TASK-BASED PROGRAMMING MODELS FOR SCALABLE DISTRIBUTED MEMORY PARALLEL ALGORITHMS

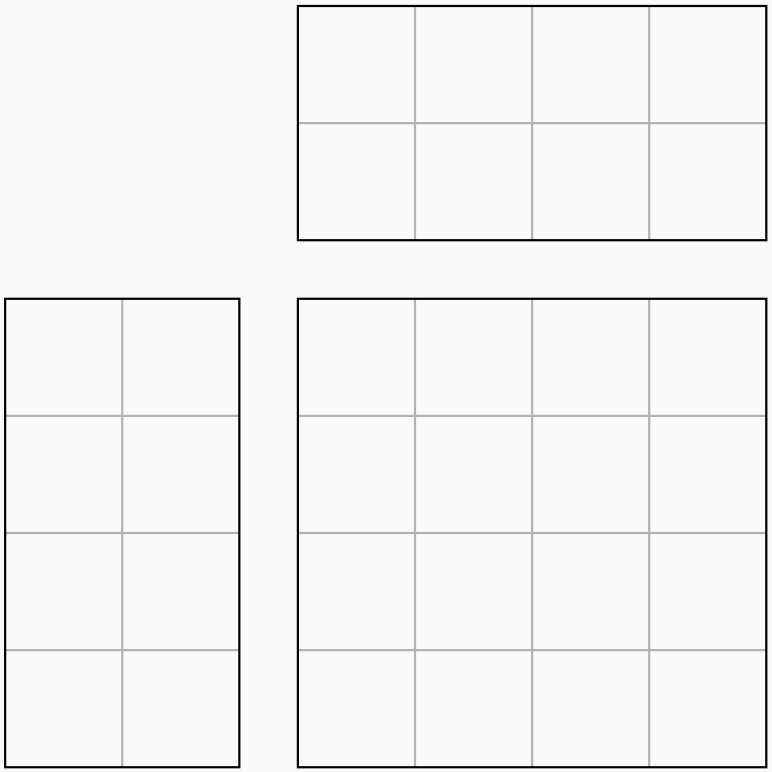
E. Agullo, A. Buttari, A. Guermouche, J. Herrmann, A. Jego

December 4, 2020

Université de Toulouse - IRIT

GENERAL MATRIX MULTIPLICATION ALGORITHMS

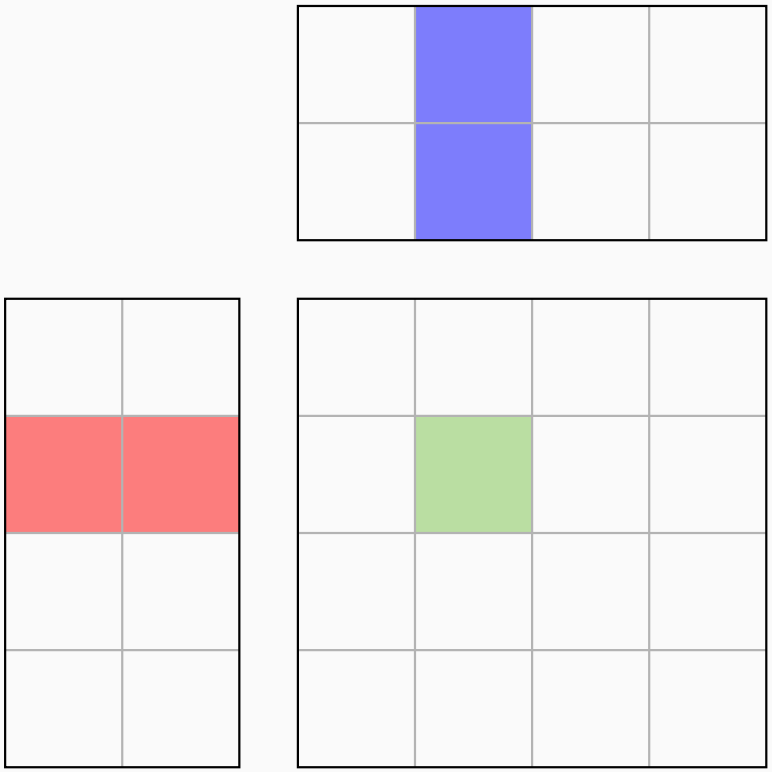
Sequential GEMM



```
do i=1,m
  do j=1,n
    do l=1,k
      call gemm( A[i,l], B[l,j], C[i,j])
    end do
  end do
end do
```

$$\forall i, j, C_{i,j} += \sum A_{i,k} B_{k,j}$$

Sequential GEMM



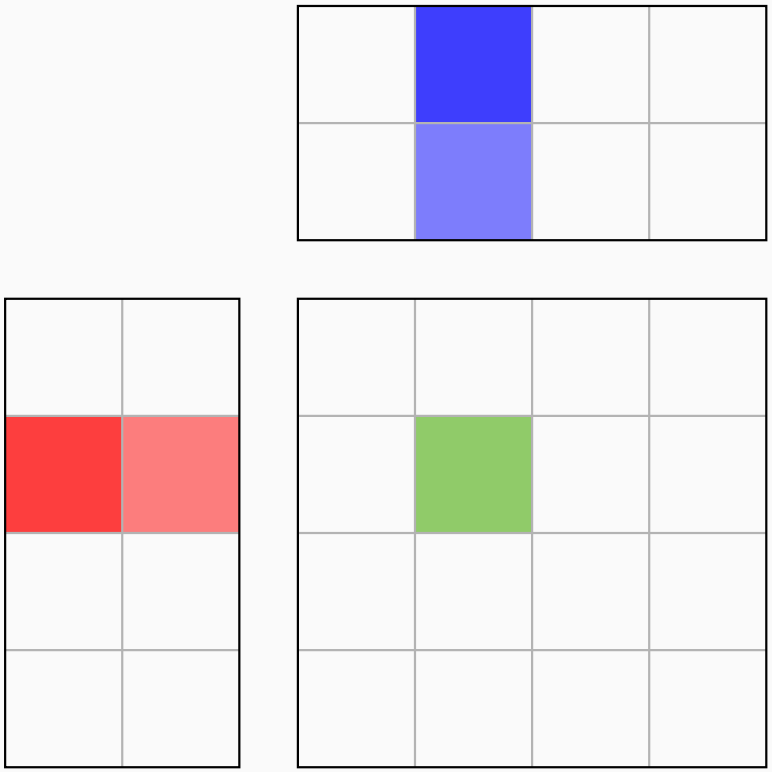
```
do i=1,m
  do j=1,n
    do l=1,k
      call gemm( A[i,l], B[l,j], C[i,j])
    end do
  end do
end do
```

$$\forall i, j, C_{i,j} += \sum A_{i,k} B_{k,j}$$

As an example

$C_{2,2} =$

Sequential GEMM



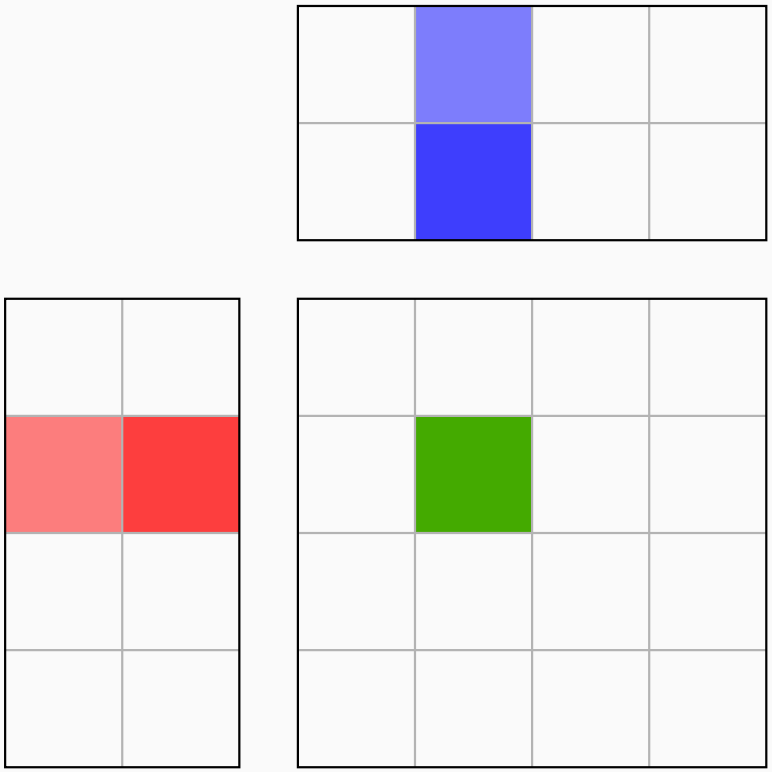
```
do i=1,m
  do j=1,n
    do l=1,k
      call gemm( A[i,l], B[l,j], C[i,j])
    end do
  end do
end do
```

$$\forall i, j, C_{i,j} += \sum A_{i,k} B_{k,j}$$

As an example

$$C_{2,2} = A_{2,1} * B_{1,2}$$

Sequential GEMM



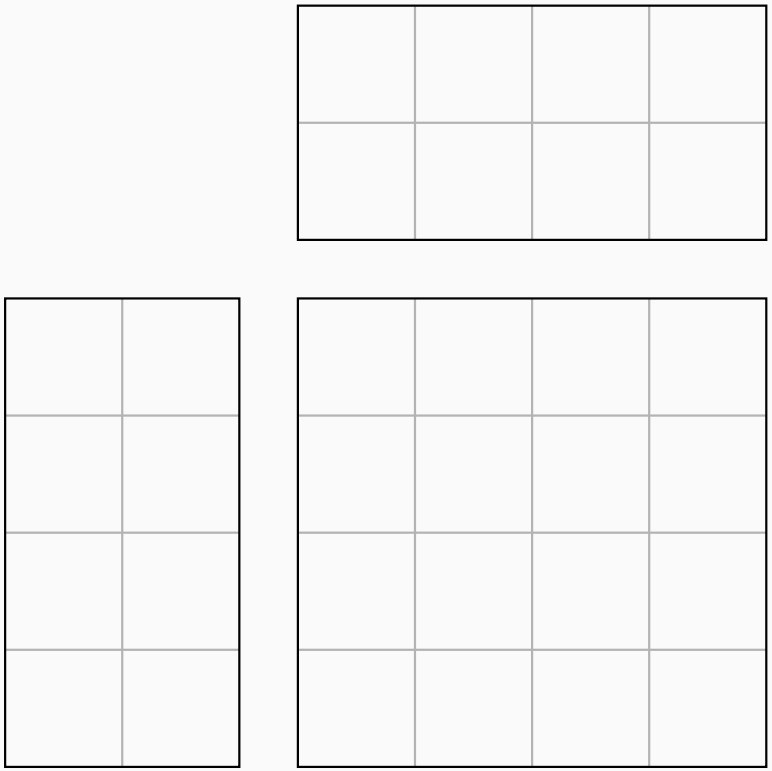
```
do i=1,m
  do j=1,n
    do l=1,k
      call gemm( A[i,l], B[l,j], C[i,j])
    end do
  end do
end do
```

$$\forall i,j, C_{i,j} += \sum A_{i,k} B_{k,j}$$

As an example

$$C_{2,2} = A_{2,1} * B_{1,2} + A_{2,2} * B_{2,2}$$

GEMM : distributed memory (Basic)



Ingredients

GEMM : distributed memory (Basic)

P0	P1	P2	P3
P4	P5	P6	P7

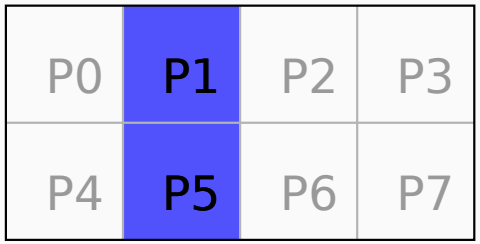
P0	P1
P4	P5
P8	P9
P12	P13

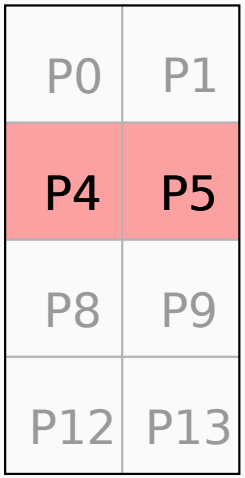
P0	P1	P2	P3
P4	P5	P6	P7
P8	P9	P10	P11
P12	P13	P14	P15

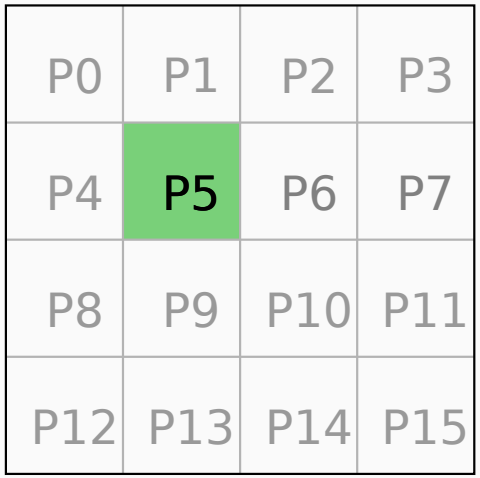
Ingredients

- Data Mapping - here, 2D Block Cyclic

GEMM : distributed memory (Basic)





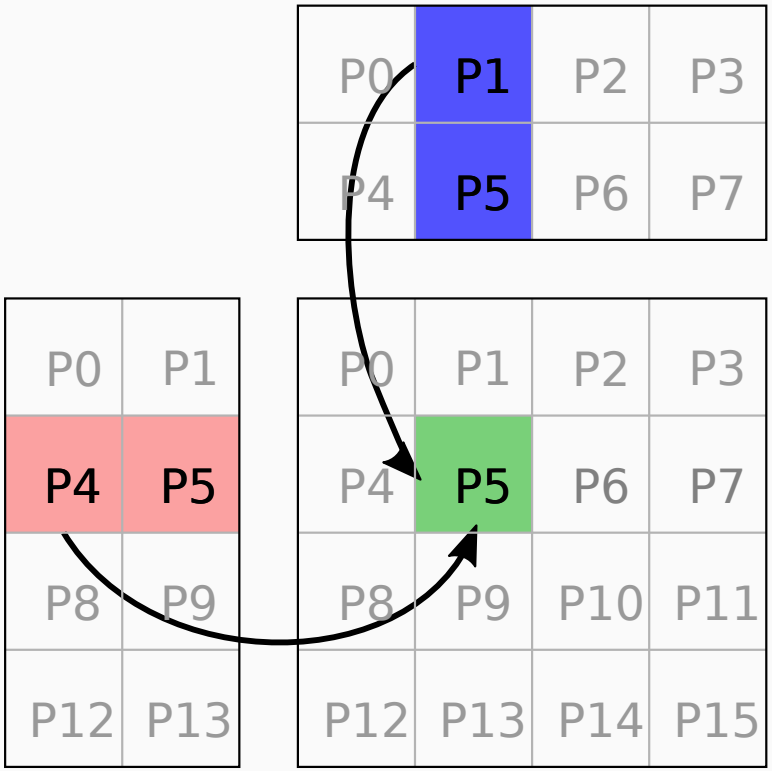


Ingredients

- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)

```
do k=1,2
  call Recv(A[i,k])
  call Recv(B[k,j])
  call gemm(A[i,k], B[k,j], C[i,j])
end do
```

GEMM : distributed memory (Basic)

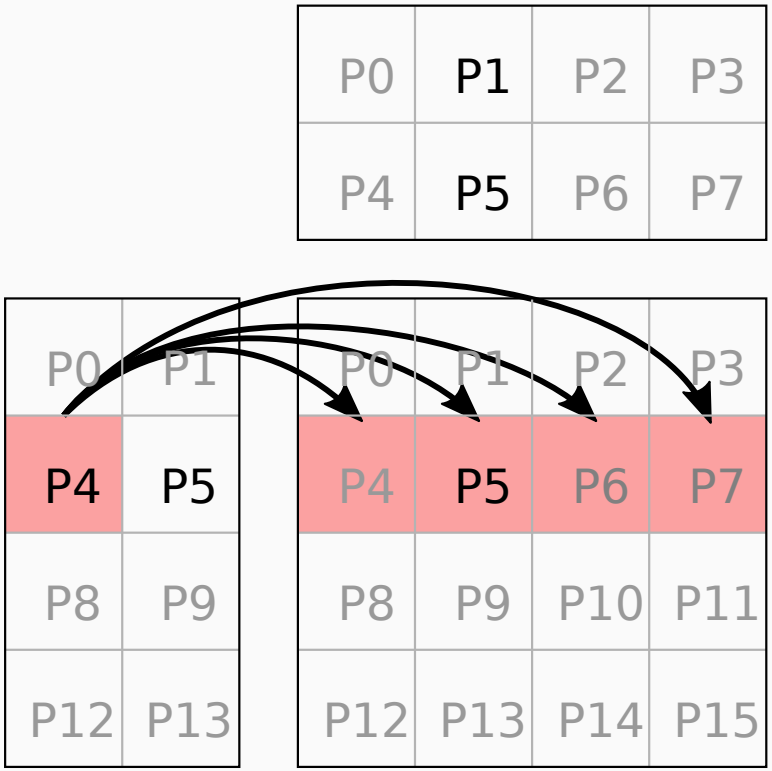


Ingredients

- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)
- Communicat°

```
do k=1,2
      call Send(A[i,j]) on my row
      call Send(B[i,j]) on my column
      call gemm(A[i,k],B[k,j],C[i,j])
end do
```

GEMM : DISTRIBUTED MEMORY (SUMMA)



Ingredients

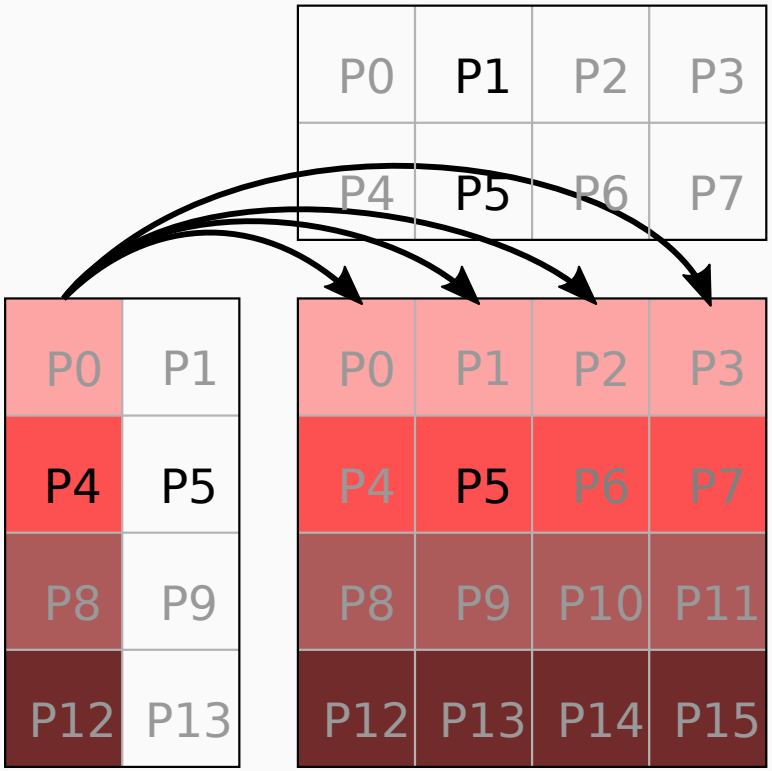
- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)
- Communicat° patterns

```
do k=1,2
      call Bcast(A[i,j]) on my row
      call Bcast(B[i,j]) on my column
      call gemm(A[i,k],B[k,j],C[i,j])
end do
```

Sender and receivers match color.

Sending block of *A* from P4

GEMM : DISTRIBUTED MEMORY (SUMMA)



Ingredients

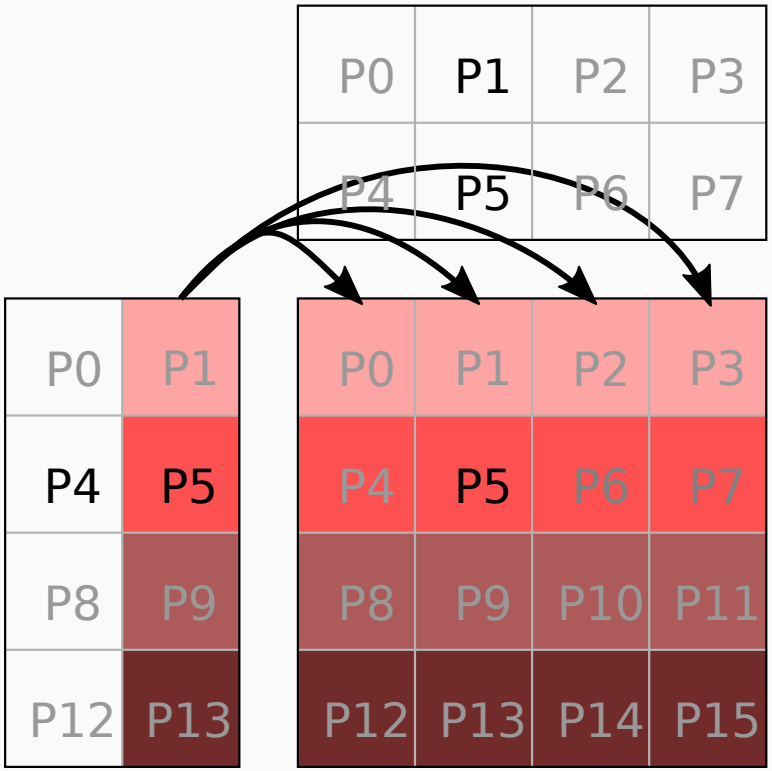
- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)
- Communicat° patterns

```
do k=1,2
      call Bcast(A[i,j]) on my row
      call Bcast(B[i,j]) on my column
      call gemm(A[i,k],B[k,j],C[i,j])
end do
```

Sender and receivers match color.

