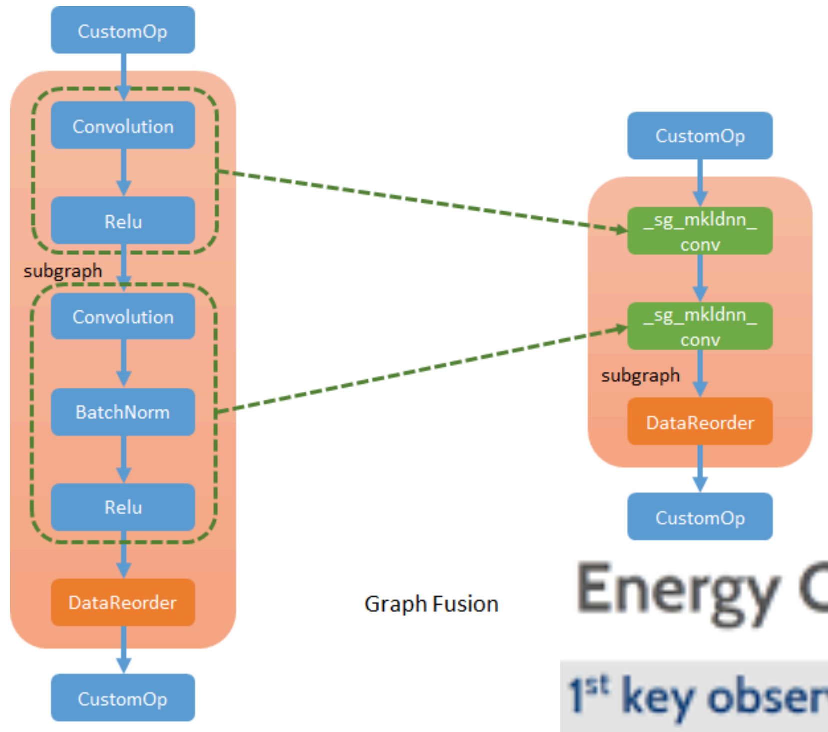
Correctness   Productivity   Performance

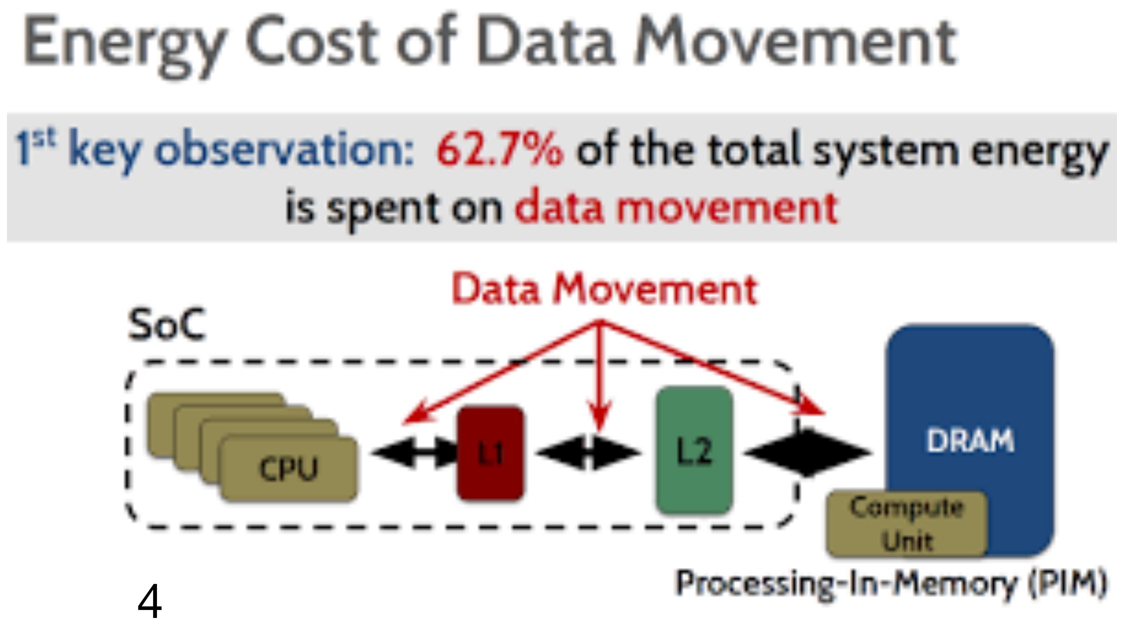
Why is achieving good performance hard?

Optimizations are intertwined with correctness

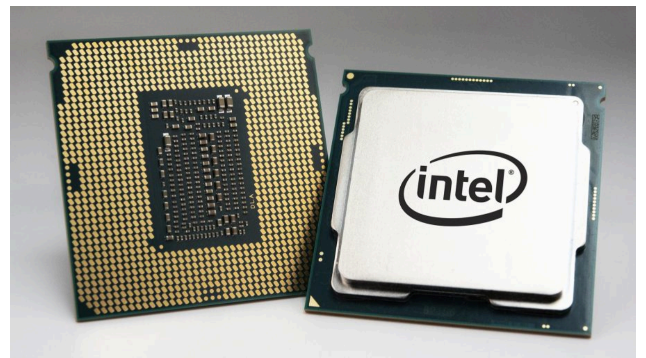
```c
int n5;
__m512d x05, x15, x25, x35;
__m512d __alpha5;
n5 = n & ~7;

x05 = _mm512_broadcastsd_pd(_mm_load_sd(&x[0]));
x15 = _mm512_broadcastsd_pd(_mm_load_sd(&x[1]));
x25 = _mm512_broadcastsd_pd(_mm_load_sd(&x[2]));
x35 = _mm512_broadcastsd_pd(_mm_load_sd(&x[3]));

__alpha5 = _mm512_broadcastsd_pd(_mm_load_sd(alpha));

for (; i < n5; i+= 8) {
  __m512d tempY;
  __m512d sum;

  sum = _mm512_loadu_pd(&ap[0][i]) * x05 +
        _mm512_loadu_pd(&ap[1][i]) * x15 +
        _mm512_loadu_pd(&ap[2][i]) * x25 +
        _mm512_loadu_pd(&ap[3][i]) * x35;

  tempY = _mm512_loadu_pd(&y[i]);
  tempY += sum *  __alpha5;
  _mm512_storeu_pd(&y[i], tempY);
}
```

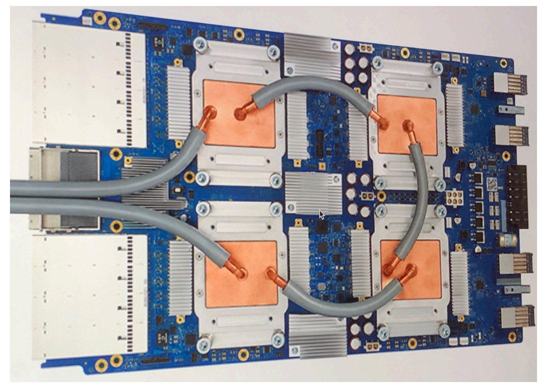
Performance doesn't compose





Variability of target architectures





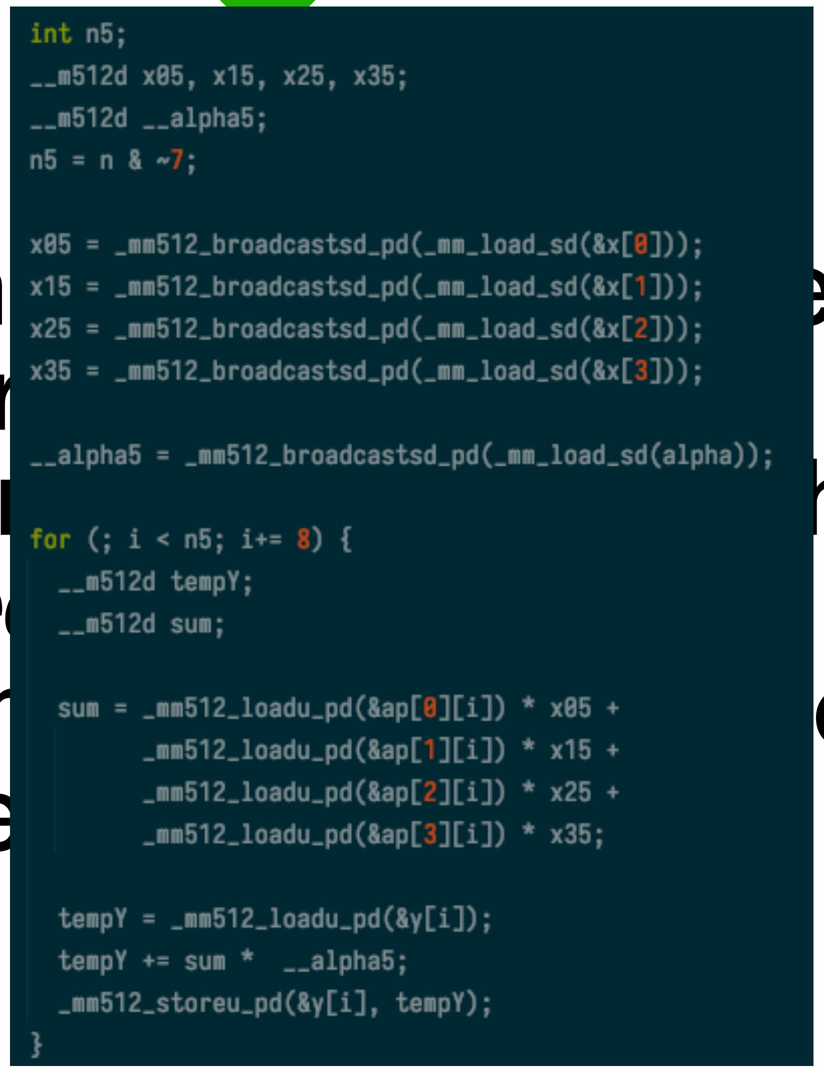
DSLs and compilation-based approaches have been successful

Halide, TVM, TACO (and many more!)

How can compilers help?

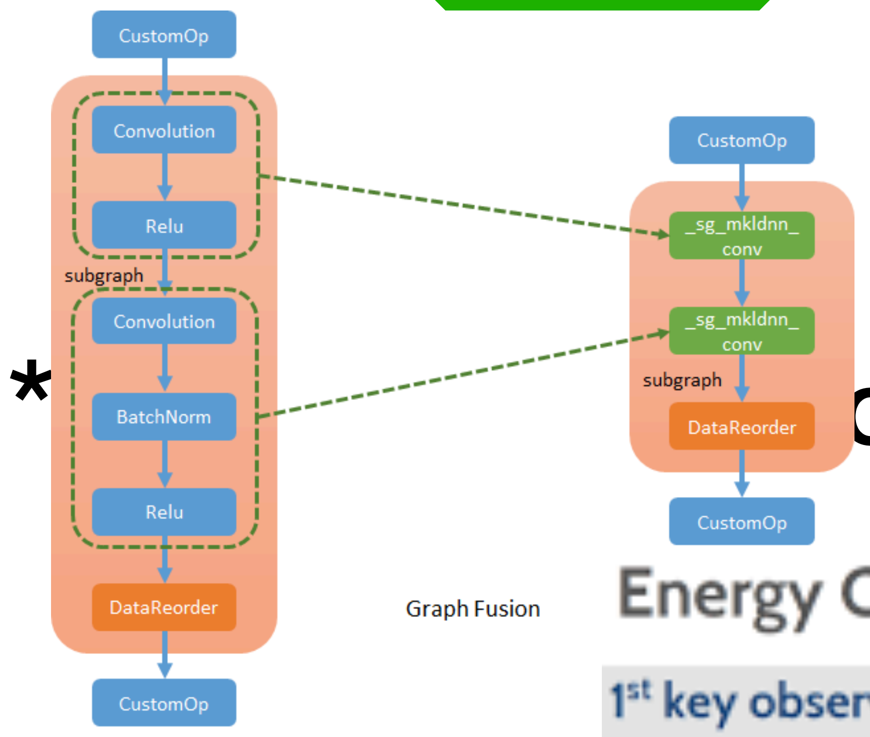
Optimizatio are intertwi d with c tness

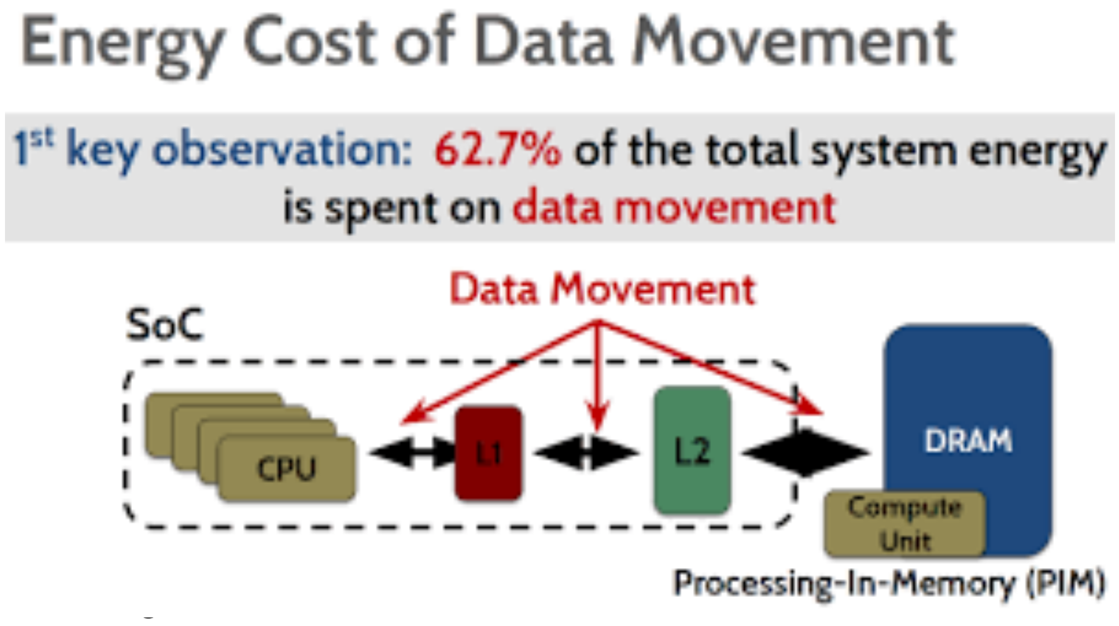
* Com ed abstr
* Algo hrough *sche*
* Opti e corre



Perform ce do 't ompose

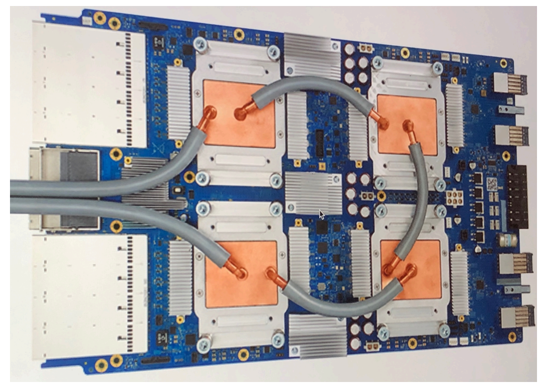
* optimizations



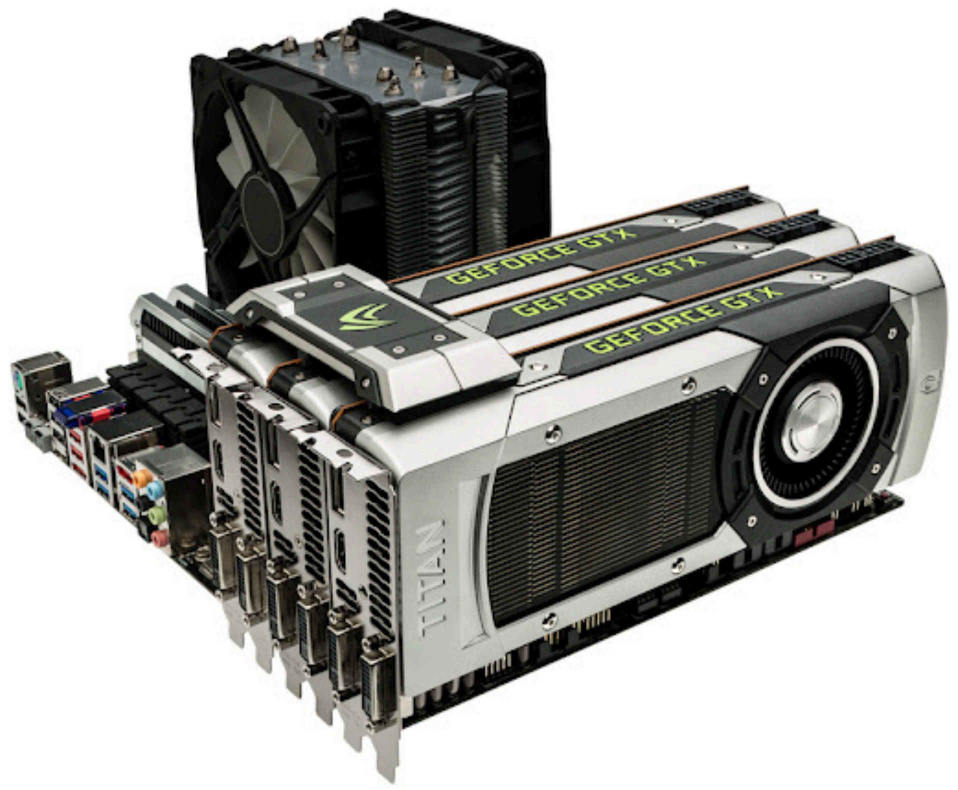


Variability c arget a ures

* eeds for each b



What about distributed systems?



Distributed systems exacerbate prior problems. We need compilers for this case too!

Optimizations are intertwined with correctness

* Inter-address space communication

Performance doesn't compose

* Expensive data movement
* Reorganize data to fit library interfaces

Variability of target architectures

* Machines are heterogenous

What are the right abstractions for distributed tensor compilation?

Simpler for end users

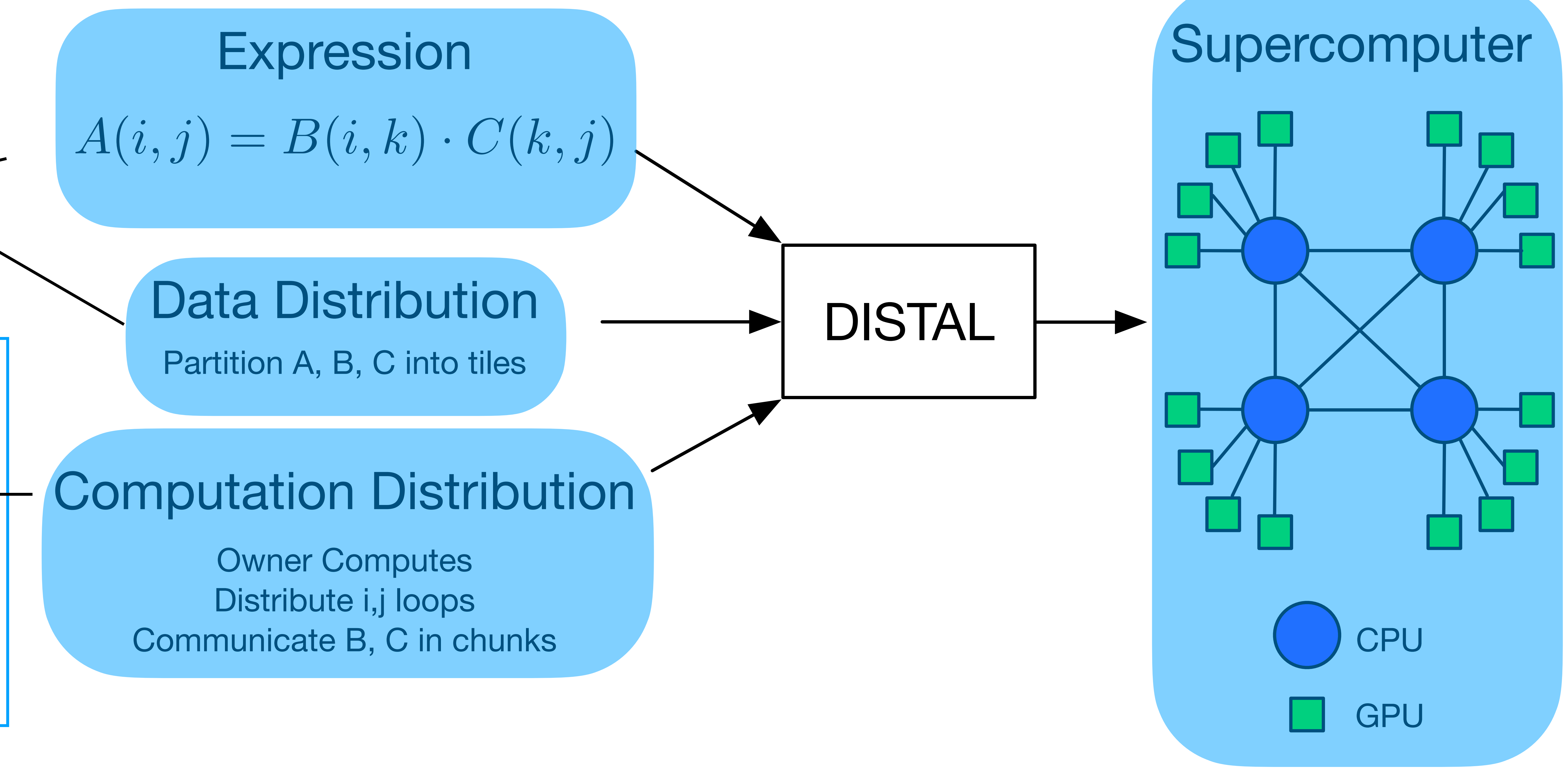
Capture existing algorithms

Generalize to all tensor programs

DISTAL: The Distributed Tensor Algebra Compiler

Decouple computation, performance optimizations, and data distribution

```
Param gx, gy, n;
Machine m(Grid(gx, gy));

Distribution tiles(m, {0, 1});
Format f({dense, dense}, tiles);
Tensor<double> a({n, n}, f), b({n, n}, f), c({n, n}, f);

IndexVar i, j, k;
a(i, j) = b(i, k) * c(k, j);

IndexVar in, jn, il, jl, ko, ki;
a.schedule()
 .divide(i, in, il, m.x).divide(j, jn, jl, m.y).divide(k, ko, ki, m.x)
 .reorder({in, jn, il, jl})
 .distribute({in, jn}, DistributedGPU)
 .reorder({ko, il, il, ki})
 .communicate(a, jn).communicate({b, c}, ko)
 .substitute({il, jl, ki}, CuBLAS::GeMM)
 ;

a.compile();
```



Modeling Machines

Expression

$$A(i,j) = B(i,k) \cdot C(k,j)$$

$$A(i,l) = B(i,j,k) \cdot C(j,l) \cdot D(k,l)$$

$$a = B(i,j,k) \cdot C(i,j,k)$$

$$A(i,j,l) = B(i,j,k) \cdot C(k,l)$$

$$A(i,j) = B(i,j,k) \cdot c(k)$$

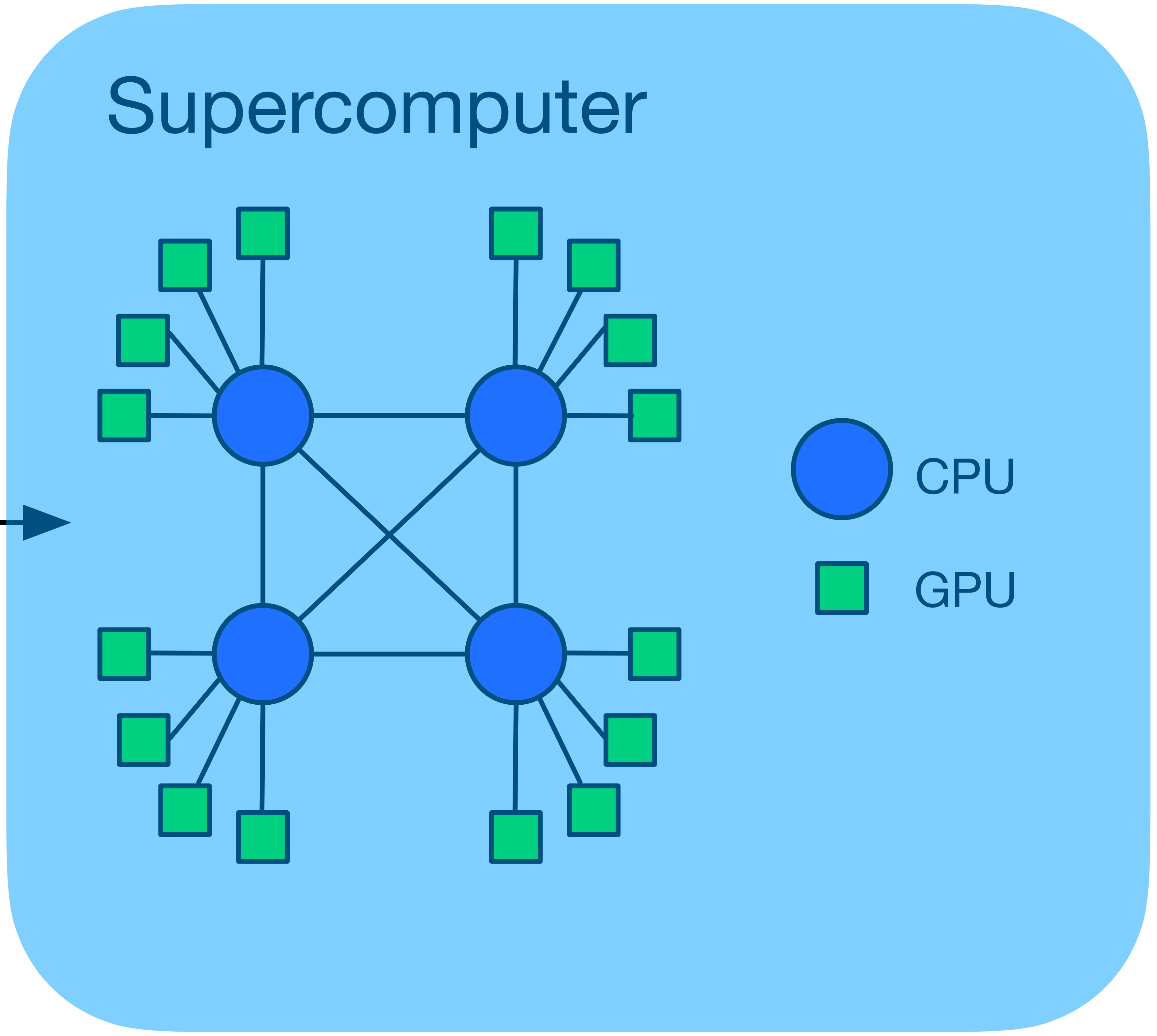
Data Distribution

Partition A into tiles
Replicate B onto all nodes
Place C onto only some nodes

Computation Distribution

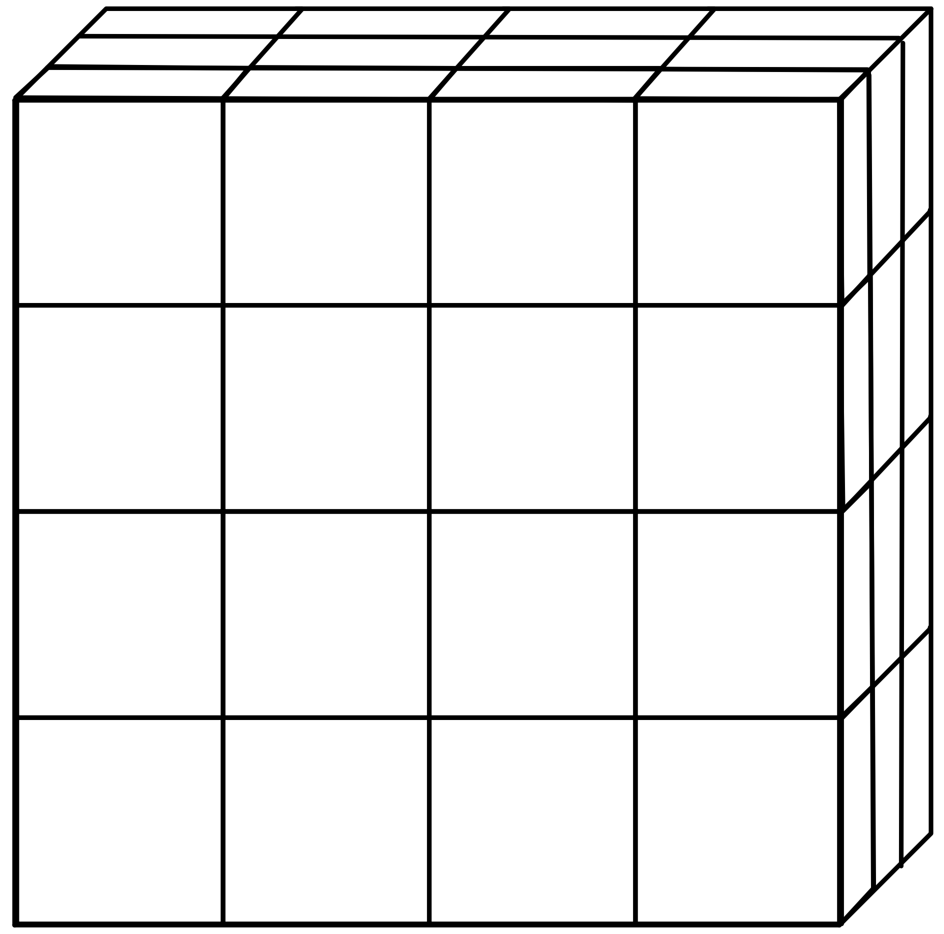
Owner Computes
Distribute i,j loops
Communicate in chunks

