TASK-BASED PROGRAMMING MODELS FOR SCALABLE ALGORITHMS

Application to matrix multiplication

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

February 10, 2022

Université de Toulouse - IRIT

INTRODUCTION

Algorithms, architectures and programming models

Architectures are becoming increasingly **large**, **heterogeneous** and **complex**. This has two effects:

- Algorithms have to handle communications to achieve high scalability
  - Communication/computation overlap
  - Communication-avoiding algorithms
- Software should deliver a reliable abstraction to build mathematical software
  - parallel programming models
  - runtime systems

Are currently available parallel programming models and runtimes expressive enough to implement scalable algorithms in a efficient, portable and maintainable way?

Algorithms, architectures and programming models

Architectures are becoming increasingly **large**, **heterogeneous** and **complex**. This has two effects:

- Algorithms have to handle communications to achieve high scalability
  - Communication/computation overlap
  - Communication-avoiding algorithms

- Software should deliver a reliable abstraction to build mathematical software
  - parallel programming models
  - runtime systems

Are currently available parallel programming models and runtimes expressive enough to implement scalable algorithms in a efficient, portable and maintainable way?

Algorithms, architectures and programming models

Architectures are becoming increasingly **large**, **heterogeneous** and **complex**. This has two effects:

- Algorithms have to handle communications to achieve high scalability
  - Communication/computation overlap
  - Communication-avoiding algorithms
- Software should deliver a reliable abstraction to build mathematical software
  - parallel programming models
  - runtime systems

Are currently available parallel programming models and runtimes expressive enough to implement scalable algorithms in a efficient, portable and maintainable way?

Algorithms, architectures and programming models

Architectures are becoming increasingly **large**, **heterogeneous** and **complex**. This has two effects:

- Algorithms have to handle communications to achieve high scalability
  - Communication/computation overlap
  - Communication-avoiding algorithms
- Software should deliver a reliable abstraction to build mathematical software
  - parallel programming models
  - runtime systems

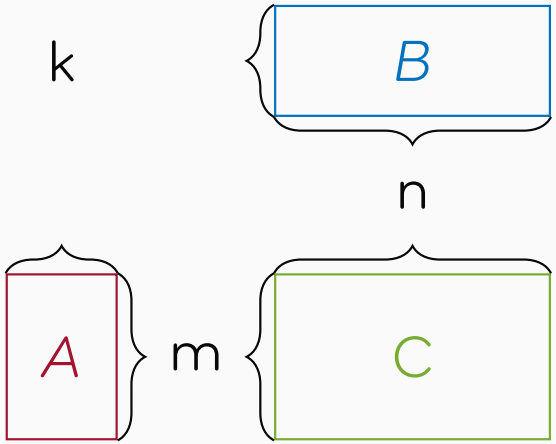
Are currently available parallel programming models and runtimes expressive enough to implement scalable algorithms in a efficient, portable and maintainable way?

Algorithms, architectures and programming models

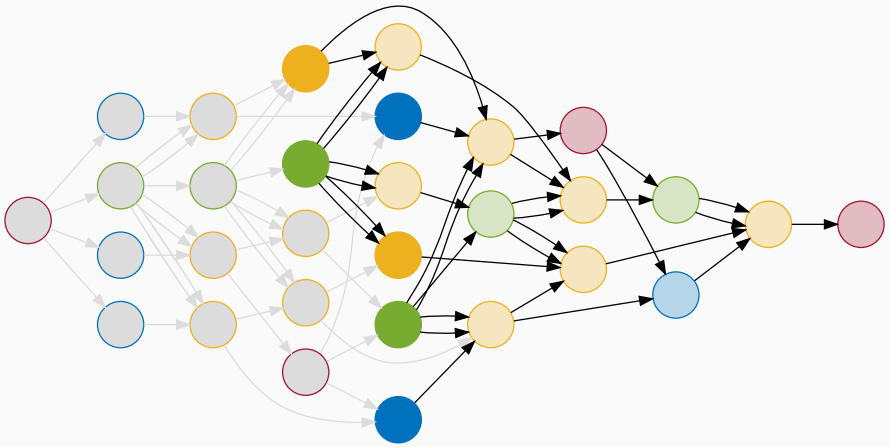
We focus on two aspects of this large question

- Algorithms: scalable dense matrix multiplication (GEMM) algorithms such as SUMMA and 2.5D SUMMA

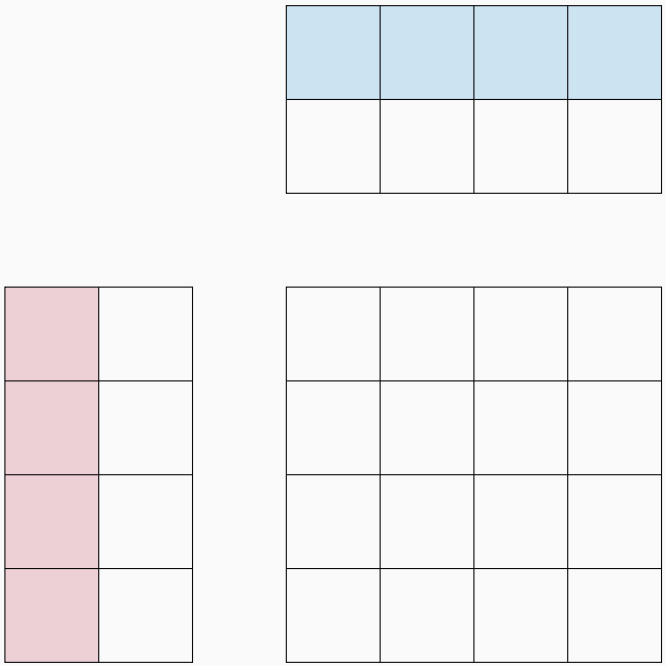
$$C = \alpha AB + \beta C$$



- Software: task-based parallel programming paradigm through the sequential task flow (STF) model



BACKGROUND

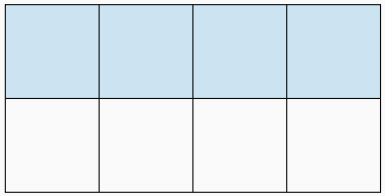


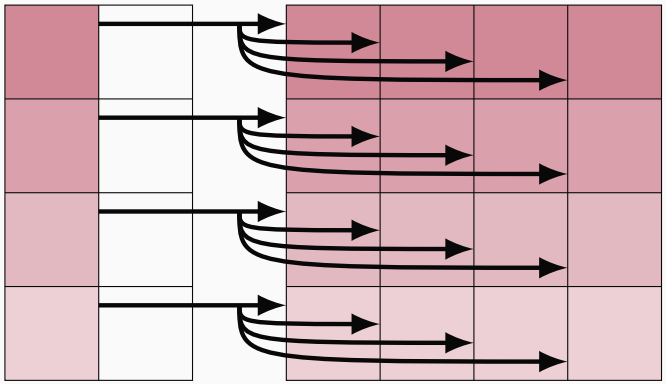
2D C-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of C computes
- **Communications**
  - row-wise broadcast of A
  - column-wise broadcast of B

```
! I am r × c
do l=1, k
   forall i, i%p==r :
      bcast(A_{i,l}, r×*)
   forall j, j%q==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall j, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C_{i,j})
end do
```