Sending blocks of *A* from **the column** of P4

GEMM : DISTRIBUTED MEMORY (SUMMA)



Ingredients

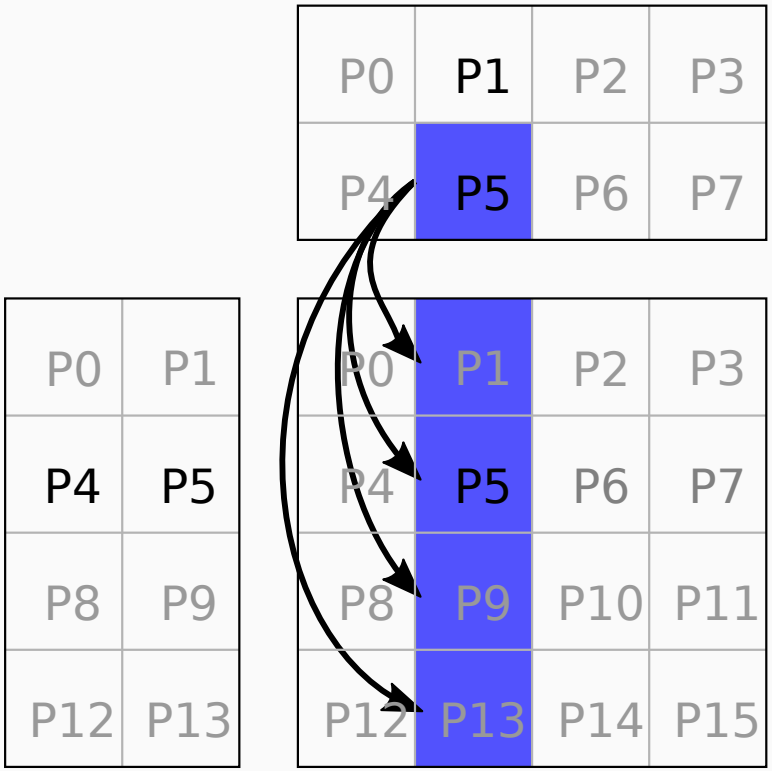
- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)
- Communicat° patterns

```
do k=1,2
      call Bcast(A[i,j]) on my row
      call Bcast(B[i,j]) on my column
      call gemm(A[i,k],B[k,j],C[i,j])
end do
```

Sender and receivers match color.

Sending blocks of *A* from **the column** of P5

GEMM : DISTRIBUTED MEMORY (SUMMA)



Ingredients

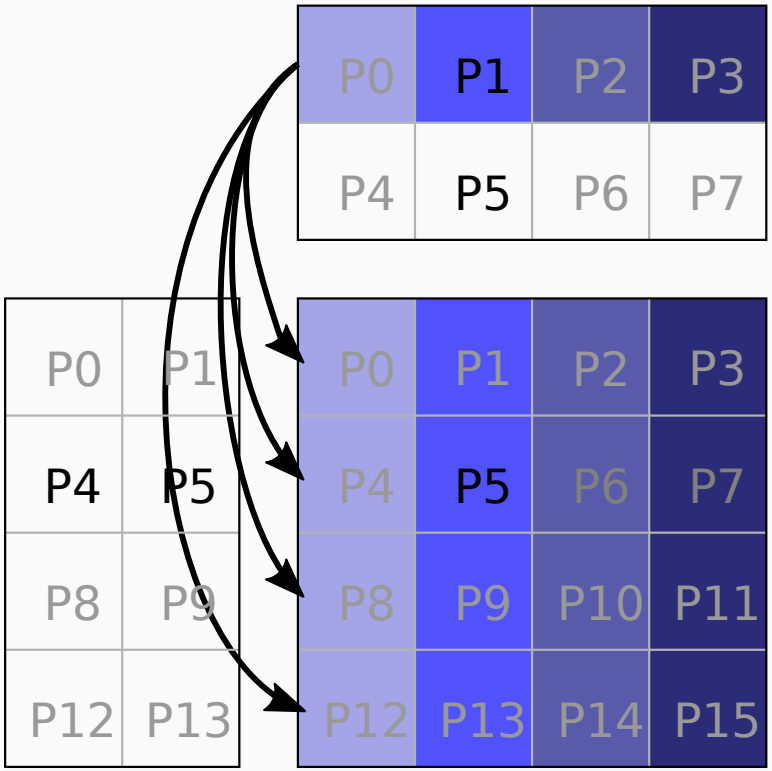
- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)
- Communicat° patterns

```
do k=1,2
      call Bcast(A[i,j]) on my row
      call Bcast(B[i,j]) on my column
      call gemm(A[i,k],B[k,j],C[i,j])
end do
```

Sender and receivers match color.

Sending block of *B* from P5

GEMM : DISTRIBUTED MEMORY (SUMMA)



Ingredients

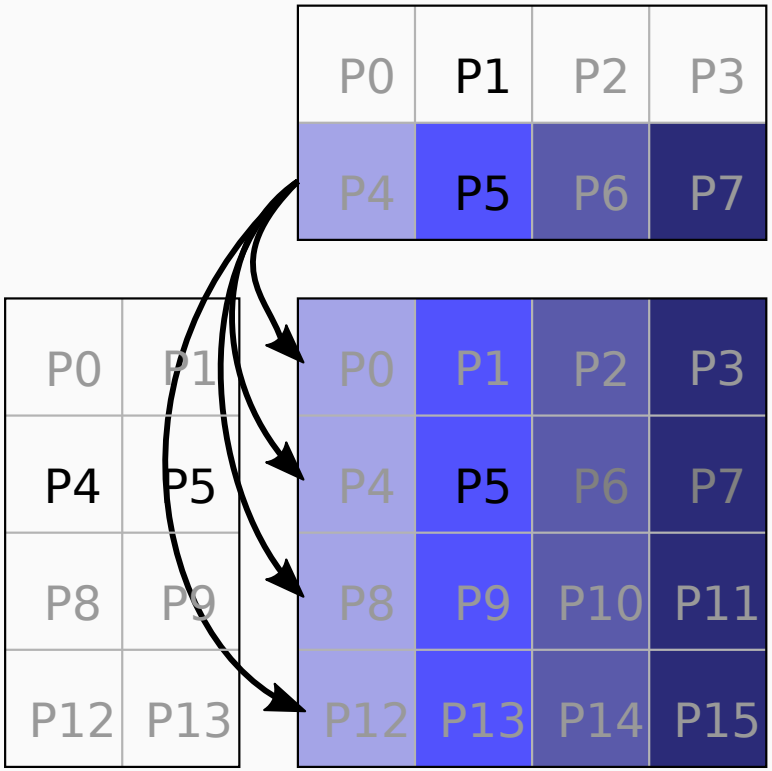
- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)
- Communicat° patterns

```
do k=1,2
      call Bcast(A[i,j]) on my row
      call Bcast(B[i,j]) on my column
      call gemm(A[i,k],B[k,j],C[i,j])
end do
```

Sender and receivers match color.

Sending blocks of *B* from **the row** of P1

GEMM : DISTRIBUTED MEMORY (SUMMA)



Sender and receivers match color.

Sending blocks of *B* from **the row** of P5

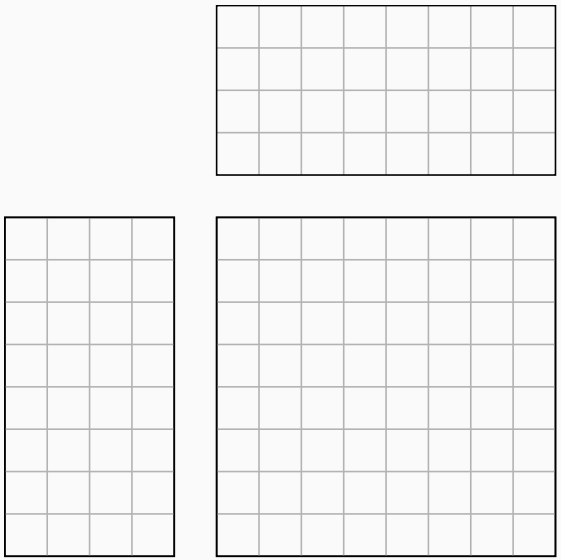
Ingredients

- Data Mapping - here, 2D Block Cyclic
- Task Mapping (= Data Mapping)
- Communicat° patterns

```
do k=1,2
      call Bcast(A[i,j]) on my row
      call Bcast(B[i,j]) on my column
      call gemm(A[i,k],B[k,j],C[i,j])
end do
```

SUMMA Algorithms

m, m, k 2D product on a $\sqrt{p} \times \sqrt{p}$ grid



Communications critical path (*latency* α and *bandwidth* β)

SUMMA Algorithms

m, m, k 2D product on a $\sqrt{p} \times \sqrt{p}$ grid

0	1	2	3	4	5	6	7
8							
16							
24							

0	1	2	3
8			
16			
24			
32			
40			
48			
56	57	58	59

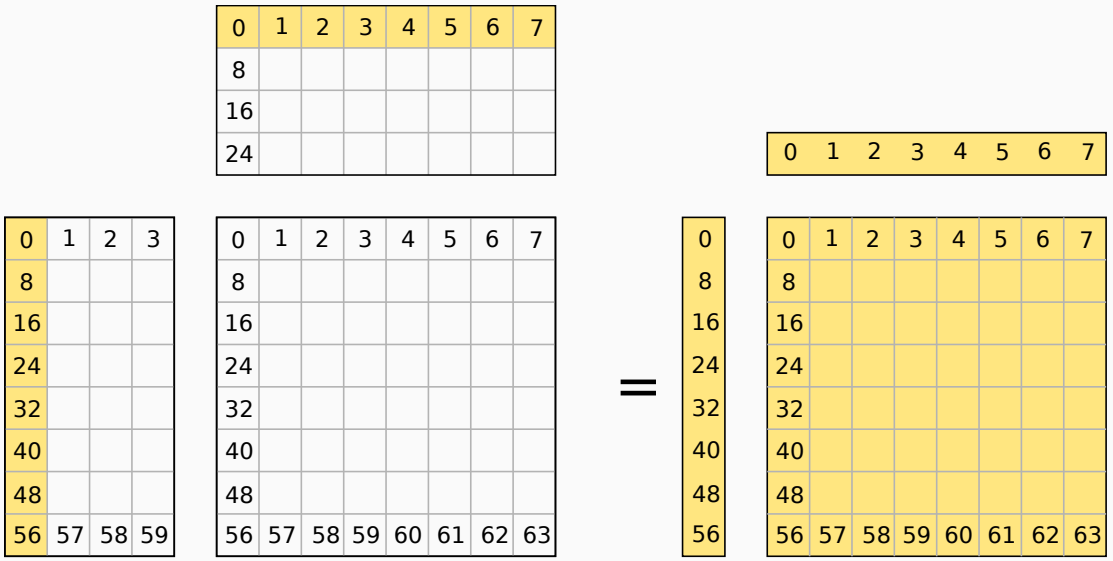
0	1	2	3	4	5	6	7
8							
16							
24							
32							
40							
48							
56	57	58	59	60	61	62	63

Communications critical path (*latency* α and *bandwidth* β)

2D: $\left(\qquad \right) \left(\alpha + \beta b^2\right)$

SUMMA Algorithms

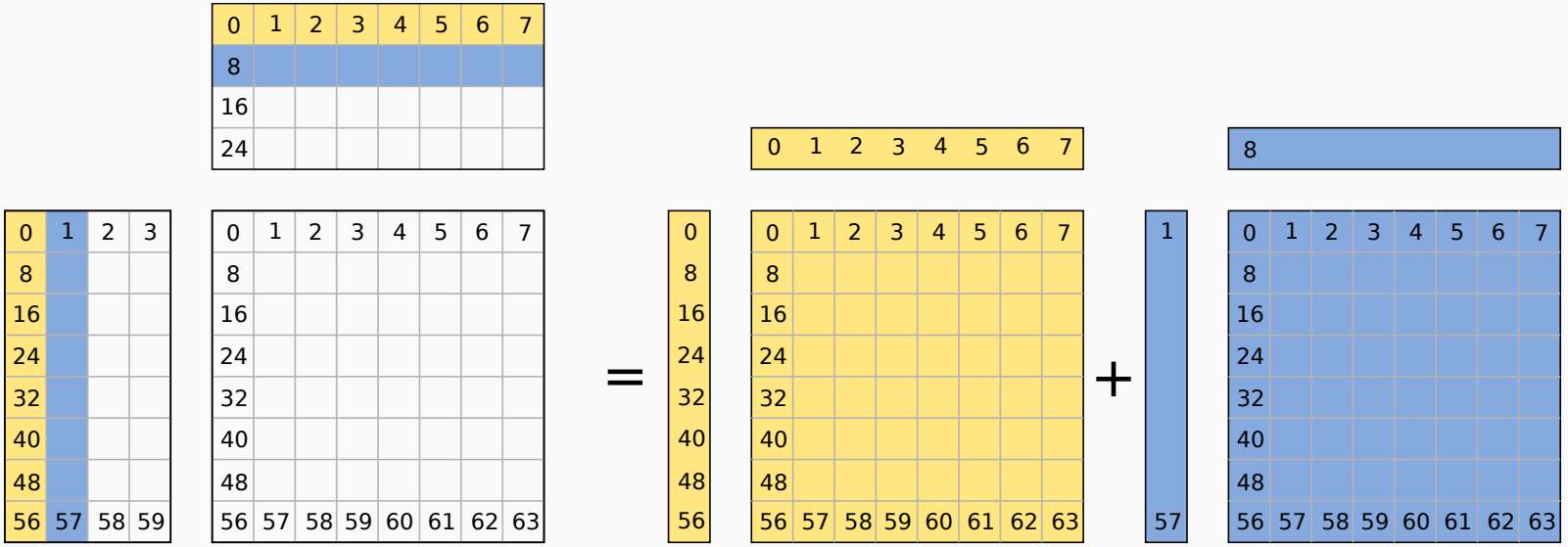
m, m, k 2D product on a $\sqrt{p} \times \sqrt{p}$ grid



Communications critical path (*latency* α and *bandwidth* β)

2D: $\left(\frac{2m}{b\sqrt{p}} log\left(\sqrt{p}\right)\right)\left(\alpha + \beta b^2\right)$

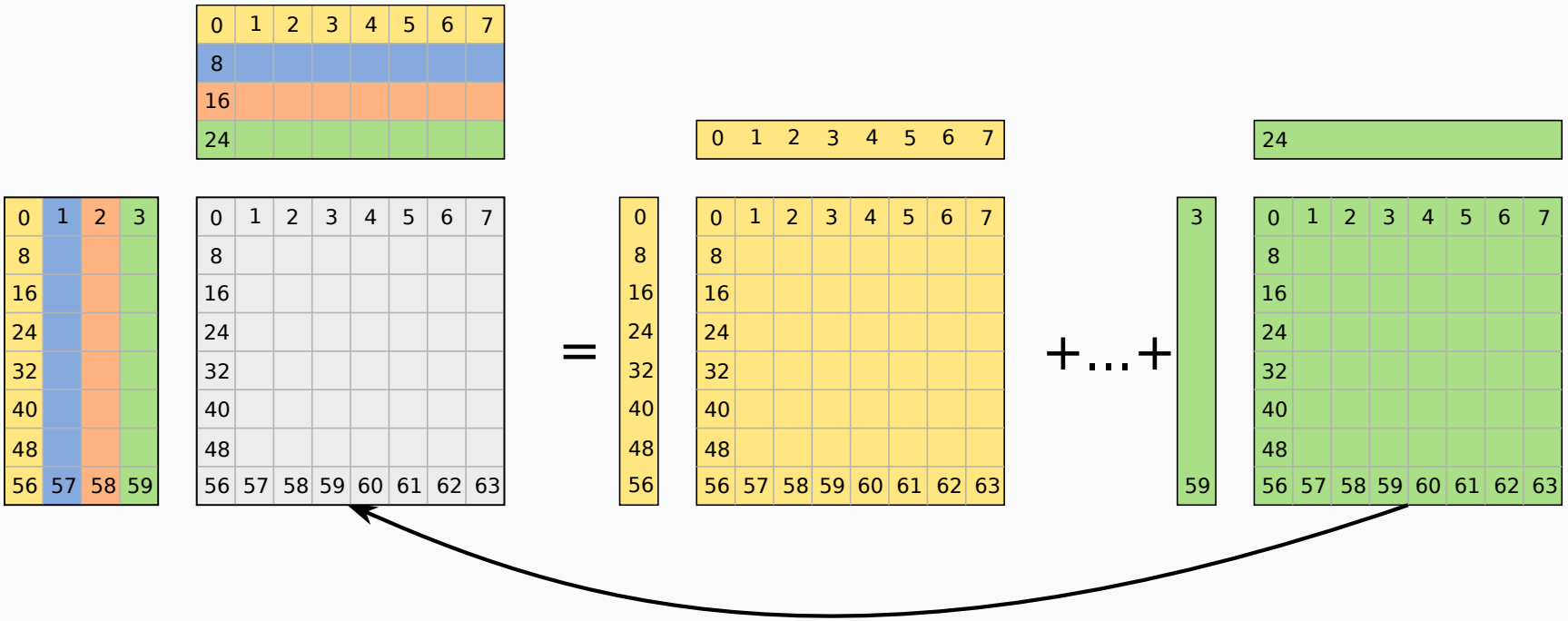
m, m, k 2D product on a $\sqrt{p} \times \sqrt{p}$ grid



Communications critical path (*latency* α and *bandwidth* β)

2D: $$\left(\frac{2m}{b\sqrt{p}} log\left(\sqrt{p}\right)\right)\left(\alpha + \beta b^2\right)$$

m, m, k 2D product on a $\sqrt{p} \times \sqrt{p}$ grid

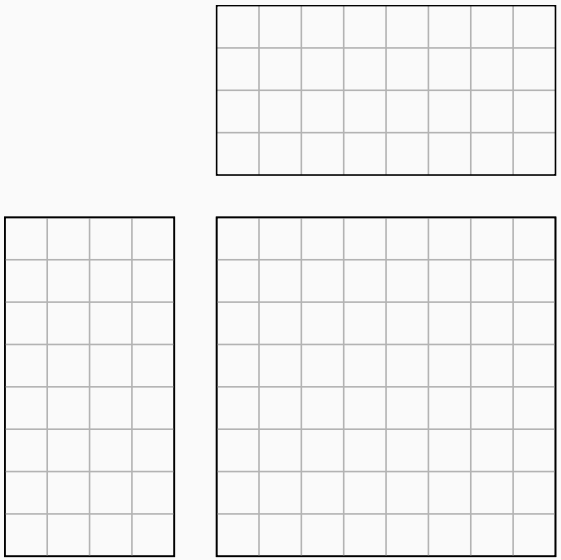


Communications critical path (*latency* α and *bandwidth* β)

2D: $\frac{k}{b}\left(\frac{2m}{b\sqrt{p}} log\left(\sqrt{p}\right)\right)\left(\alpha + \beta b^2\right)$

SUMMA Algorithms

m, m, k 3D product on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ grid

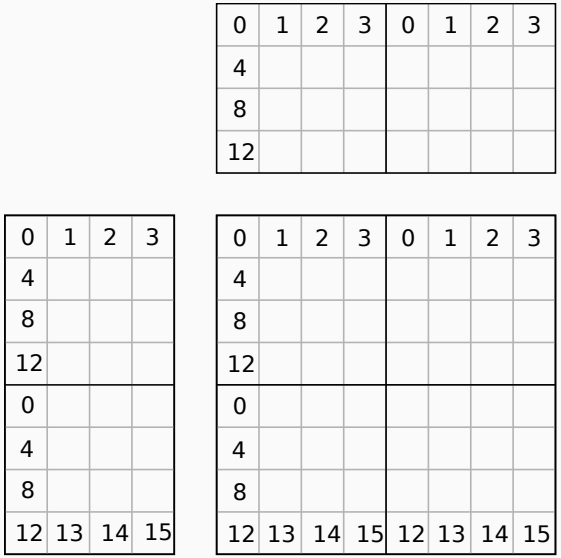


Communications critical path (*latency* α and *bandwidth* β)

2D: $\frac{k}{b}\left(\frac{2m}{b\sqrt{p}} log\left(\sqrt{p}\right)\right)\left(\alpha + \beta b^2\right)$