DISTAL



View machines as hyper-rectangular grids of processors, where each processor has a local memory

- Expose locality in the physical machine
- Structure machine like target computations



Distributing Data

Expression

$$A(i,j) = B(i,k) \cdot C(k,j)$$

$$A(i,l) = B(i,j,k) \cdot C(j,l) \cdot D(k,l)$$

$$a = B(i,j,k) \cdot C(i,j,k)$$

$$A(i,j,l) = B(i,j,k) \cdot C(k,l)$$

$$A(i,j) = B(i,j,k) \cdot c(k)$$

Data Distribution

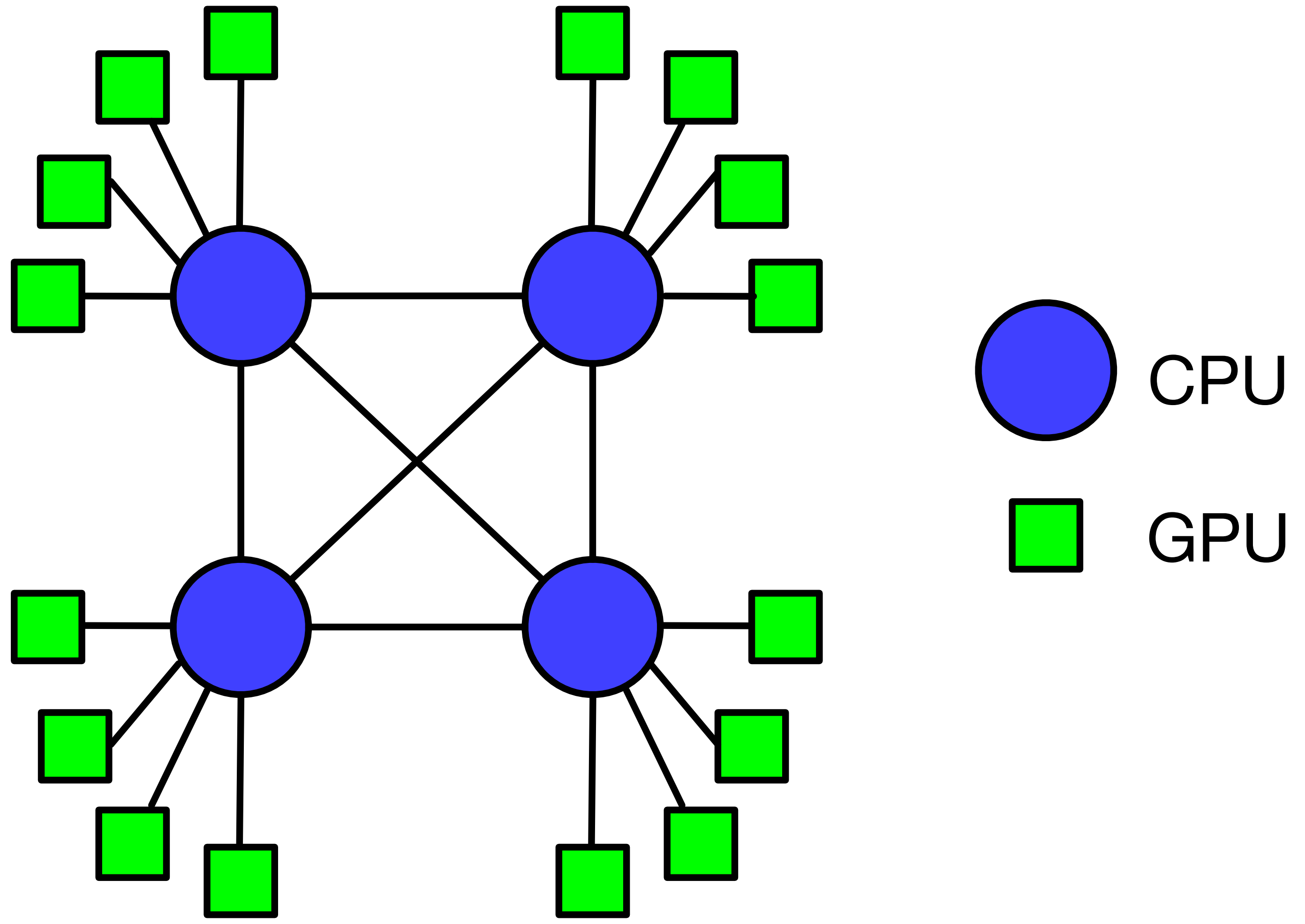
Partition A into tiles
Replicate B onto all nodes
Place C onto only some nodes

Computation Distribution

Owner Computes
Distribute i,j loops
Communicate in chunks

DISTAL → Supercomputer

Supercomputer



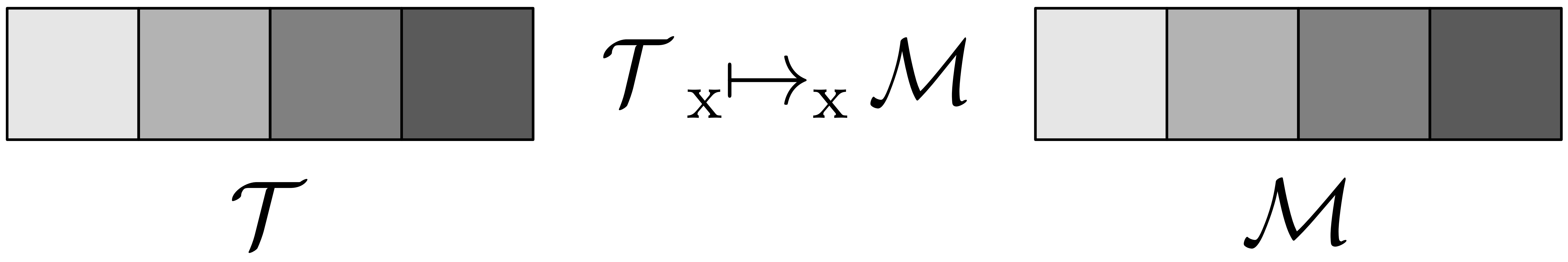
Tensor Distribution Notation

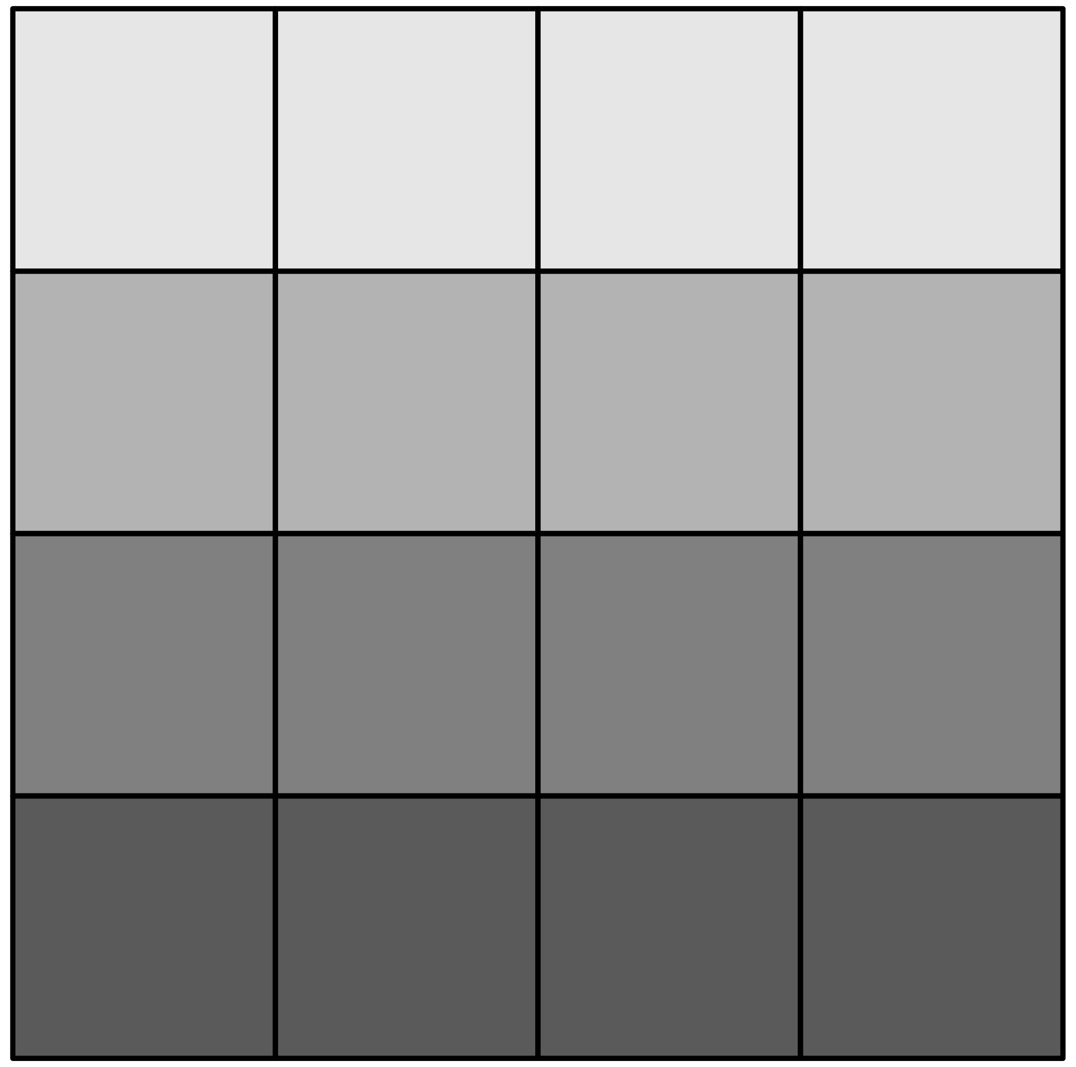
Describe how dimensions of a tensor $\mathcal{T}$ map onto a machine $\mathcal{M}$

$$\mathcal{T}_{\mathrm{xy}} \mapsto_{\mathrm{x}} \mathcal{M}$$

Name each dimension of $\mathcal{T}$ and $\mathcal{M}$

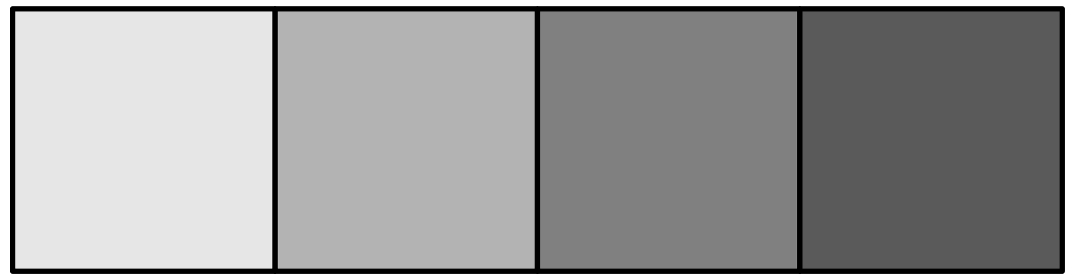
Dimensions of $\mathcal{T}$ are partitioned by dimensions of $\mathcal{M}$ with the same name