Geijn and Watts, 1997



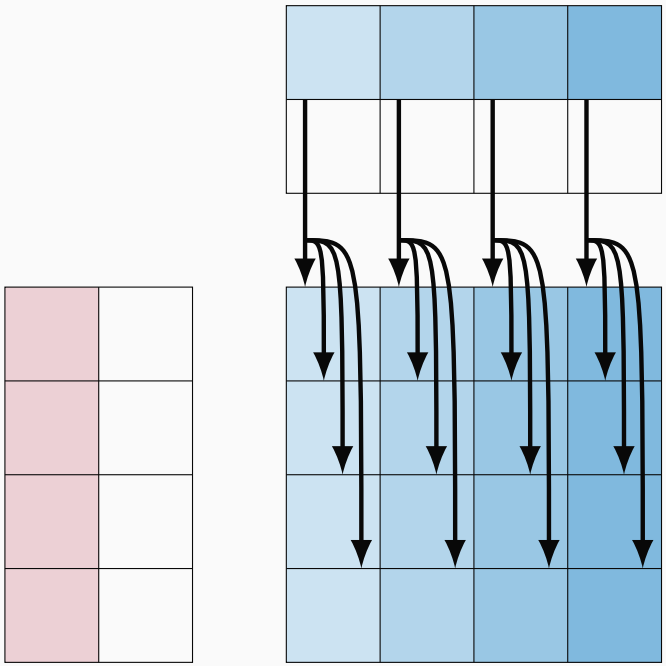


2D C-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of C computes
- **Communications**
  - row-wise **broadcast** of A
  - column-wise broadcast of B

```
! I am r × c
do l=1, k
   forall i, i%p==r :
      bcast(A_{i,l}, r×*)
   forall j, j%q==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall j, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C_{i,j})
end do
```

Geijn and Watts, 1997

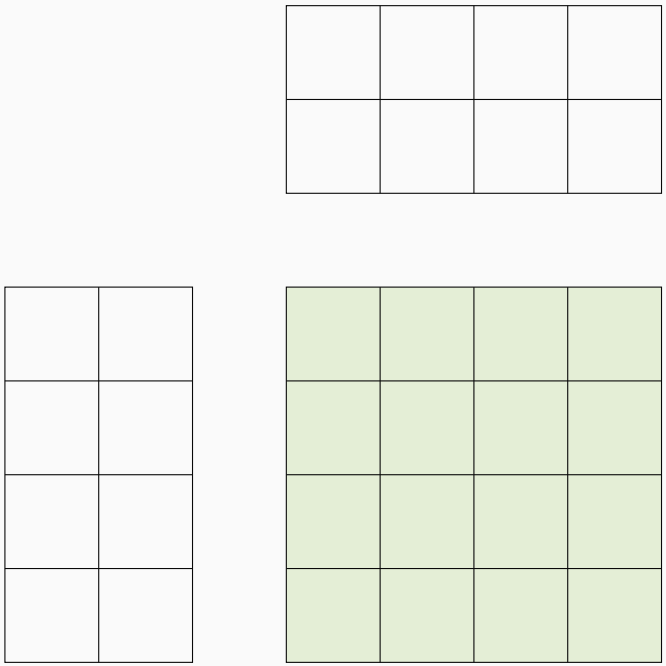


2D C-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of C computes
- **Communications**
  - row-wise broadcast of A
  - column-wise **broadcast** of B

```
! I am r × c
do l=1, k
   forall i, i%p==r :
      bcast(A_{i,l}, r×*)
   forall j, j%q==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall j, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C_{i,j})
end do
```

Geijn and Watts, 1997

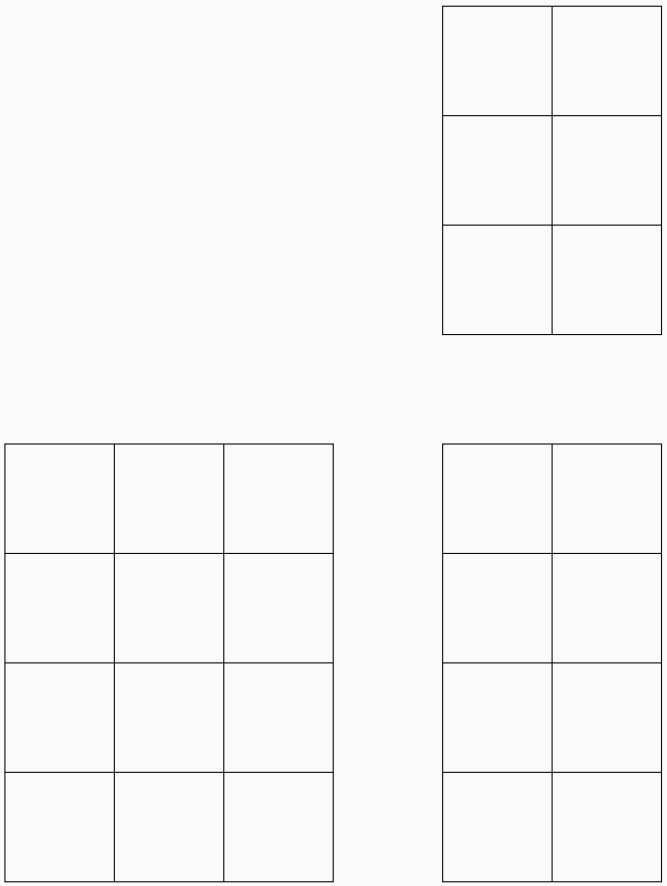


2D C-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of C **computes**
- **Communications**
  - row-wise broadcast of A
  - column-wise broadcast of B

```
! I am r × c
do l=1, k
   forall i, i%p==r :
      bcast(A_{i,l}, r×*)
   forall j, j%q==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall j, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C_{i,j})
end do
```