SUMMA Algorithms

m, m, k 3D product on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ grid



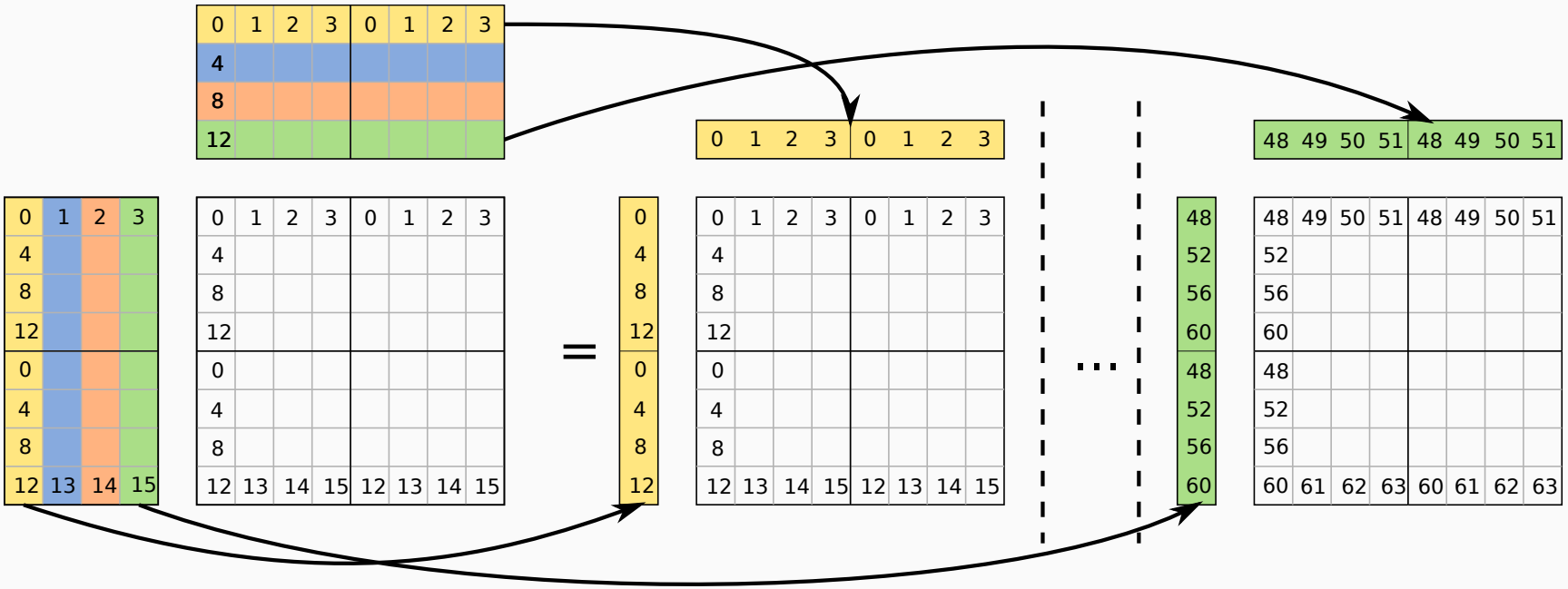
Communications critical path (*latency* α and *bandwidth* β)

2D: $\frac{k}{b}\left(\frac{2m}{b\sqrt{p}} log\left(\sqrt{p}\right)\right)\left(\alpha + \beta b^2\right)$

3D: $\left(\right)\left(\alpha + \beta b^2\right)$

SUMMA Algorithms

m, m, k 3D product on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ grid

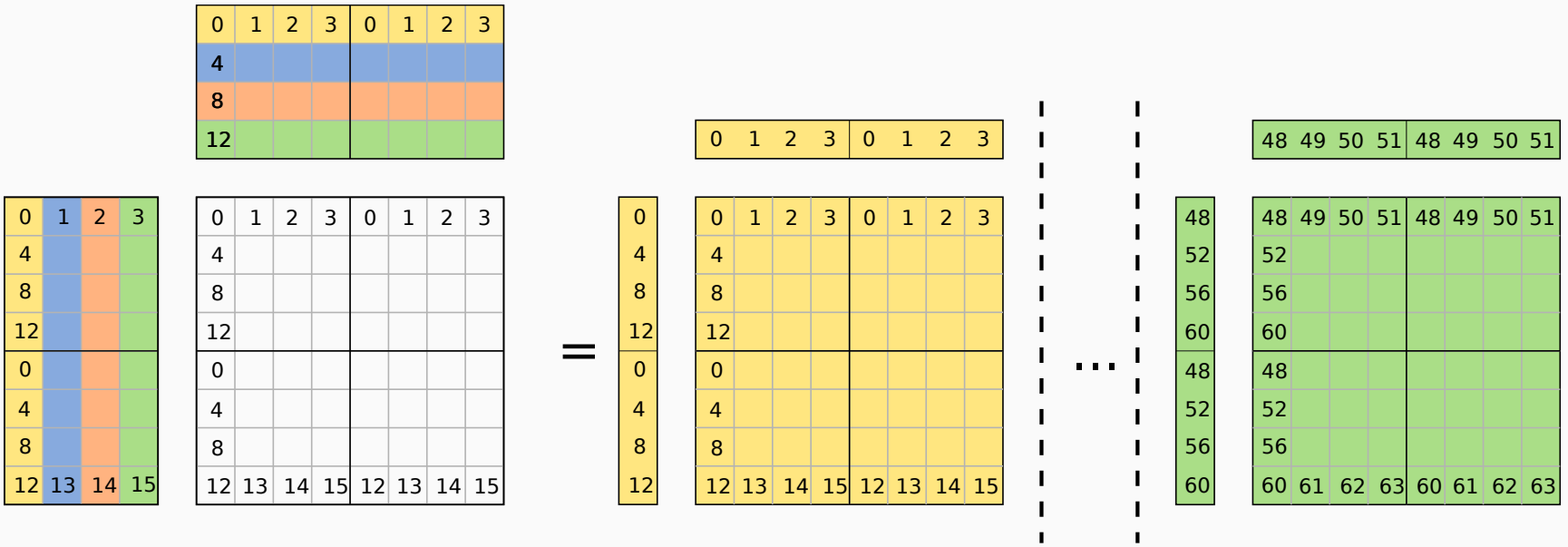


Communications critical path (*latency* α and *bandwidth* β)

2D: $\frac{k}{b}\left(\frac{2m}{b\sqrt{p}} log\left(\sqrt{p}\right)\right)\left(\alpha + \beta b^2\right)$

3D: $\left(\frac{2mk}{b^2\sqrt[3]{p^2}}\right)\left(\alpha + \beta b^2\right)$

m, m, k 3D product on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ grid

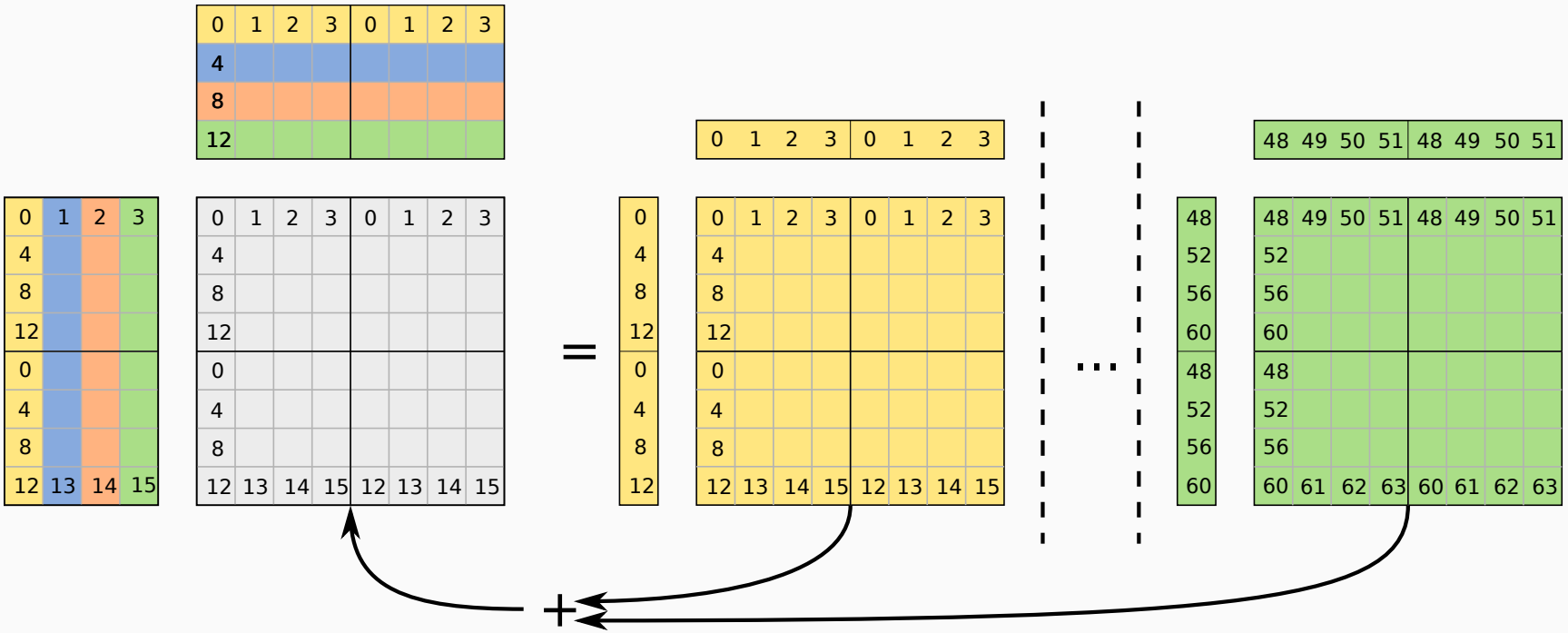


Communications critical path (*latency* α and *bandwidth* β)

2D: $\frac{k}{b}\left(\frac{2m}{b\sqrt{p}}log\left(\sqrt{p}\right)\right)\left(\alpha+\beta b^2\right)$

3D: $\left(\frac{2mk}{b^2\sqrt[3]{p^2}}+\frac{k}{b\sqrt[3]{p}}\left(\frac{2m}{b\sqrt[3]{p}}log\left(\sqrt[3]{p}\right)\right)\right)\left(\alpha+\beta b^2\right)$

m, m, k 3D product on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ grid

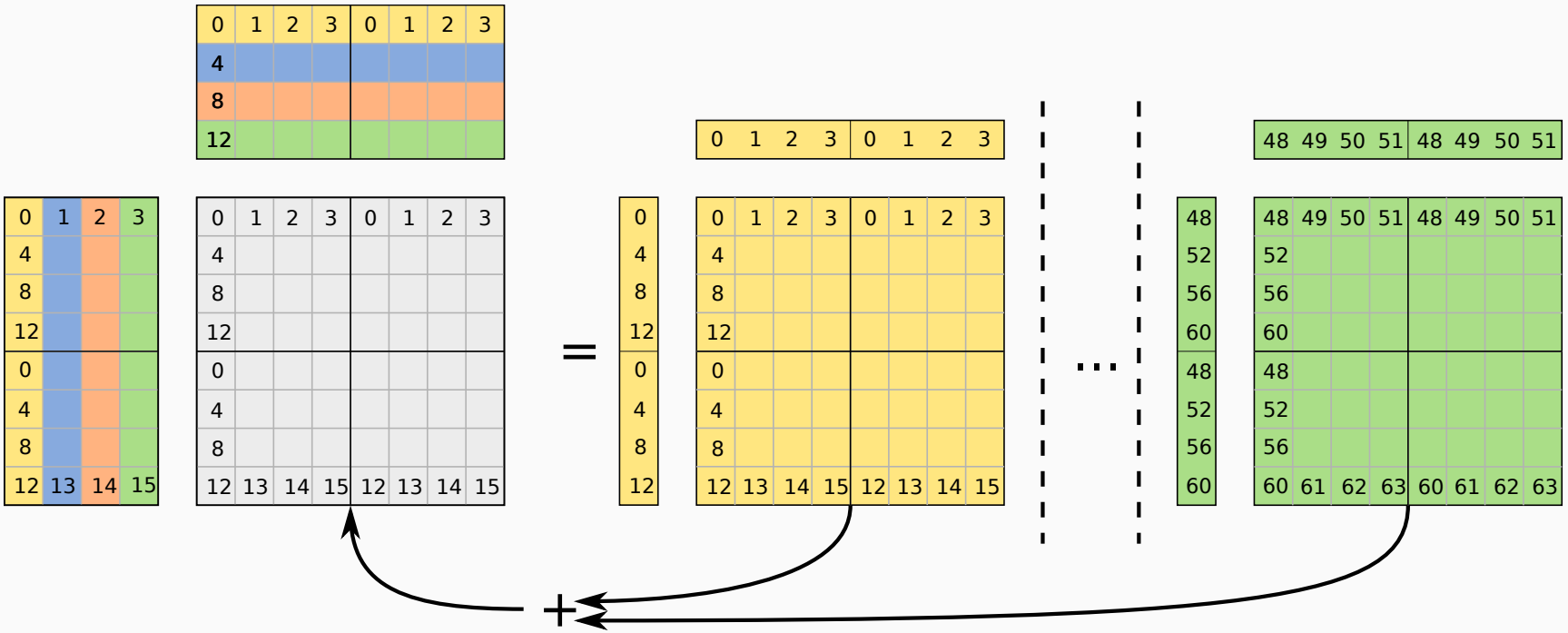


Communications critical path (*latency* α and *bandwidth* β)

2D: $\frac{k}{b}\left(\frac{2m}{b\sqrt{p}}log\left(\sqrt{p}\right)\right)\left(\alpha+\beta b^2\right)$

3D: $\left(\frac{2mk}{b^2\sqrt[3]{p^2}}+\frac{k}{b\sqrt[3]{p}}\left(\frac{2m}{b\sqrt[3]{p}}log\left(\sqrt[3]{p}\right)\right)+\frac{m^2}{b^2\sqrt[3]{p^2}}log\left(\sqrt[3]{p}\right)\right)\left(\alpha+\beta b^2\right)$

m, m, k 3D product on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ grid



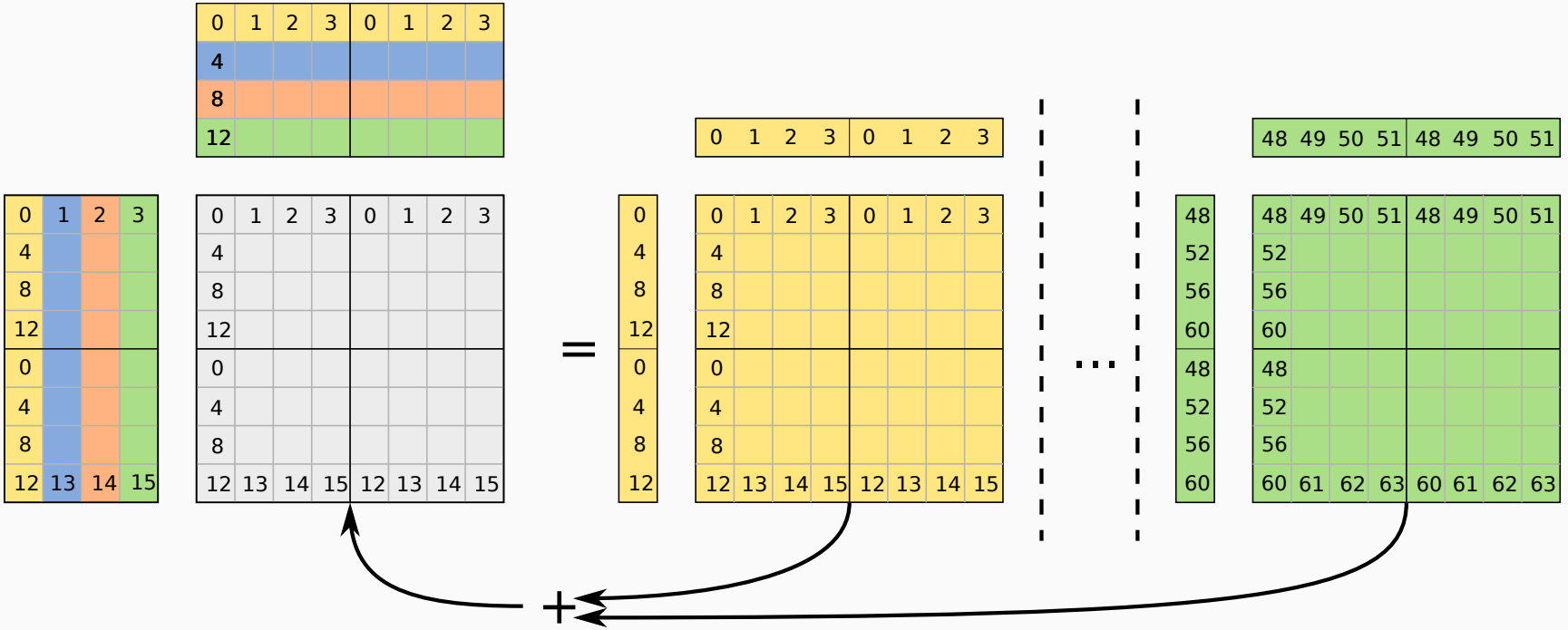
Communications critical path (*latency* α and *bandwidth* β)

2D: $4\left(\frac{2\times 8}{8} log\left(8\right)\right)\left(\alpha+\beta b^2\right)$

3D: $\left(\frac{2\times 8\times 4}{4^2}+\frac{4}{4}\left(\frac{2\times 8}{4} log\left(4\right)\right)+\frac{8\times 8}{4^2} log\left(4\right)\right)\left(\alpha+\beta b^2\right)$

SUMMA Algorithms

m, m, k 3D product on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ grid



Communications critical path (*latency* α and *bandwidth* β)

2D: $(24log\,(2))\left(\alpha + \beta b^2\right)$

3D: $(4 + 16log\,(2))\left(\alpha + \beta b^2\right)$

SUMMA is implemented in reknown libraries

SUMMA is implemented in reknown libraries

- Multiple SUMMA variants are implemented in `ScalaPACK`
- A variant using pipeline is implemented in `Chameleon` using `StarPU`

The `PARSEC` runtime has helped with distributed `GEMM` algorithms

SUMMA is implemented in reknown libraries

- Multiple SUMMA variants are implemented in `ScalaPACK`
- A variant using pipeline is implemented in `Chameleon` using `StarPU`

The `PARSEC` runtime has helped with distributed `GEMM` algorithms

These implementations use either

- **explicit communication management**
- **specific data structures** required by their programming models

SUMMA

SUMMA

- Data Mapping
- Task Mapping to describe 2.5D decomposition
- Collective communications

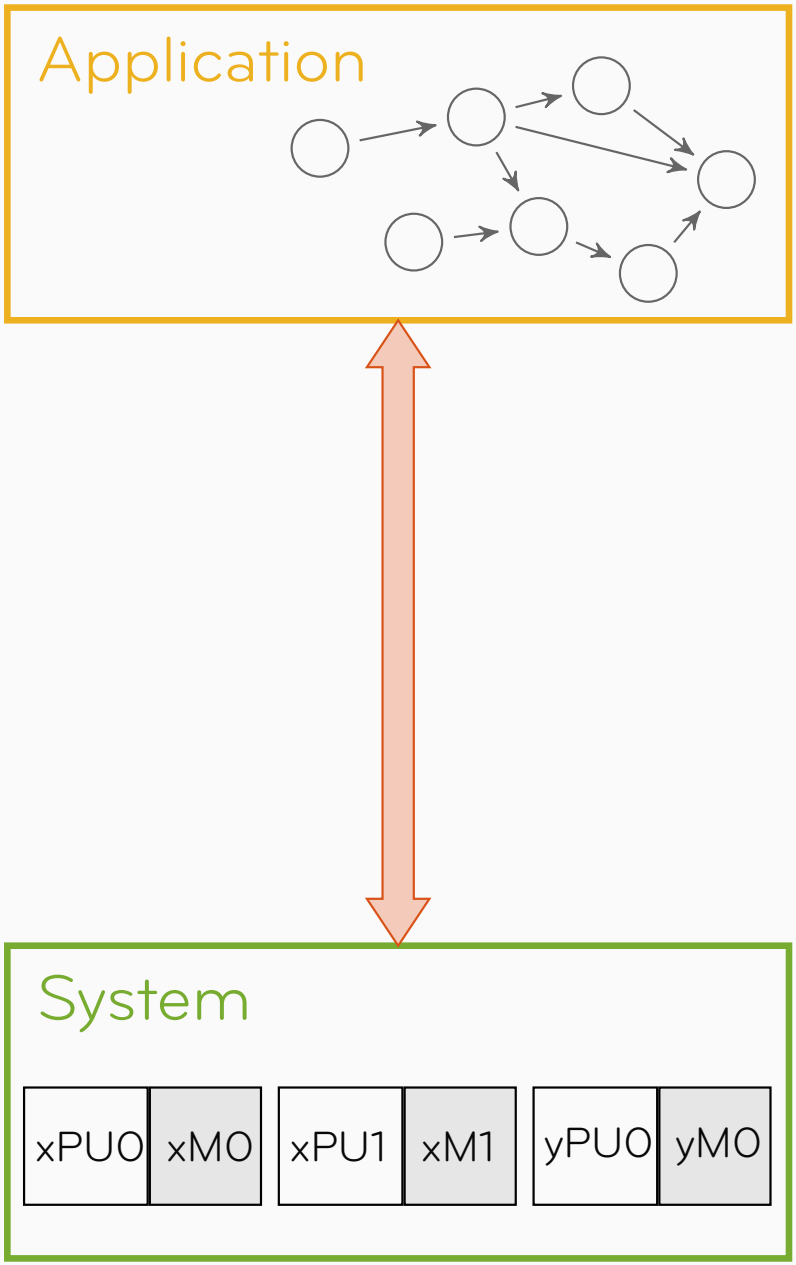
Objectives

SUMMA

- Data Mapping
- Task Mapping to describe 2.5D decomposition
- Collective communications