$\mathcal{T} \; {}_{x}{\mapsto}_{x} \; \mathcal{M}$
$\mathcal{T}$
$\mathcal{M}$

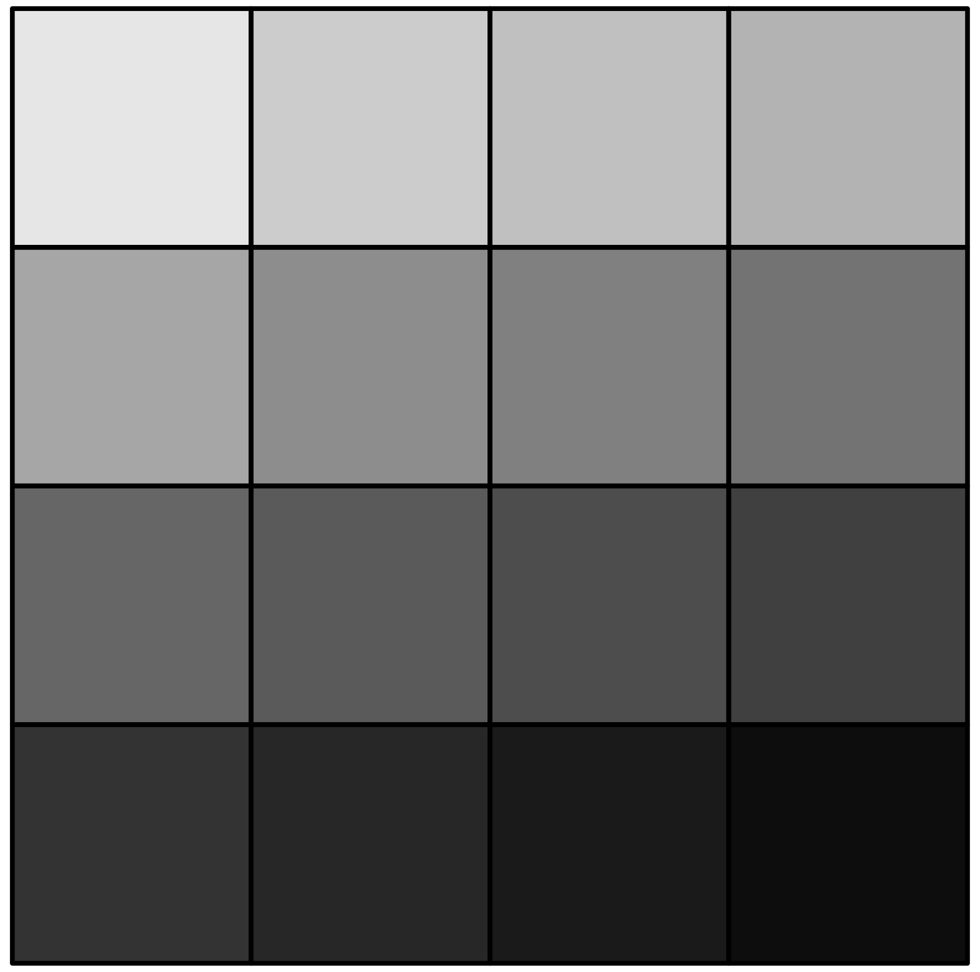


$\mathcal{T}$

$$\mathcal{T}\ {}_{\mathrm{xy}}\!\mapsto_{\mathrm{x}} \mathcal{M}$$

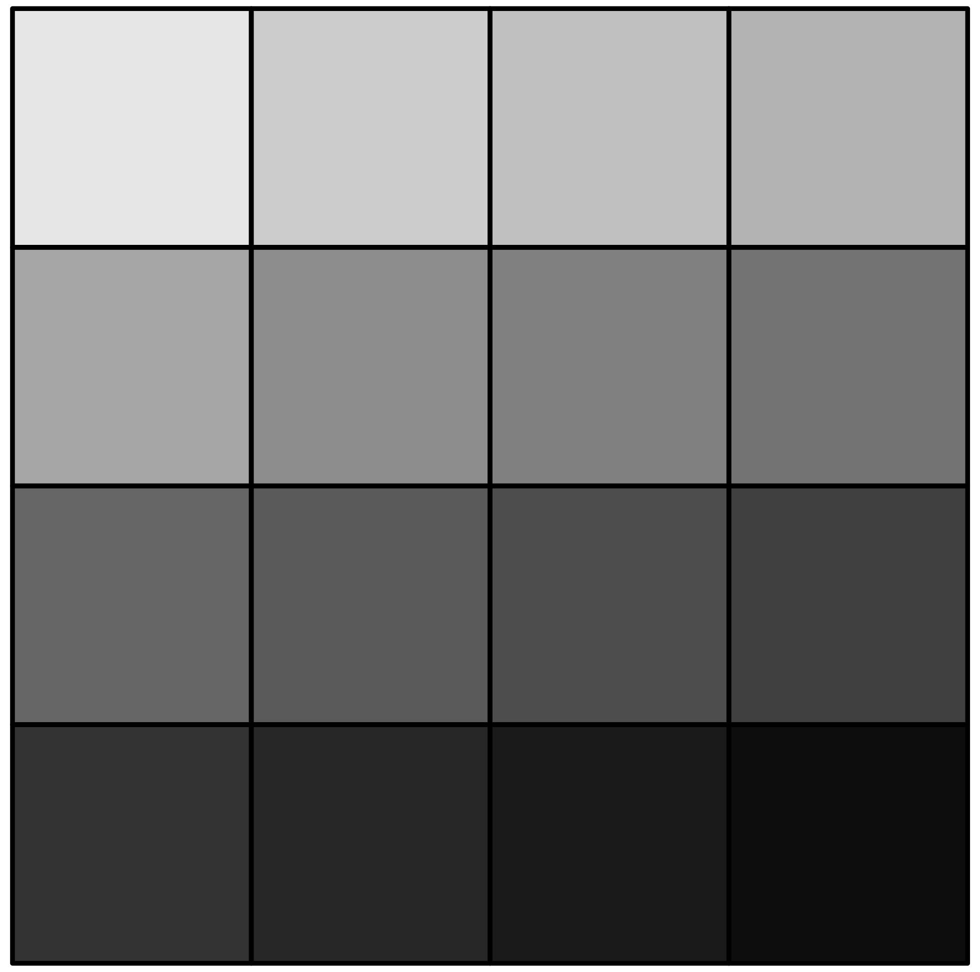


$\mathcal{M}$

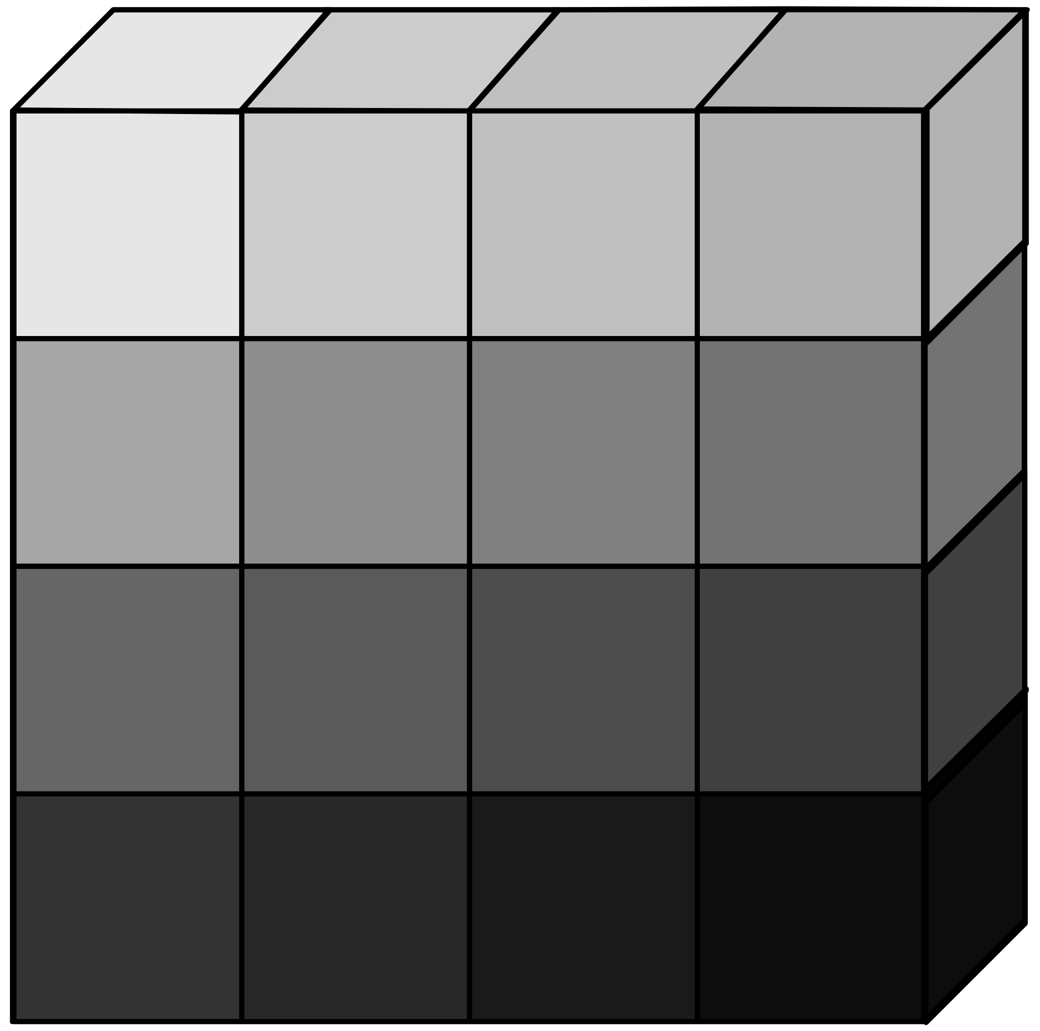


$\mathcal{T}$

$$\mathcal{T}\ {}_{\mathrm{xy}}\!\mapsto_{\mathrm{xy}}\ \mathcal{M}$$

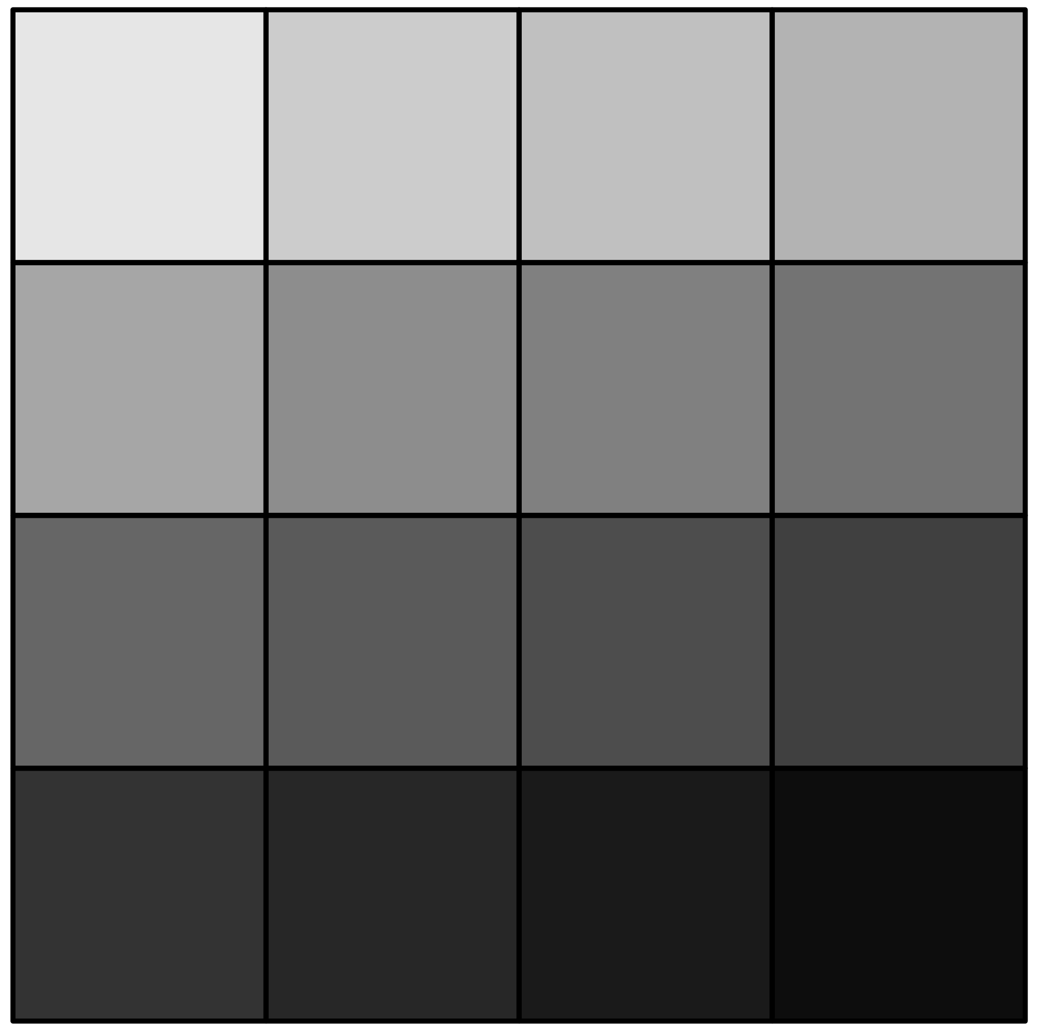


$\mathcal{M}$

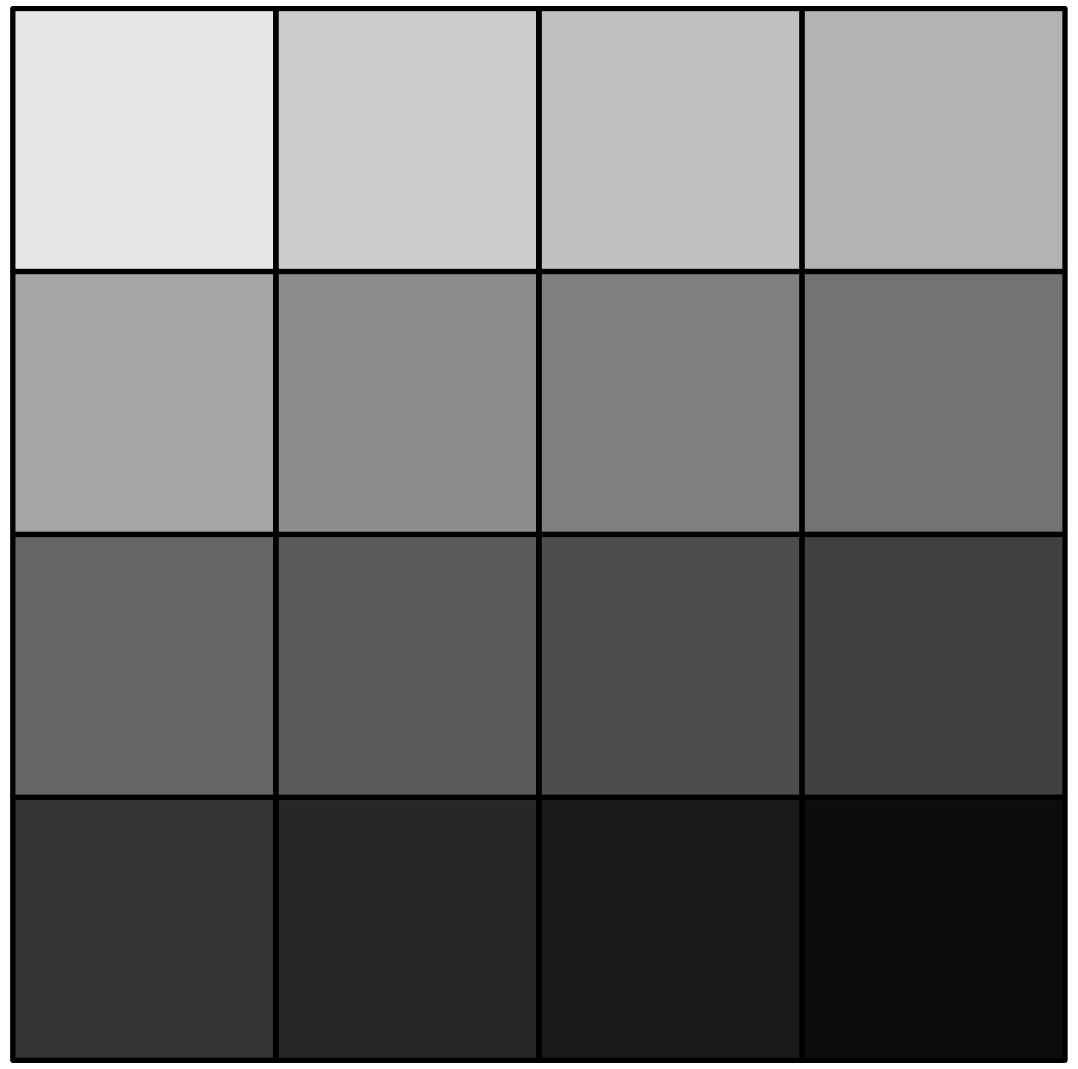


$\mathcal{T}$

$$\mathcal{T}_{\mathrm{xyz} \mapsto \mathrm{xy}} \mathcal{M}$$

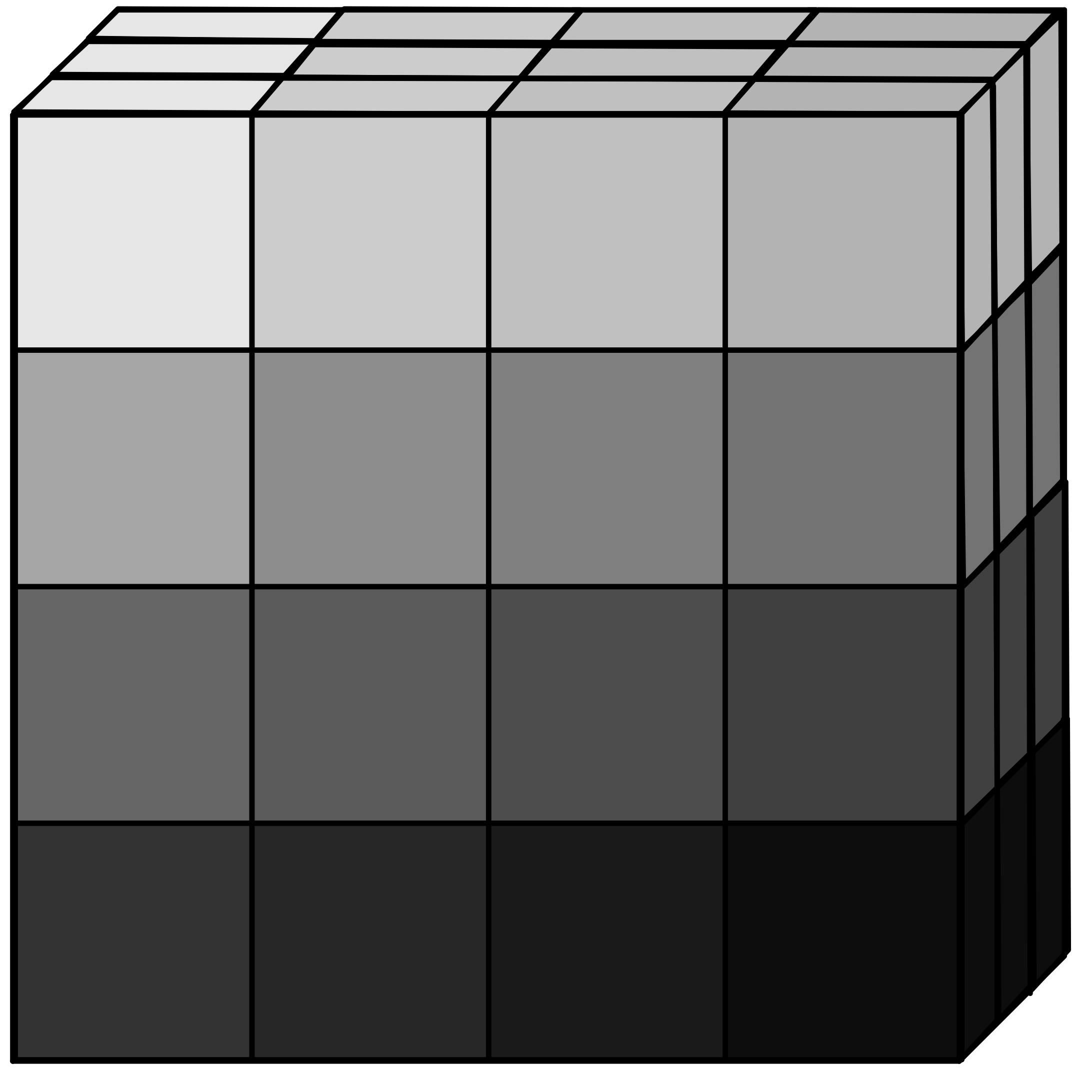


$\mathcal{M}$

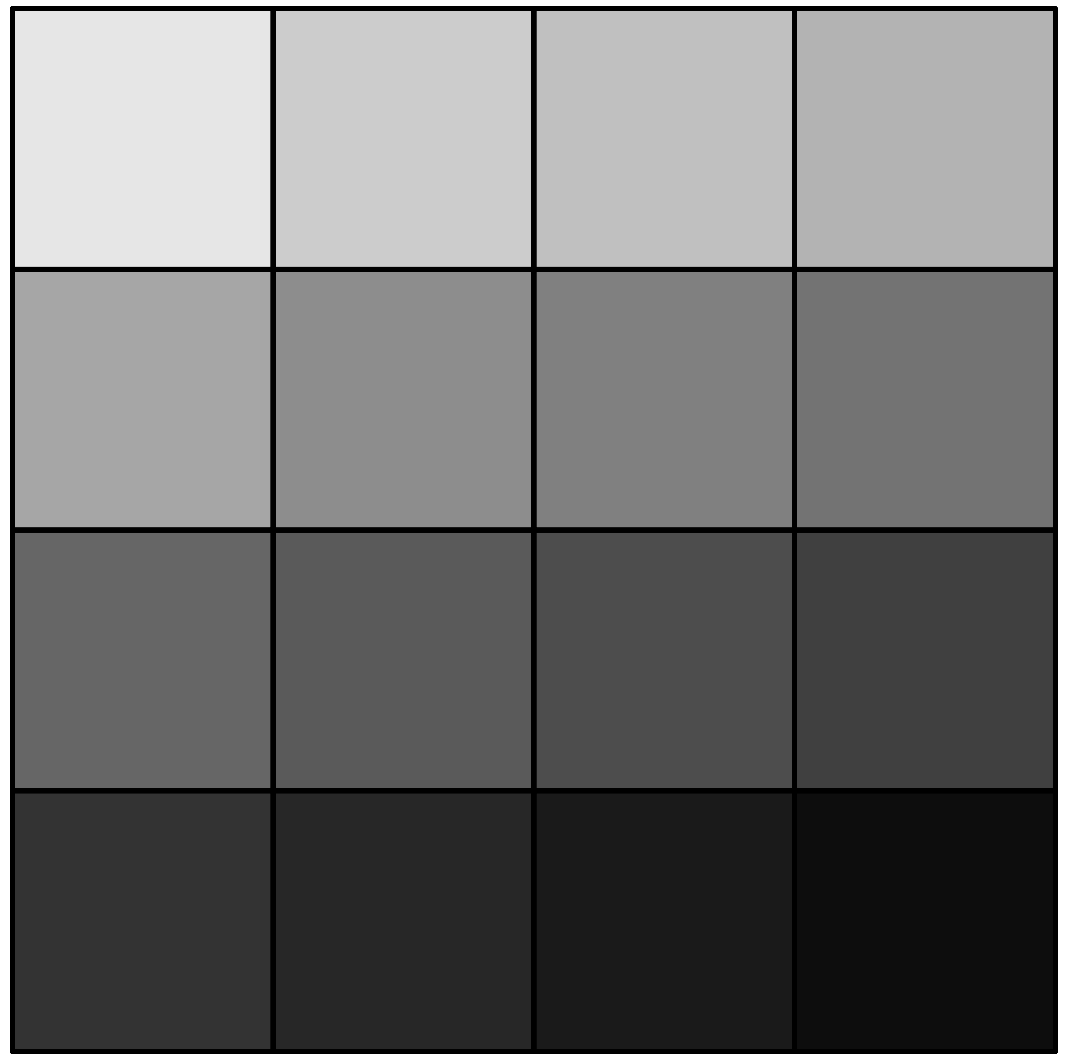


$\mathcal{T}$

$$\mathcal{T}\ {}_{\mathrm{xy}}\!\mapsto_{\mathrm{xy}^*} \mathcal{M}$$

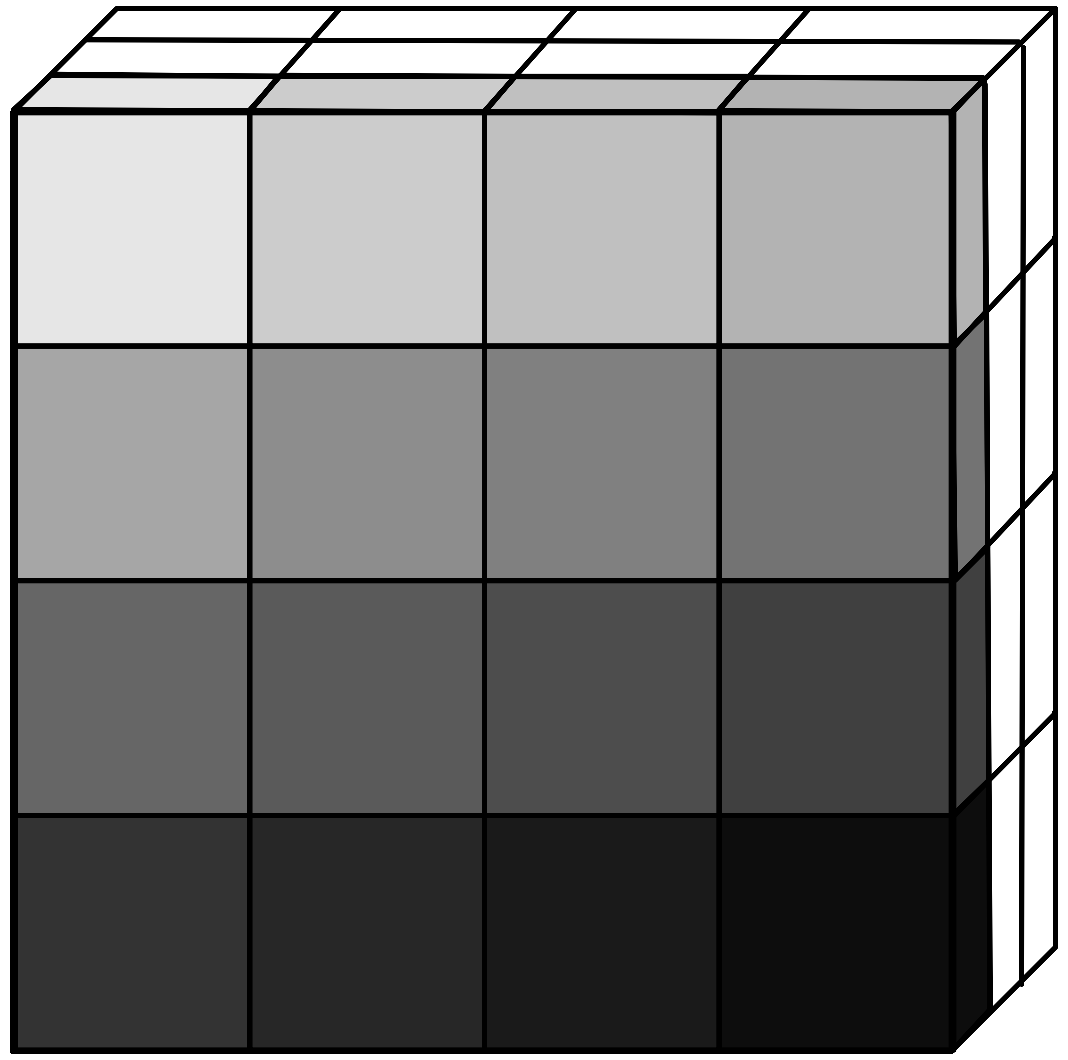


$\mathcal{M}$



$\mathcal{T}$

$$\mathcal{T}_{\mathrm{xy} \mapsto \mathrm{xy0}}\, \mathcal{M}$$



$\mathcal{M}$

Distributing Computation

Expression

$$A(i,j) = B(i,k) \cdot C(k,j)$$

$$A(i,l) = B(i,j,k) \cdot C(j,l) \cdot D(k,l)$$

$$a = B(i,j,k) \cdot C(i,j,k)$$

$$A(i,j,l) = B(i,j,k) \cdot C(k,l)$$

$$A(i,j) = B(i,j,k) \cdot c(k)$$

Data Distribution

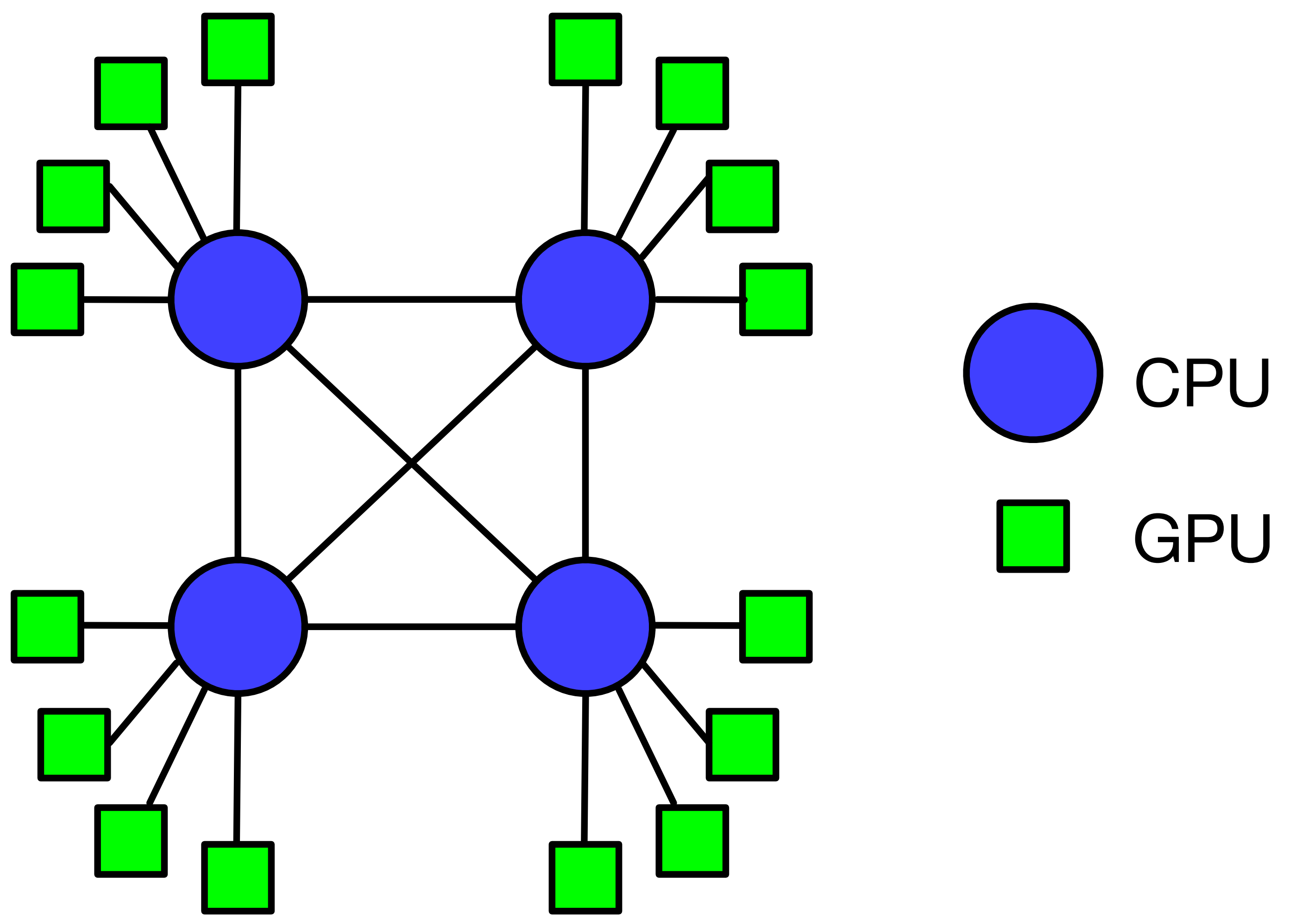
Partition A into tiles
Replicate B onto all nodes
Place C onto only some nodes

Computation Distribution

Owner Computes
Distribute i,j loops
Communicate in chunks

DISTAL

Supercomputer



Iteration Spaces

Hyper-rectangular grid of points representing each point in a set of nested loops

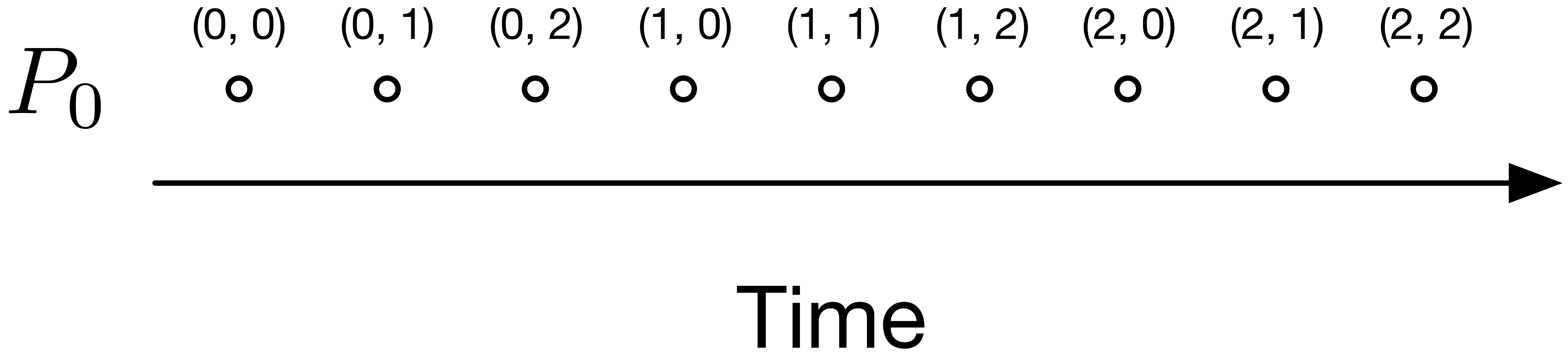
$$\forall_i \; a(i) = \sum_j b(j) \quad (i, j) \in \{0, 1, 2\} \times \{0, 1, 2\}$$

```
for i in (0, len(a)):
  for j in (0, len(b)):
    a[i] += b[j]
```

Execution Spaces

Space of all processors in $\mathcal{M}$ cross a time dimension

$$\forall_i \; a(i) = \sum_j b(j) \quad (i, j) \in \{0, 1, 2\} \times \{0, 1, 2\}$$

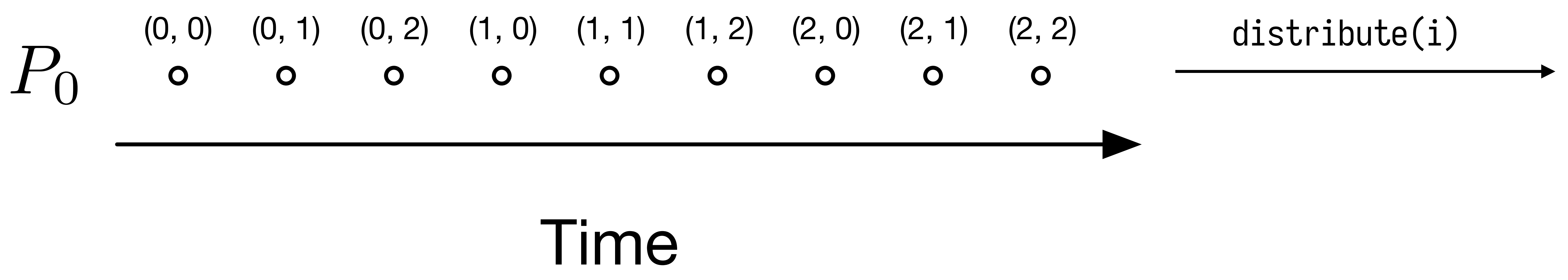


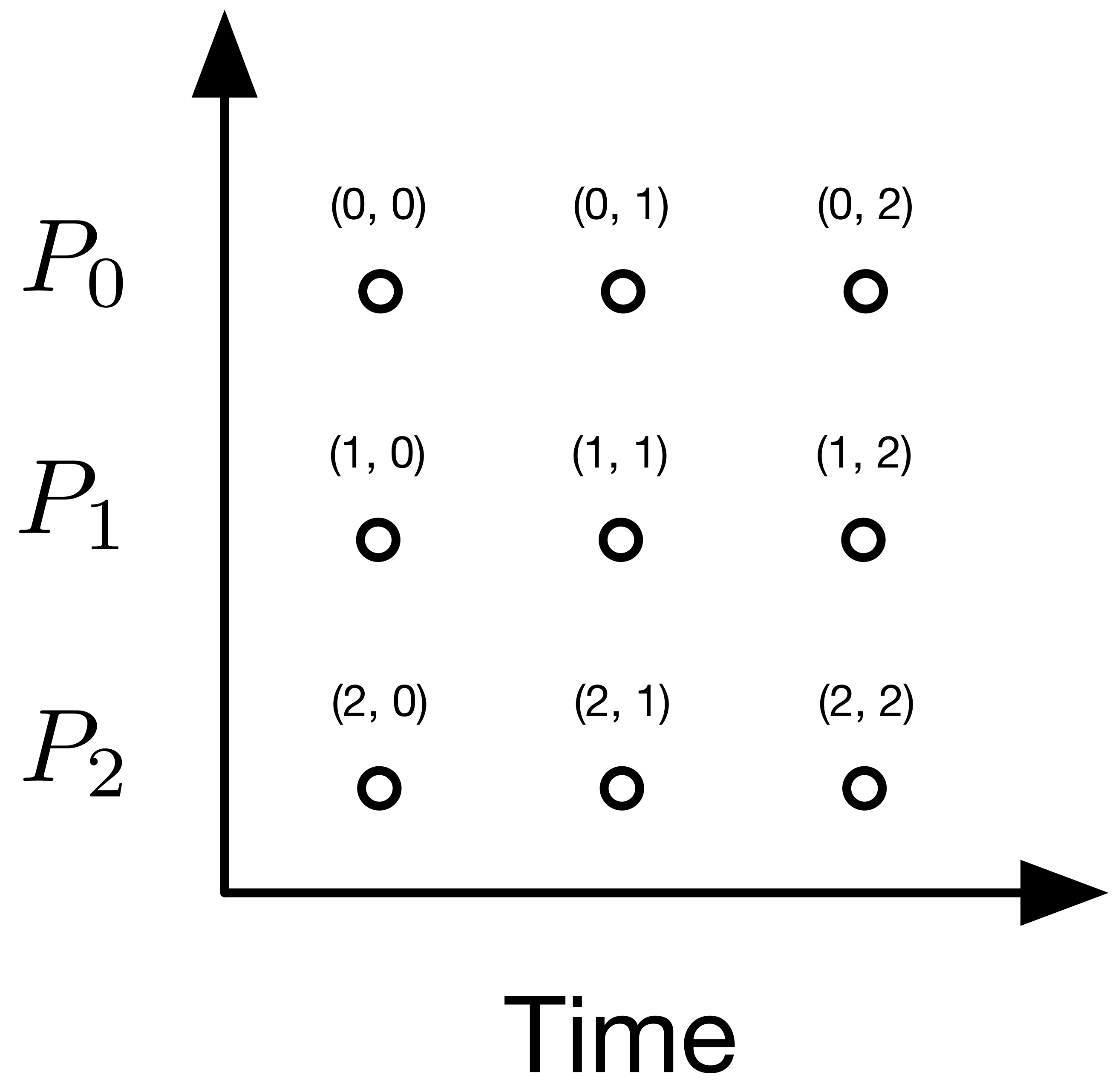
Scheduling

Change execution of iteration space through scheduling transformations

distribute

$$\forall_i \; a(i) = \sum_j b(j) \qquad (i, j) \in \{0, 1, 2\} \times \{0, 1, 2\}$$

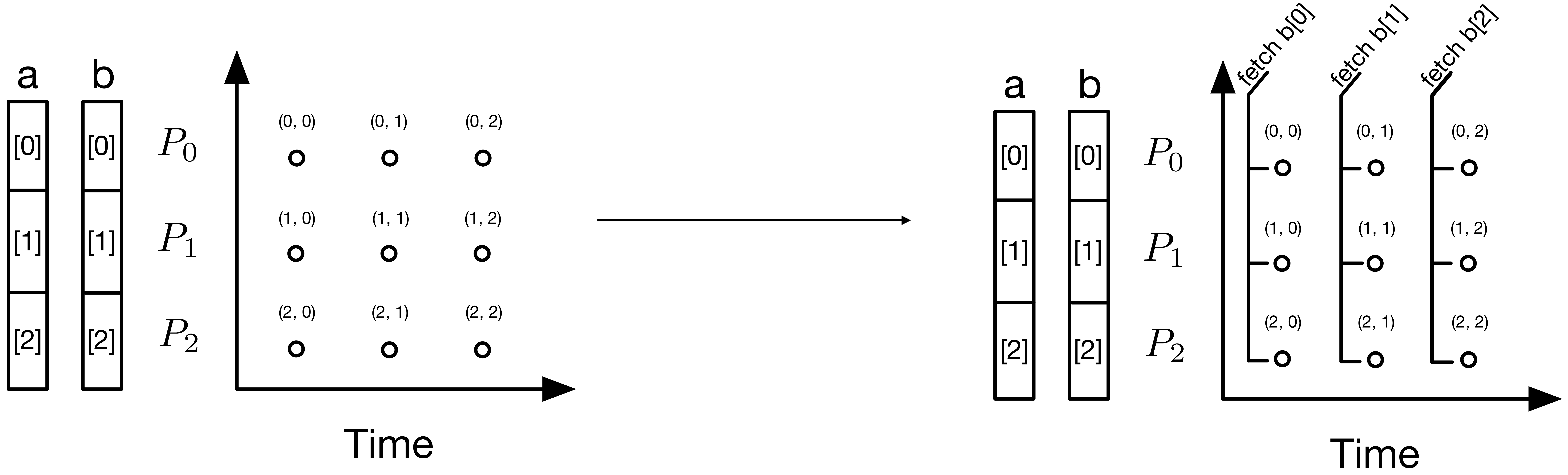




What about communication?

$$\forall_i \; a(i) = \sum_j b(j) \quad \text{s.t.} \quad a \;{}_{\mathrm{x}}{\mapsto}_{\mathrm{x}}\, \mathcal{M} \quad b \;{}_{\mathrm{x}}{\mapsto}_{\mathrm{x}}\, \mathcal{M}$$

Insert communication at each iteration space point, as needed

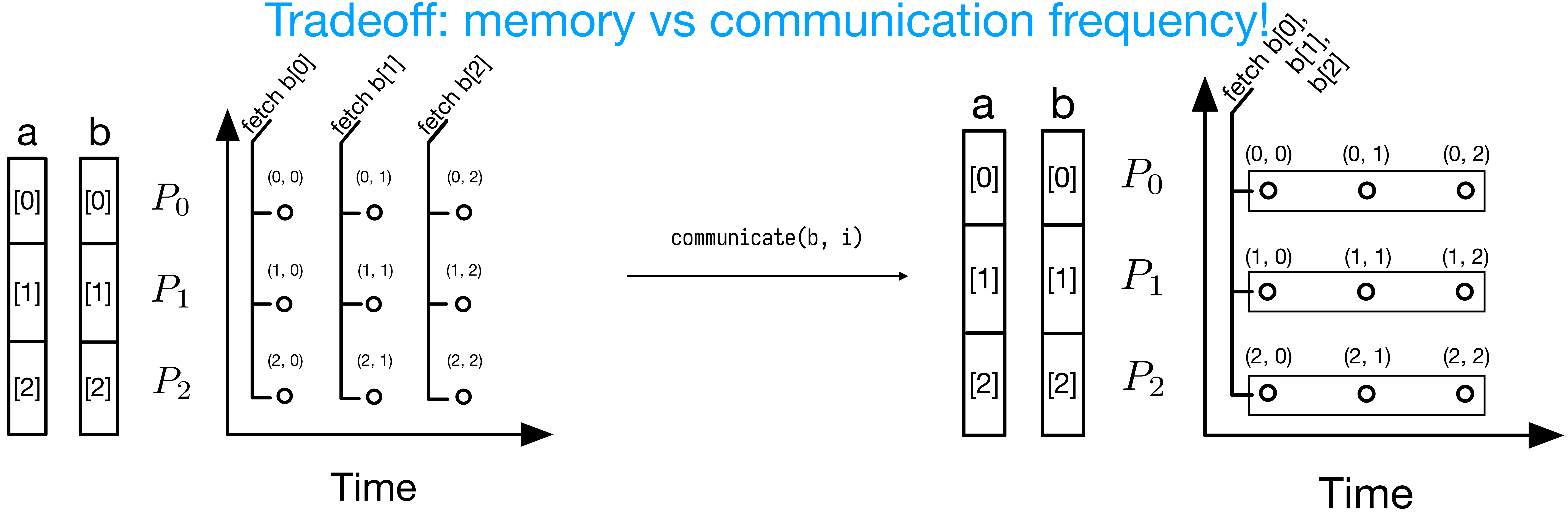


communicate

$$\forall_i \; a(i) = \sum_j b(j)$$

s.t. $a \,{}_{\mathrm{x}}\!\mapsto_{\mathrm{x}} \mathcal{M} \quad b \,{}_{\mathrm{x}}\!\mapsto_{\mathrm{x}} \mathcal{M}$

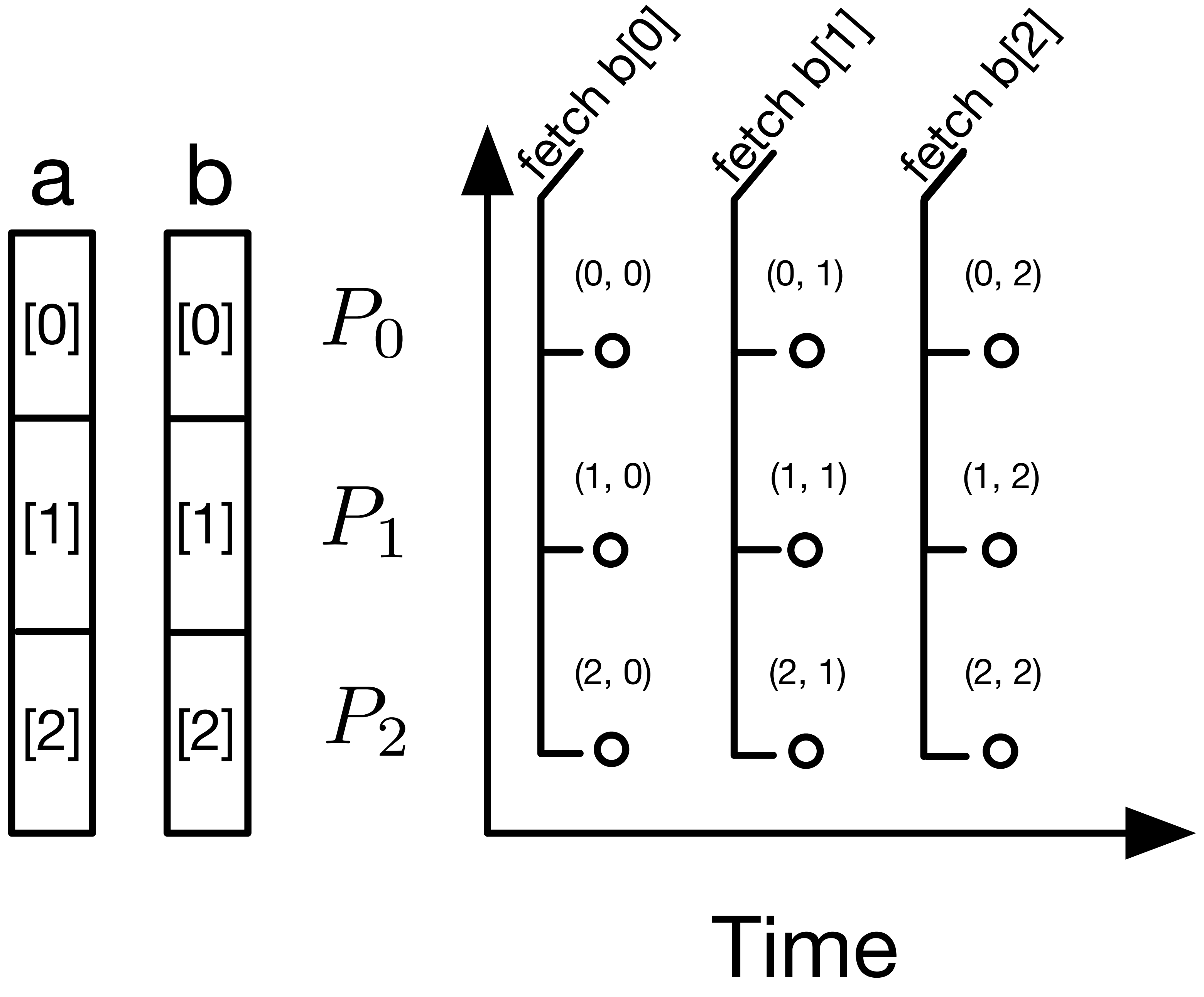
Tradeoff: memory vs communication frequency!