Geijn and Watts, 1997

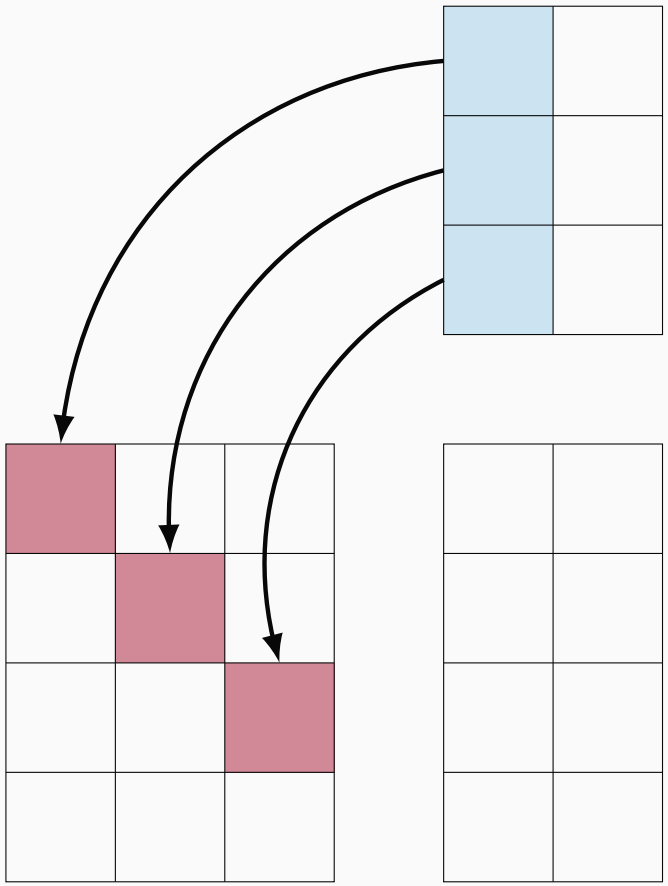


2D A-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of A computes
- **Communications**
  - column-wise scatter+broadcast of B
  - row-wise reduce of C

```
! I am r × c
do j=1, n
   forall l, l%p==r.and.l%q==c :
      recv(B_{l,j}, l%p×j%q)
   forall l, l%p==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall l, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C^r_{i,i})
      reduce(C^r_{i,i}, i%r×j%q)
end do
```

Geijn and Watts, 1997

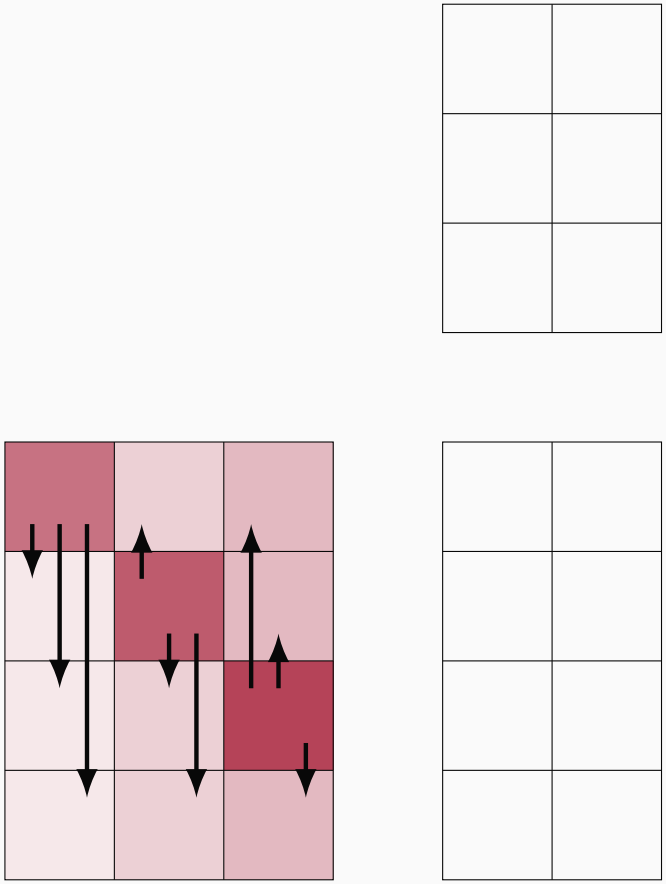


2D A-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of A computes
- **Communications**
  - column-wise **scatter**+broadcast of B
  - row-wise reduce of C

```
! I am r × c
do j=1, n
   forall l, l%p==r.and.l%q==c :
      recv(B_{l,j}, l%p×j%q)
   forall l, l%p==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall l, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C^r_{i,i})
      reduce(C^r_{i,i}, i%r×j%q)
end do
```

Geijn and Watts, 1997

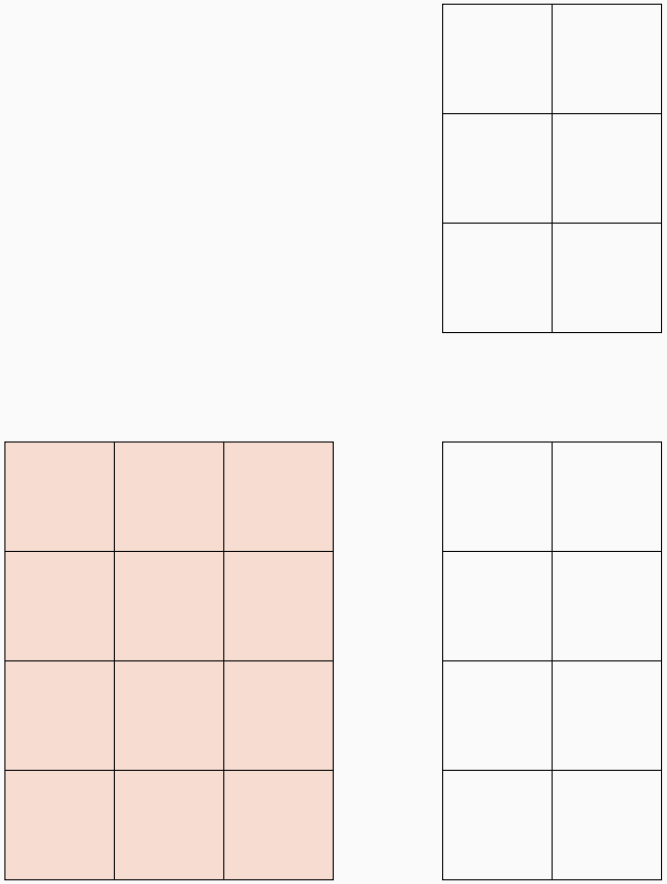


2D A-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of A computes
- **Communications**
  - column-wise scatter+**broadcast** of B
  - row-wise reduce of C

```
! I am r × c
do j=1, n
   forall l, l%p==r.and.l%q==c :
      recv(B_{l,j}, l%p×j%q)
   forall l, l%p==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall l, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C^r_{i,j})
      reduce(C^r_{i,j}, i%r×j%q)
end do
```

Geijn and Watts, 1997

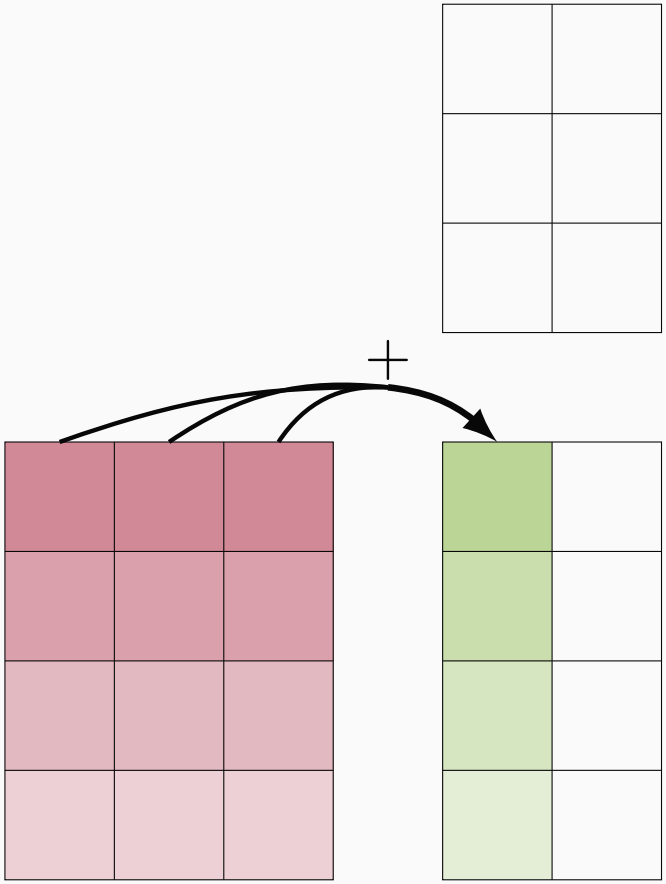


2D A-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of A **computes**
- **Communications**
  - column-wise scatter+broadcast of B
  - row-wise reduce of C

```
! I am r × c
do j=1, n
   forall l, l%p==r.and.l%q==c :
      recv(B_{l,j}, l%p×j%q)
   forall l, l%p==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall l, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C^r_{i,i})
      reduce(C^r_{i,i}, i%r×j%q)
end do
```