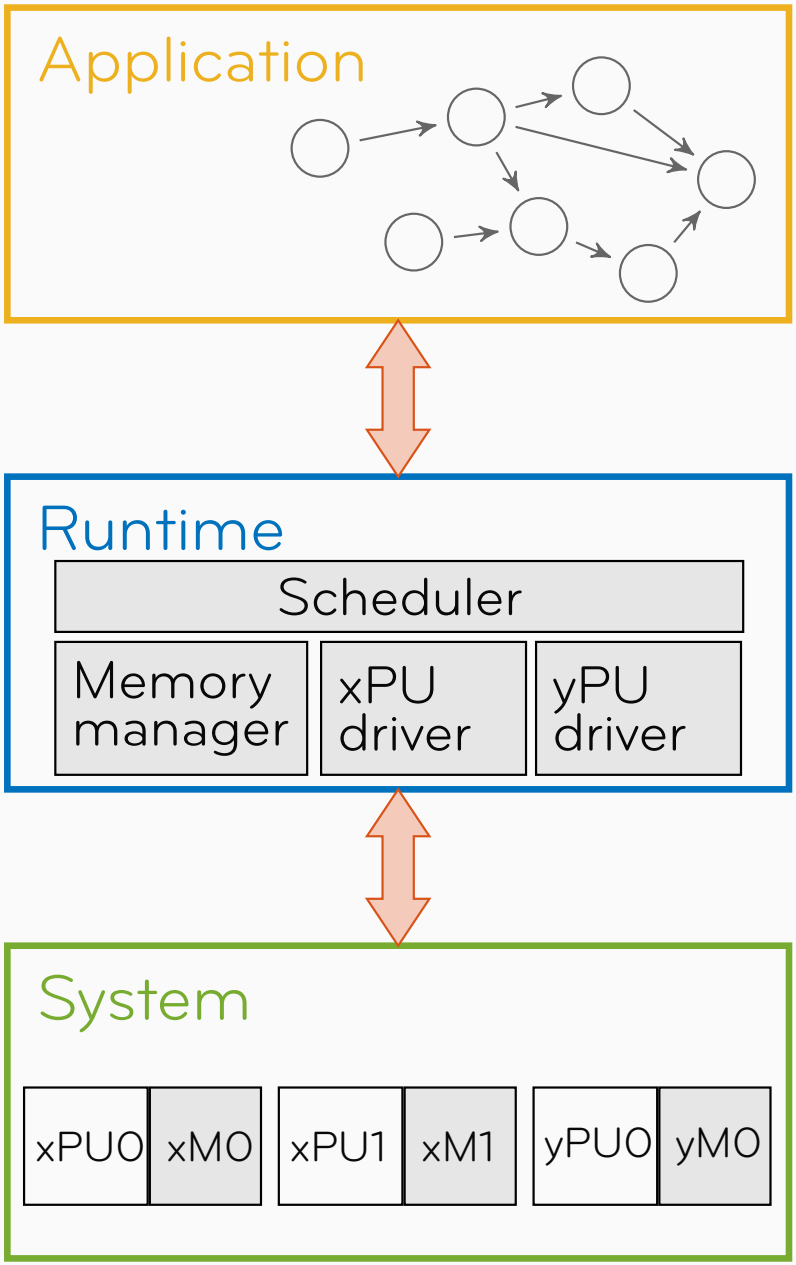
Can we feature **complex** communications and reduction **patterns** using **programming models** offered by runtime systems ?

Runtimes



- The classical approach is based on a mixture of technologies (e.g., MPI+OpenMP+CUDA) which
  - requires a big programming effort
  - is difficult to maintain and update
  - is prone to (performance) portability issues

Runtimes

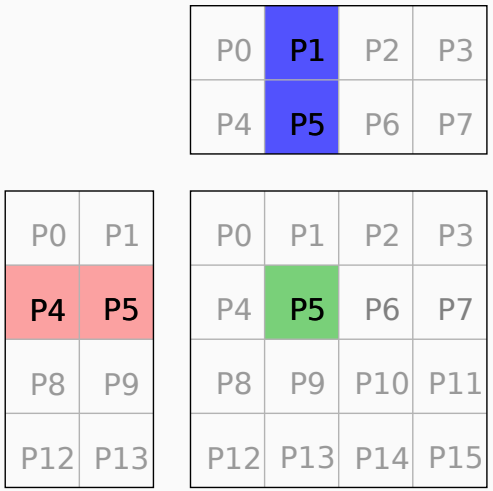


- The classical approach is based on a mixture of technologies (e.g., MPI+OpenMP+CUDA) which
  - requires a big programming effort
  - is difficult to maintain and update
  - is prone to (performance) portability issues

- runtimes provide an abstraction layer that hides the architecture details:
  - portable programming interface
  - handling of data
  - support for multiple architectures
  - customizable scheduling policies
  - ...

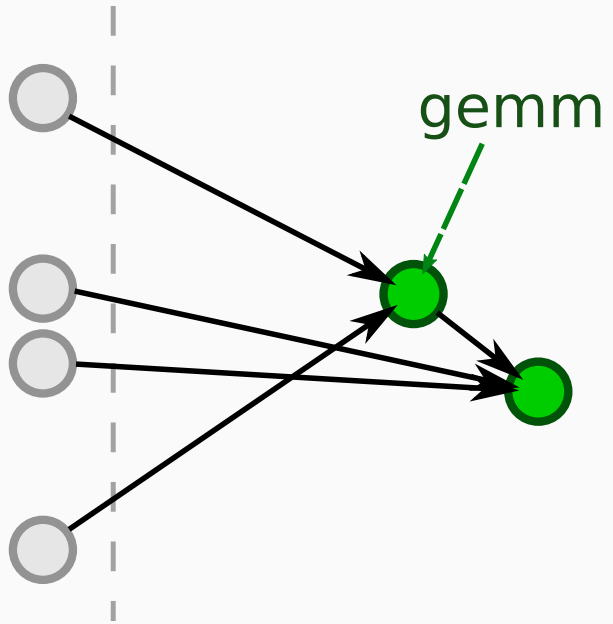
Task-based Matrix Multiplication with STF

```
do i=1,m
  do j=1,n
    do l=1,k
      call gemm(
                  A[i,l],
                  B[l,j],
                  C[i,j] )
    end do
  end do
end do
```



Directed Acyclic Graph (DAG) :

- **Node** : Elementary task (`gemm`)
- **Edge** : Sequential dependency



DAG for $C_{2,2}$

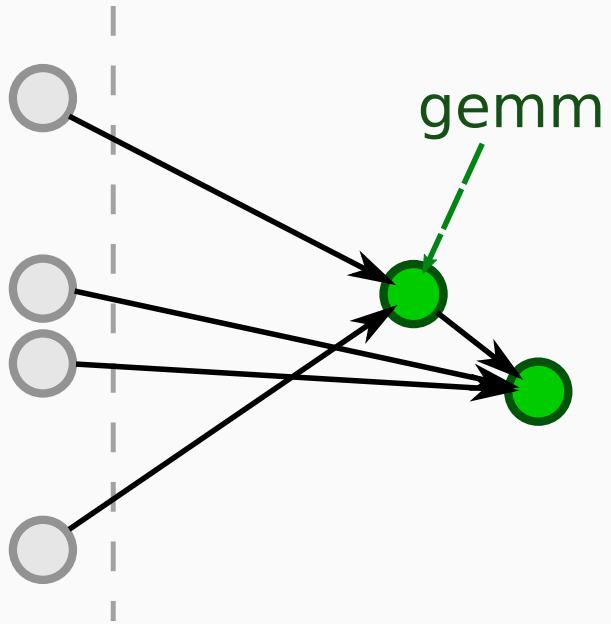
Task-based Matrix Multiplication with STF

```
do i=1,m
  do j=1,n
    do l=1,k
      call gemm(
                  A[i,l],
                  B[l,j],
                  C[i,j] )
    end do
  end do
end do
```

Sequential **T**ask **F**low : how to describe the DAG to the runtime

Directed Acyclic Graph (DAG) :

- **Node** : Elementary task (`gemm`)
- **Edge** : Sequential dependency



DAG for $C_{2,2}$

Task-based Matrix Multiplication with STF

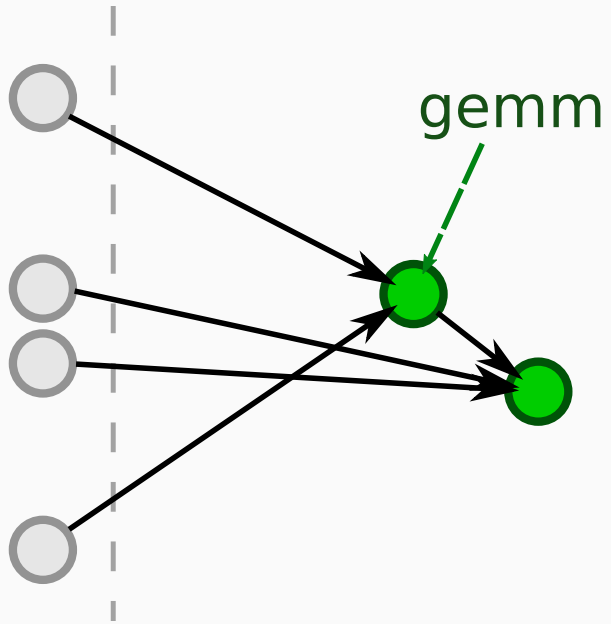
```
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l],
                   B[l,j],
                   C[i,j])
    end do
  end do
end do
```

Sequential **T**ask **F**low : how to describe the DAG to the runtime

- Submit tasks **sequentially**

Directed Acyclic Graph (DAG) :

- **Node** : Elementary task (`gemm`)
- **Edge** : Sequential dependency



DAG for $C_{2,2}$

Task-based Matrix Multiplication with STF

```
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```

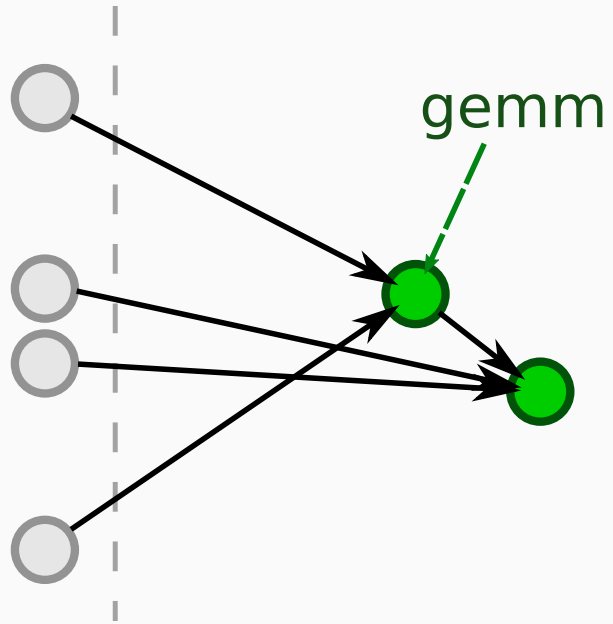
Sequential **T**ask **F**low : how to describe the DAG to the runtime

- Submit tasks **sequentially**
- Access modes `Read`, `Write`, `ReadWrite`

This works well in **shared memory** !

Directed Acyclic Graph (DAG) :

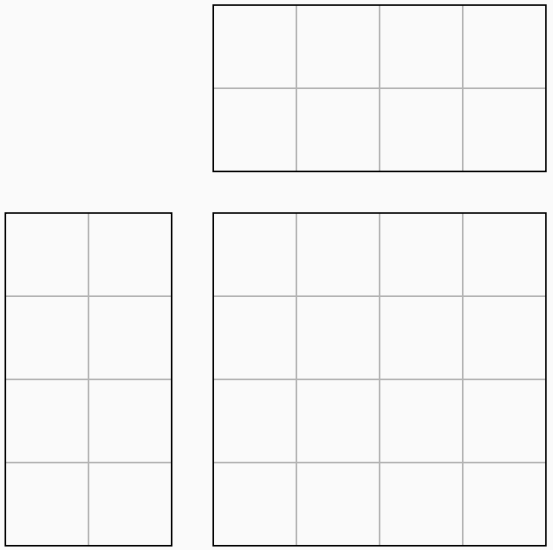
- **Node** : Elementary task (`gemm`)
- **Edge** : Sequential dependency



DAG for $C_{2,2}$

Baseline distributed programming model

```
!
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```

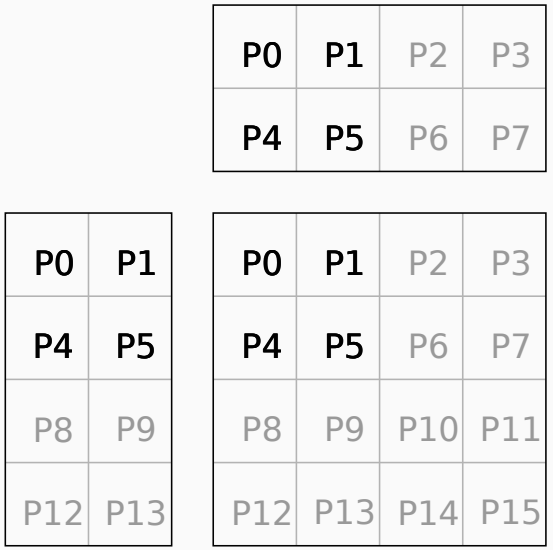


Baseline model (M0)

E. Agullo, O. Aumage, M. Faverge, N. Furmento et al., 2016

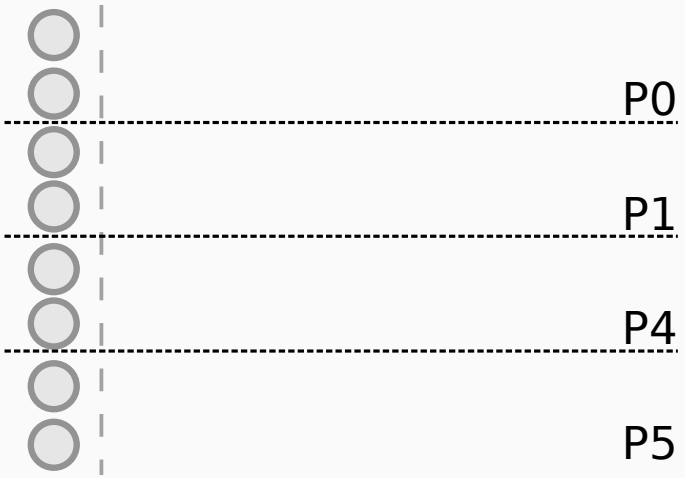
Baseline distributed programming model

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```



Baseline model (M0)

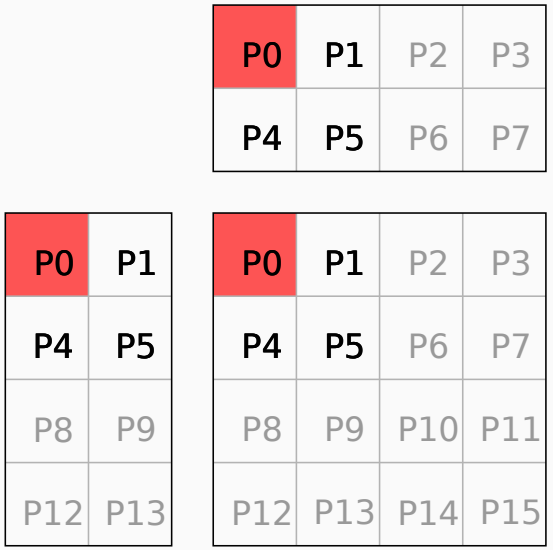
- M0-0 *Data Mapping*



E. Agullo, O. Aumage, M. Faverge, N. Furmento et al., 2016

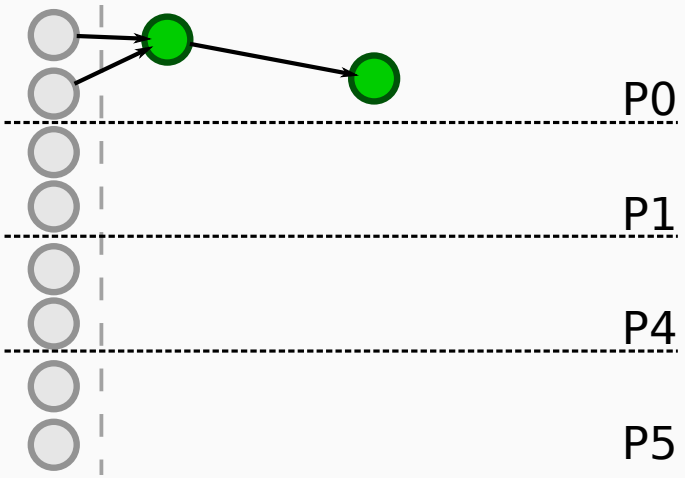
Baseline distributed programming model

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```



Baseline model (M0)

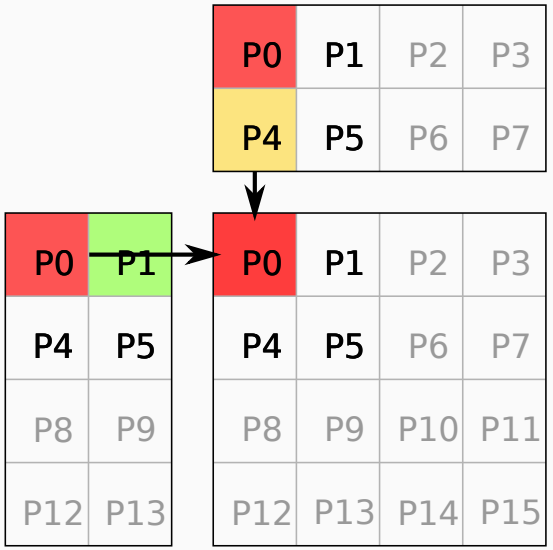
- M0-0 *Data Mapping*
- M0-1 *Task Mapping* **inferred** from DM (RW)



E. Agullo, O. Aumage, M. Faverge, N. Furmento et al., 2016

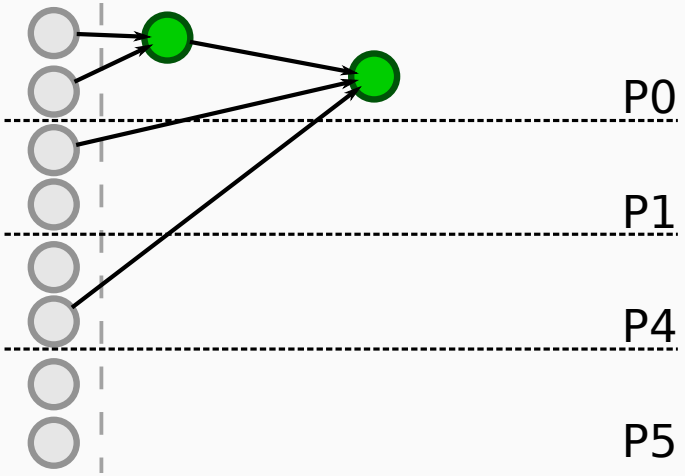
Baseline distributed programming model

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```



Baseline model (M0)

- M0-0 *Data Mapping*
- M0-1 *Task Mapping* **inferred** from DM (RW)
- M0-2 Point-to-Point Comm. **inferred** from TM



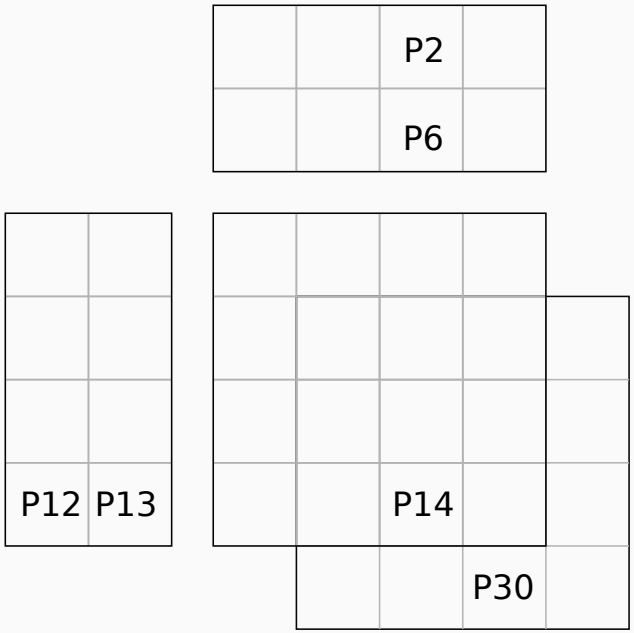
E. Agullo, O. Aumage, M. Faverge, N. Furmento et al., 2016

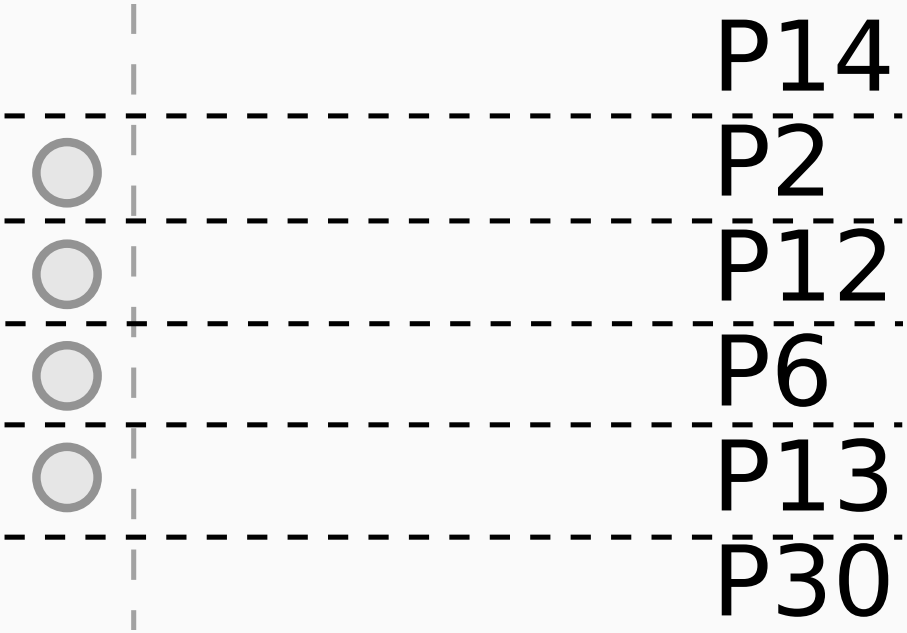
Limits on implementing a task-based 2.5D GEMM algorithm

Limit L1 : how to distribute a dot product ?