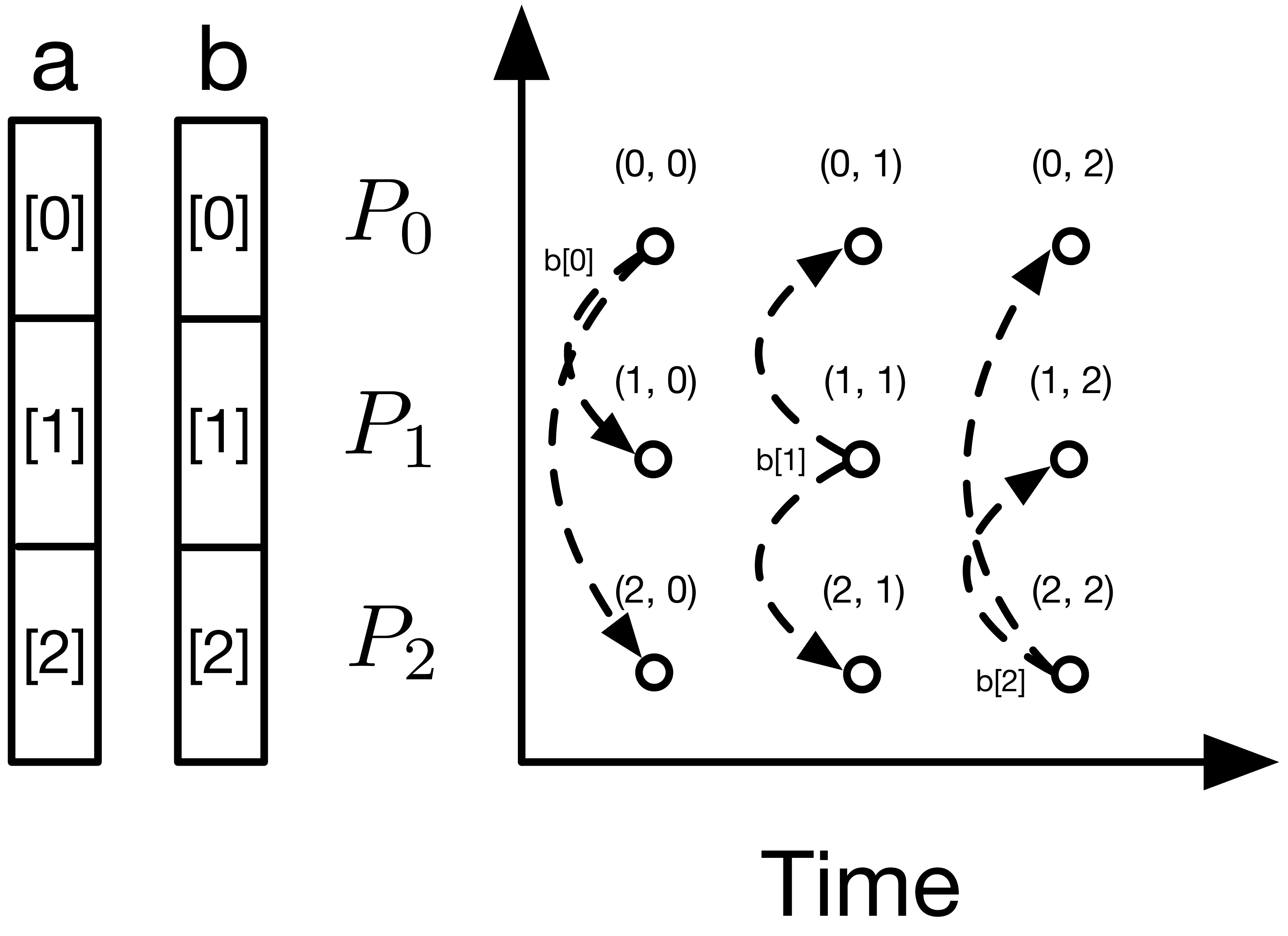


Breaking symmetry with rotate

$$\forall_i \; a(i) = \sum_j b(j)$$

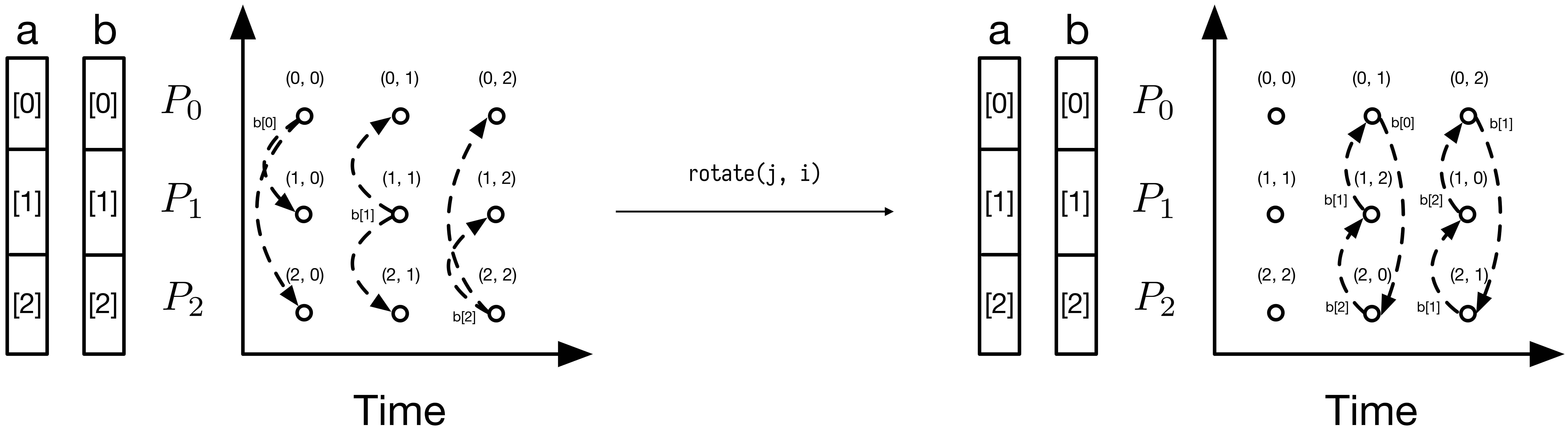
s.t. $a \; {}_{\mathrm{x}}{\mapsto}_{\mathrm{x}} \mathcal{M} \quad b \; {}_{\mathrm{x}}{\mapsto}_{\mathrm{x}} \mathcal{M}$



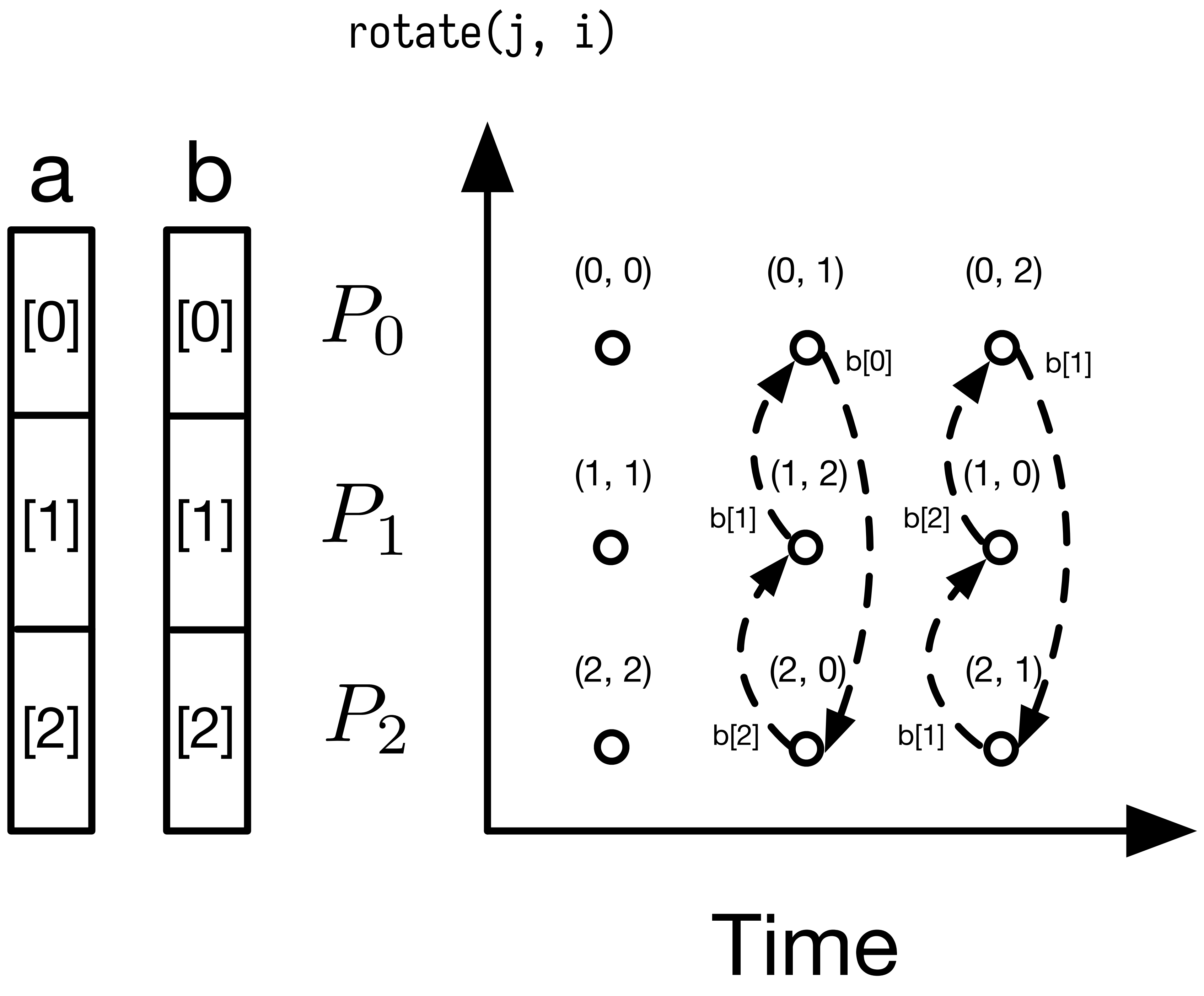


rotate

$$\forall_i \ a(i) = \sum_j b(j) \quad \text{s.t.} \quad a \ {}_{\mathrm{x}}{\mapsto}_{\mathrm{x}} \mathcal{M} \quad b \ {}_{\mathrm{x}}{\mapsto}_{\mathrm{x}} \mathcal{M}$$



rotate



Use the modulus operator!

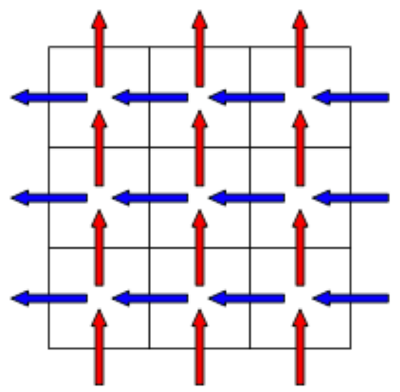
```
for j in (0, extent(j)):
  …
```

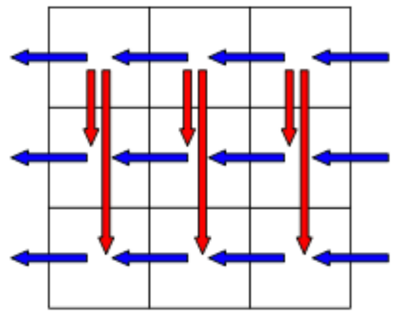
→

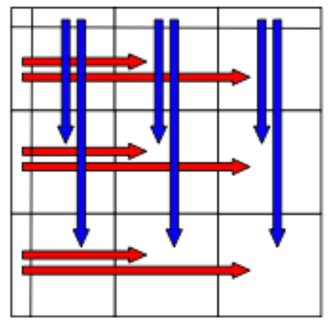
```
for j' in (0, extent(j)):
  j = j' + i mod extent(j)
  …
```

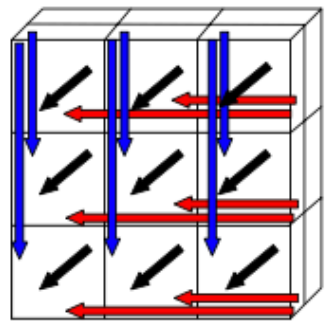
How expressive are these abstractions?

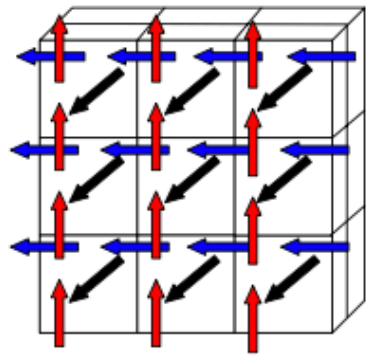
	Comm. Pattern	Target Machine	Data Distribution	Schedule

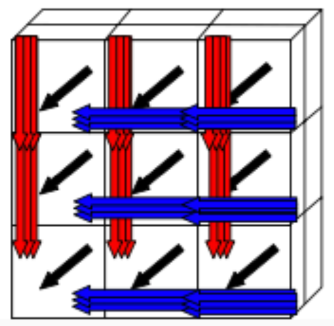
Cannon's Algorithm		$\mathcal{M}(gx, gy)$	$A\ {}_{ij \mapsto ij}\ \mathcal{M}$
			$B\ {}_{ij \mapsto ij}\ \mathcal{M}$
			$C\ {}_{ij \mapsto ij}\ \mathcal{M}$
				`.distribute({i, j}, {in, jn}, {il, jl}, Grid(gx, gy))`
				`.divide(k, ko, ki, gx)`
				`.reorder({ko, il, jl, ki})`
				`.rotate(ko, {in, jn}, kos)`
				`.communicate(A, jn)`
				`.communicate({B, C}, kos)`

PUMMA		$\mathcal{M}(gx, gy)$	$A\ {}_{ij \mapsto ij}\ \mathcal{M}$
			$B\ {}_{ij \mapsto ij}\ \mathcal{M}$
			$C\ {}_{ij \mapsto ij}\ \mathcal{M}$
				`.distribute({i, j}, {in, jn}, {il, jl}, Grid(gx, gy))`
				`.divide(k, ko, ki, gx)`
				`.reorder({ko, il, jl, ki})`
				`.rotate(ko, {in}, kos)`
				`.communicate(A, jn)`
				`.communicate({B, C}, kos)`

SUMMA		$\mathcal{M}(gx, gy)$	$A\ {}_{ij \mapsto ij}\ \mathcal{M}$
			$B\ {}_{ij \mapsto ij}\ \mathcal{M}$
			$C\ {}_{ij \mapsto ij}\ \mathcal{M}$
				`.distribute({i, j}, {in, jn}, {il, jl}, Grid(gx, gy))`
				`.split(k, ko, ki, chunkSize)`
				`.reorder({ko, il, jl, ki})`
				`.communicate(A, jn)`
				`.communicate({B, C}, ko)`

Johnson's Algorithm		$\mathcal{M}(\sqrt[3]{p}, \sqrt[3]{p}, \sqrt[3]{p})$	$A\ {}_{ij \mapsto ij0}\ \mathcal{M}$
			$B\ {}_{ik \mapsto i0k}\ \mathcal{M}$
			$C\ {}_{kj \mapsto 0jk}\ \mathcal{M}$
				`.distribute({i, j, k}, {in, jn, kn},`
				`{il, jl, kl}, Grid(`$\sqrt[3]{p}$`, `$\sqrt[3]{p}$`, `$\sqrt[3]{p}$`))`
				`.communicate({A, B, C}, kn)`

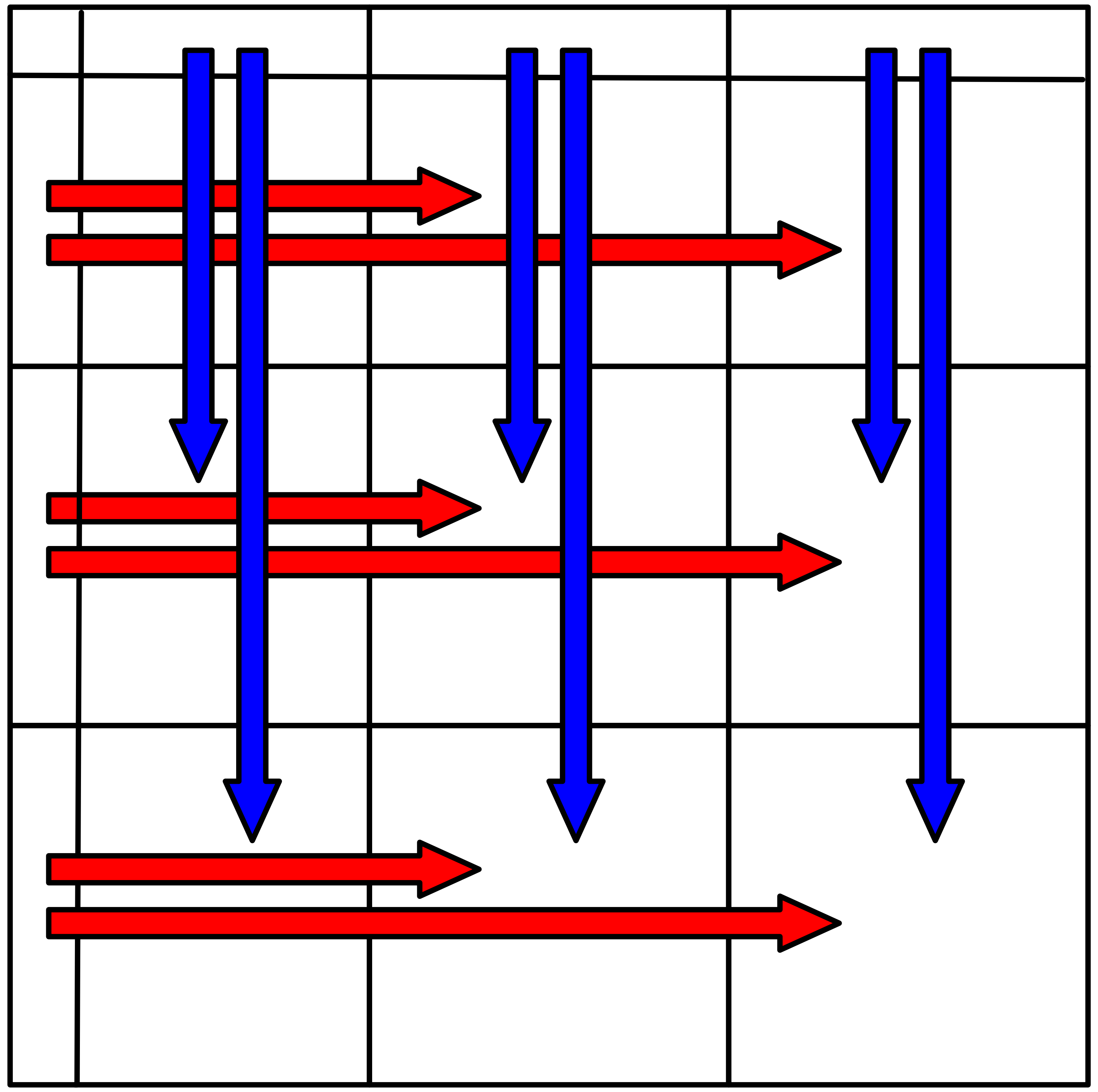
Solomonik's Algorithm		$\mathcal{M}(\sqrt{\frac{p}{c}}, \sqrt{\frac{p}{c}}, c)$	$A\ {}_{ij \mapsto ij0}\ \mathcal{M}$
			$B\ {}_{ij \mapsto ij0}\ \mathcal{M}$
			$C\ {}_{ij \mapsto ij0}\ \mathcal{M}$
				`.distribute({i, j, k}, {in, jn, kn},`
				`{il, jl, kl}, Grid(`$\sqrt{\frac{p}{c}}$`, `$\sqrt{\frac{p}{c}}$`, c))`
				`.divide(kl, k1, k2, `$\sqrt{\frac{p}{c^3}}$`)`
				`.reorder({k1, il, jl, k2})`
				`.rotate(k1, {in, jn}, k1s)`
				`.communicate(A, jn)`
				`.communicate({B, C}, k1s)`

COSMA		induced by schedule	induced by schedule	`// gx, gy, gz, numSteps computed by COSMA scheduler.`
				`.distribute({i, j, k}, {in, jn, kn}`
				`{il, jl, kl}, Grid(gx, gy, gz))`
				`.divide(kl, klo, kli, numSteps)`
				`.reorder({klo, il, jl, kli})`
				`.communicate(A, kn)`
				`.communicate({B, C}, klo)`

SUMMA Algorithm

A(i, j) = B(i, k) * C(k, j)

```
# Arrange p processors into a 2D grid.
# Assign a tile of A, B, C to each processor.
for all P_ij in parallel:
  for kc in (0, k, chunkSize):
    B_l = row broadcast the kc to kc+chunkSize columns of B
    C_l = col broadcast the kc to kc+chunkSize rows of C
    A += B_l * C_l
```

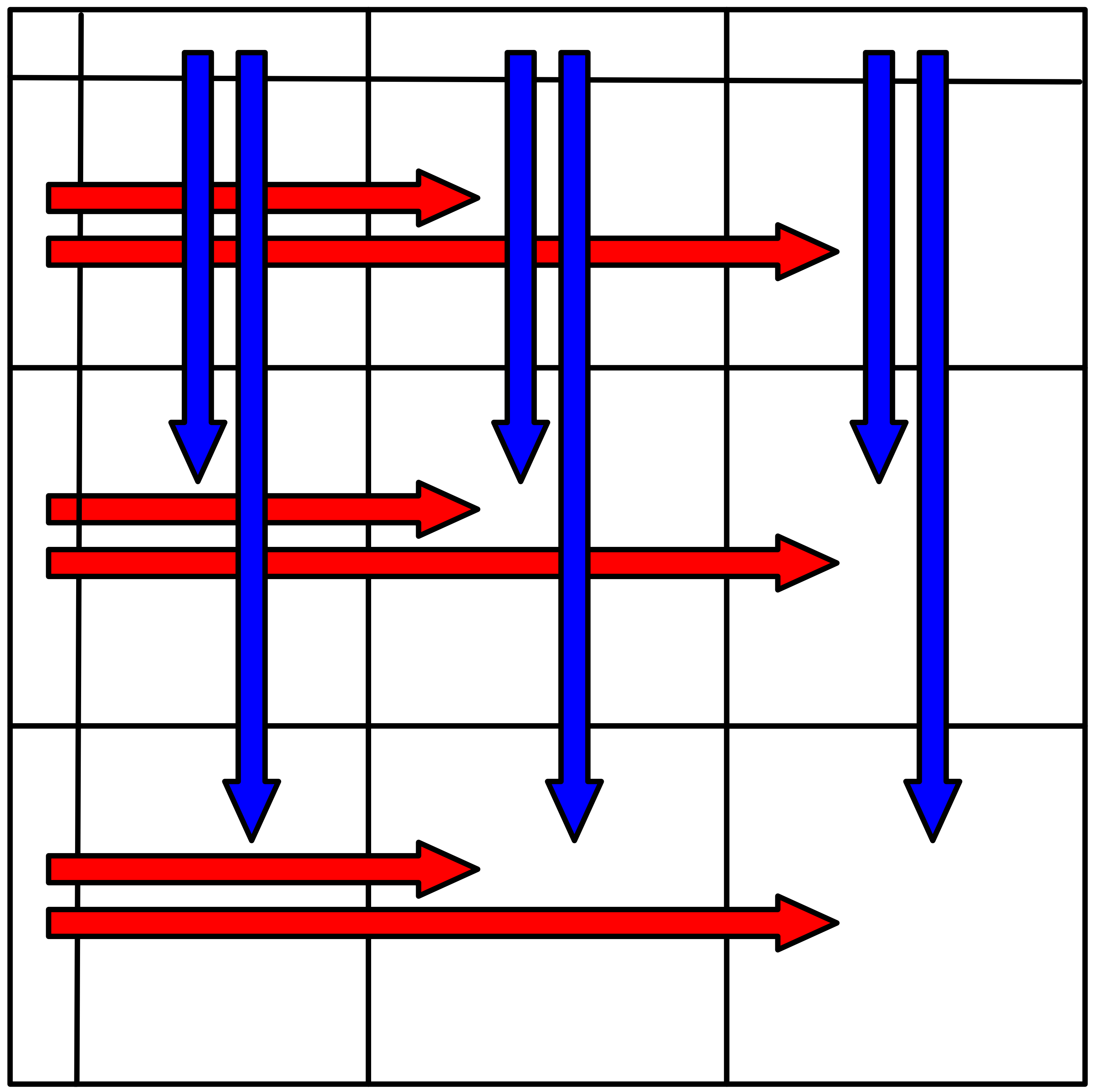


$$\mathcal{M} = \mathsf{Grid}(gx, gy)$$

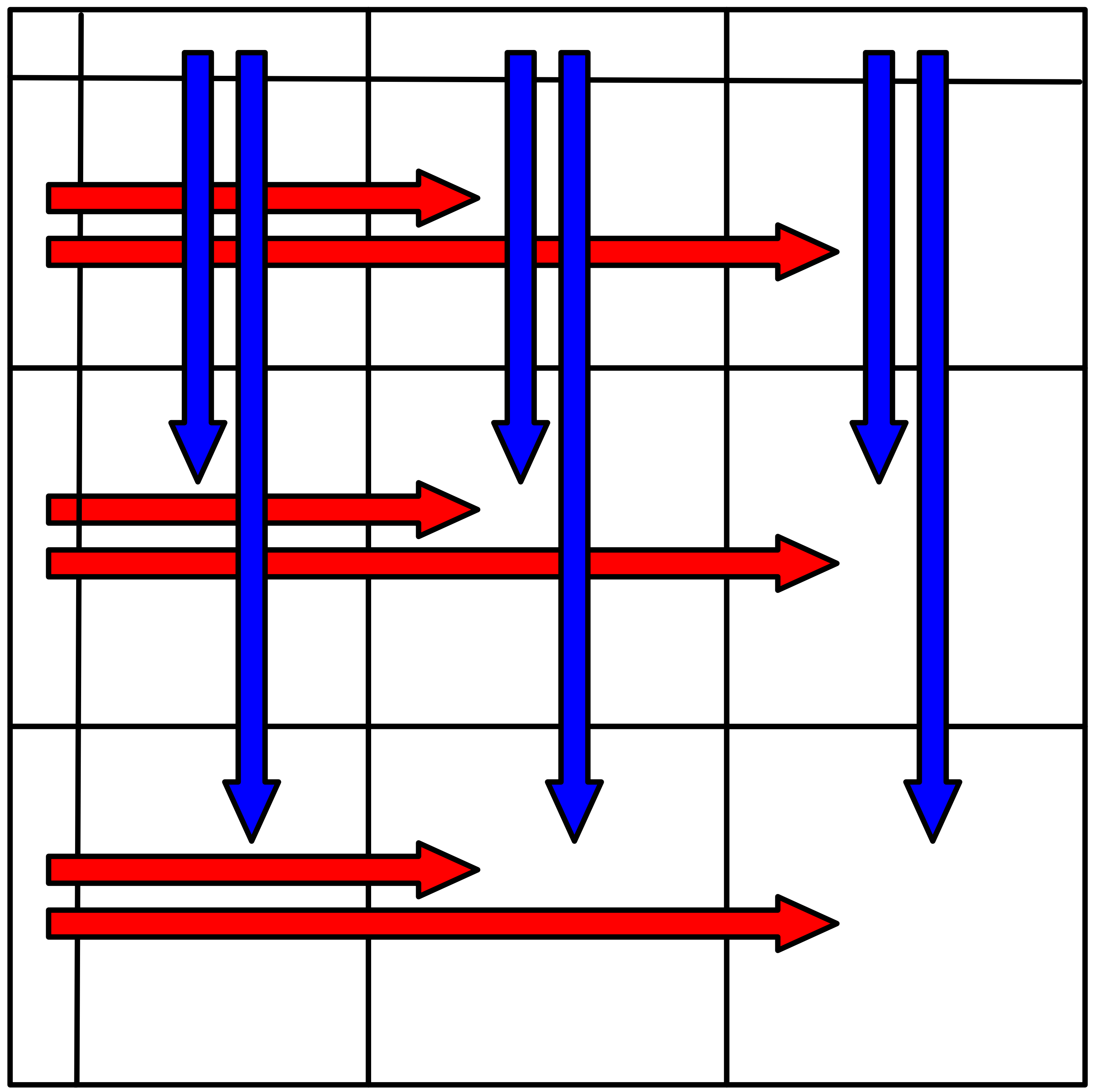
$$A \;{}_{\mathrm{xy}}{\mapsto}_{\mathrm{xy}}\; \mathcal{M}$$