Geijn and Watts, 1997

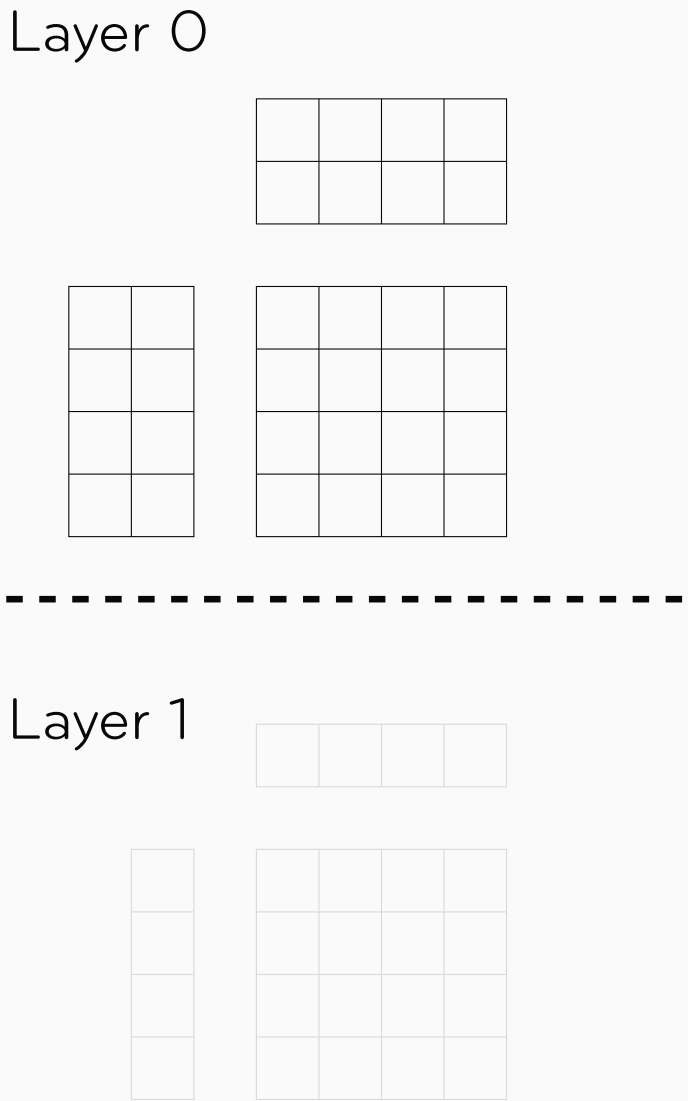


2D A-stationary

- **Data mapping**
  - 2D block-cyclic
- **Task mapping**
  - owner of A computes
- **Communications**
  - column-wise scatter+broadcast of B
  - row-wise **reduce** of C

```
! I am r × c
do j=1, n
   forall l, l%p==r.and.l%q==c :
      recv(B_{l,j}, l%p×j%q)
   forall l, l%p==c :
      bcast(B_{l,j}, *×c)
   forall i, i%p==r :
      forall l, j%q==c :
         call gemm(A_{i,l}, B_{l,j}, C^r_{i,i})
      reduce(C^r_{i,i}, i%r×j%q)
end do
```

Geijn and Watts, 1997



2.5D C-stationary

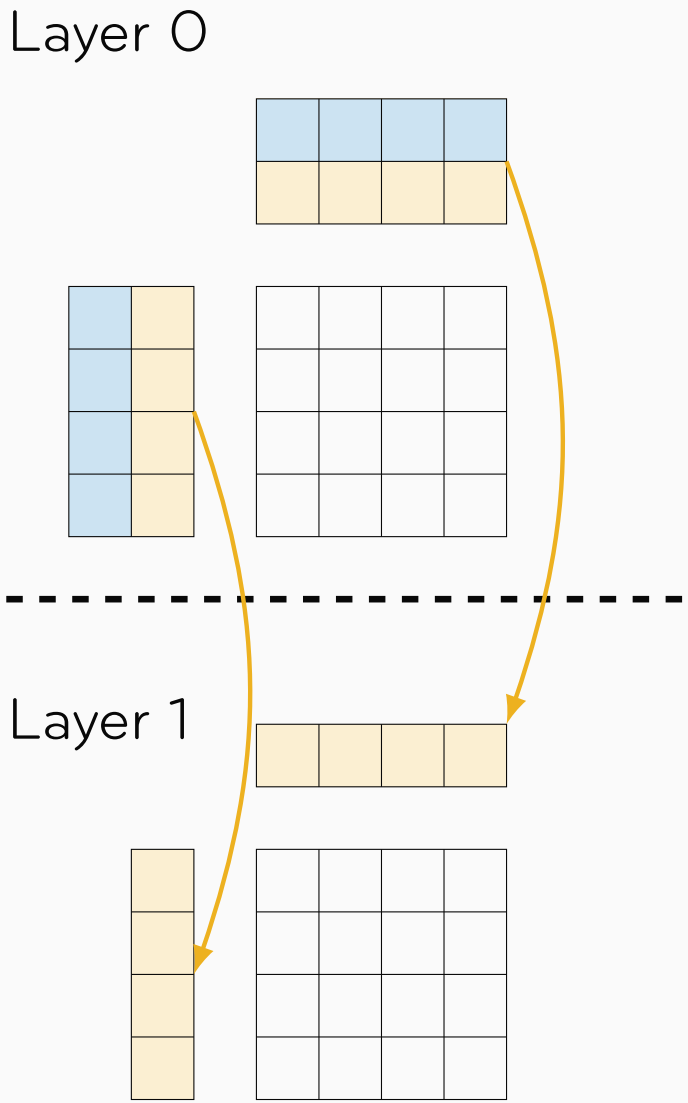
- **Data mapping**
  - 2D block-cyclic on 0^{th} layer
- **Task mapping**
  - Not trivially related to data mapping
- **Communications**
  - row-wise scatter+broadcast of A
  - column-wise scatter+broadcast of B
  - aisle-wise reduce of C

```
! I am r × c × h
scatter A_{i,j} on layer j/h
scatter B_{i,j} on layer i/h

forall layer=1,h:
   call SUMMA_2DC(A[:, (layer : layer + 1) * k/h],
                  B[(layer : layer + 1) * k/h, :], C^h)

forall i, i%p==r :
  forall j, j%q==c :
    reduce(C^h_{i,j}, r×c×0)
```