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```

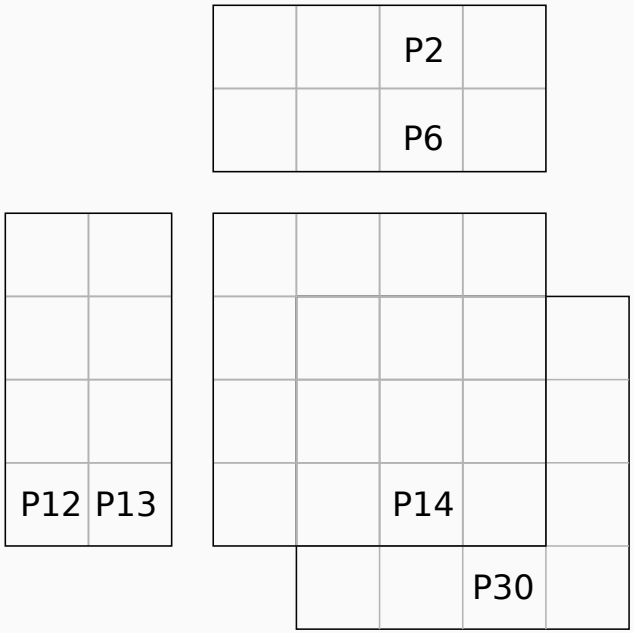
Limit L1

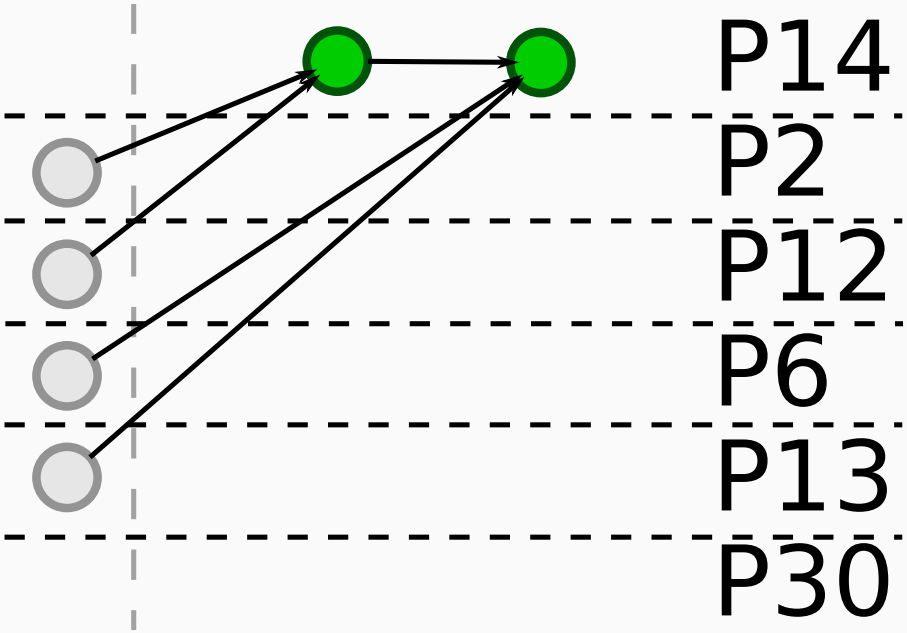




Limit L1 : how to distribute a dot product ?

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```



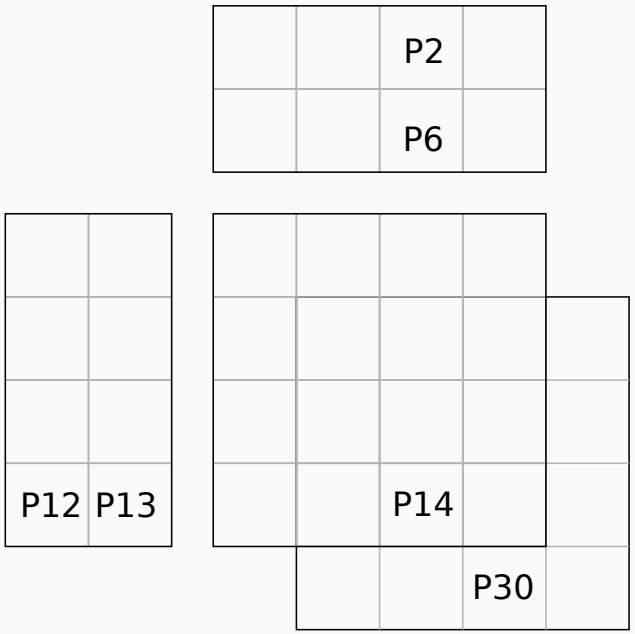


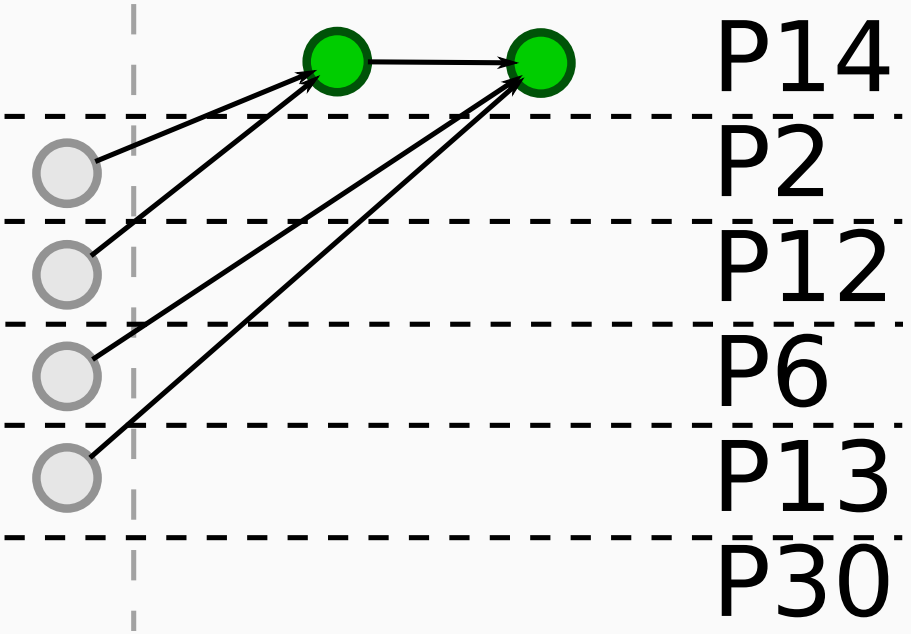
Limit L1

M0-1 : the Data Mapping forces P14 to execute the tasks

Limit L2 : how compact is it ?

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```





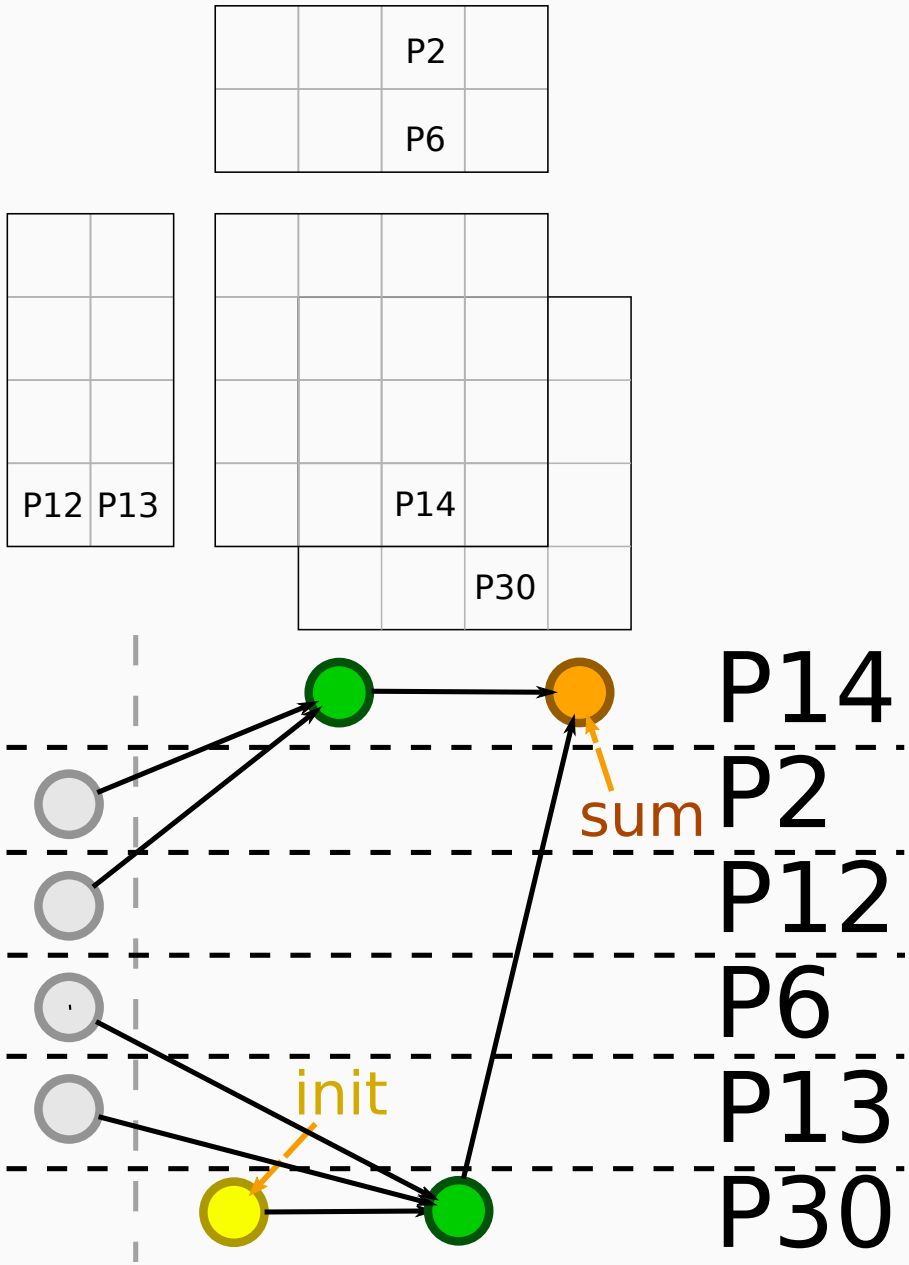
Limit L2

New tasks end up in the DAG

- `init` tasks
- `sum` tasks

Limit L2 : how compact is it ?

```
! data_map: A,B,C distributed on a grid
instanciate C_copy  on every layer
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C_copy[i,j]:RW)
    end do
  end do
end do
do l=1,h
  do i=1,m; do j=1,n
    call submit(sum,
                C_copy[i,j]:R,
                C[i,j]:RW )
  end do
end do
```



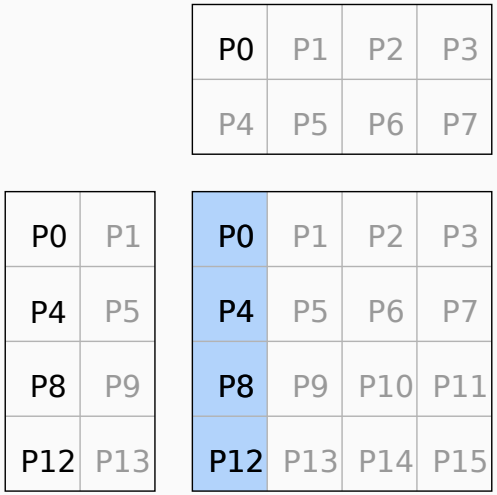
Limit L2

New tasks end up in the DAG

- `init` tasks
- `sum` tasks

Limit L3 : collective communications ?

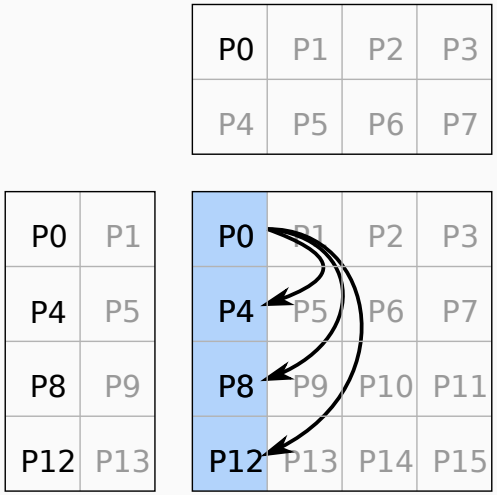
```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```



Limit L3

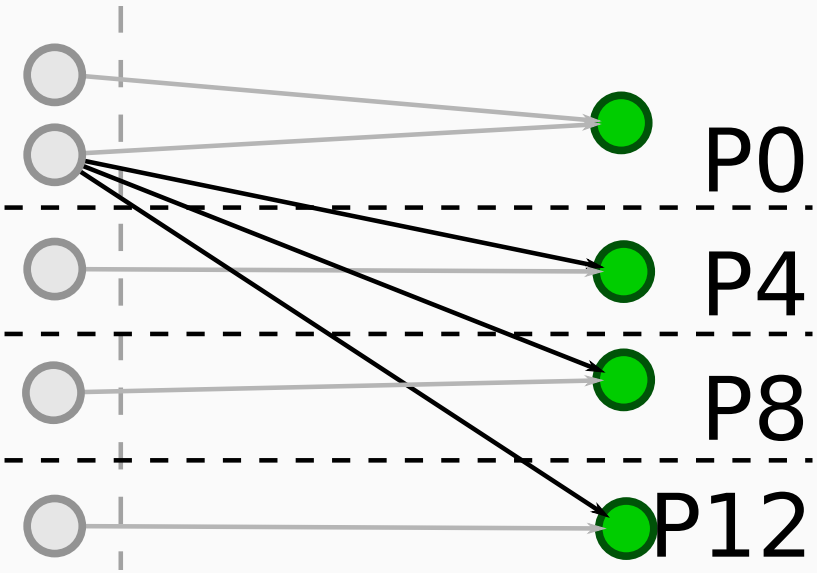
Limit L3 : collective communications ?

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)
    end do
  end do
end do
```



Limit L3

M0-2 : *Point-to-point* Communications



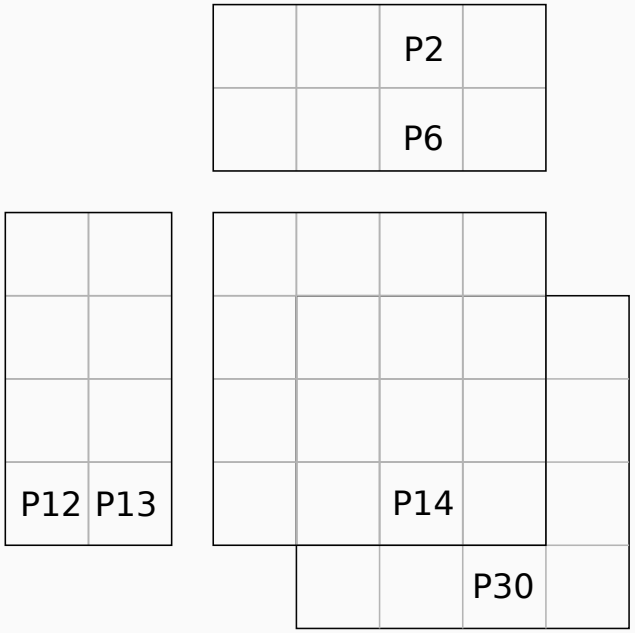
Limits on `M0`

- **L1** Over-constrained Task Mapping
- **L2** Tedious reduction pattern
- **L3** Point-to-point Communications)

EXTENDING THE BASELINE MODEL

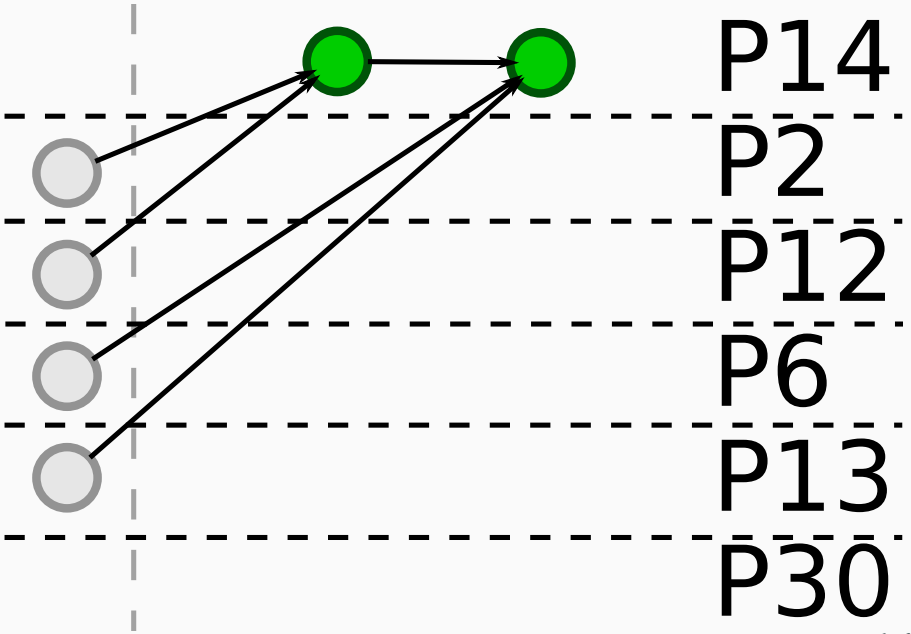
Distributing a dot product (L1)

```
!data_map: A,B,C distributed on a grid
!
do i=1,m
  do j=1,n
    do l=1,k
      call submit(gemm,
                  A[i,l]:R,
                  B[l,j]:R,
                  C[i,j]:RW)
    end do
  end do
end do
```



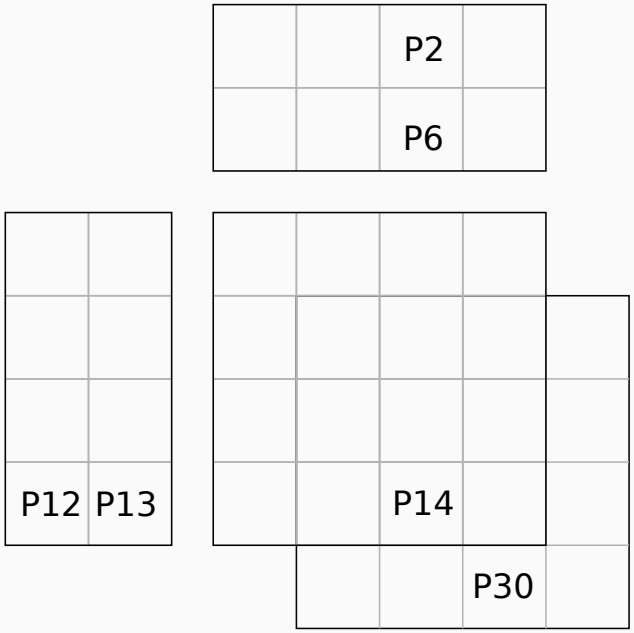
Extended model MX

- M0-0 Data Mapping
- M0-1 *Task Mapping inferred* from Data Mapping
- M0-2 Communications inferred from Task Mapping



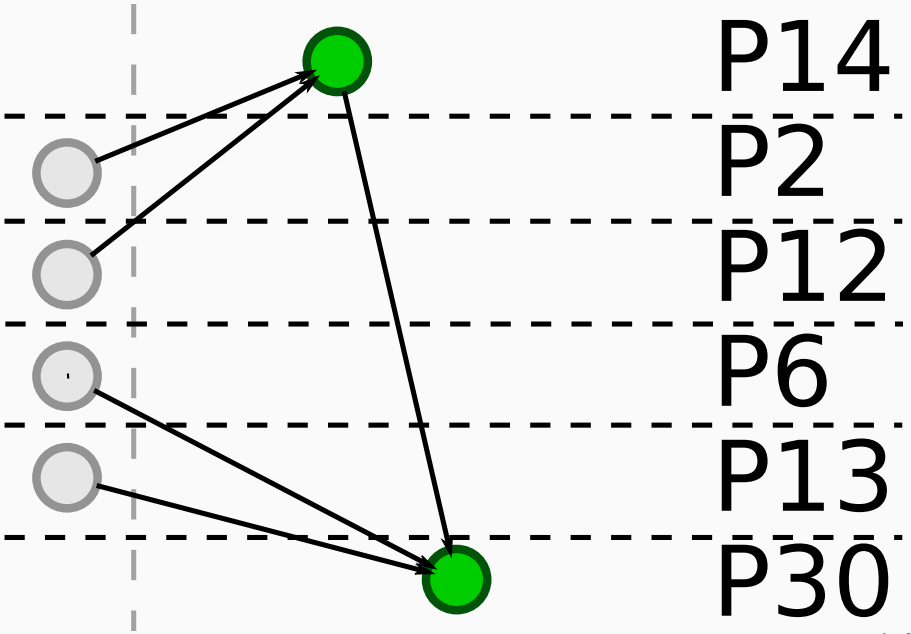
Distributing a dot product (L1)