$$B \;{}_{\mathrm{xy}}{\mapsto}_{\mathrm{xy}}\; \mathcal{M}$$

$$C \;{}_{\mathrm{xy}}{\mapsto}_{\mathrm{xy}}\; \mathcal{M}$$

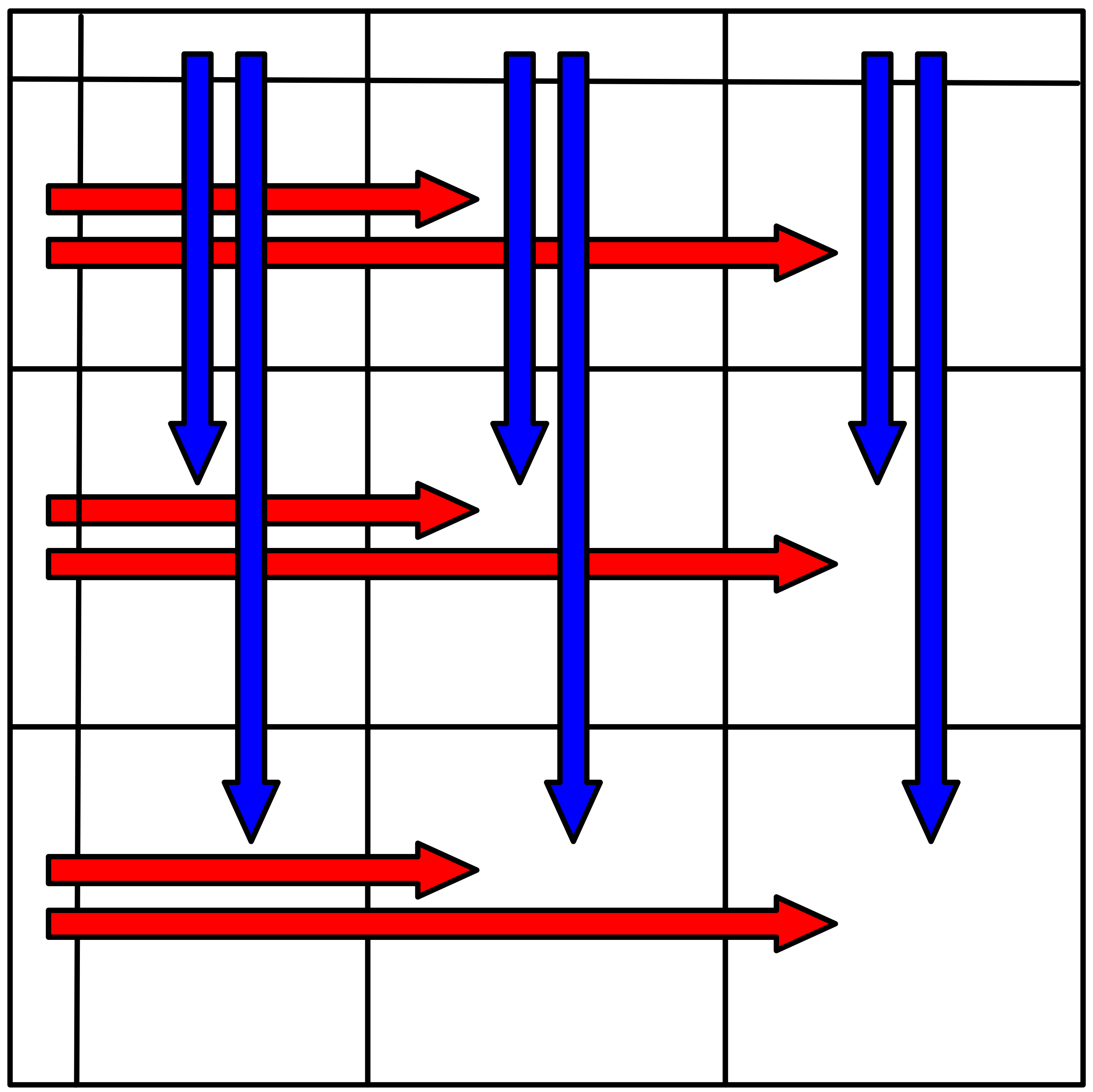


```
for i:
  for j:
    for k:
      A(i, j) += B(i, k) * C(k, j)
```



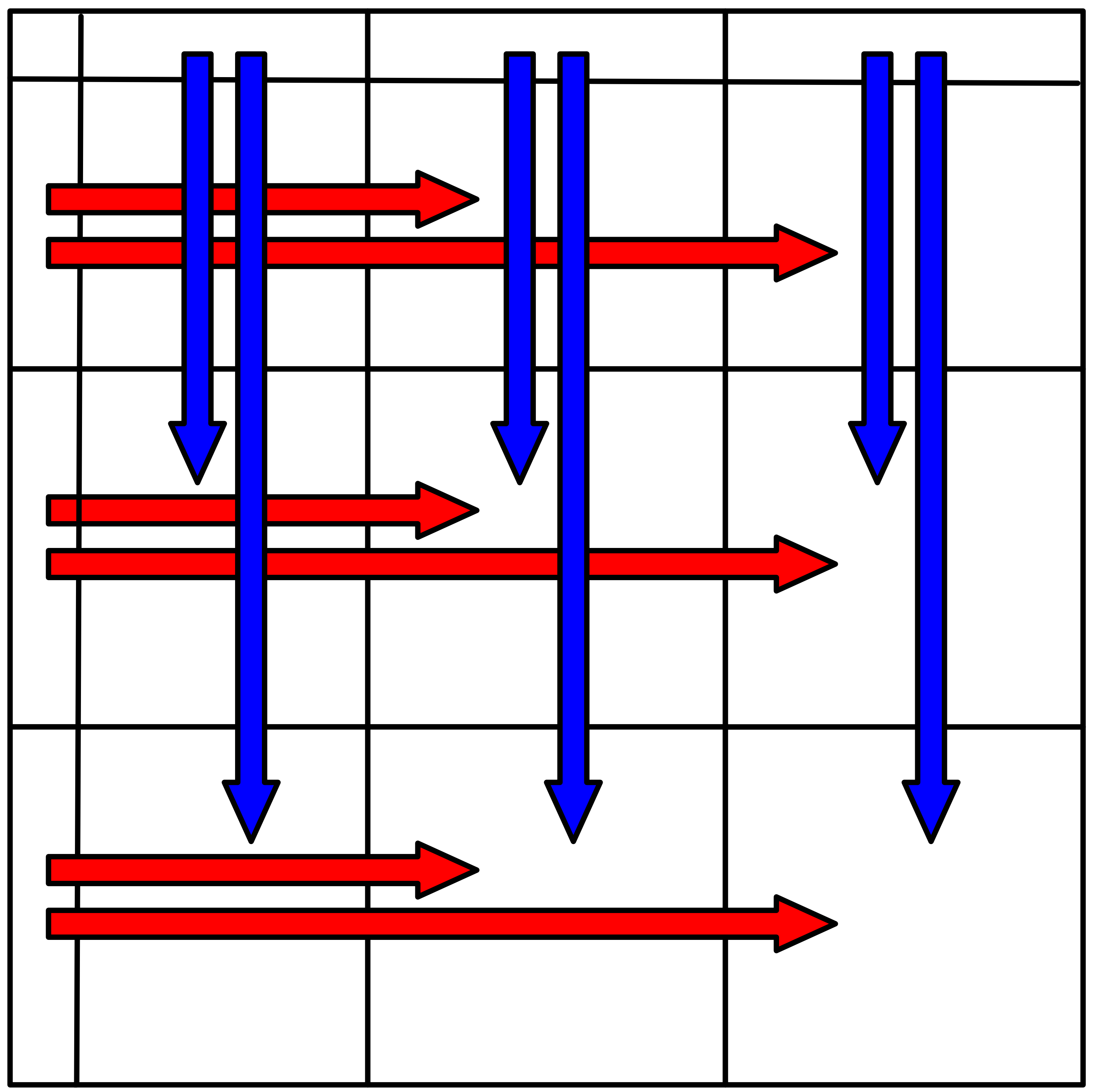
```
divide(i, il, in, gx)
divide(j, jl, jn, gy)
reorder({in, jn, il, jl})
```

```
for in:
  for jn:
    for il:
      for jl:
        for k:
          A(i, j) += B(i, k) * C(k, j)
```



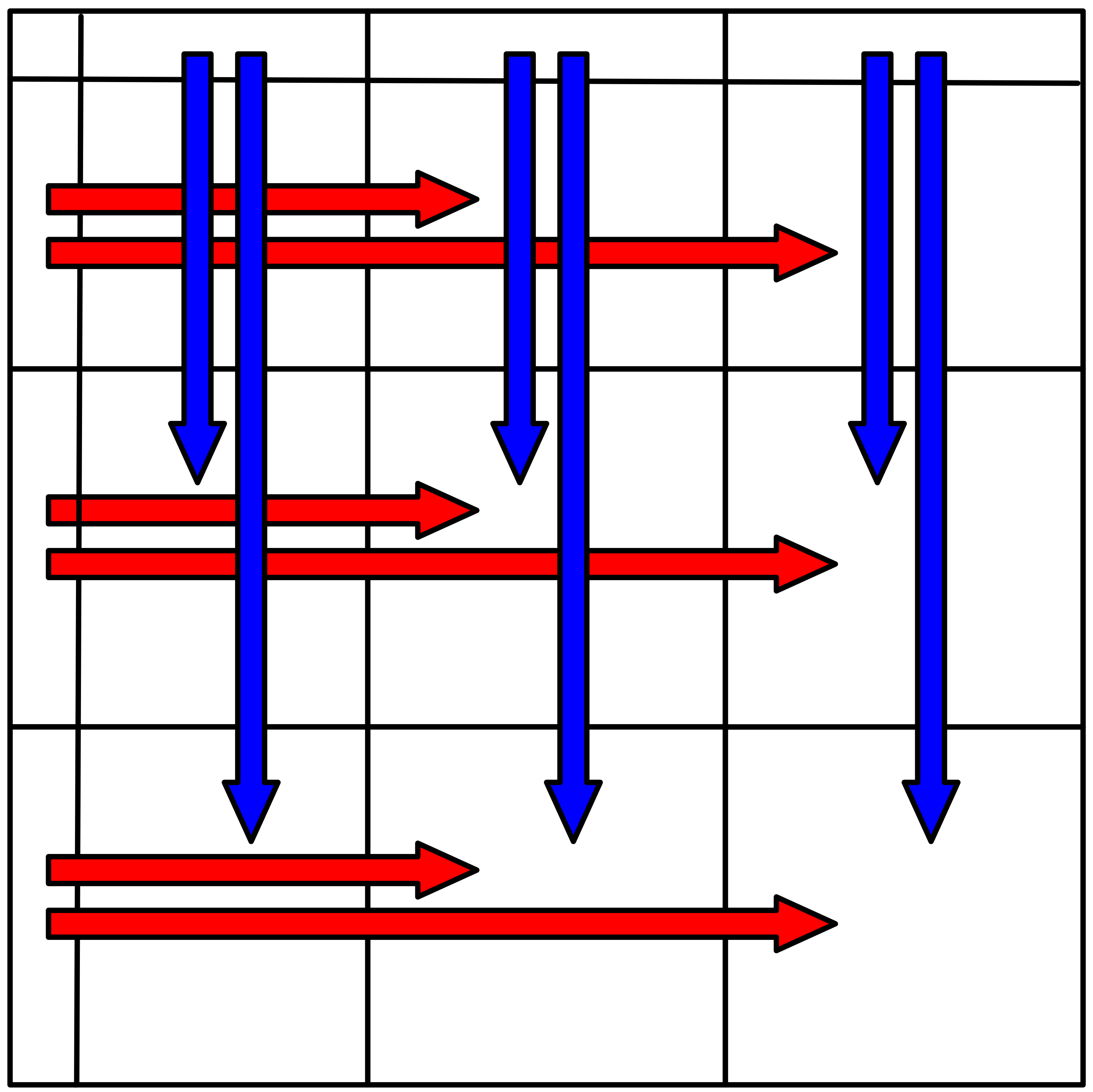
```
divide(i, il, in, gx)
divide(j, jl, jn, gx)
reorder({in, jn, il, jl})
split(k, ko, ki, chunkSize)
reorder(ko, il, jl, ki)
```

```
for in:
  for jn:
    for ko:
      for il:
        for jl:
          for ki:
            A(i, j) += B(i, k) * C(k, j)
```



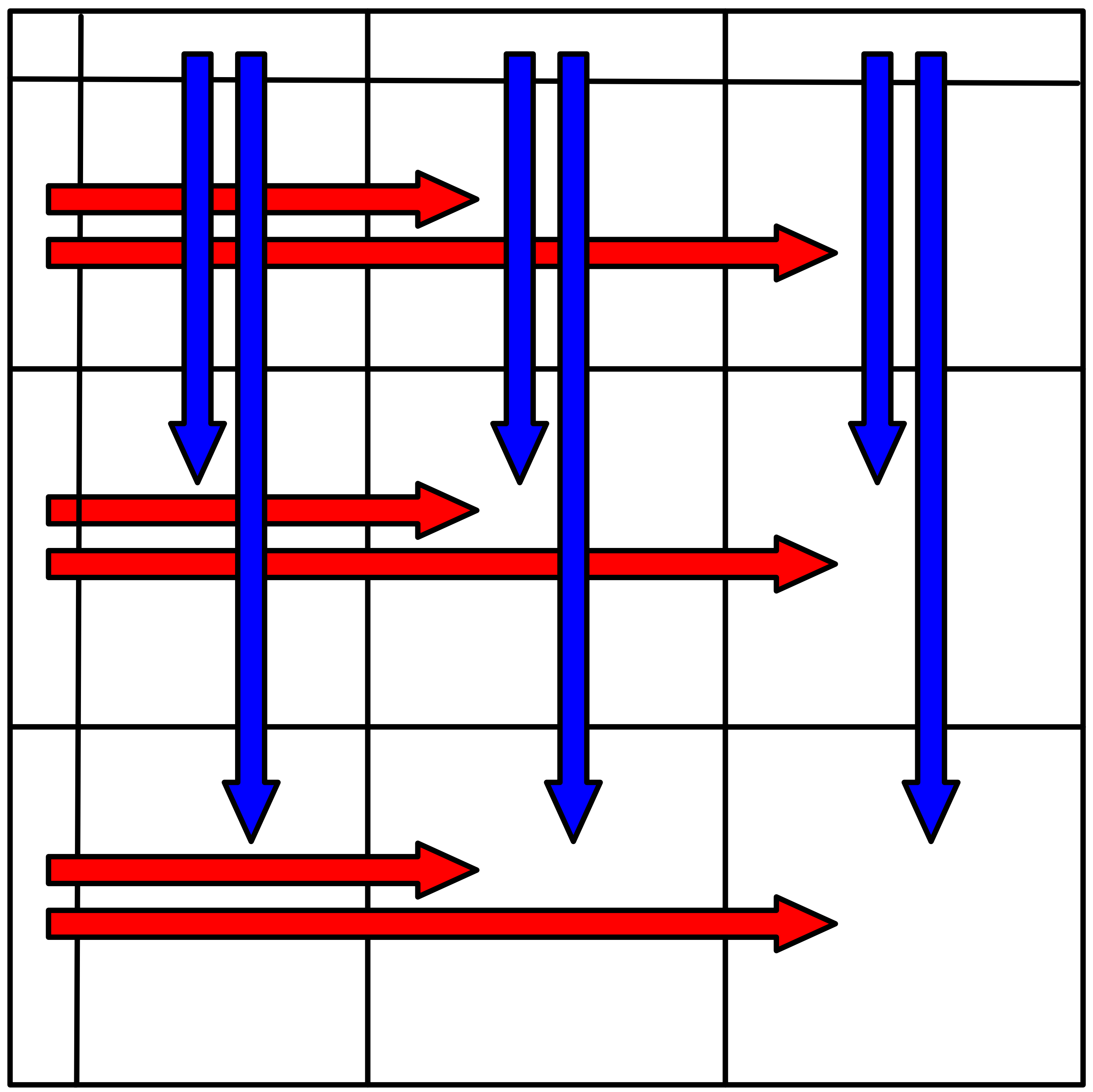
```
divide(i, il, in, gx)
divide(j, jl, jn, gx)
reorder({in, jn, il, jl})
split(k, ko, ki, chunkSize)
reorder(ko, il, jl, ki)
distribute(in, jn)
```

```
distributed for in, jn:
  for ko:
    for il:
      for jl:
        for ki:
          A(i, j) += B(i, k) * C(k, j)
```



```
divide(i, il, in, gx)
divide(j, jl, jn, gx)
reorder({in, jn, il, jl})
split(k, ko, ki, chunkSize)
reorder(ko, il, jl, ki)
distribute(in, jn)
communicate(A, jn)
communicate({B, C}, ko)
```

```
distributed for in, jn:
  communicate A
  for ko:
    communicate B, C
    for il:
      for jl:
        for ki:
          A(i, j) += B(i, k) * C(k, j)
```



Compilation Process

Expression

$$A(i,j) = B(i,k) \cdot C(k,j)$$

$$A(i,l) = B(i,j,k) \cdot C(j,l) \cdot D(k,l)$$

$$a = B(i,j,k) \cdot C(i,j,k)$$

$$A(i,j,l) = B(i,j,k) \cdot C(k,l)$$

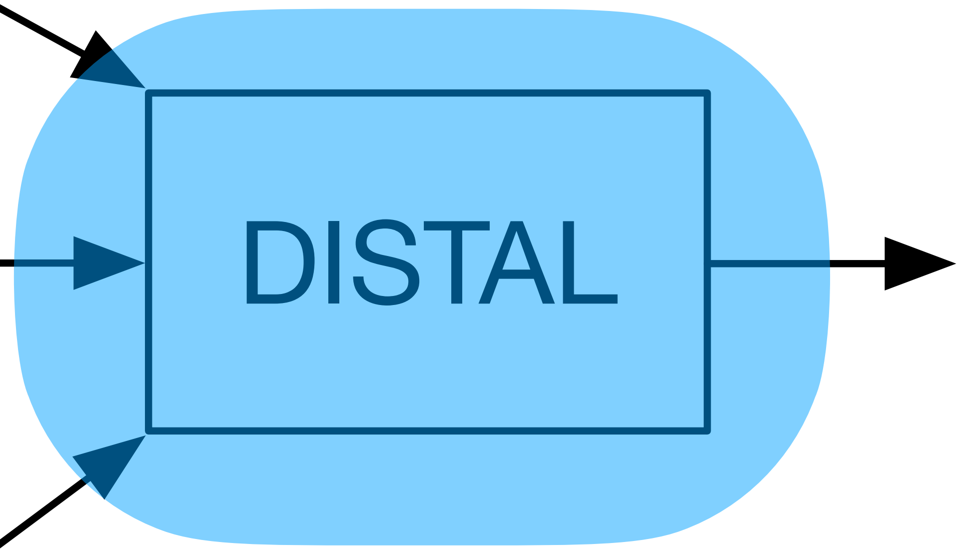
$$A(i,j) = B(i,j,k) \cdot c(k)$$

Data Distribution

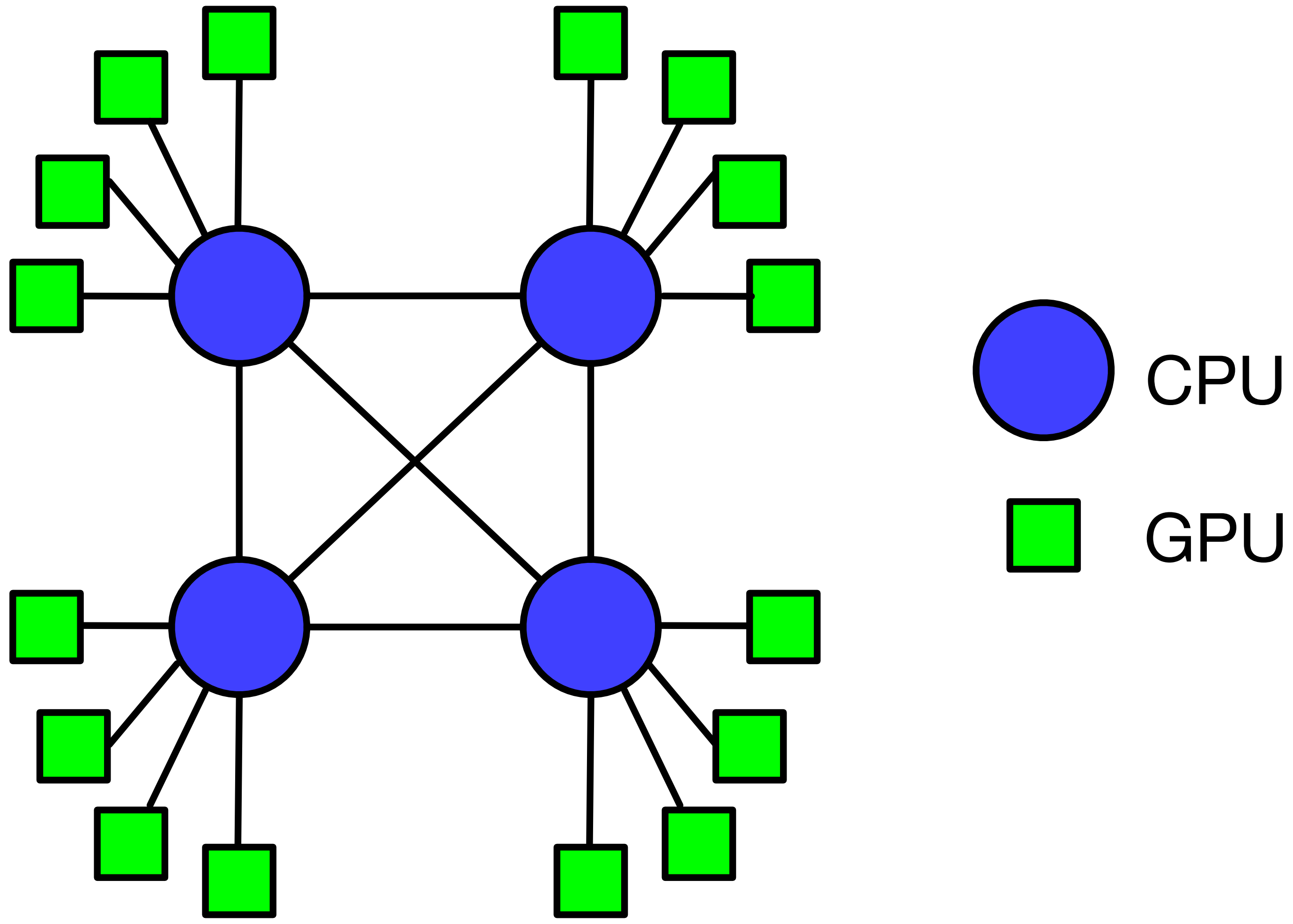
Partition A into tiles
Replicate B onto all nodes
Place C onto only some nodes

Computation Distribution

Owner Computes
Distribute i,j loops
Communicate in chunks



Supercomputer



Translate to Concrete Index Notation

$$A(i,j) = B(i,j,k) \cdot c(k) \longrightarrow \forall_i \forall_j \forall_k \; A(i,j) \mathrel{+}= B(i,j,k) \cdot c(k)$$

Iteration order unspecified

Specified Iteration Order

Scheduling operations rewrite Concrete Index Notation

$$\ldots\ \forall_i\ S \xrightarrow{\text{divide}(i, i_o, i_i, c)} \ldots\ \forall_{i_o}\forall_{i_i}\ S \text{ s.t. } \text{divide}(i, i_o, i_i, c)$$

$$\ldots\ \forall_i\ S \xrightarrow{\text{distribute}(i)} \ldots\ \forall_i\ S \text{ s.t. } \text{distribute}(i)$$

$$\ldots\ \forall_I\forall_t\ S \xrightarrow{\text{rotate}(t, I, r)} \ldots\forall_I\forall_r\ S \text{ s.t. } \text{rotate}(t, I, r)$$

$$\ldots\ \forall_i\ S \xrightarrow{\text{communicate}(\mathcal{T}, i)} \ldots\ \forall_i\ S \text{ s.t. } \text{communicate}(\mathcal{T}, i)$$

Target specific backend handles these constructs now!

Compiling Tensor Distribution Notation

$$\mathcal{T}\ {}_{\mathrm{xy}\mapsto_{\mathrm{x}}}\ \mathcal{M}$$

$$\downarrow$$

$$\forall_x \forall_y\ \mathcal{T}(x, y)$$

$$\downarrow$$

$$\forall_{xo} \forall_{xi} \forall_y\ \mathcal{T}(x, y) \text{ s.t. divide(x, xo, xi, gx)}$$

$$\downarrow$$

$$\forall_{xo} \forall_{xi} \forall_y\ \mathcal{T}(x, y) \text{ s.t. divide(x, xo, xi, gx)}, \text{distribute(xo)}, \text{comm.}(\mathcal{T}, \text{xo})$$

What’s the backend?