Irony, Toledo, and Tiskin, 2004



2.5D C-stationary

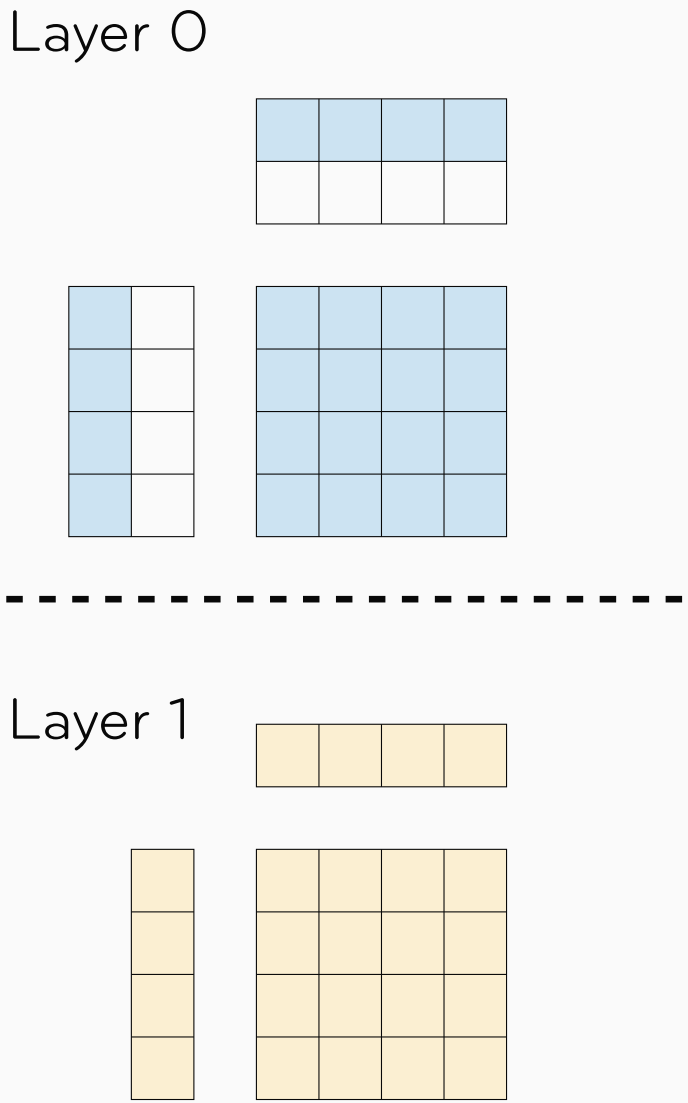
- **Data mapping**
  - 2D block-cyclic on 0^{th} layer
- **Task mapping**
  - Not trivially related to data mapping
- **Communications**
  - row-wise **scatter**+broadcast of A
  - column-wise **scatter**+broadcast of B
  - aisle-wise reduce of C

```
! I am r × c × h
scatter A_{i,j} on layer j/h
scatter B_{i,j} on layer i/h

forall layer=1,h:
   call SUMMA_2DC(A[:, (layer : layer + 1) * k/h],
                  B[(layer : layer + 1) * k/h, :], C^h)

forall i, i%p==r :
  forall j, j%q==c :
    reduce(C^h_{i,j}, r×c×0)
```

Irony, Toledo, and Tiskin, 2004



2.5D C-stationary

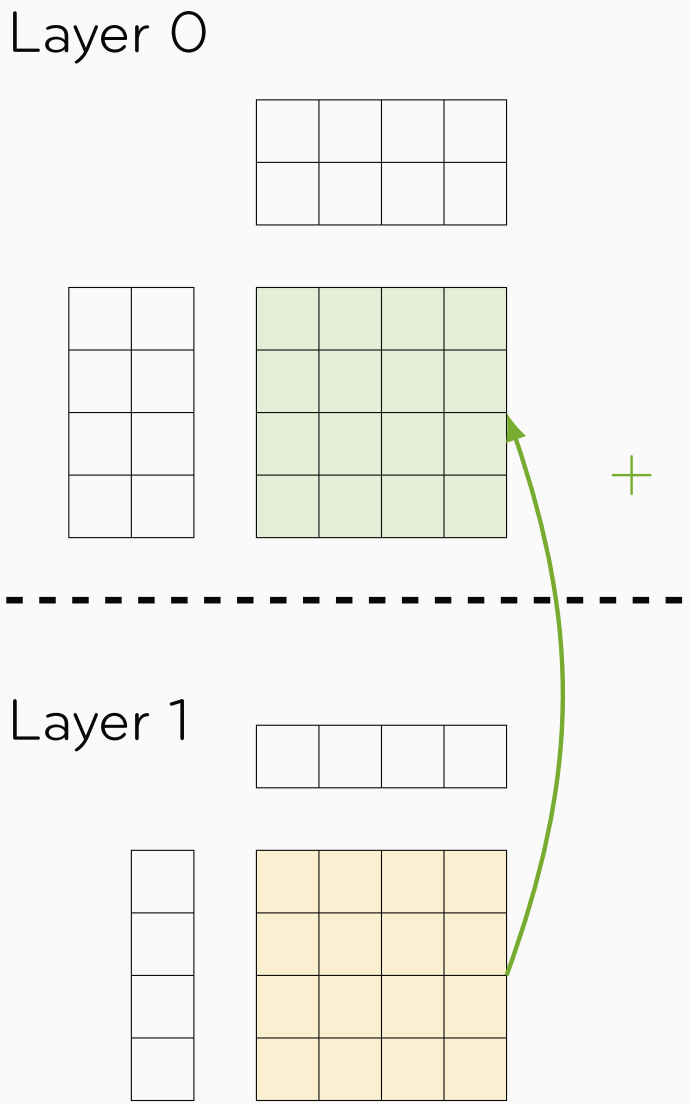
- **Data mapping**
  - 2D block-cyclic on 0^{th} layer
- **Task mapping**
  - Not trivially related to data mapping
- **Communications**
  - row-wise scatter+**broadcast** of A
  - column-wise scatter+**broadcast** of B
  - aisle-wise reduce of C

```
! I am r × c × h
scatter A_{i,j} on layer j/h
scatter B_{i,j} on layer i/h

forall layer=1,h:
   call SUMMA_2DC(A[:, (layer : layer + 1) * k/h],
                  B[(layer : layer + 1) * k/h, :], C^h)

forall i, i%p==r :
  forall j, j%q==c :
    reduce(C^h_{i,j}, r×c×0)
```

Irony, Toledo, and Tiskin, 2004



2.5D C-stationary

- **Data mapping**
  - 2D block-cyclic on 0^{th} layer
- **Task mapping**
  - Not trivially related to data mapping
- **Communications**
  - row-wise scatter+broadcast of A
  - column-wise scatter+broadcast of B
  - aisle-wise **reduce** of C

```
! I am r × c × h
scatter A_{i,j} on layer j/h
scatter B_{i,j} on layer i/h

forall layer=1,h:
   call SUMMA_2DC(A[:, (layer : layer + 1) * k/h],
                  B[(layer : layer + 1) * k/h, :], C^h)

forall i, i%p==r :
  forall j, j%q==c :
    reduce(C^h_{i,j}, r×c×0)
```