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]
do i=1,m
  do j=1,n
    do l=1,k
      call submit(gemm,
                  A[i,l]:R,
                  B[l,j]:R,
                  C[i,j]:RW,
                  task_map[i,j,l])
    end do
  end do
end do
! reduction tasks submission
```



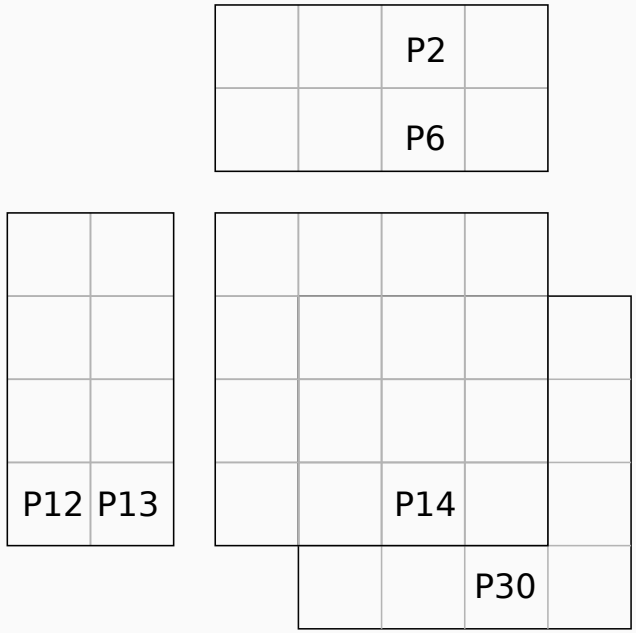
Extended model MX

- MO-0 Data Mapping
- MX-1 **Explicit** *Task Mapping*
- MO-2 Communications inferred from Task Mapping



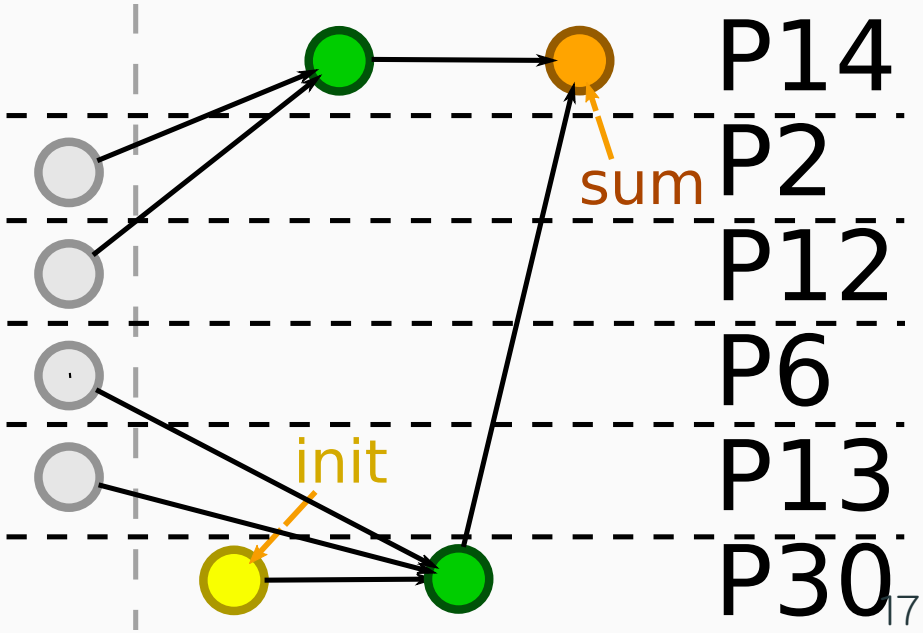
Delegating the reduction pattern (L2)

```
!data_map: A,B,C distributed on one layer
!task_map: executing node of A[i,l]B[l,j]
instanciate C_copy on every layer
do i=1,m
  do j=1,n
    do l=1,k
      call submit(gemm,
                  A[i,l]:R,
                  B[l,j]:R,
                  C_copy[i,j]:RW,
                  task_map[i,j,l])
    end do
  end do
end do
reduce C using C_copy
```



Extended model MX

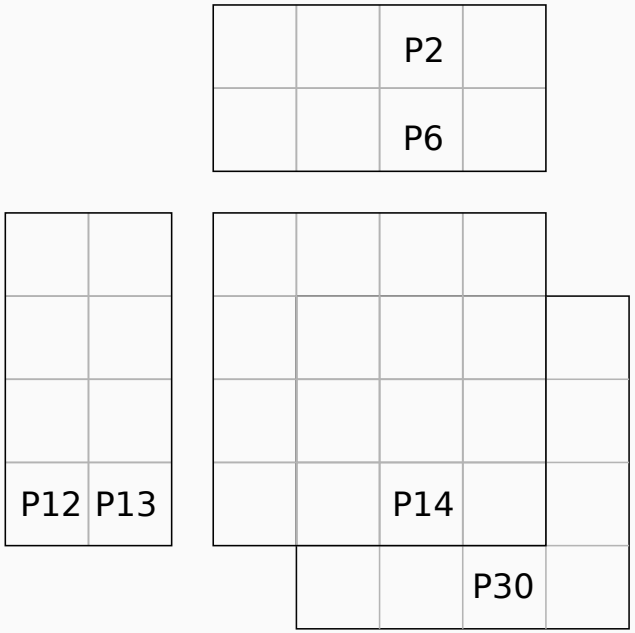
- M0-0 Data Mapping
- MX-1 Explicit Task Mapping
- M0-2 Communications inferred from Task Mapping
- MX-3



Delegating the reduction pattern (L2)

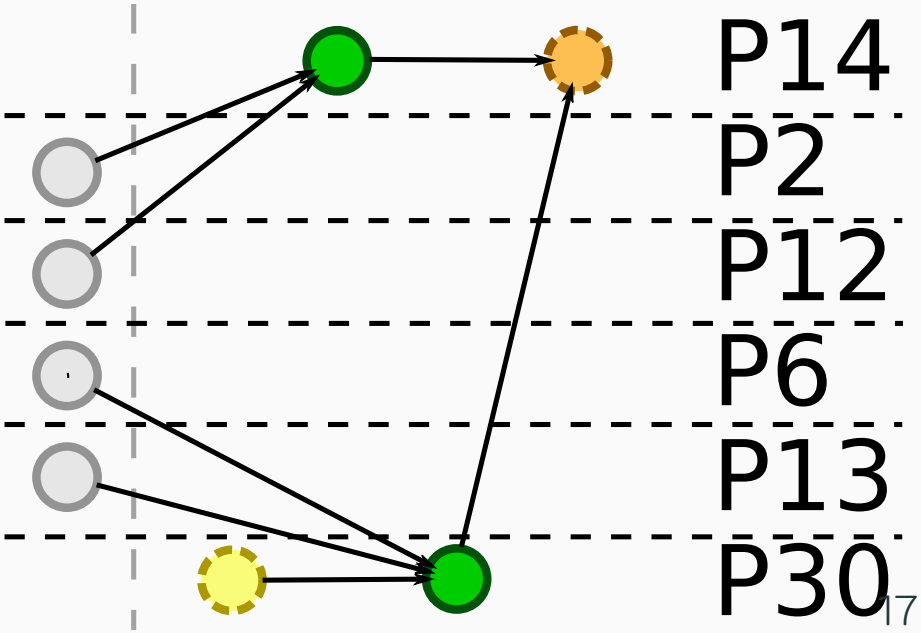
```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]

do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:MPI_REDUX,
                   task_map[i,j,l])
    end do
  end do
end do
```



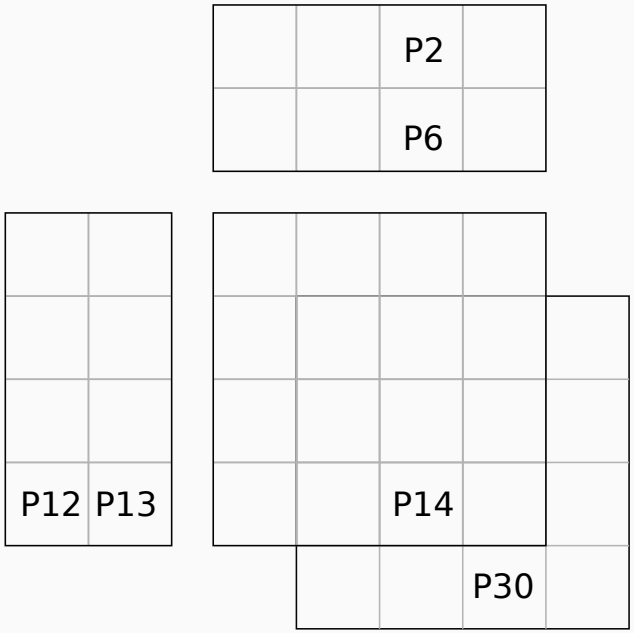
Extended model MX

- MO-0 Data Mapping
- MX-1 Explicit Task Mapping
- MO-2 Communications inferred from Task Mapping
- MX-3 **Implicit** reduction pattern



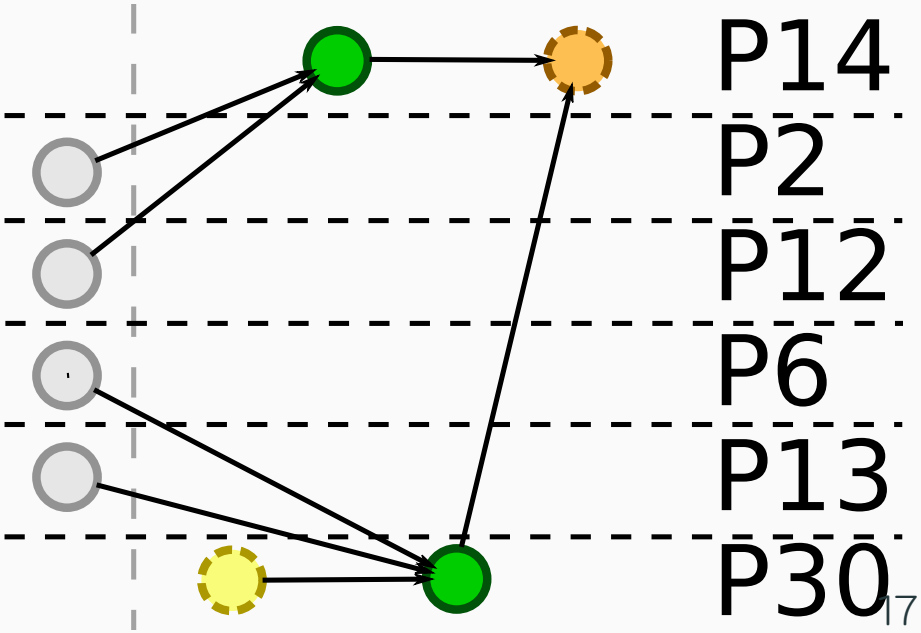
Delegating the reduction pattern (L2)

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,1]B[1,j]
call bind_methods(C, init, sum)
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:MPI_REDUX,
                   task_map[i,j,l])
    end do
  end do
end do
```



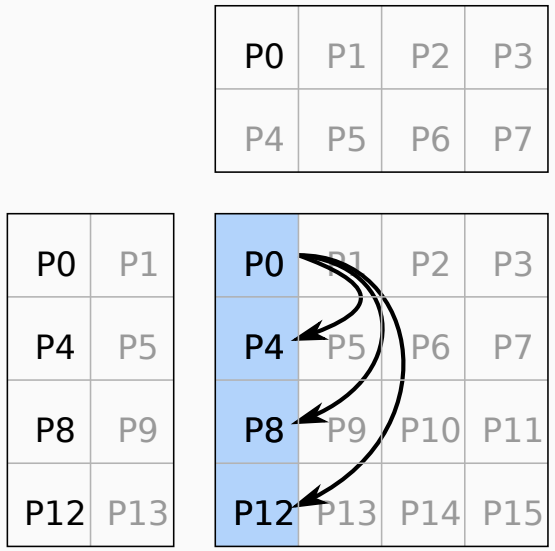
Extended model MX

- M0-0 Data Mapping
- MX-1 Explicit Task Mapping
- M0-2 Communications inferred from Task Mapping
- MX-3 **Implicit** reduction pattern



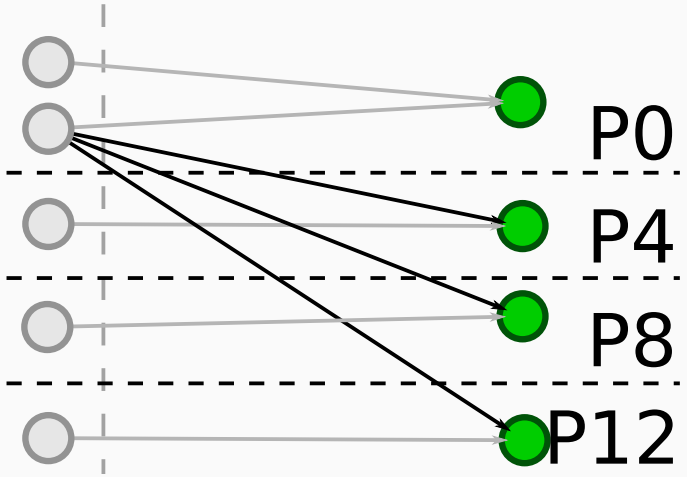
Detecting collective communications (L3)

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]
!dyn_tree: type of broadcast tree
call bind_methods(C, init, sum)
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:MPI_REDUX,
                   task_map[i,j,l])
    end do
  end do
end do
```



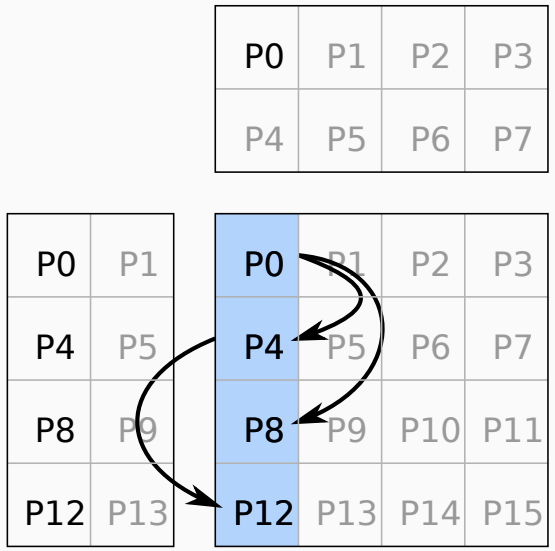
Extended model MX

- MO-0 Data Mapping
- MX-1 Explicit Task Mapping
- MX-2 Comms. from TM
- MX-3 Implicit reduction pattern



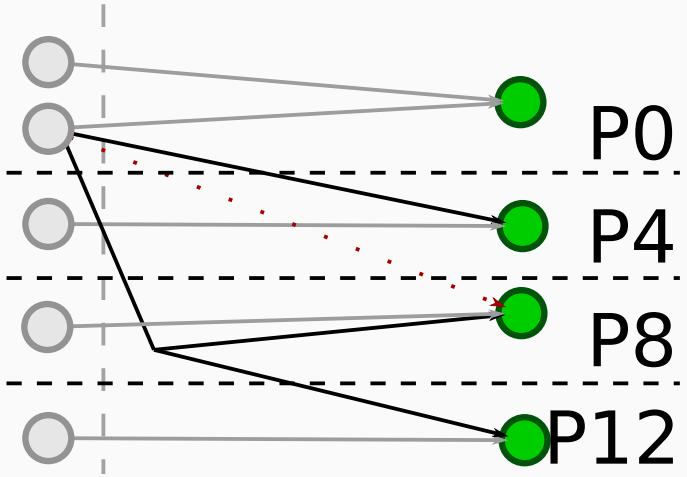
Detecting collective communications (L3)

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]
!dyn_tree: type of broadcast tree
call bind_methods(C, init, sum)
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:MPI_REDUX,
                   task_map[i,j,l])
    end do
  end do
end do
```



Extended model MX

- MO-0 Data Mapping
- MX-1 Explicit Task Mapping
- MX-2 **Coll.** Comms. from TM
- MX-3 Implicit reduction pattern



A. Denis, E. Jeannot, P. Swartvagher, S. Thibault et al, 2020

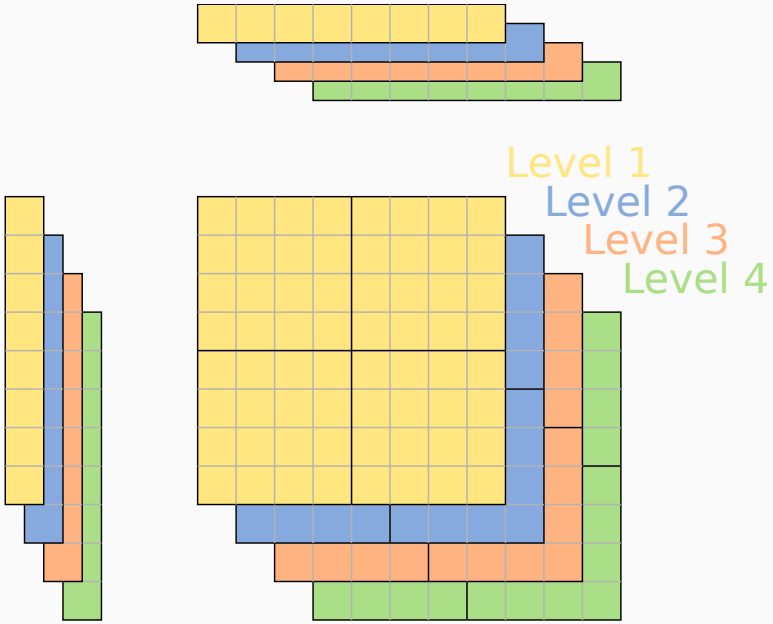
IDEAL EXPRESSION

Task-based 2.5D distributed GEMM

```
!data_map: A,B,C distributed on a grid
!
!

do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)

    end do
  end do
end do
```



Extended model (MX)

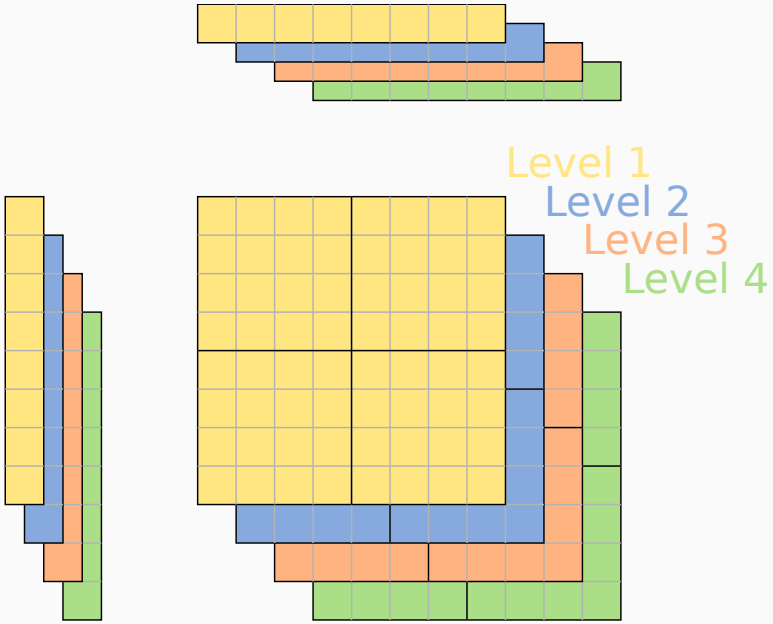
- M0-0 *Data Mapping*

Task-based 2.5D distributed GEMM

```
!data_map: A,B,C distributed on a grid
!
!

do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW)

    end do
  end do
end do
```



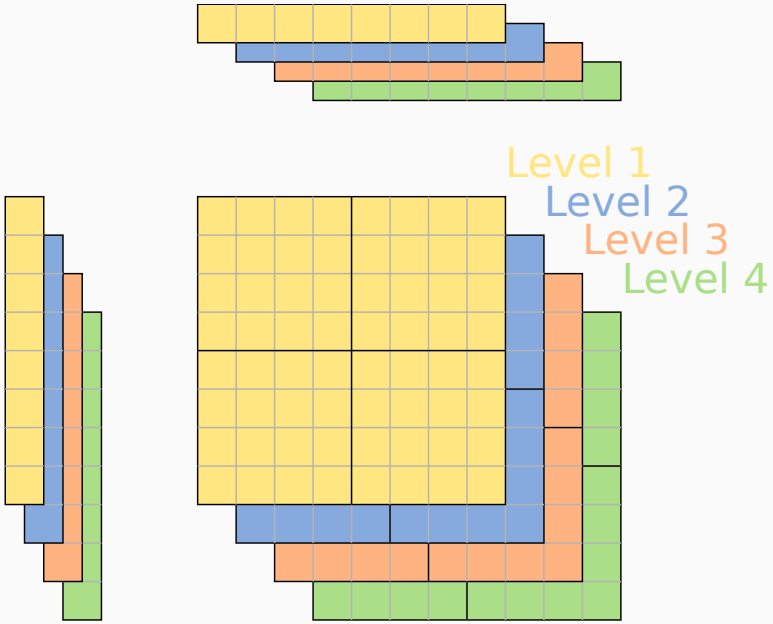
Extended model (MX)