Legion

Distributed task-based runtime system

- Tasks operate on bulk data
- System moves memory between processors for tasks to use

`distribute` → launch a set of tasks

`communicate` → tell Legion the data to transfer

`rotate` → perform a loop transformation with modulus

Evaluation

Comparisons

Distributed GEMM — evaluate performance on a highly optimized kernel

- Compare against COSMA, Cyclops Tensor Framework, ScaLAPACK

Higher order tensor kernels — evaluate performance on the long tail

- Compare against Cyclops Tensor Framework

Results (Methodology)

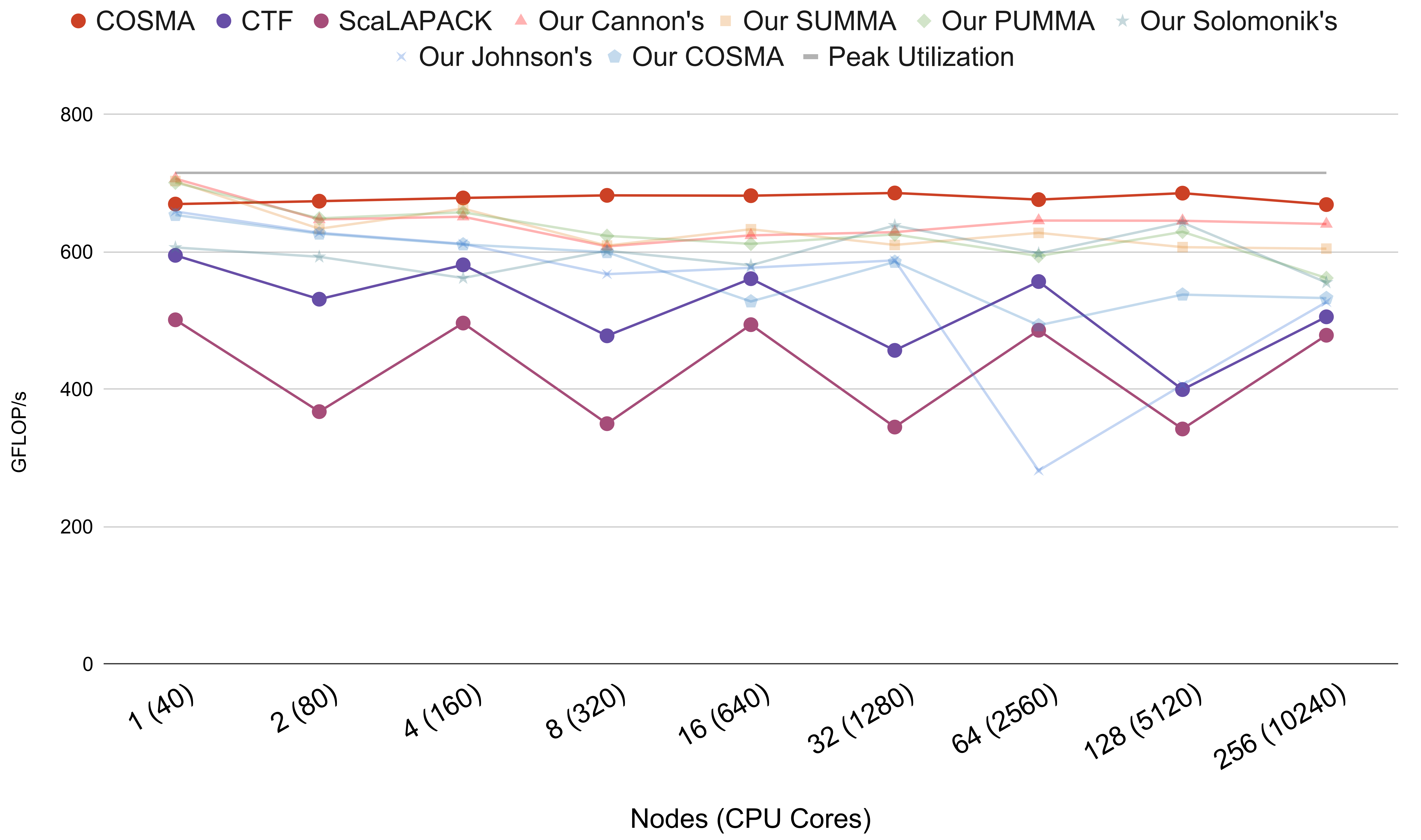
Experiments run on up to 256 nodes of Lassen (4 V100 GPUs/node, 40 Power9 CPUs/node, IB interconnect)

All systems configured to use the same BLAS / CuBLAS for GEMM

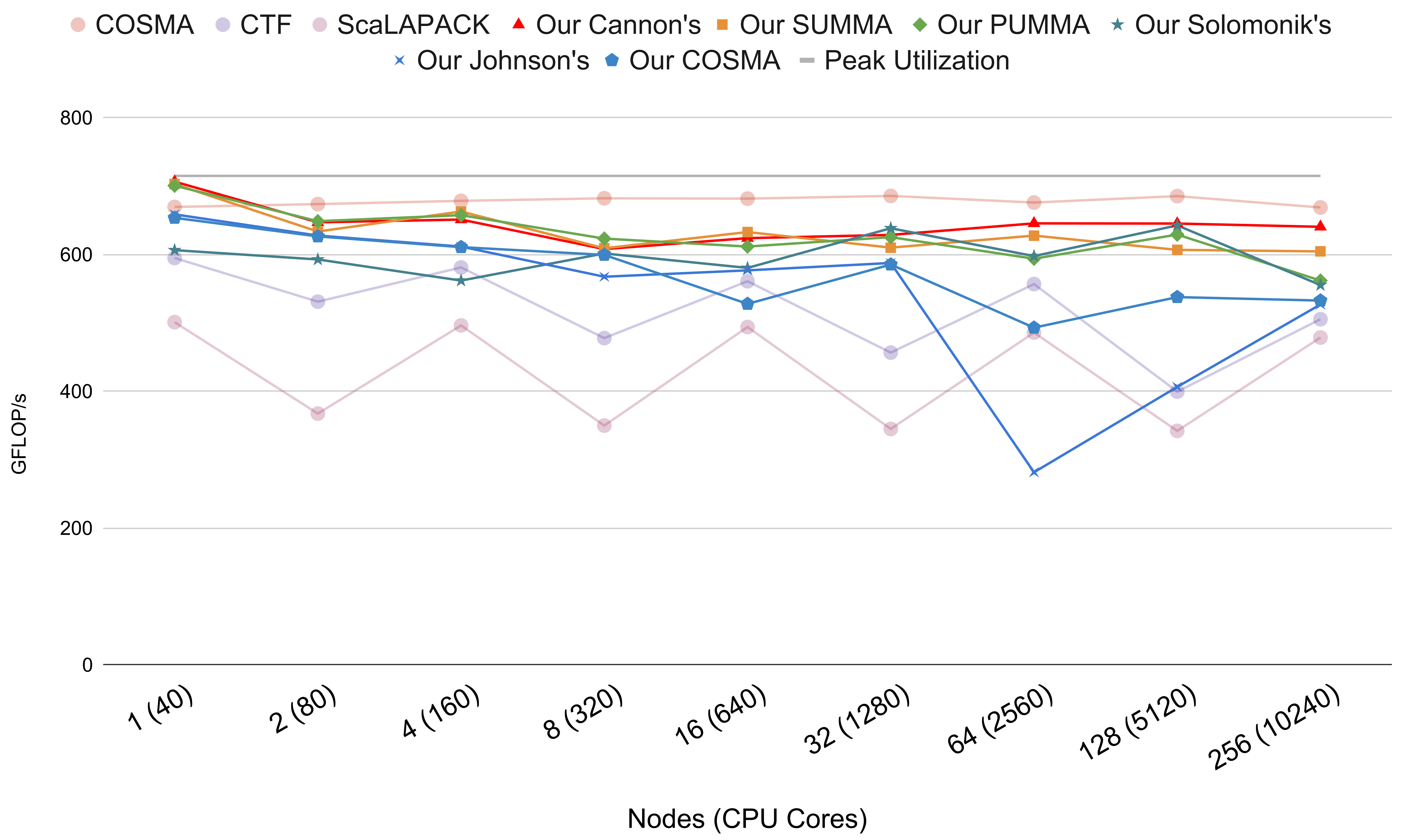
All experiments are weak-scaling (memory / node stays constant)

Results reported in GFLOP/s (compute bound) and GB/s (bandwidth bound) per node

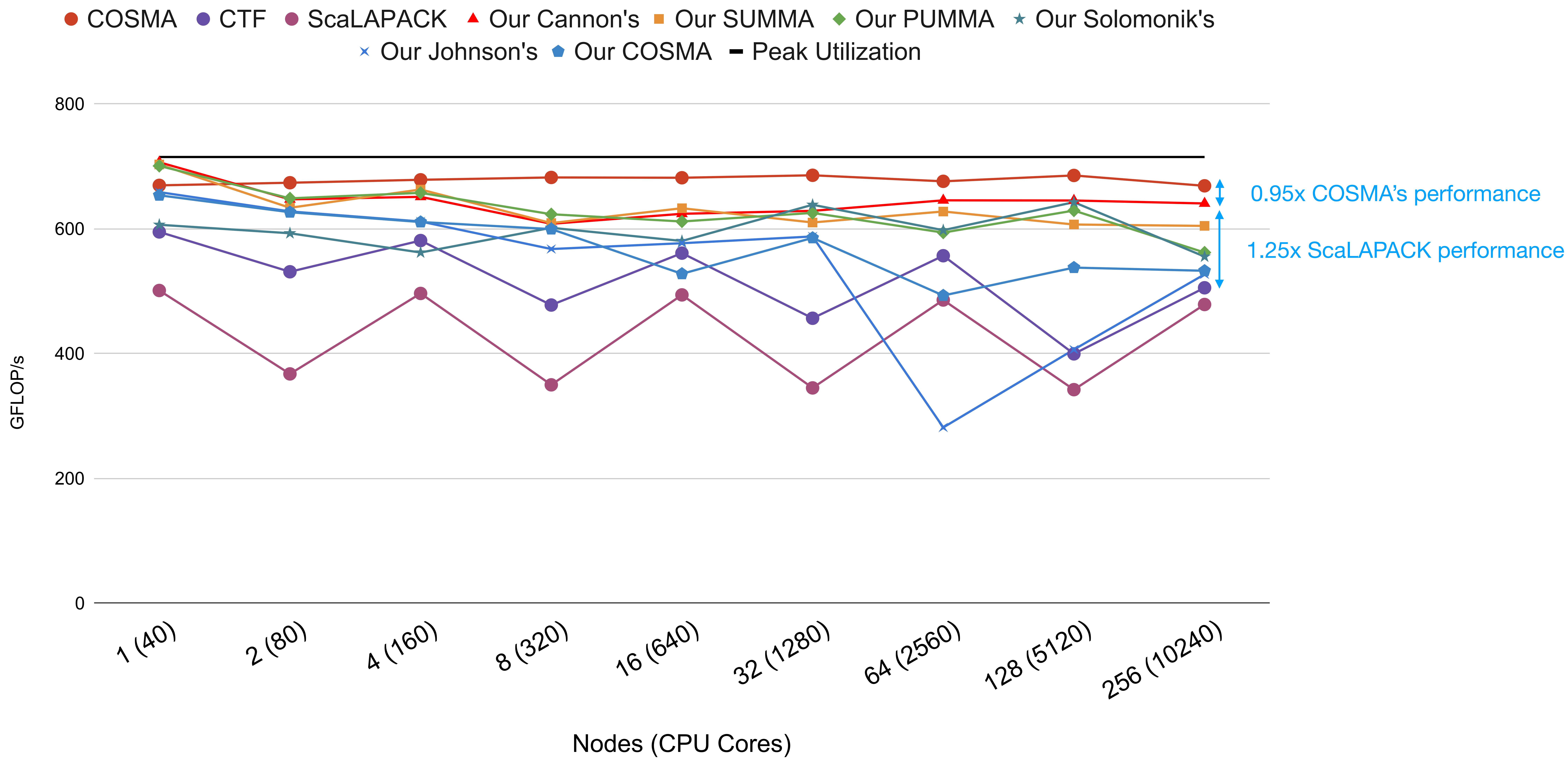
GEMM (CPU)


COSMA
CTF
ScaLAPACK
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our COSMA
Peak Utilization
800
600
400
200
0
GFLOP/s
1 (40)
2 (80)
4 (160)
8 (320)
16 (640)
32 (1280)
64 (2560)
128 (5120)
256 (10240)
Nodes (CPU Cores)

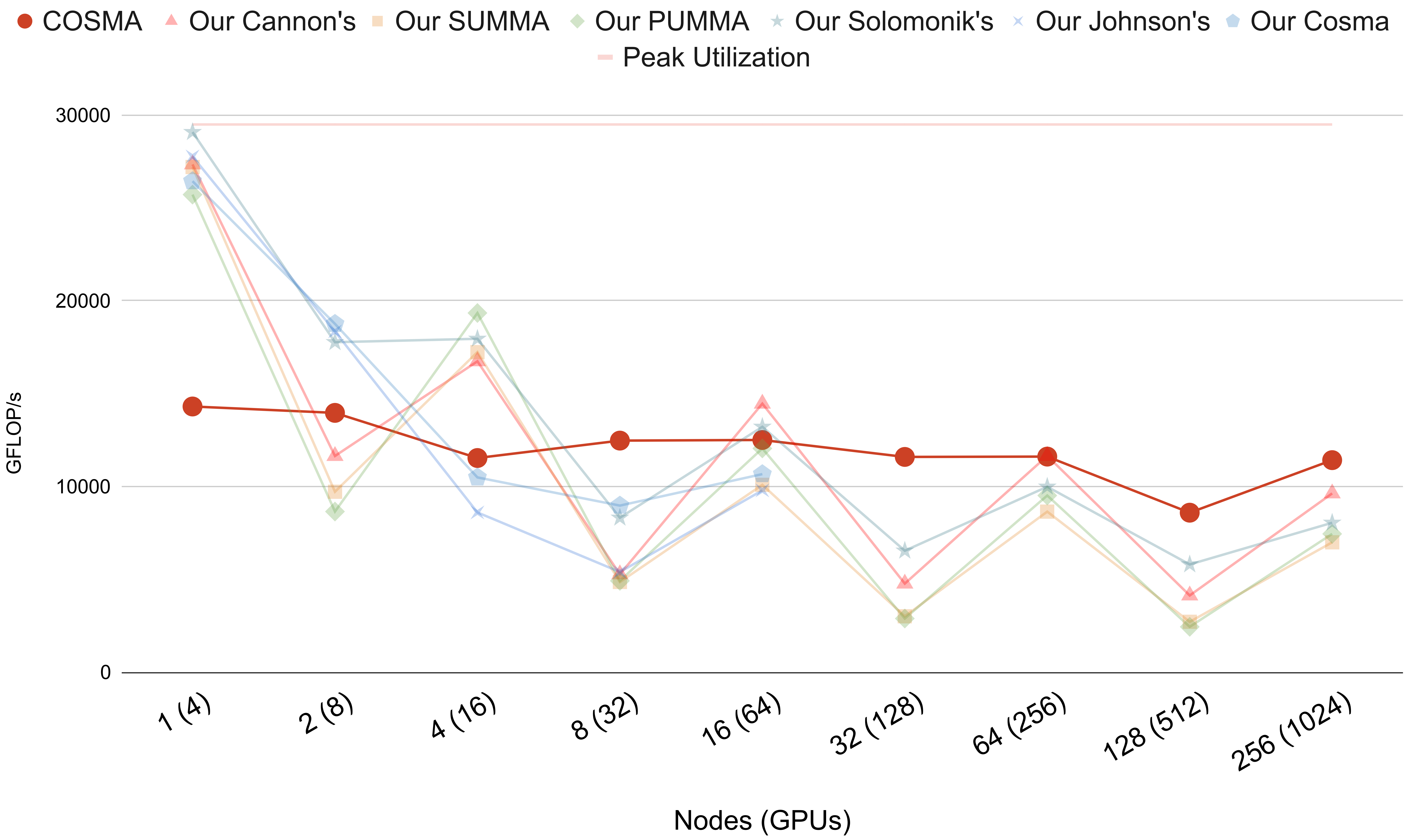
GEMM (CPU)


COSMA
CTF
ScaLAPACK
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our COSMA
Peak Utilization
800
600
400
200
0
GFLOP/s
1 (40)
2 (80)
4 (160)
8 (320)
16 (640)
32 (1280)
64 (2560)
128 (5120)
256 (10240)
Nodes (CPU Cores)

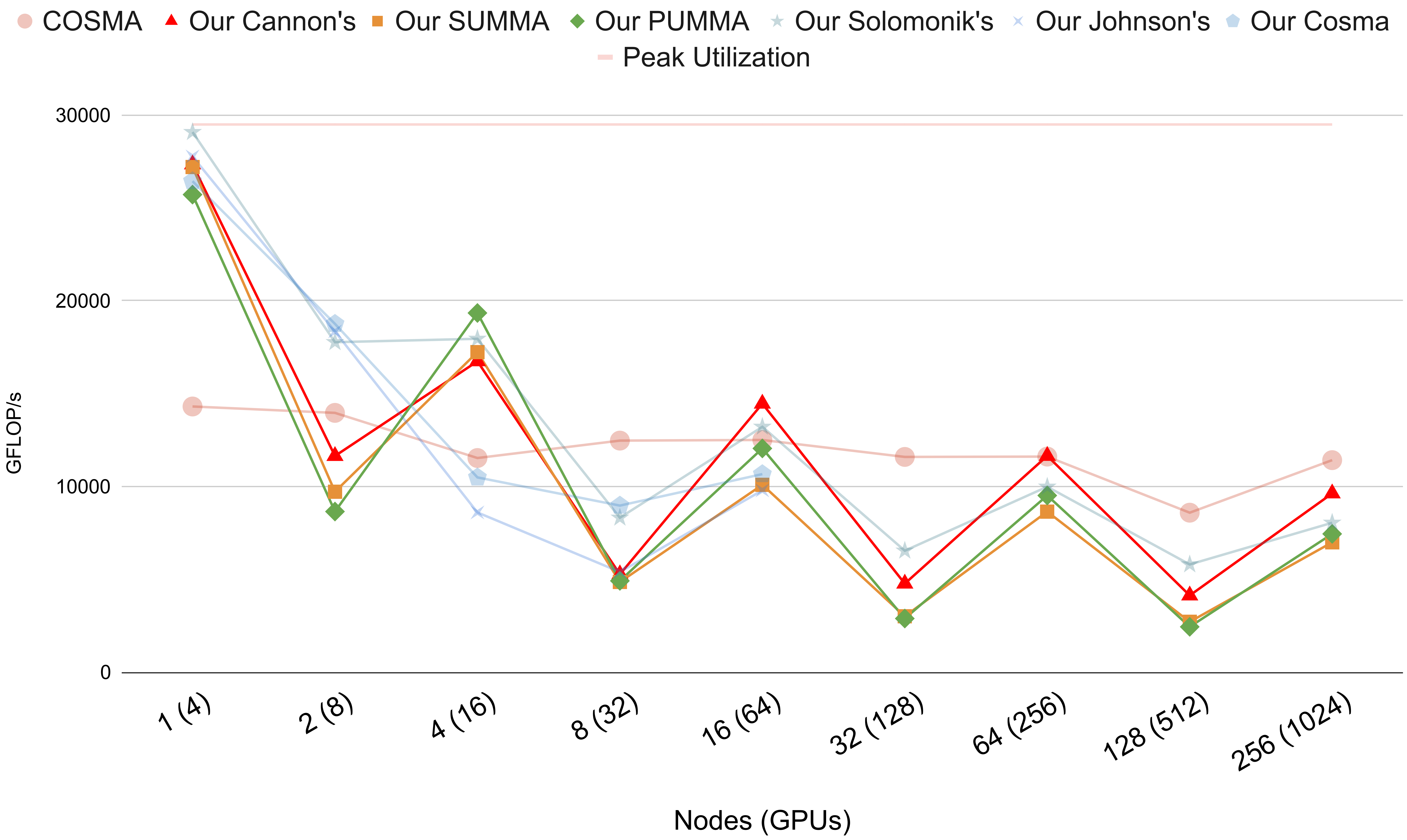
GEMM (CPU)


COSMA
CTF
ScaLAPACK
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our COSMA
Peak Utilization
800
600
400
200
0
GFLOP/s
1 (40)
2 (80)
4 (160)
8 (320)
16 (640)
32 (1280)
64 (2560)
128 (5120)
256 (10240)
Nodes (CPU Cores)
0.95x COSMA's performance
1.25x ScaLAPACK performance

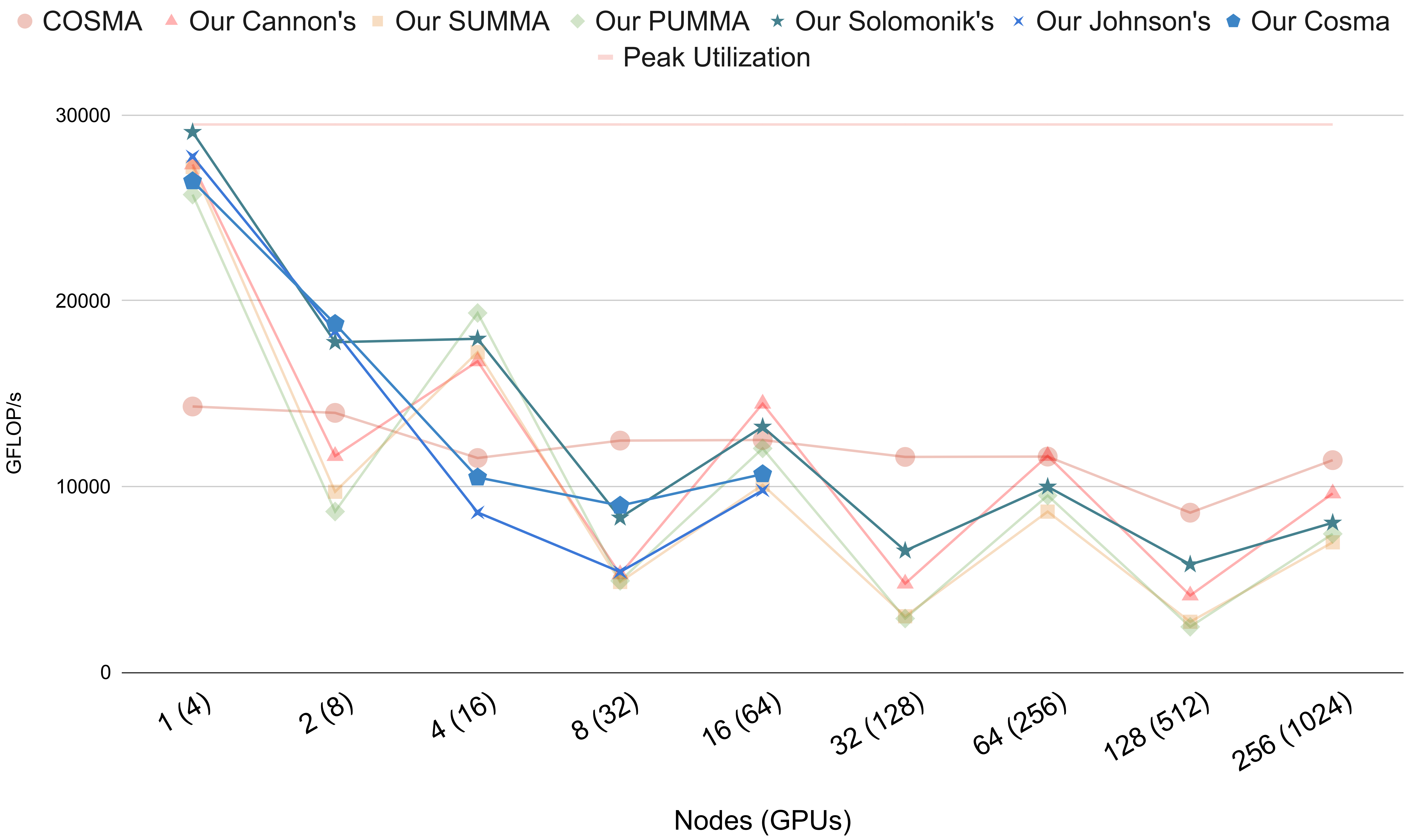
GEMM (GPU)


COSMA
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our Cosma
Peak Utilization
30000
20000
10000
0
GFLOP/s
1 (4)
2 (8)
4 (16)
8 (32)
16 (64)
32 (128)
64 (256)
128 (512)
256 (1024)
Nodes (GPUs)

GEMM (GPU)

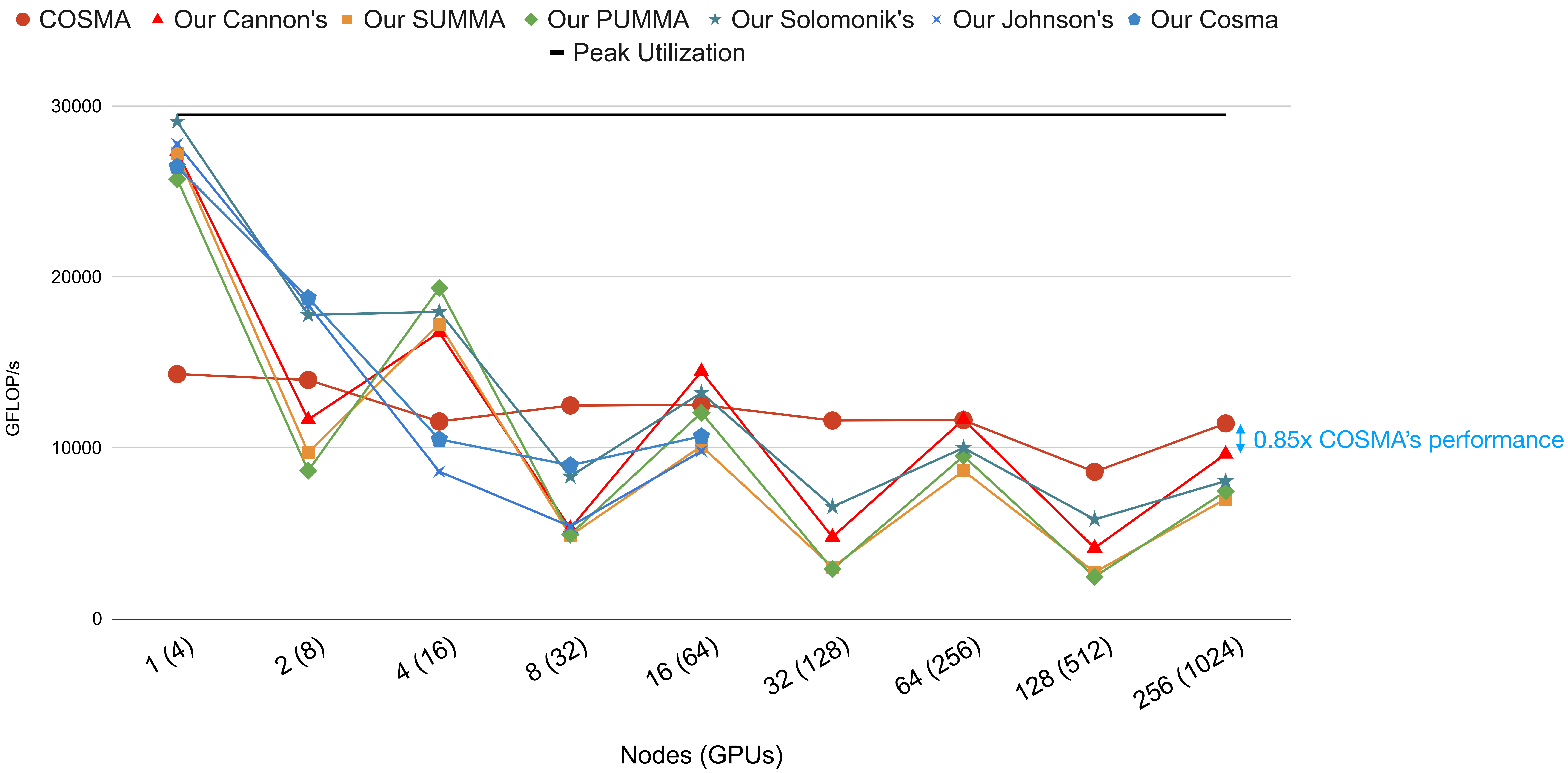

COSMA
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our Cosma
Peak Utilization
30000
20000
10000
0
GFLOP/s
1 (4)
2 (8)
4 (16)
8 (32)
16 (64)
32 (128)
64 (256)
128 (512)
256 (1024)
Nodes (GPUs)

GEMM (GPU)



COSMA
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our Cosma
Peak Utilization
30000
20000
10000
0
GFLOP/s
1 (4)
2 (8)
4 (16)
8 (32)
16 (64)
32 (128)
64 (256)
128 (512)
256 (1024)
Nodes (GPUs)

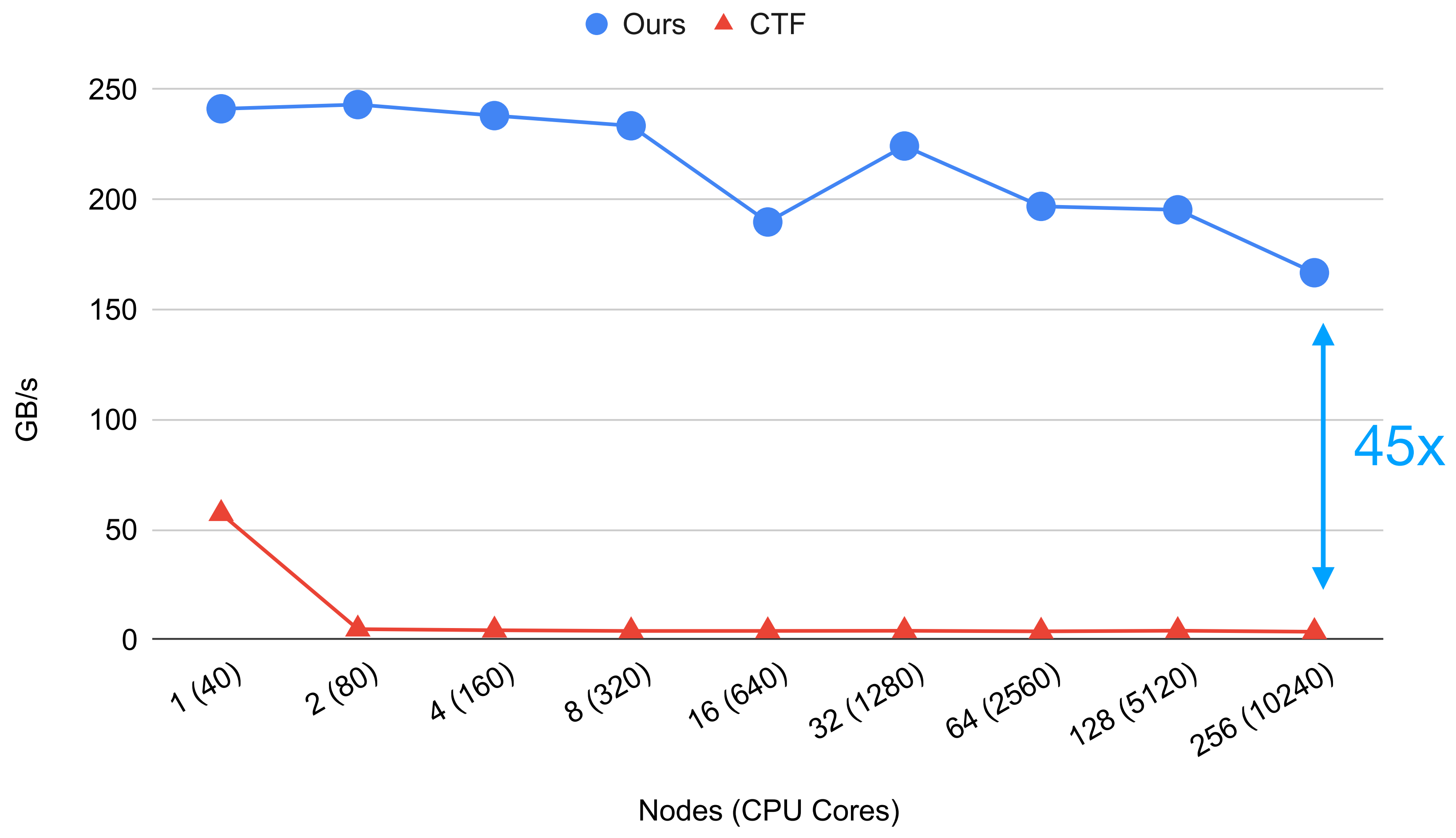
GEMM (GPU)