Irony, Toledo, and Tiskin, 2004

The sequential task flow (STF) Model

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
```

Shared-memory STF GEMM

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


DAG of GEMM

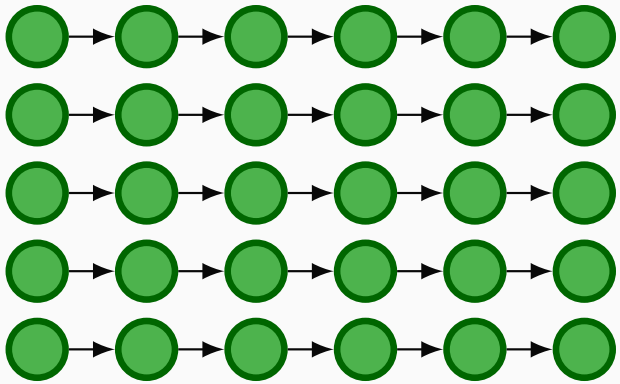
The sequential task flow (STF) Model

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
```

Shared-memory STF GEMM

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



DAG of GEMM

Objectives

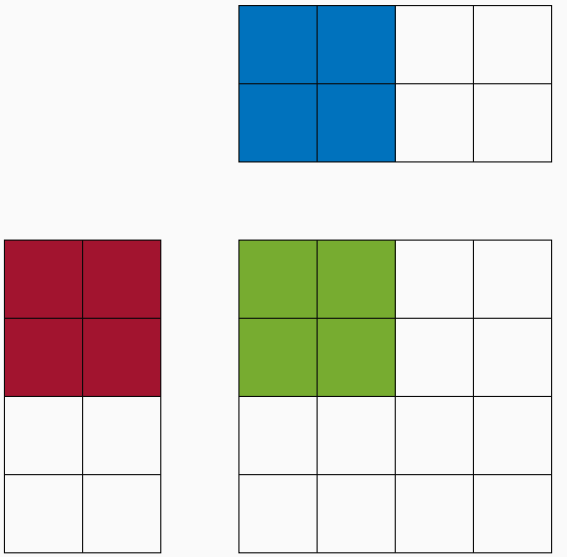
- Can we use the **STF model** to express a **scalable distributed-memory GEMM** algorithm ?

- How **efficient** is the resulting expression compared to reference libraries ?

DISTRIBUTED-MEMORY STF GEMM ALGORITHM

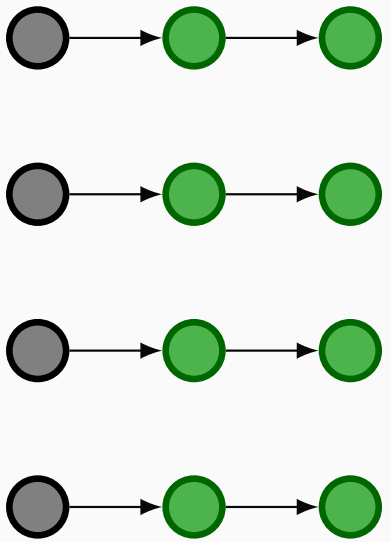
Baseline STF model

```
!
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(gemm,
                       A_{i,l}:R,
                       B_{l,j}:R,
                       C_{i,j}:RW)
    end do
  end do
end do
```



Baseline model (M0)[a]

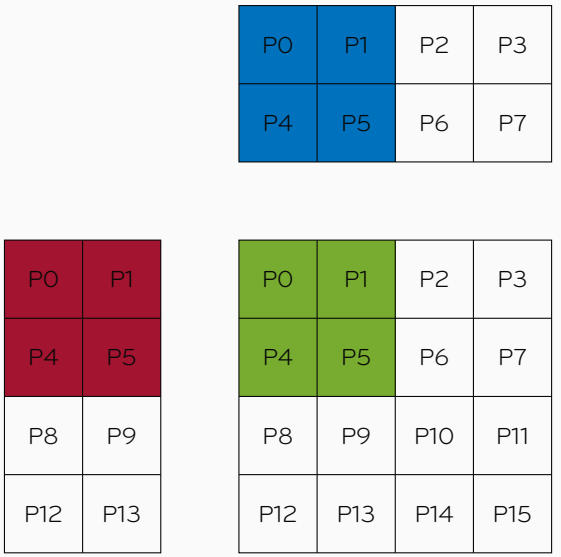
- M0-0 Data mapping
- M0-1 Task mapping **inferred** from DM (RW)
- M0-2 Point-to-point comms. **inferred** from TM



[a] Agullo, Aumage, Faverge, Furmento, Pruvost, Sergent, and Thibault, 2017.