- M0-0 *Data Mapping*
- M0-1 *Task Mapping* **inferred** from Data Mapping (RW)

Task-based 2.5D distributed GEMM

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]
!

do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW,
                   task_map[i,j,l])
    end do
  end do
end do
```



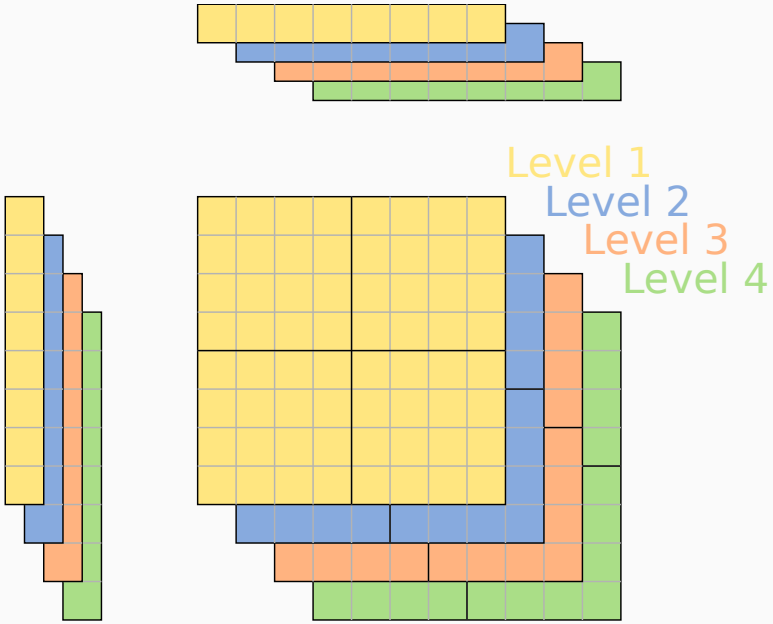
Extended model (MX)

- MO-0 *Data Mapping*
- MX-1 **Explicit** *Task Mapping*

Task-based 2.5D distributed GEMM

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]
!

do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW,
                   task_map[i,j,l])
    end do
  end do
end do
```



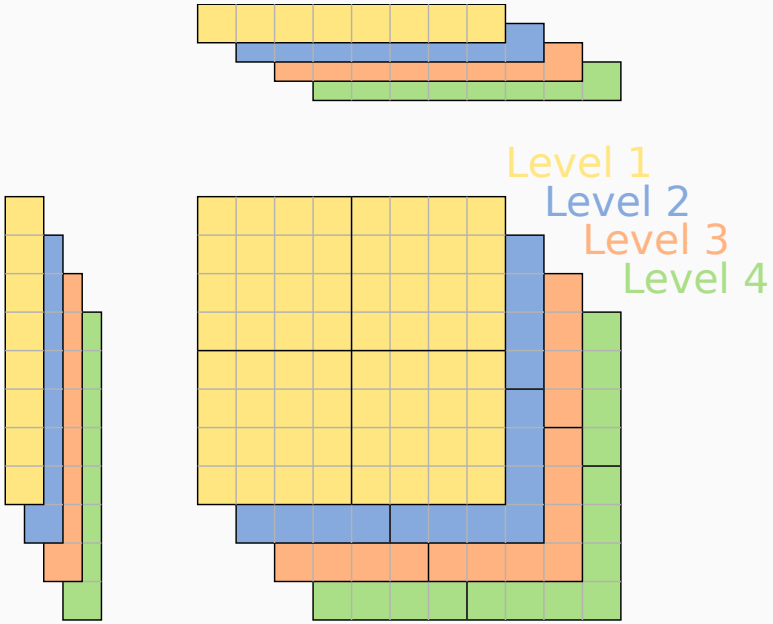
Extended model (MX)

- M0-0 *Data Mapping*
- MX-1 **Explicit** *Task Mapping*
- M0-2 Communicat° **inferred** from Task Mapping

Task-based 2.5D distributed GEMM

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]
!dyn_tree: type of diffusion tree

do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:RW,
                   task_map[i,j,l])
    end do
  end do
end do
```

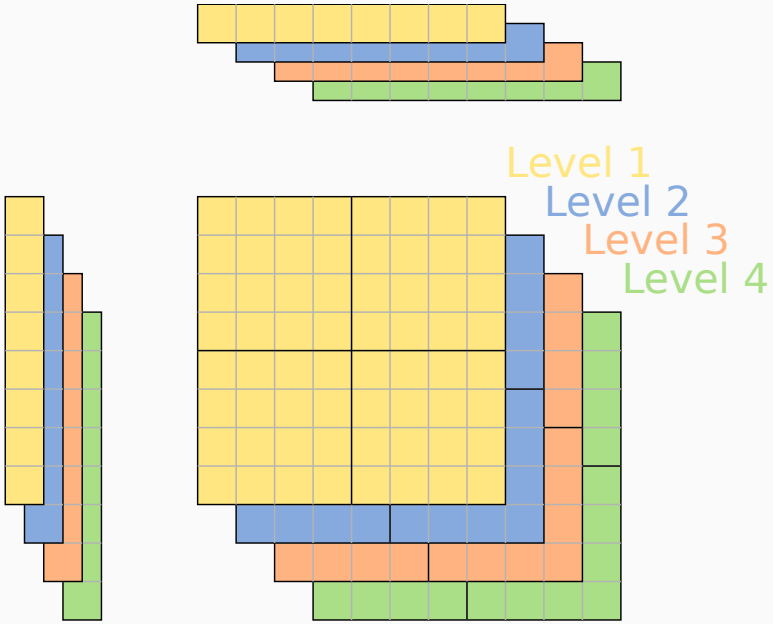


Extended model (MX)

- MO-0 *Data Mapping*
- MX-1 **Explicit** *Task Mapping*
- MX-2 *Collective* Communicat° **inferred** from Task Mapping

Task-based 2.5D distributed GEMM

```
!data_map: A,B,C distributed on a grid
!task_map: executing node of A[i,l]B[l,j]
!dyn_tree: type of broadcast tree
call bind_methods(C, init, red)
do i=1,m
  do j=1,n
    do l=1,k
      call submit( gemm,
                   A[i,l]:R,
                   B[l,j]:R,
                   C[i,j]:MPI_REDUX,
                   task_map[i,j,l])
    end do
  end do
end do
```



Extended model (MX)

- M0-0 *Data Mapping*
- MX-1 **Explicit** *Task Mapping*
- MX-2 *Collective* Communicat° **inferred** from Task Mapping
- MX-3 **implicit** reduction pattern

ACTUAL IMPLEMENTATION AND RESULTS

Tools used

- StarPU
  - STF-based *runtime*
  - Developed by Storm Team in INRIA - Bordeaux

Tools used

- StarPU
  - STF-based *runtime*
  - Developed by Storm Team in INRIA - Bordeaux

- **L1** StarPU provides the API

Tools used

- StarPU
  - STF-based *runtime*
  - Developed by Storm Team in INRIA - Bordeaux

- **L1** StarPU provides the API
- **L2** StarPU provides the API *in shared memory*
  - Adaptations are required for distributed computing

Tools used

- StarPU
  - STF-based *runtime*
  - Developed by Storm Team in INRIA - Bordeaux
- NewMadeleine
  - Communications engine based on `MPI` standards
  - Developed by TaDaaM Team in INRIA- Bordeaux

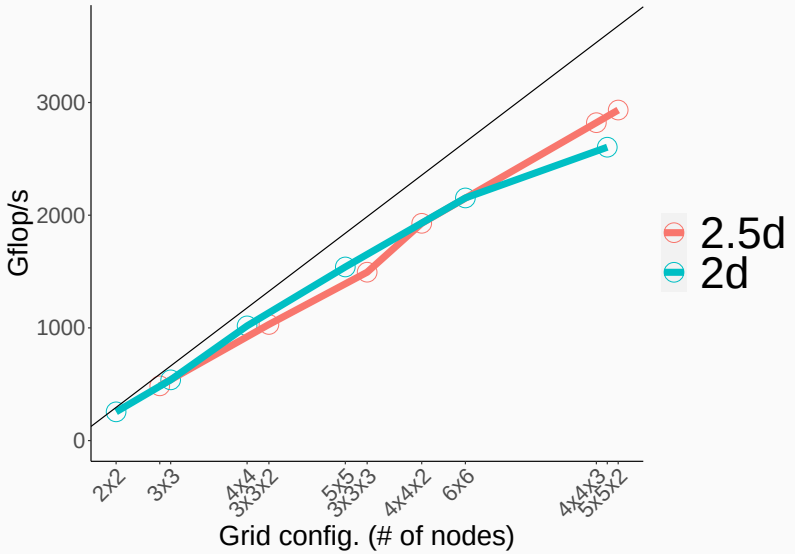
- **L1** StarPU provides the API
- **L2** StarPU provides the API *in shared memory*
  - Adaptations are required for distributed computing
- **L3** NewMadeleine & StarPU provide the detection
  - Ongoing research (Philippe Swartvagher's PhD. thesis)

Preliminary Results

Ran on **CALMIP** Olympe with *Intel Gold Skylake CPUs*

- 1 core used per node
- Strong scaling using sq. mat. of size 65k

≈ 80% peak performance

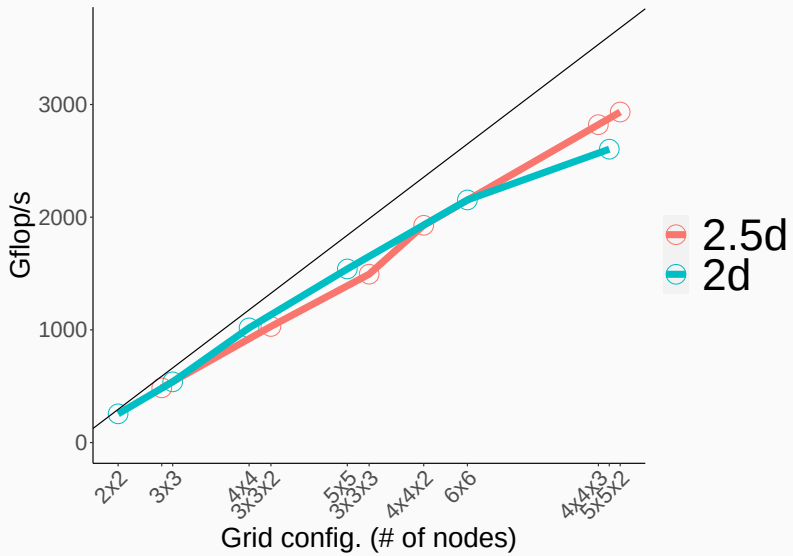


Preliminary Results

Ran on **CALMIP** Olympe with *Intel Gold Skylake CPUs*

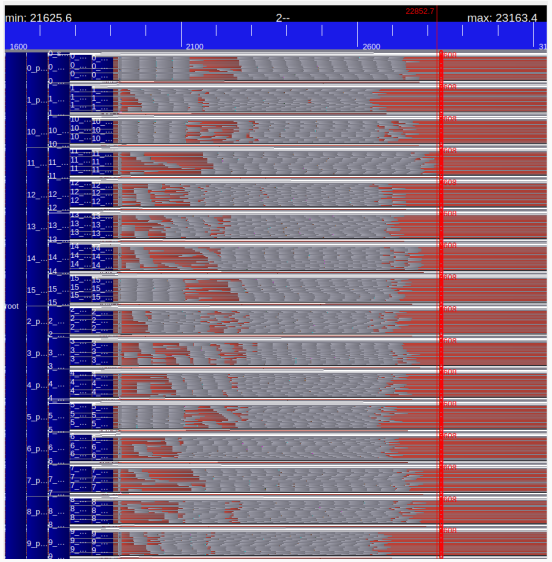
- 1 core used per node
- Strong scaling using sq. mat. of size 65k

≈ 80% peak performance



- 16 cores used per node
- sq. mat. of size 16k
- limited to 2D

≈ 55% peak performance



Trace legend

- **grey** : gemm tasks
- **red** : idle time

CONCLUSION

Contributions

- Study of extensions to the STF model to **compactly** write **distributed** algorithms
  - Model `M0` $\rightarrow$ Model `MX`
- Design and implementation of a prototype using complex patterns
  - Exchanges on `StarPU` (see gitlab issue#2, issue#3)

Conclusion

Contributions

- Study of extensions to the STF model to **compactly** write **distributed** algorithms
  - Model `M0` $\rightarrow$ Model `MX`
- Design and implementation of a prototype using complex patterns
  - Exchanges on `StarPU` (see gitlab issue#2, issue#3)

Future Works

- Broaden the scope of the experiments
- Further extend the model
  - Implement other algorithms (i.e. factorizations)
  - Adapt to sparse algebra

Thank you

APPENDIX

Behaviour

Test Program

With $2n$ nodes (n pairs)

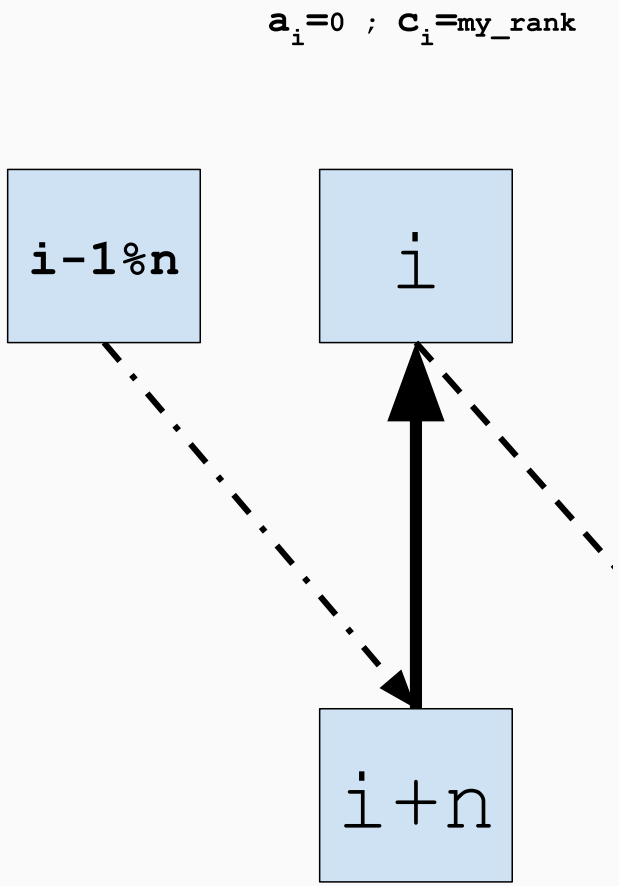
Adjusting the reduction

Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
```

Behaviour



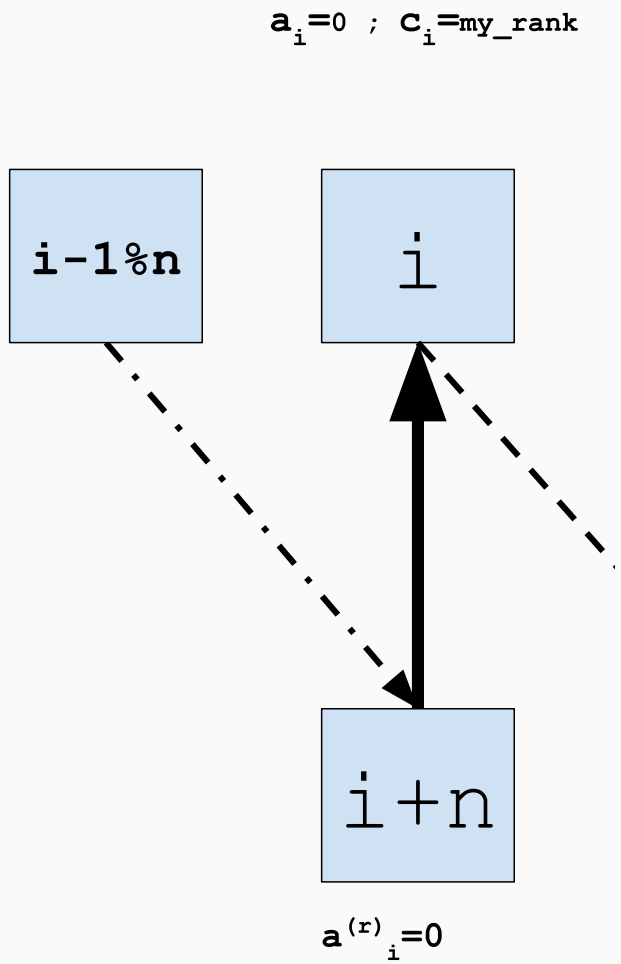
Adjusting the reduction

Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
```

Behaviour

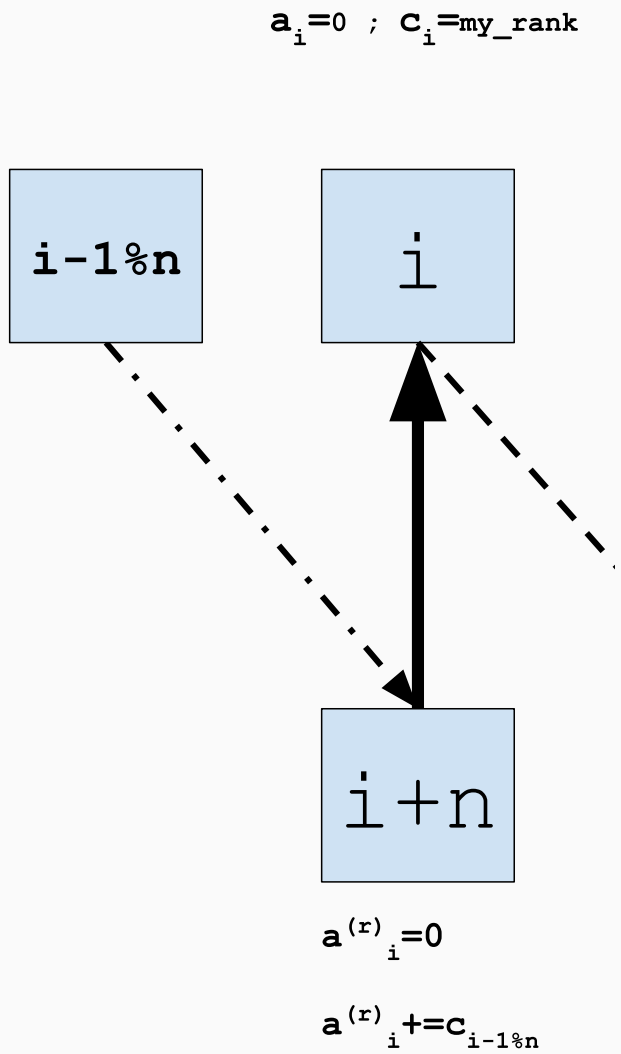


Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
```