COSMA
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our Cosma
Peak Utilization
30000
20000
10000
0
GFLOP/s
0.85x COSMA's performance
1 (4)
2 (8)
4 (16)
8 (32)
16 (64)
32 (128)
64 (256)
128 (512)
256 (1024)
Nodes (GPUs)

Higher Order Tensor Operations (CPU)

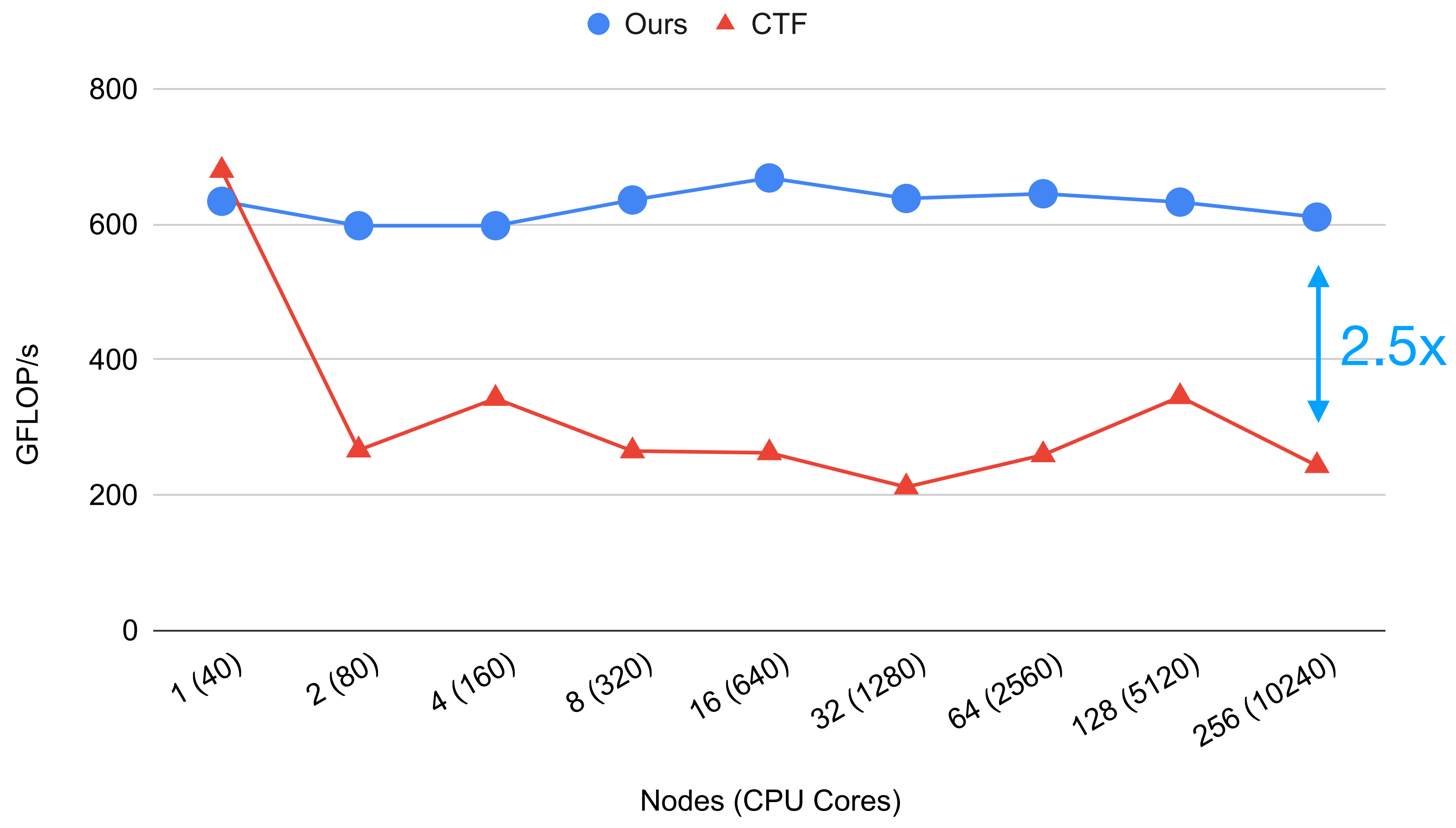
TTV

$$A_{ij} = B_{ijk} \cdot C_k$$



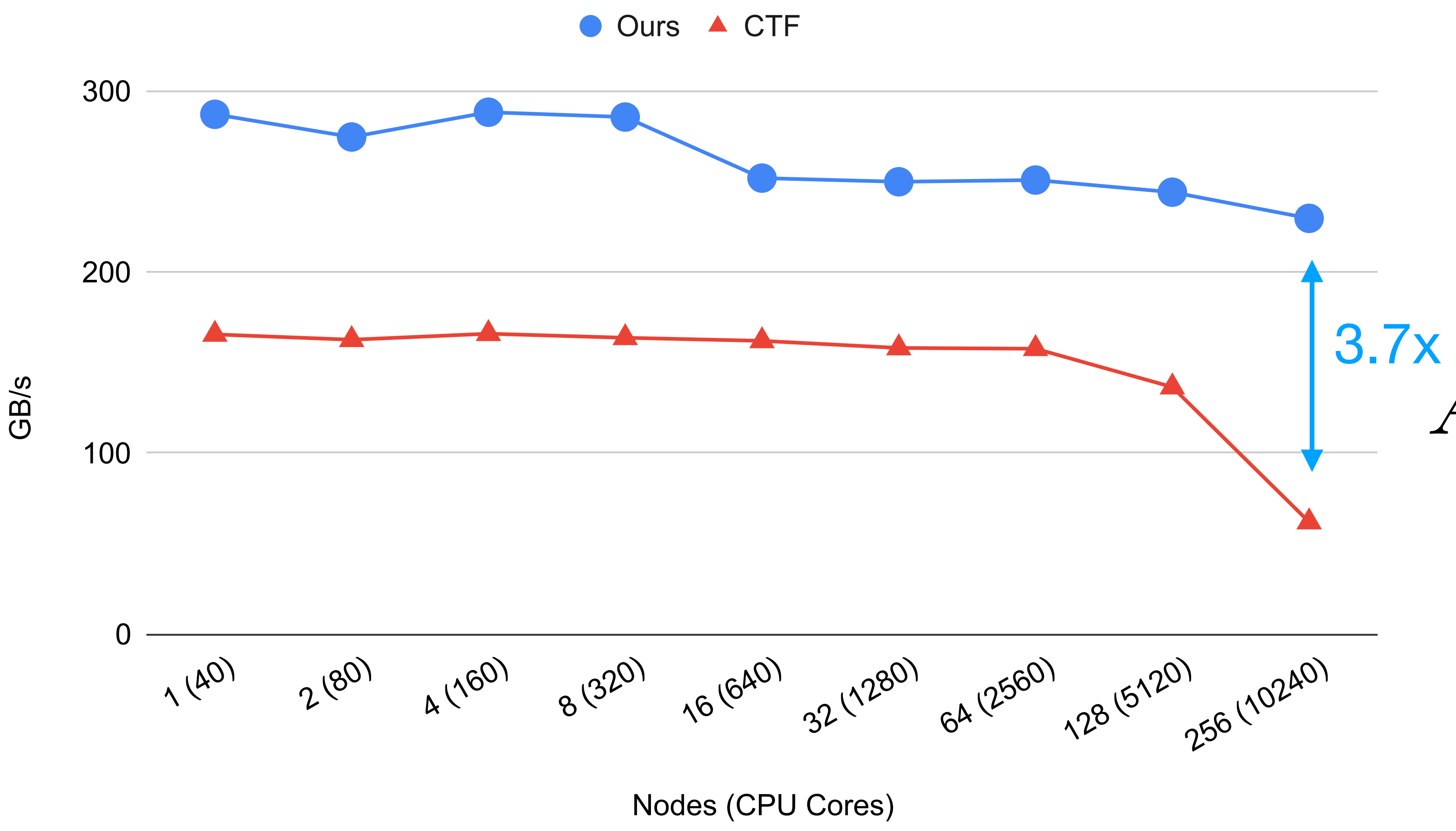
TTM

$$A_{ijl} = B_{ijk} \cdot C_{kl}$$



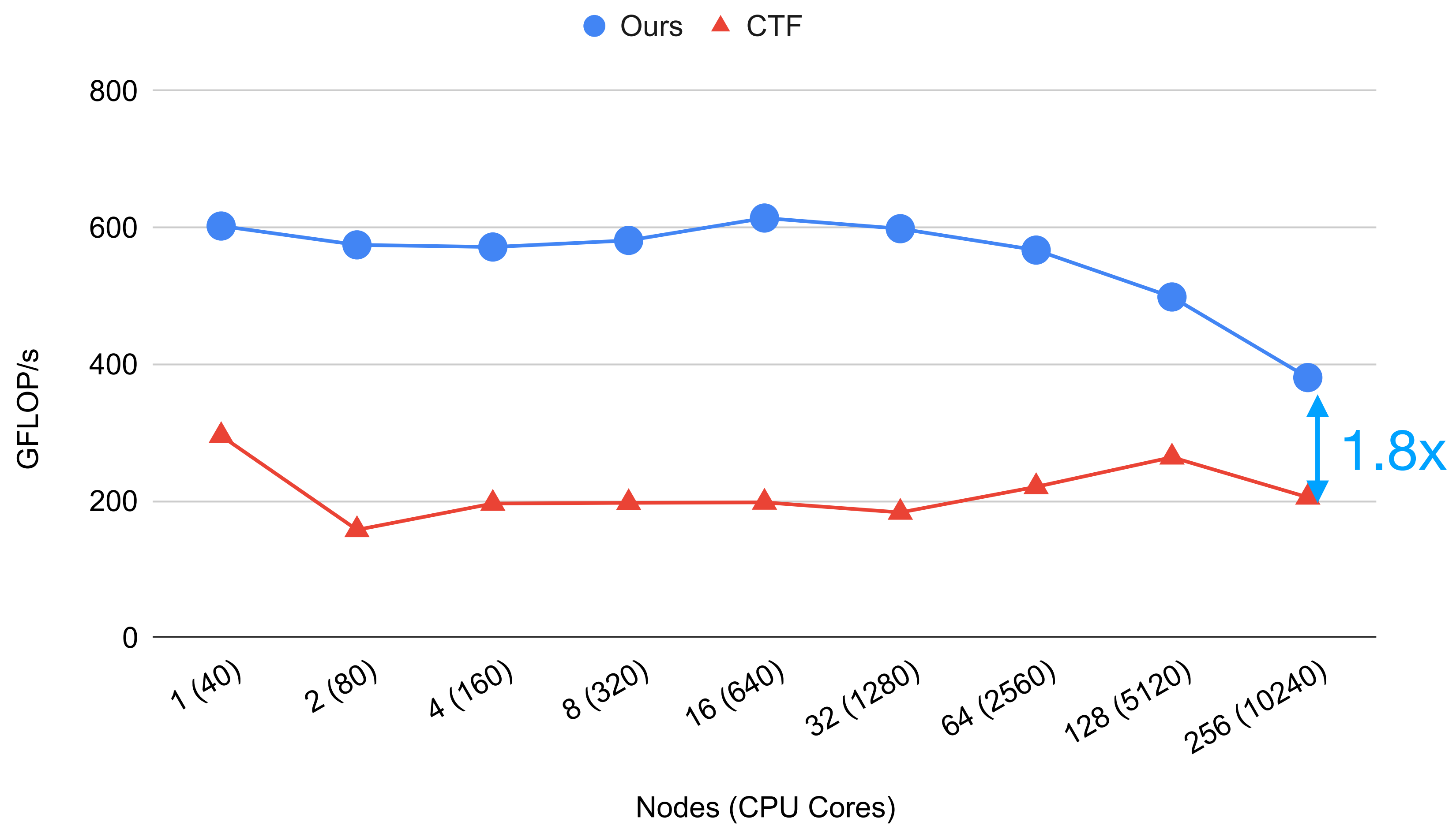
InnerProd

$$a = B_{ijk} \cdot C_{ijk}$$



MTTKRP

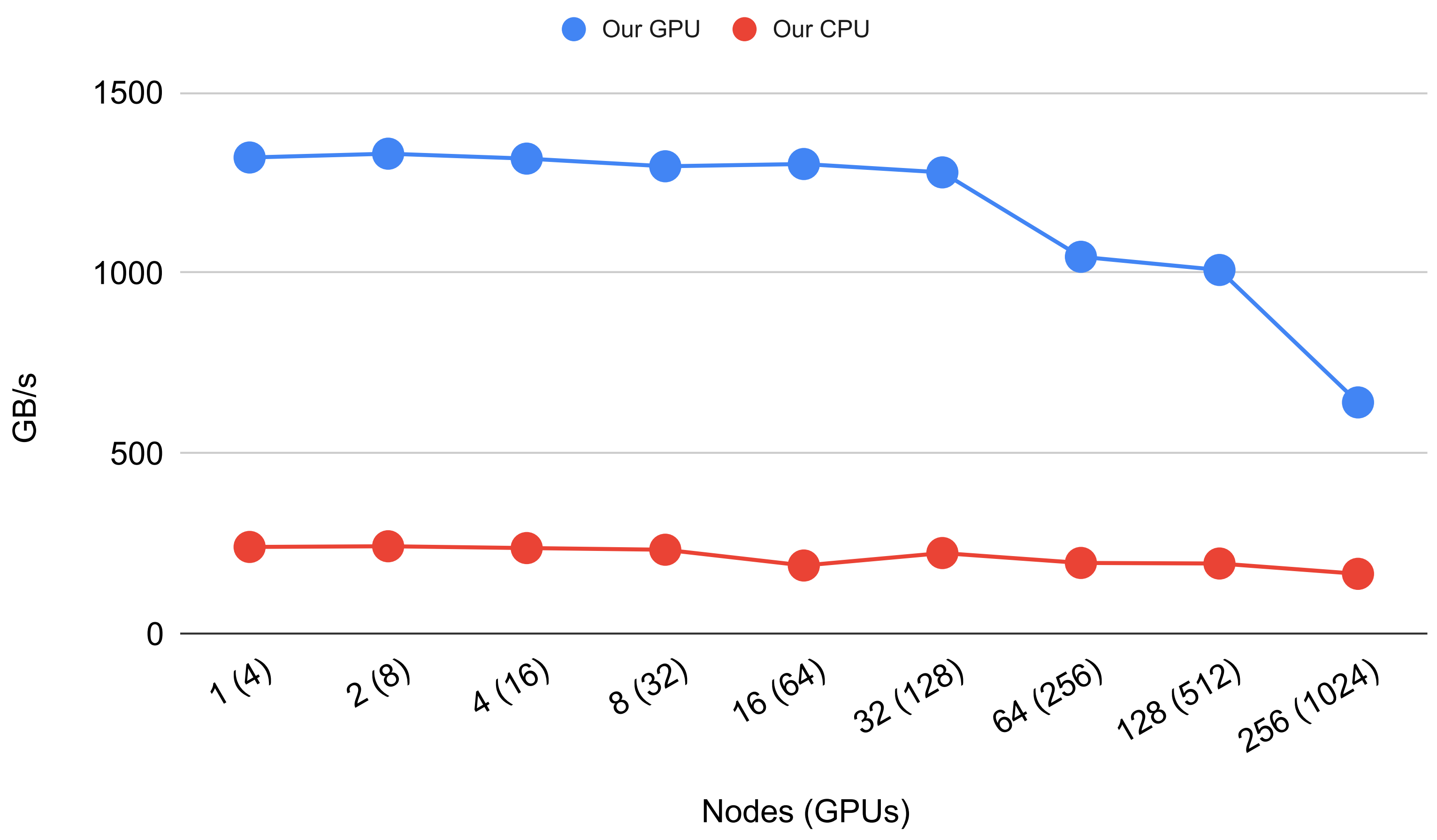
$$A_{il} = B_{ijk} \cdot C_{jl} \cdot D_{kl}$$



Higher Order Tensor Operations (GPU)

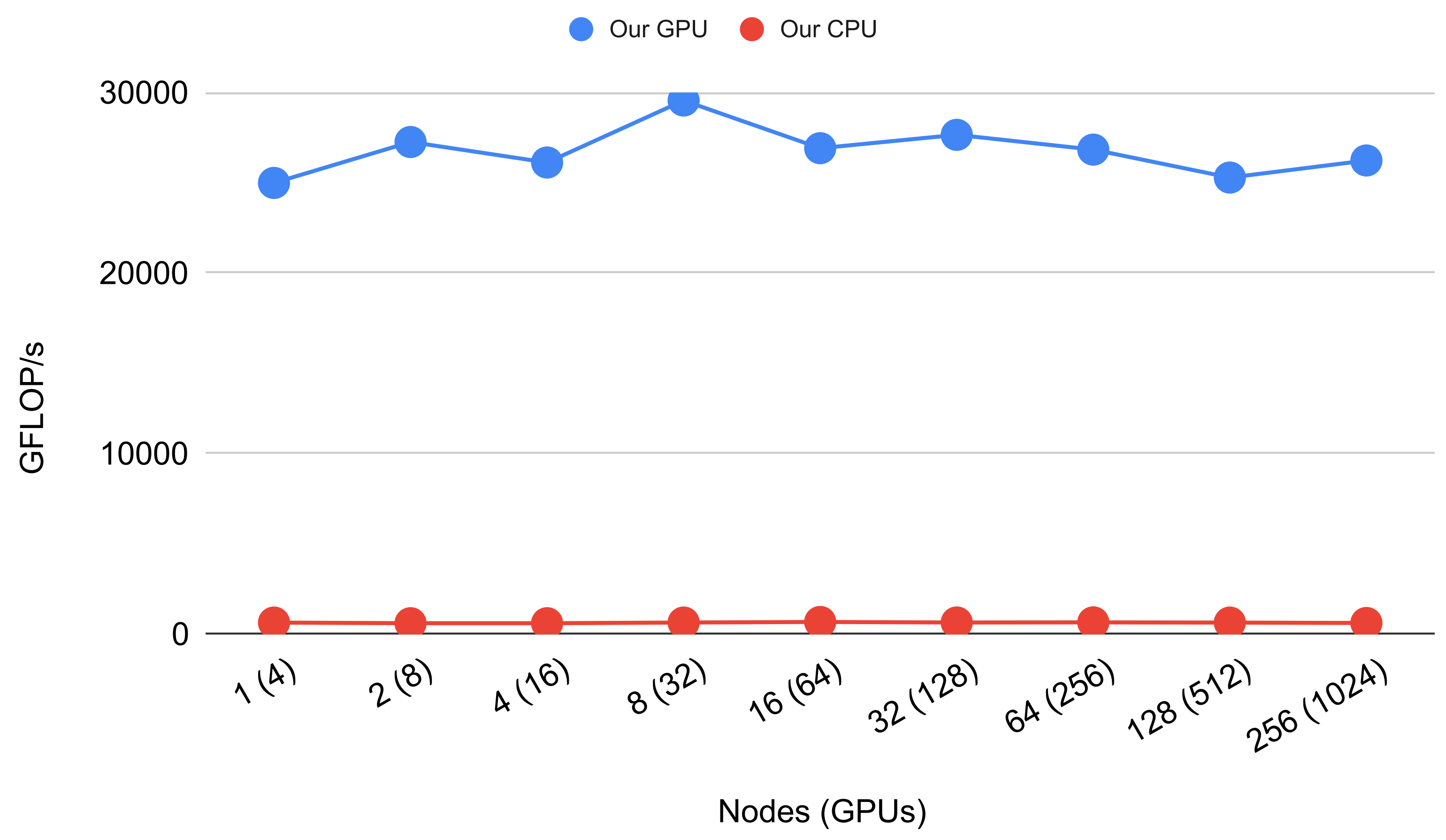
TTV

$$A_{ij} = B_{ijk} \cdot C_k$$



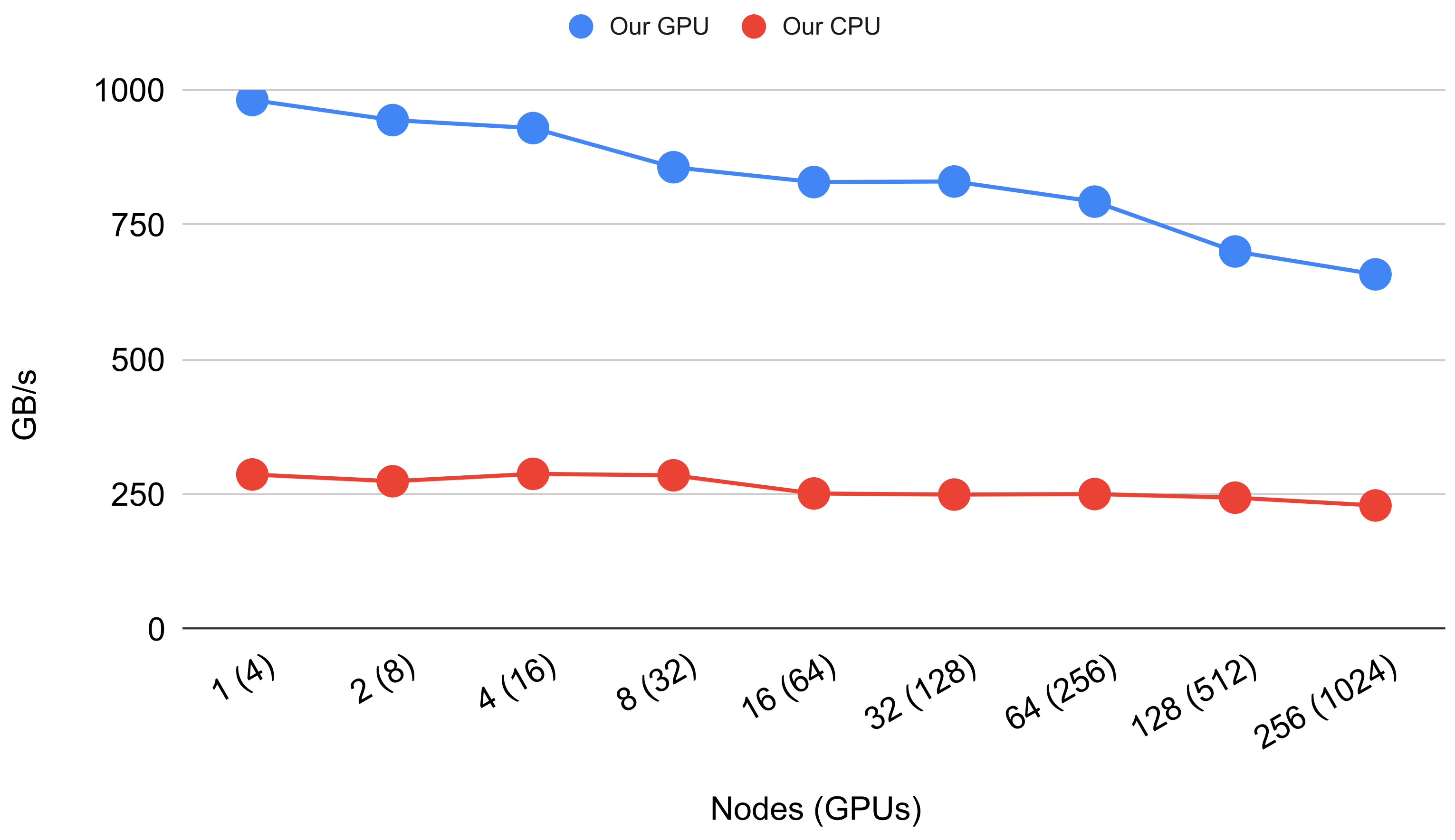
TTM

$$A_{ijl} = B_{ijk} \cdot C_{kl}$$



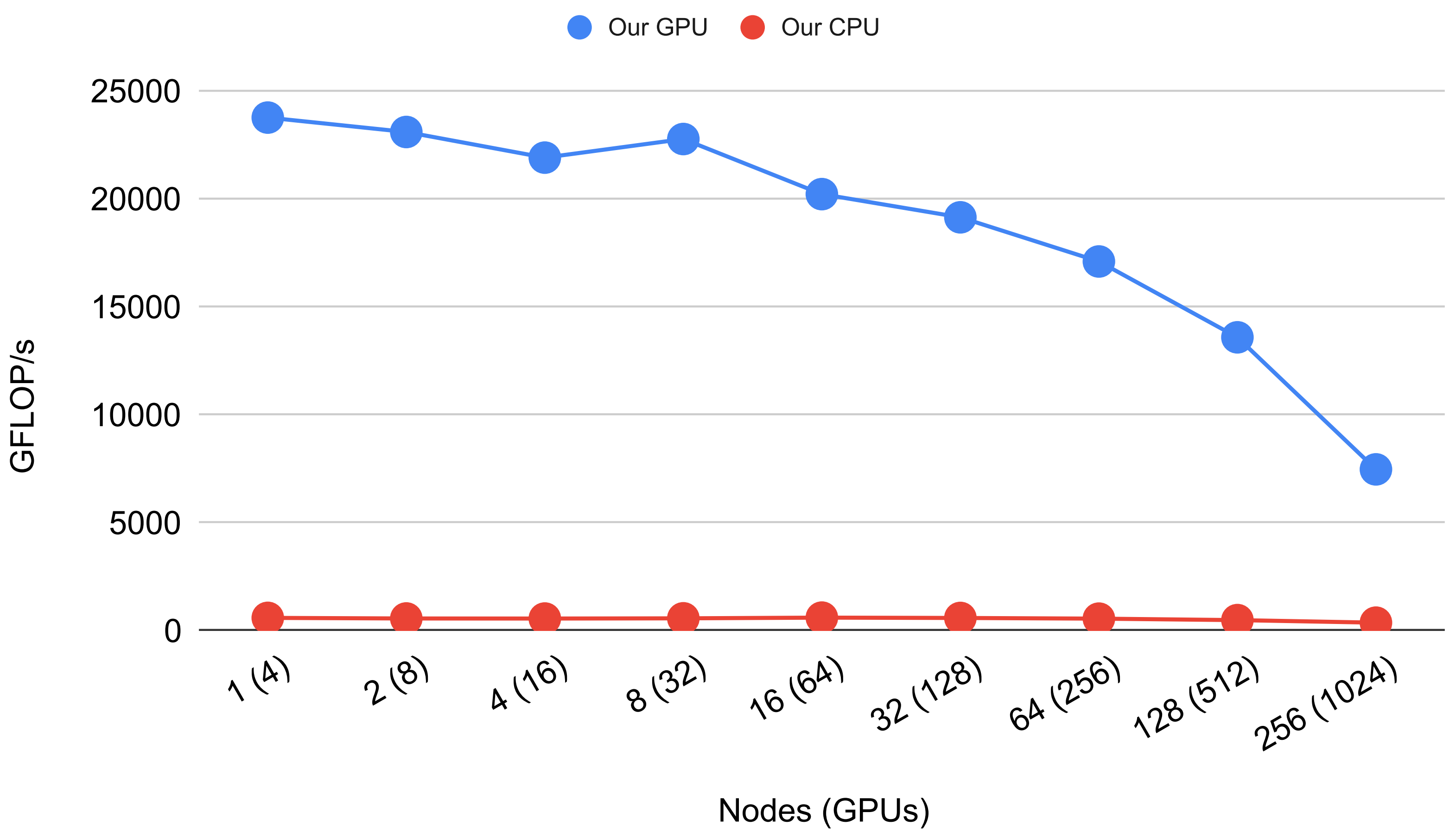
InnerProd

$$a = B_{ijk} \cdot C_{ijk}$$



MTTKRP

$$A_{il} = B_{ijk} \cdot C_{jl} \cdot D_{kl}$$



Conclusion

DISTAL combines separate specifications of data and computation distribution

DISTAL can represent many existing algorithms

DISTAL can achieve high performance

Future work — extending to sparse tensors.
Vision: distributed implementations of ANY tensor program with ANY tensor formats!