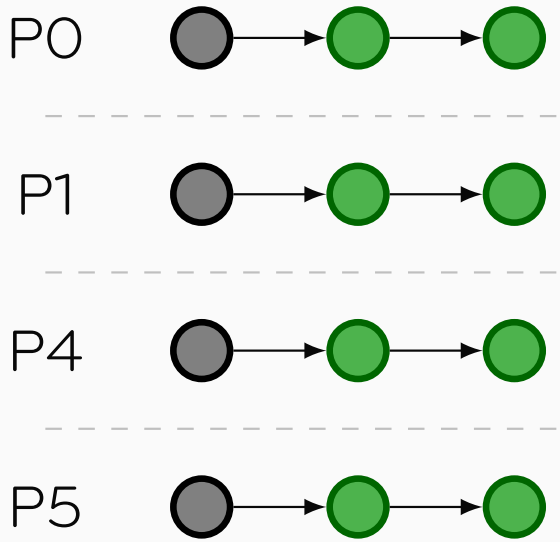
Baseline STF model

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(gemm,
                       A_{i,l}:R,
                       B_{l,j}:R,
                       C_{i,j}:RW)
    end do
  end do
end do
```



Baseline model (M0)[a]

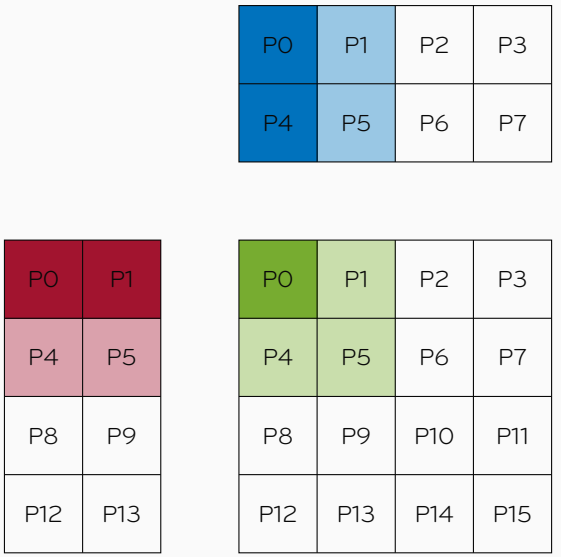
- M0-0 Data mapping
- M0-1 Task mapping **inferred** from DM (RW)
- M0-2 Point-to-point comms. **inferred** from TM



[a] Agullo, Aumage, Faverge, Furmento, Pruvost, Sergent, and Thibault, 2017.

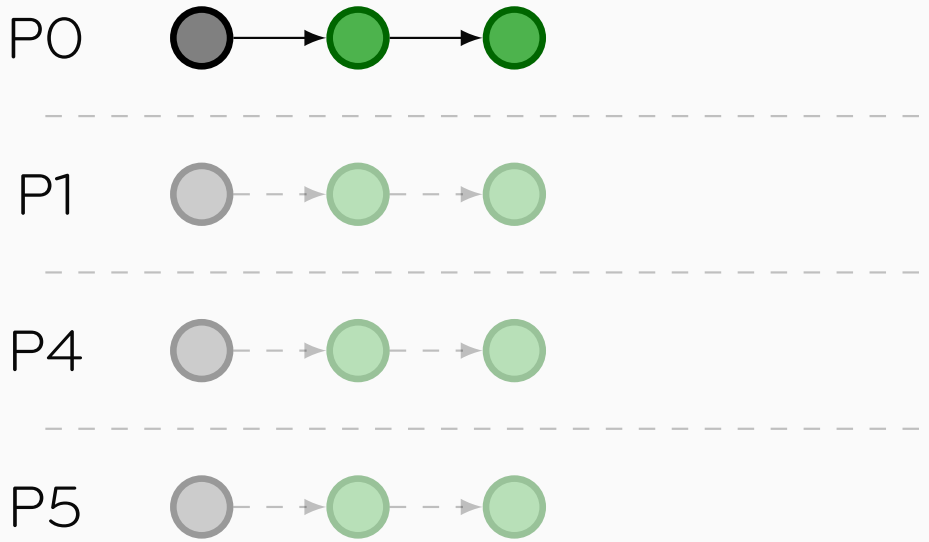
Baseline STF model

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(gemm,
                       A_{i,l}:R,
                       B_{l,j}:R,
                       C_{i,j}:RW)
    end do
  end do
end do
```



Baseline model (M0)[a]

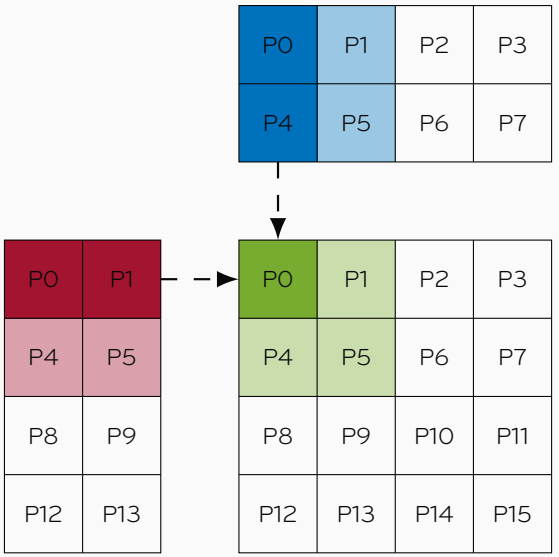
- M0-0 Data mapping
- M0-1 Task mapping **inferred** from DM (RW)
- M0-2 Point-to-point comms. **inferred** from TM



[a] Agullo, Aumage, Faverge, Furmento, Pruvost, Sergent, and Thibault, 2017.

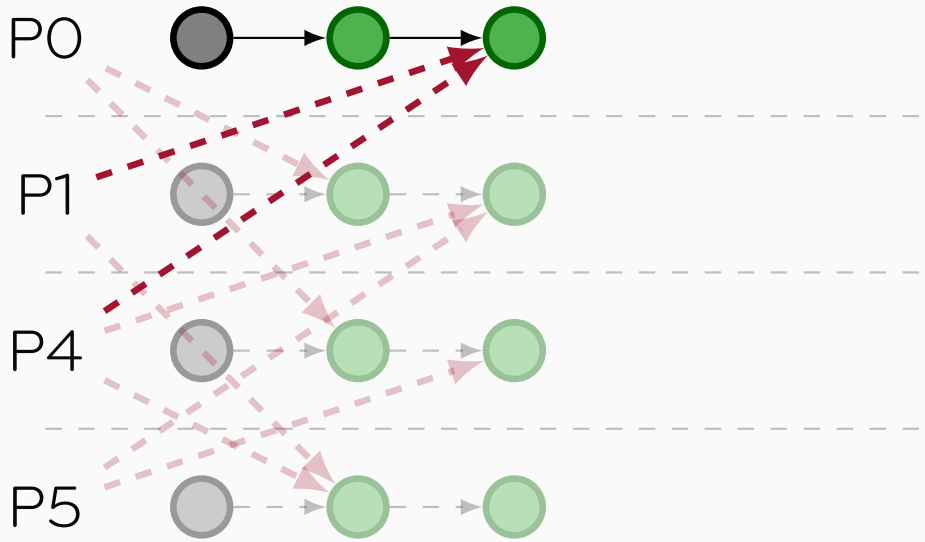
Baseline STF model

```
! data_map: A,B,C distributed on a grid
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(gemm,
                       A_{i,l}:R,
                       B_{l,j}:R,
                       C_{i,j}:RW)
    end do
  end do
end do
```



Baseline model (M0)[a]

- M0-0 Data mapping
- M0-1 Task mapping **inferred** from DM (RW)
- M0-2 Point-to-point comms. **inferred** from TM



[a] Agullo, Aumage, Faverge, Furmento, Pruvost, Sergent, and Thibault, 2017.

Limits to the baseline model

The baseline STF model `M0` does not allow the expression of the features of SUMMA algorithms

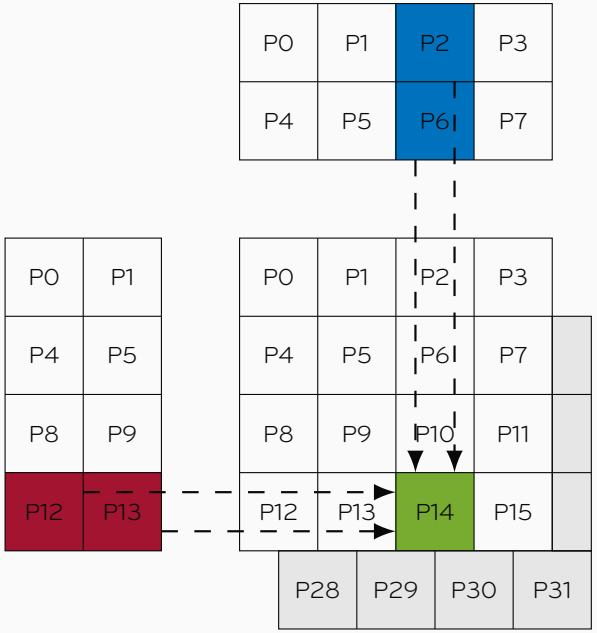
- `M0-1` : Task mapping is over-constrained by data mapping
- `M0-2` : Point-to-point communications patterns are inefficient
- `M0-3` : Reduction patterns are hard to make

We have identified key functionalities required in a STF model `MX` from `M0` to allow the expression of SUMMA algorithms.