Behaviour

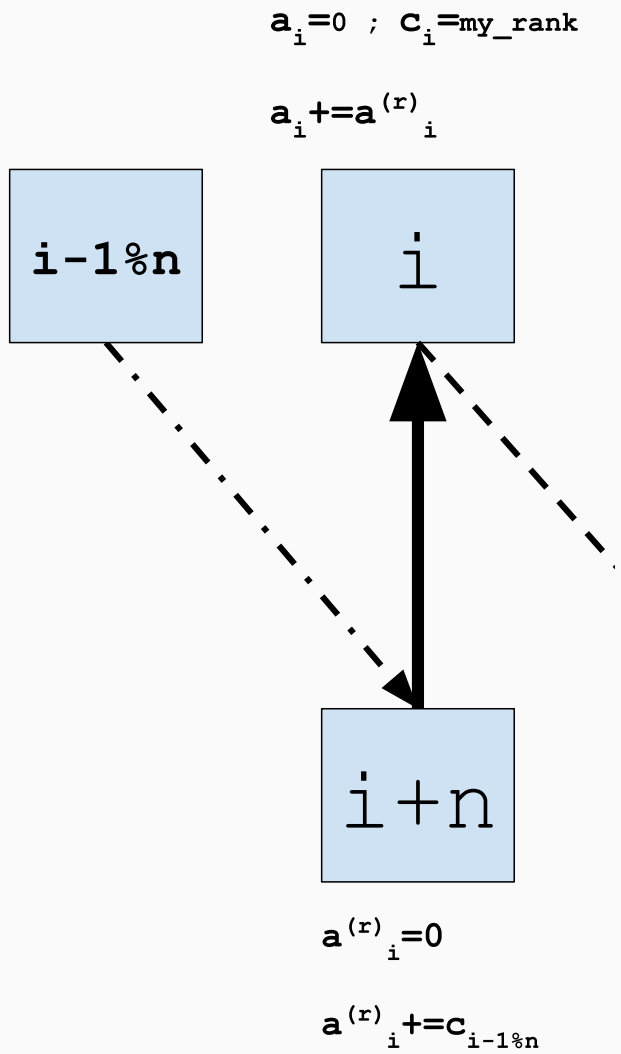


Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
a_i += a_i^(r)
```

Behaviour

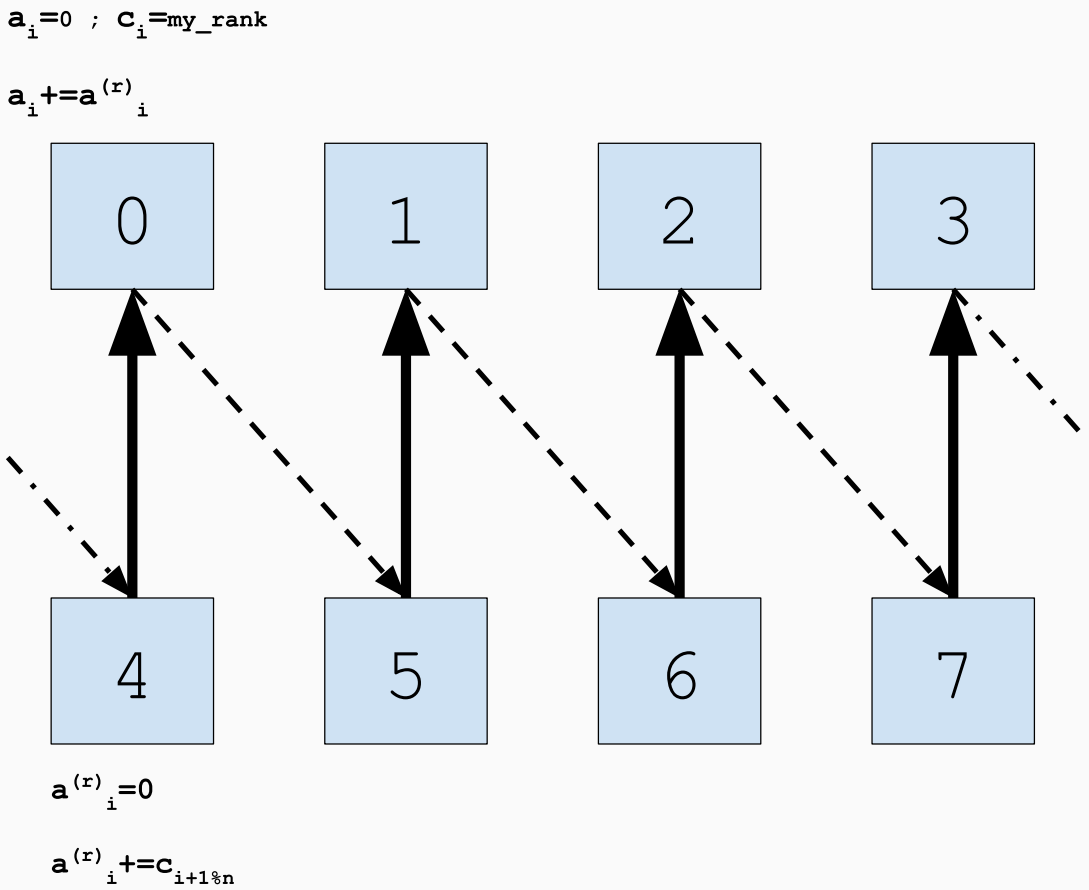


Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
a_i += a_i^(r)
```

Behaviour (n=4)



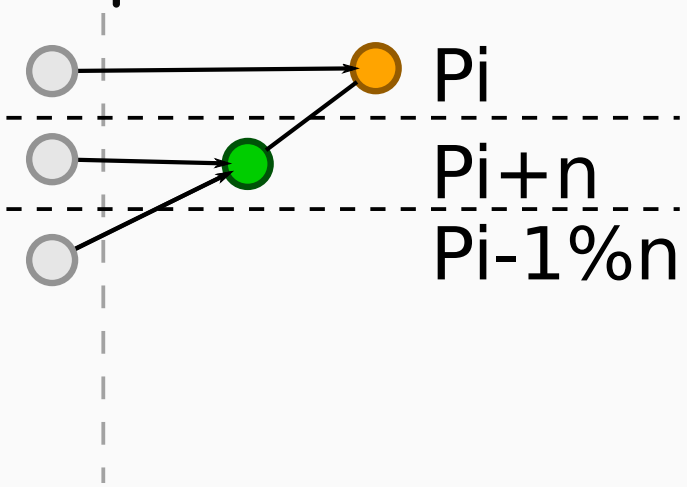
Adjusting the reduction

Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
a_i += a_i^(r)
```

Expected DAG



Adjusting the reduction

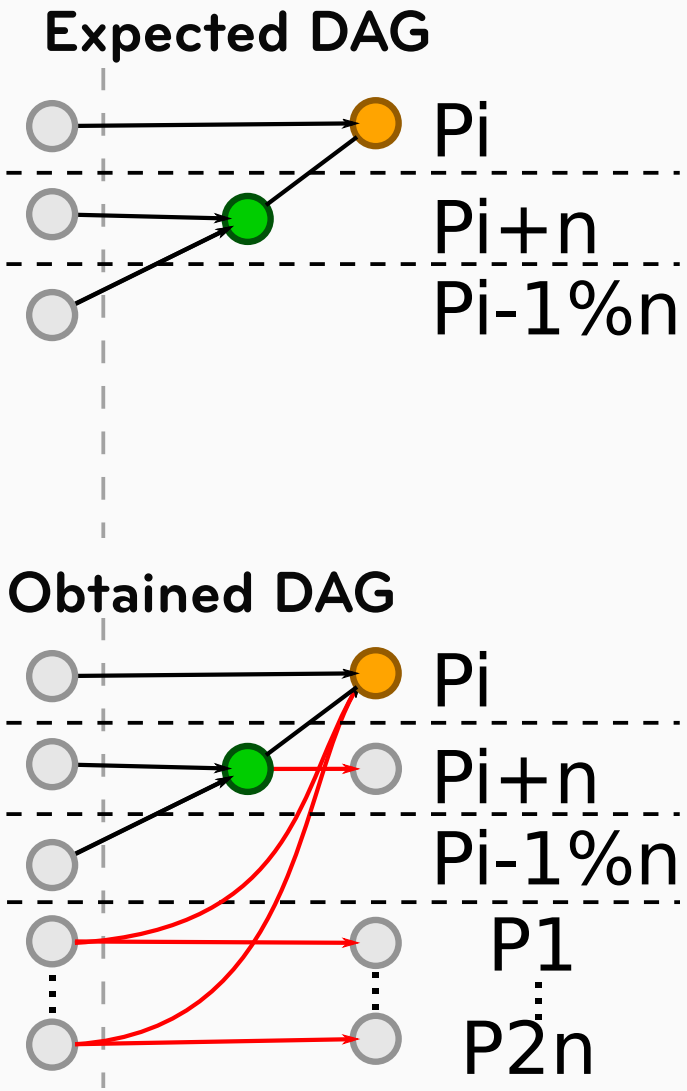
Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
a_i += a_i^(r)
```

Issues

- All nodes partake in the reduction



Adjusting the reduction

Test Program

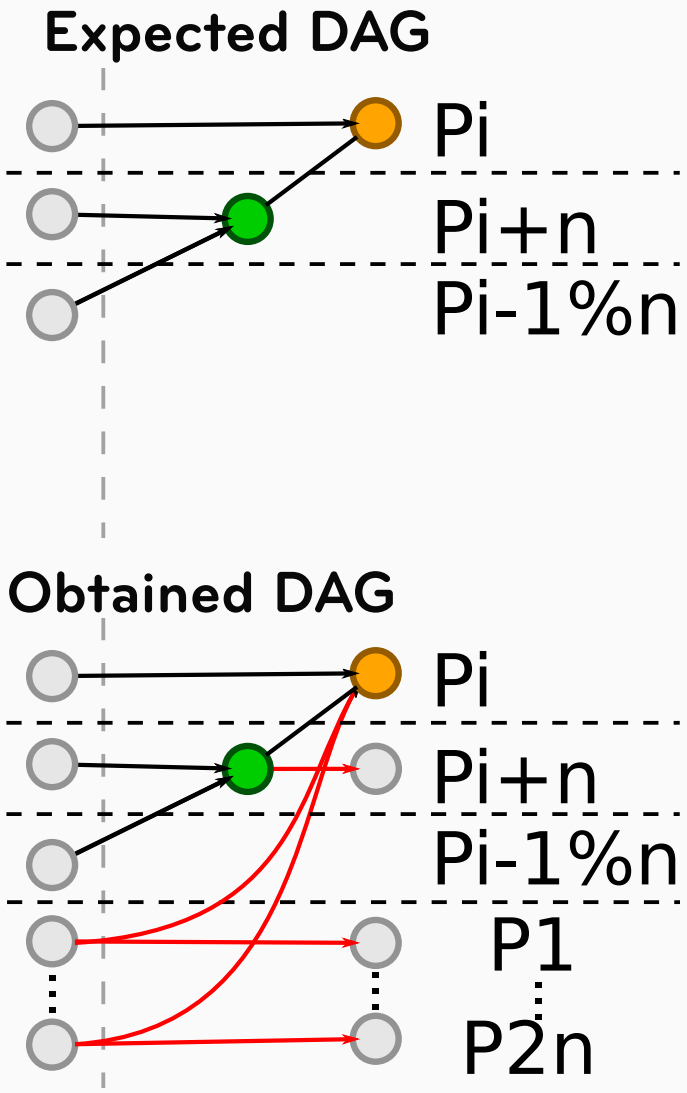
With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
a_i += a_i^(r)
```

Issues

- All nodes partake in the reduction

$\Rightarrow$ Many instances w/ `REDUX` access



Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
a_i += a_i^(r)
```

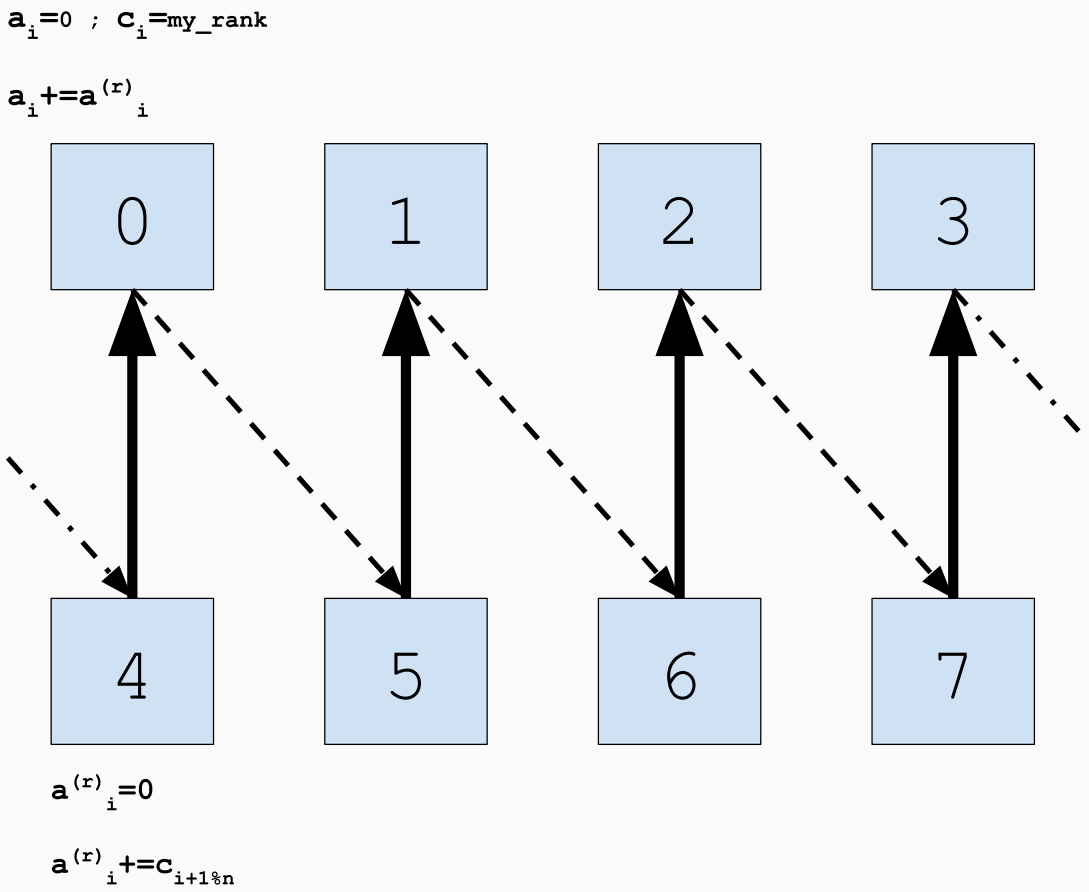
Issues

- All nodes partake in the reduction

$\Rightarrow$ Many instances w/ `REDUX` access

Solution

Splitting `MPI_COMM_WORLD`



Test Program

With $2n$ nodes (n pairs)

```
a_i = 0, c_i = my_rank
a_i^(r) = 0
a_i^(r) += c_(i-1%n)
a_i += a_i^(r)
```

Issues

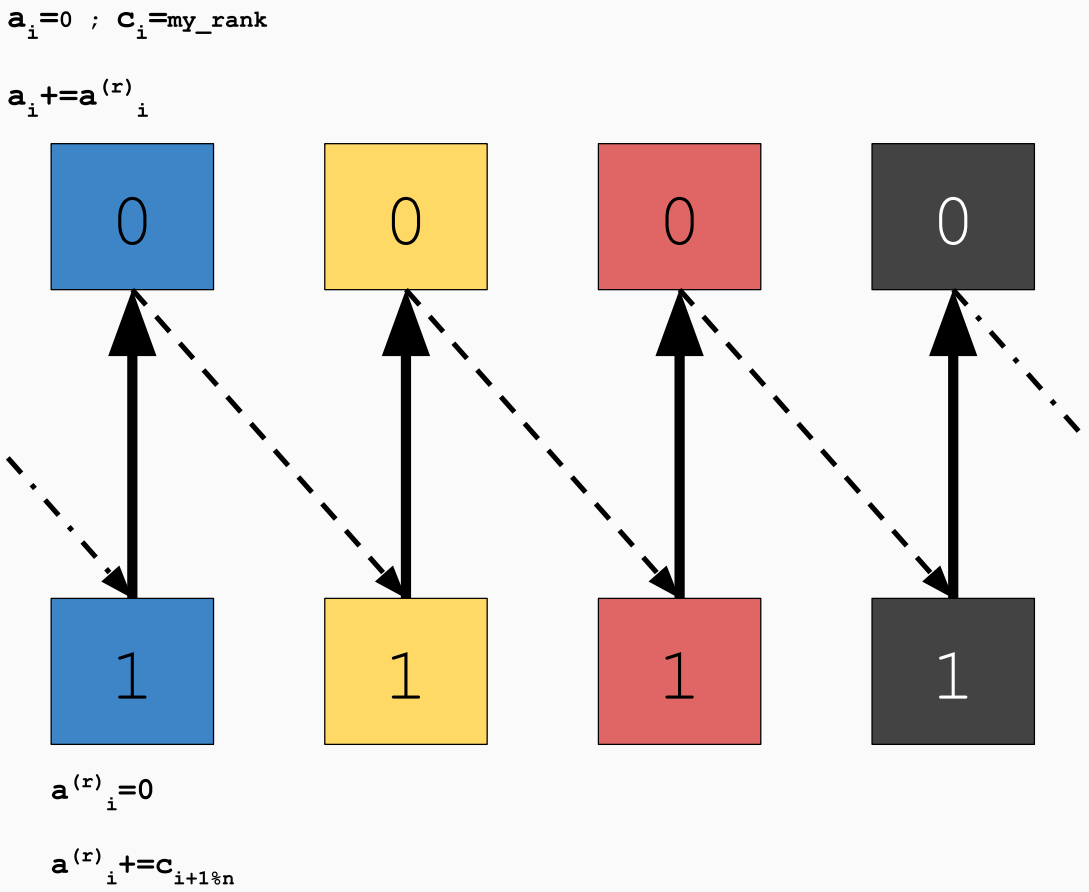
- All nodes partake in the reduction

⇒ Many instances w/ `REDUX` access

Solution

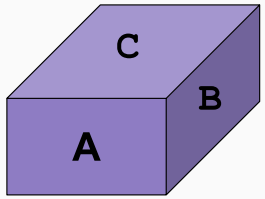
- One communicator per pair to *reduce*

Splitting `MPI_COMM_WORLD`



Tensor representation

`one element ↔ one task`



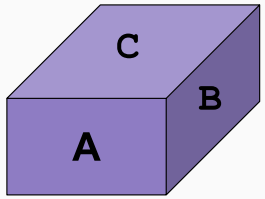
Reminder

- $C_{ij} += A_{il} * B_{lj}$ on $task_{ijl}$

SUMMA variants

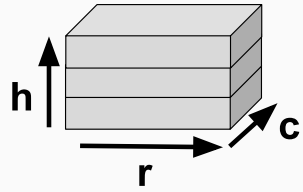
Tensor representation

`one element ↔ one task`



Grid of nodes

`h layers of r*c nodes`

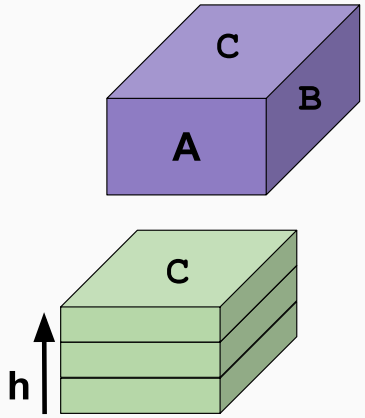


Reminder

- $C_{ij} += A_{il} * B_{lj}$ on $task_{ijl}$
- X_{ij} on $j\%c + c * (i\%r)$ (2D Block Cyclic distrib.)

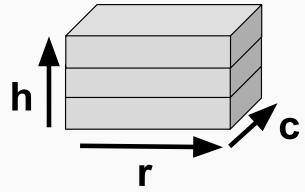
Tensor representation

`one element ↔ one task`



Grid of nodes

`h layers of r*c nodes`



Task_map SUMMA-C

- $task_{ijl} :=$
  $j\%c + c * (i\%r) + r * c * (l\%h)$

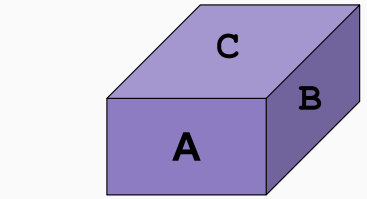
Reminder

- $C_{ij} += A_{il} * B_{lj}$ on $task_{ijl}$
- X_{ij} on $j\%c + c * (i\%r)$ (2D Block Cyclic distrib.)

SUMMA variants

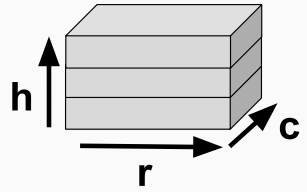
Tensor representation

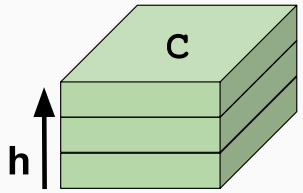
one element ↔ one task

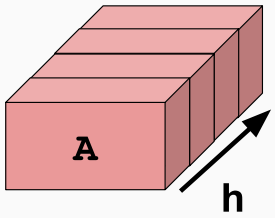


Grid of nodes

h layers of r*c nodes







Task_map SUMMA-C

- $task_{ijl} := j\%c + c * (i\%r) + r * c * (l\%h)$

Task_map SUMMA-A

- $task_{ijl} := l\%c + c * (i\%r) + r * c * (j\%h)$

Reminder

- $C_{ij} += A_{il} * B_{lj}$ on $task_{ijl}$
- X_{ij} on $j\%c + c * (i\%r)$ (2D Block Cyclic distrib.)

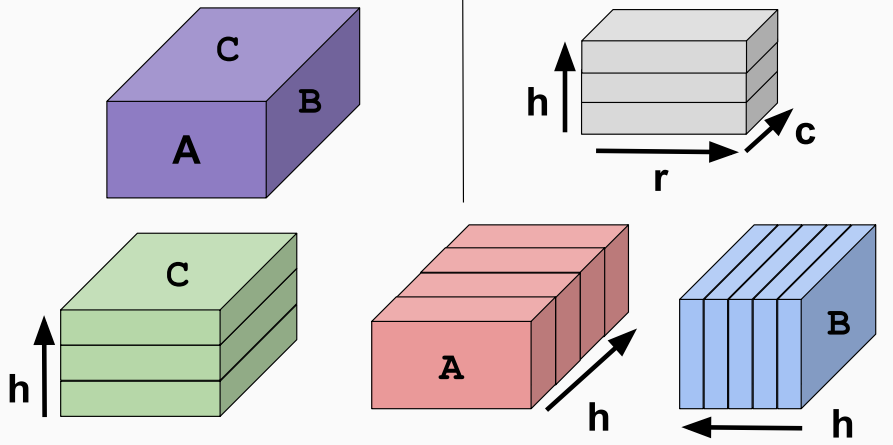
SUMMA variants

Tensor representation

one element ↔ one task

Grid of nodes

h layers of r*c nodes



Task_map SUMMA-C

- $task_{ijl} :=$
  $j\%c + c * (i\%r) + r * c * (l\%h)$

Task_map SUMMA-A

- $task_{ijl} :=$
  $l\%c + c * (i\%r) + r * c * (j\%h)$

Task_map SUMMA-B

- $task_{ijl} :=$
  $j\%c + c * (l\%r) + r * c * (i\%h)$

Reminder

- $C_{ij} += A_{il} * B_{lj}$ on $task_{ijl}$
- X_{ij} on $j\%c + c * (i\%r)$ (2D Block Cyclic distrib.)