Contact: rohany@cs.stanford.edu

Extra slides

DISTAL

- Decouple computation, performance optimizations, and data distribution
- Extension to TACO

Einsum notation programs

×

Format-based data distribution

×

Schedule for compute distribution

=

Too many to code by hand!

Expression

$$A(i,j) = B(i,k) \cdot C(k,j)$$
$$A(i,l) = B(i,j,k) \cdot C(j,l) \cdot D(k,l)$$
$$a = B(i,j,k) \cdot C(i,j,k)$$
$$A(i,j,l) = B(i,j,k) \cdot C(k,l)$$
$$A(i,j) = B(i,j,k) \cdot c(k)$$

Data Distribution

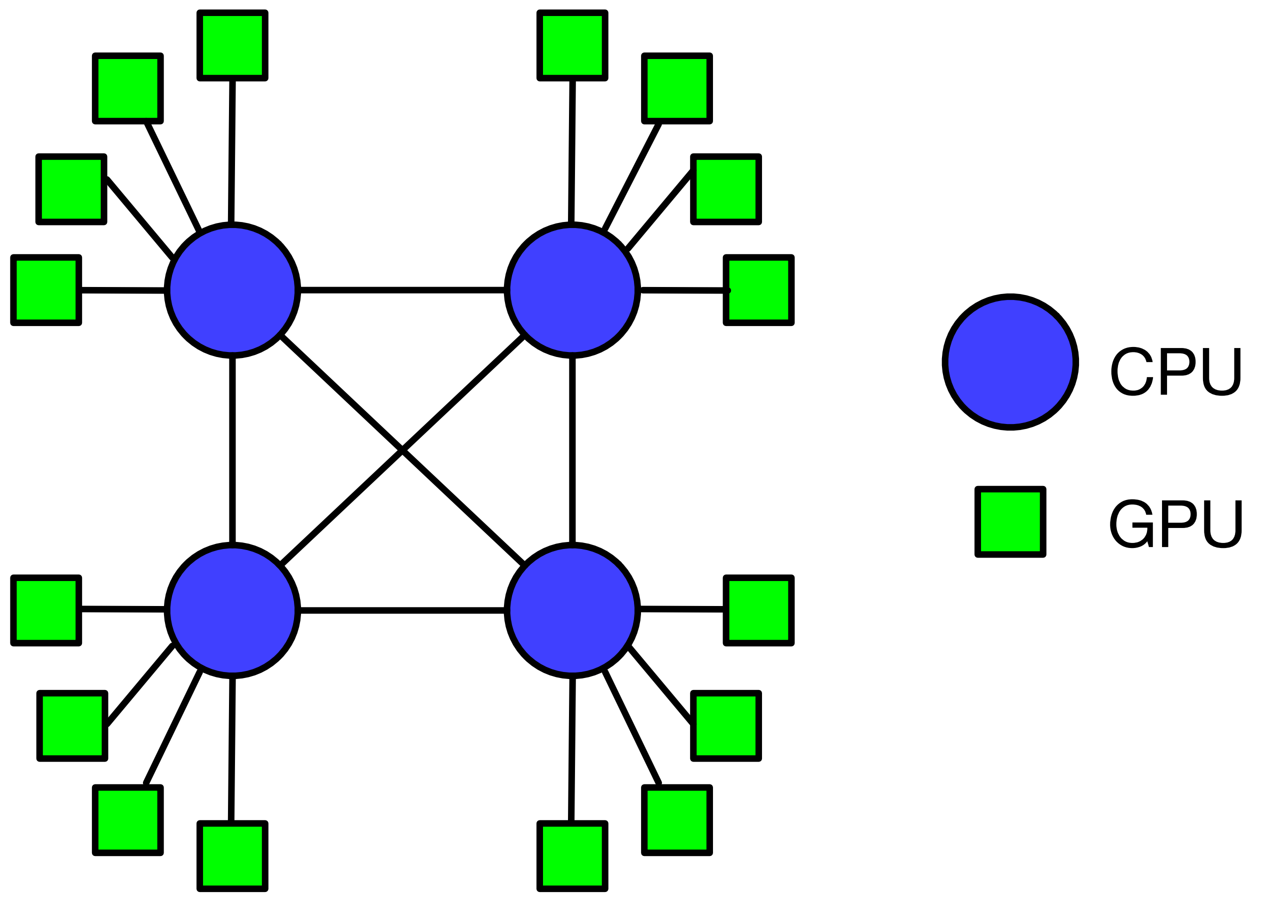
Partition A into tiles
Replicate B onto all nodes
Place C onto only some nodes

Computation Distribution

Owner Computes
Distribute i,j loops
Communicate in chunks

→ DISTAL →

Supercomputer



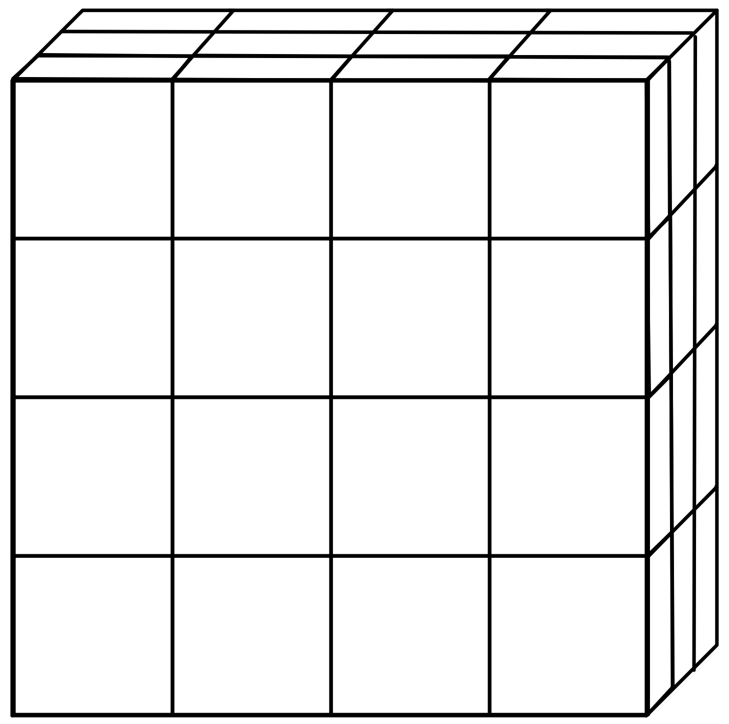
Legion

Handles many features necessary for performance on modern machines

- Overlap of communication and computation
- Data movement through deep memory hierarchies
- Native support for accelerators
- Control over placement of computation and data

Legion API

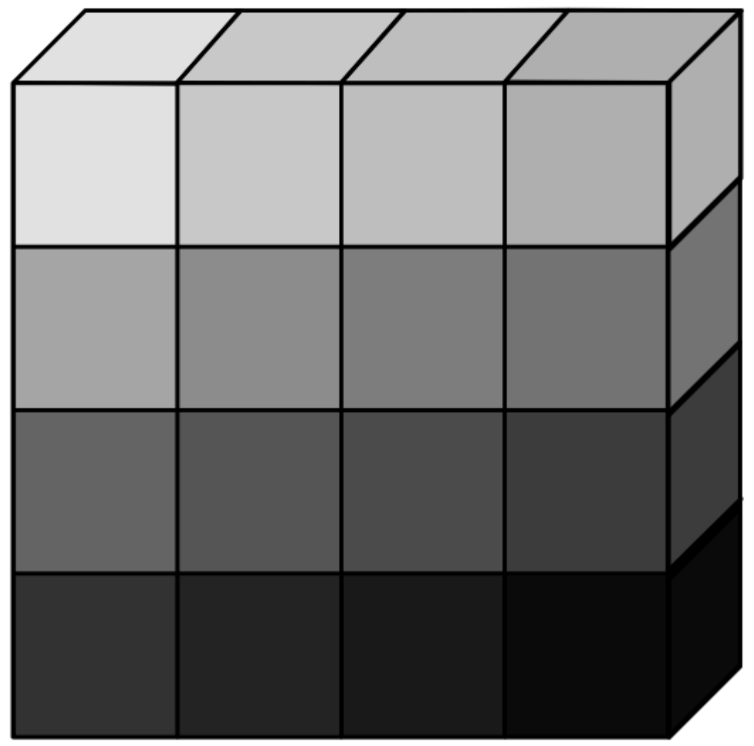
Regions



Tasks

```
void task(r1 : Region, r2 : Region, r3 : Region) { … }
```

Partitions



$\mathcal{M}$

Mapping

Lowering Concrete Index Notation to Legion

$$\forall_i \ \forall_j \ A_{ij} \mathrel{+}= B_{ij}$$

$$\forall_{in}\forall_{jn}\forall_{il}\forall_{jl} \ A_{ij} \mathrel{+}= B_{ij} \text{ s.t. } \text{div.(i, in, il, gx), div.(j, jn, jl, gy), dist.([in, jn]), comm.([A, B], jn)}$$

```
for in:
  for jn:
    for il:
      for jl:
```

Lowering Concrete Index Notation to Legion

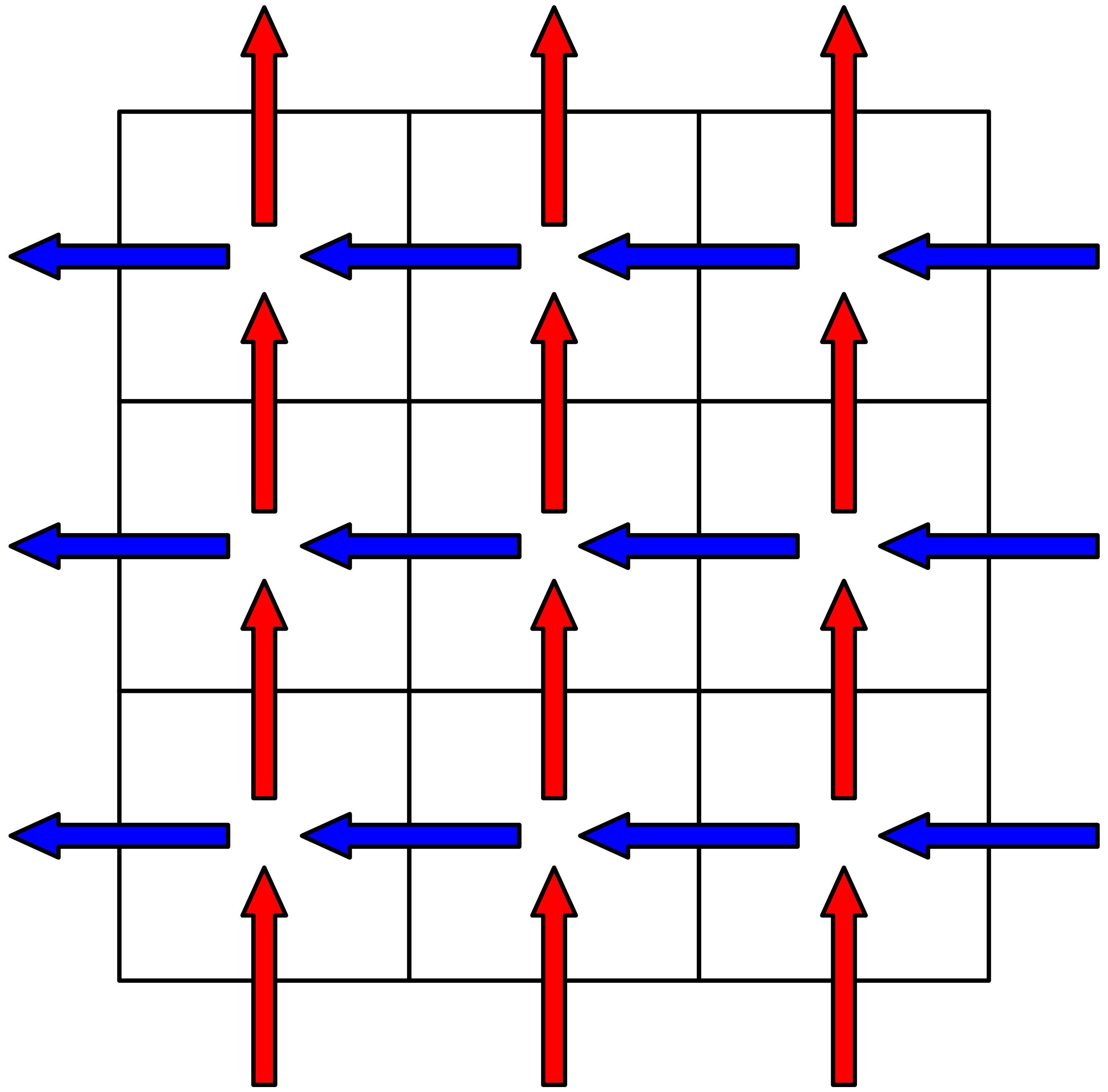
$\forall_{in}\forall_{jn}\forall_{il}\forall_{jl}\ A_{ij} \mathrel{+}= B_{ij}$ s.t. div.(i, in, il, gx), div.(j, jn, jl, gy), dist.([in, jn]), comm.([A, B], jn)

```
ind
  A
  B
  f
```

```
art):
```

```
for in:
  for jn:
    for il:
      for jl:
        i = in * (il.hi - il.lo) + il
        j = jn * (jl.hi - jl.lo) + jl
        A(i, j) += B(i, j)
```

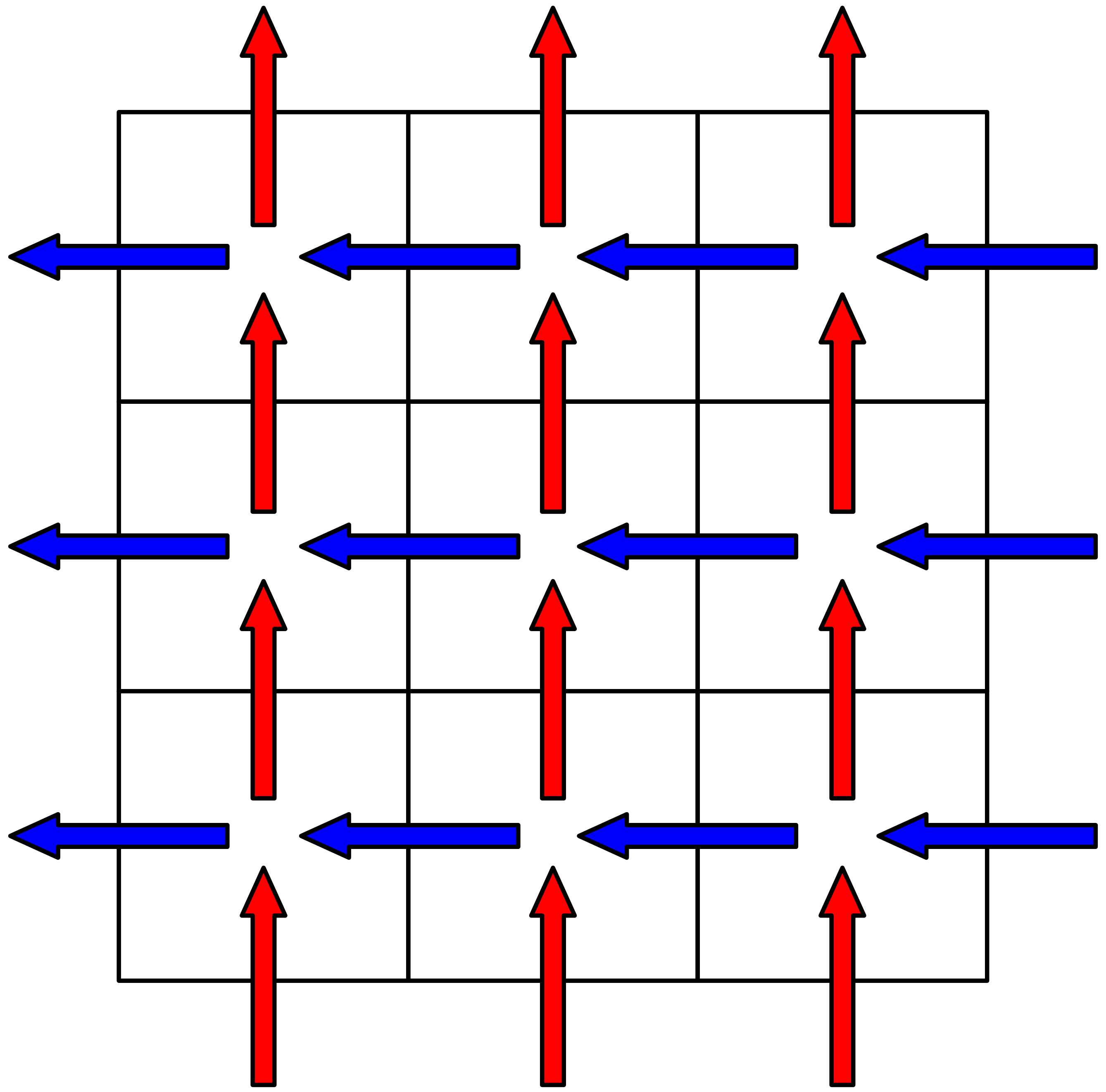
Cannon’s Algorithm



```
# Arrange $p$ processors into a 2D grid, $\sqrt{p} \times \sqrt{p}$.
# Assign a tile of $A$, $B$, $C$ to each processor.
# Perform an initial data shift.
for all $P_{ij}$ in parallel:
  shift $B_{ij}$ $i$ spaces to the left
  shift $C_{ij}$ $j$ spaces upwards
for all $P_{ij}$ in parallel:
  for k in (0, $\sqrt{p}$):
    $A_{ij}$ += $B_{ij}$ * $C_{ij}$
    shift $B_{ij}$ to the left
    shift $C_{ij}$ upwards
```

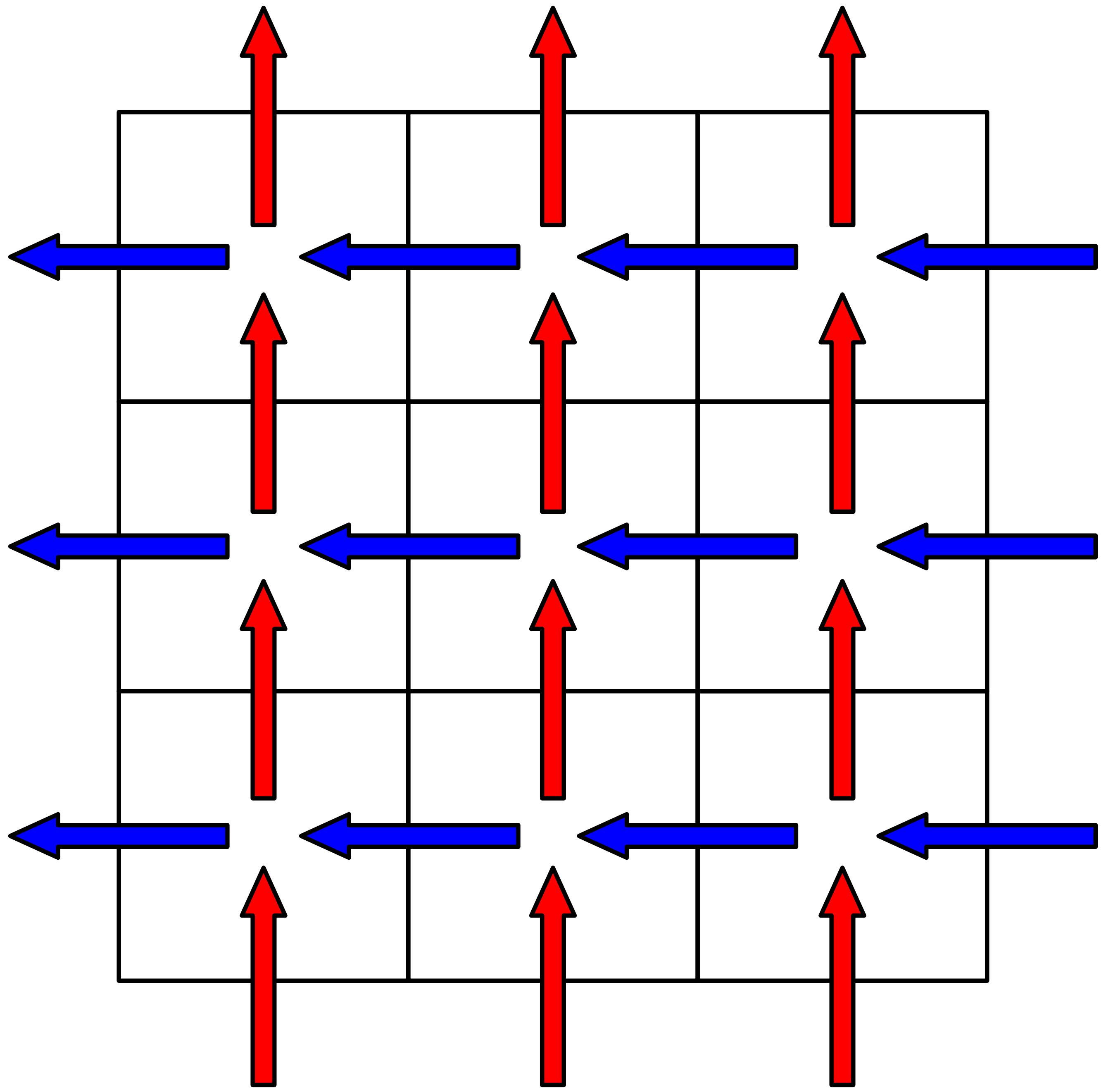
```
divide(i, il, in, gx)
divide(j, jl, jn, gx)
reorder({in, jn, il, jl})
split(k, ko, ki, chunkSize)
reorder(ko, il, jl, ki)
distribute(in, jn)
communicate(A, jn)
communicate({B, C}, ko)
```

```
distributed for in, jn:
  communicate A
  for ko:
    communicate B, C
    for il:
      for jl:
        for ki:
          A(i, j) += B(i, k) * C(k, j)
```



```
distributed for in, jn:
  communicate A
  for kr:
    communicate B, C
    for il:
      for jl:
        for ki:
          A(i, j) += B(i, k) * C(k, j)
```

```
divide(i, in, il, gx)
divide(j, jn, jl, gy)
divide(k, ko, ki, gx)
rotate(ko, {in, jn}, kr)
reorder({in, jn, kr, il, jl, ki})
distribute(in, jn)
communicate(A, jn)
communicate({B, C}, kr)
```

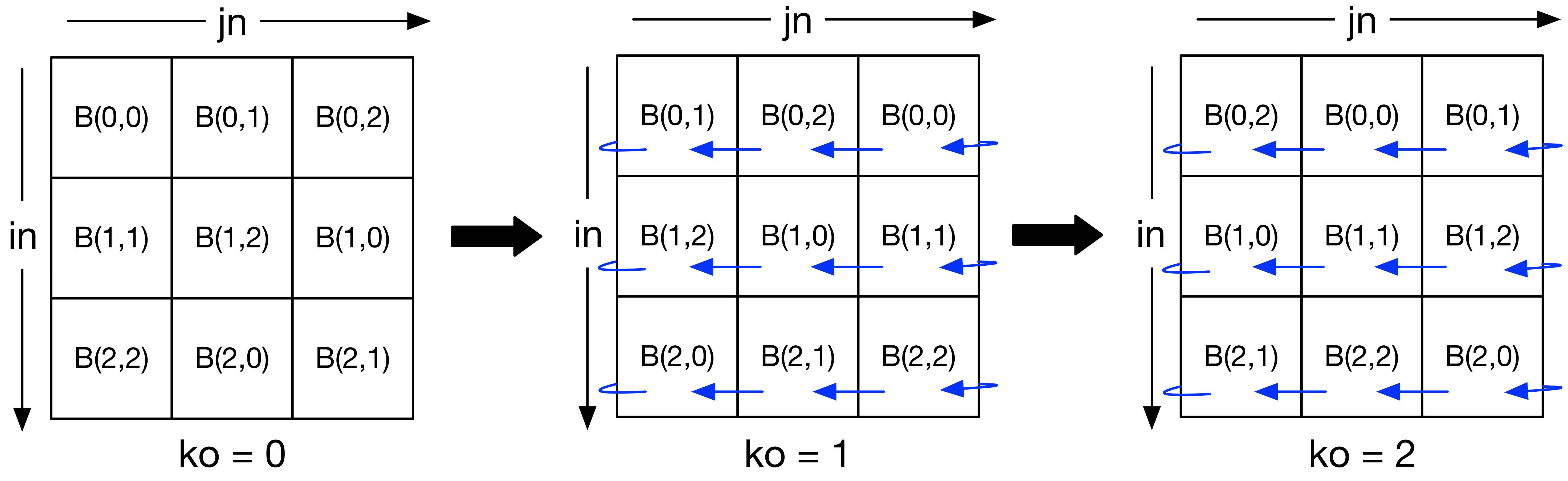


Emergence of systolic communication

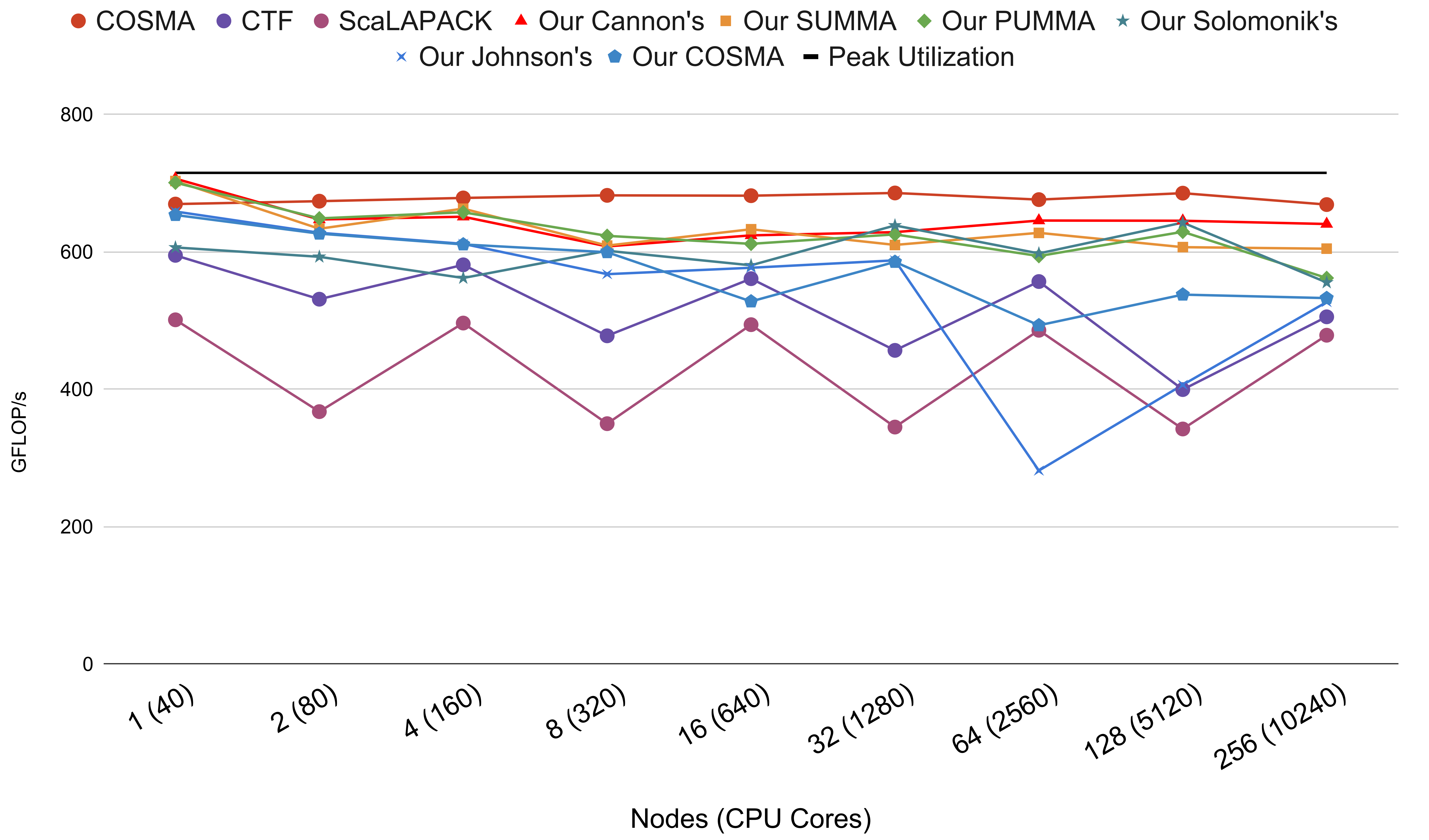
$A(i, j) = B(i, k) * C(k, j)$

$kr = ko + in + jn \bmod 3$

```
distributed for in, jn:
  communicate A
  for kr:
    communicate B, C
    for il:
      for jl:
        for ki:
          A(i, j) += B(i, k) * C(k, j)
```



GEMM (CPU)


COSMA
CTF
ScaLAPACK
Our Cannon's
Our SUMMA
Our PUMMA
Our Solomonik's
Our Johnson's
Our COSMA
Peak Utilization
800
600
400
200
0
GFLOP/s
1 (40)
2 (80)
4 (160)
8 (320)
16 (640)
32 (1280)
64 (2560)
128 (5120)
256 (10240)
Nodes (CPU Cores)