Key 1 : Task mapping

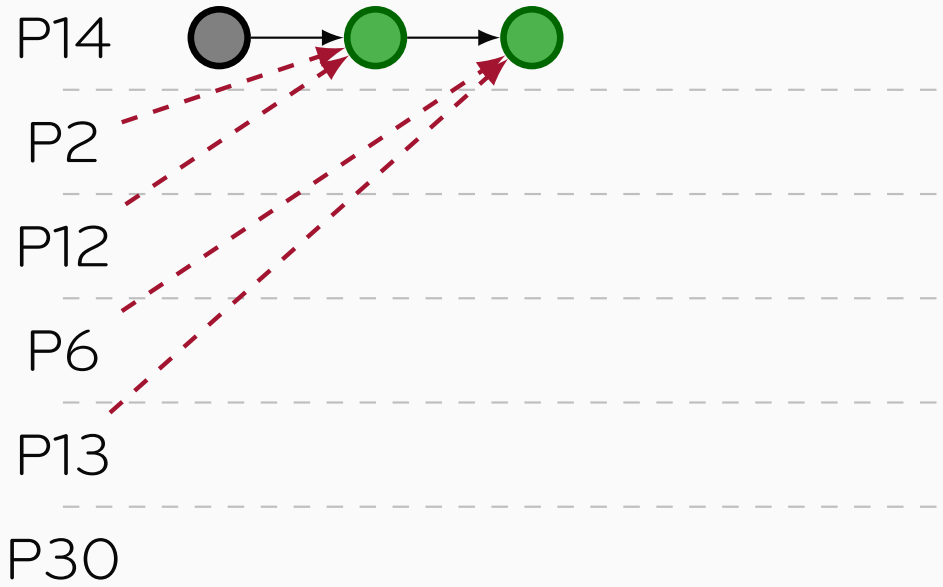
```
!data_map: A,B,C distributed on one layer
!
!

do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(gemm,
                       A_{i,l}:R,
                       B_{l,j}:R,
                       C_{i,j}:RW)
    end do
  end do
end do
```



Advanced model MX

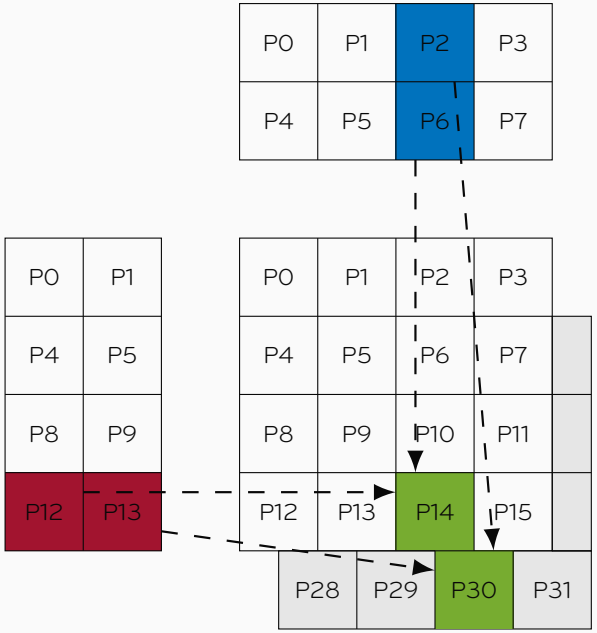
- M0-0 Data mapping
- M0-1 Inferred task mapping
- M0-2 Comms. inferred



Key 1 : Task mapping

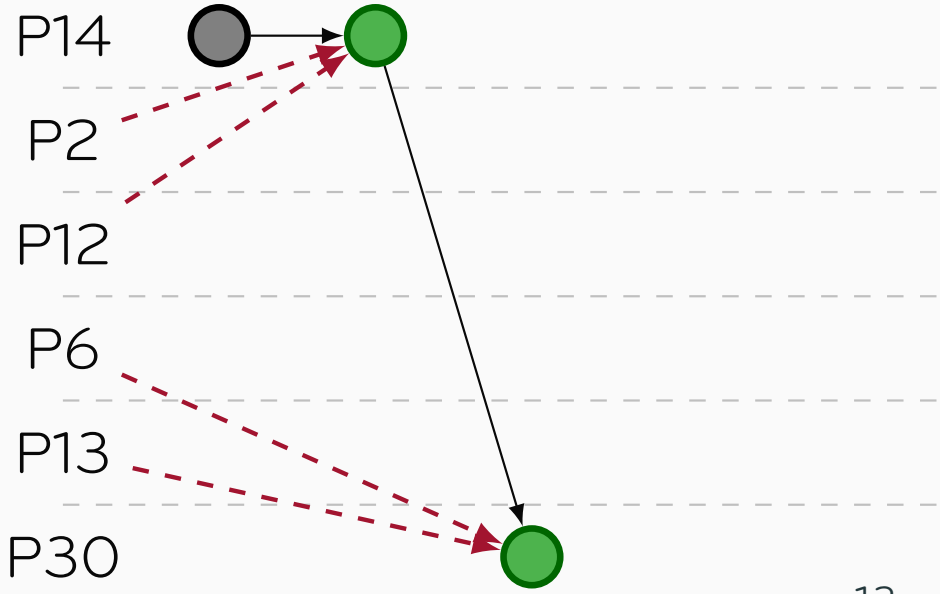
```
!data_map: A,B,C distributed on one layer
!task_map: executing node of A_{i,l}B_{l,j}
!

do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(gemm,
                       A_{i,l}:R,
                       B_{l,j}:R,
                       C_{i,j}:RW,
                       task_map[i,j,l])
    end do
  end do
end do
```



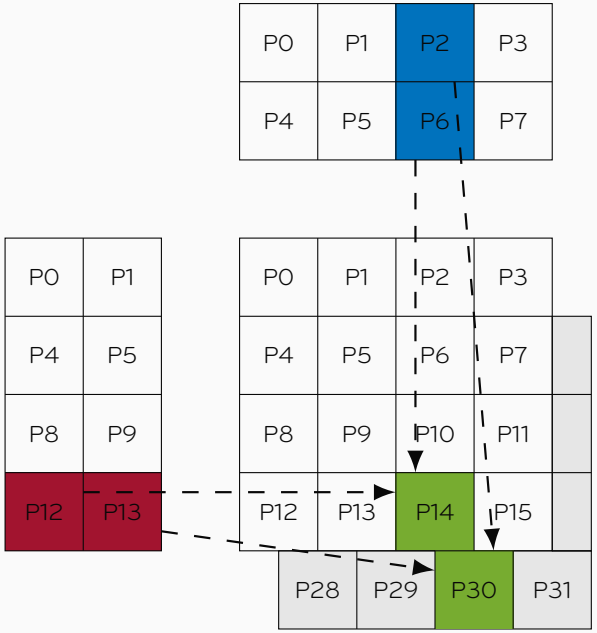
Advanced model MX

- M0-0 Data mapping
- MX-1 **Explicit** task mapping
- M0-2 Comms. inferred



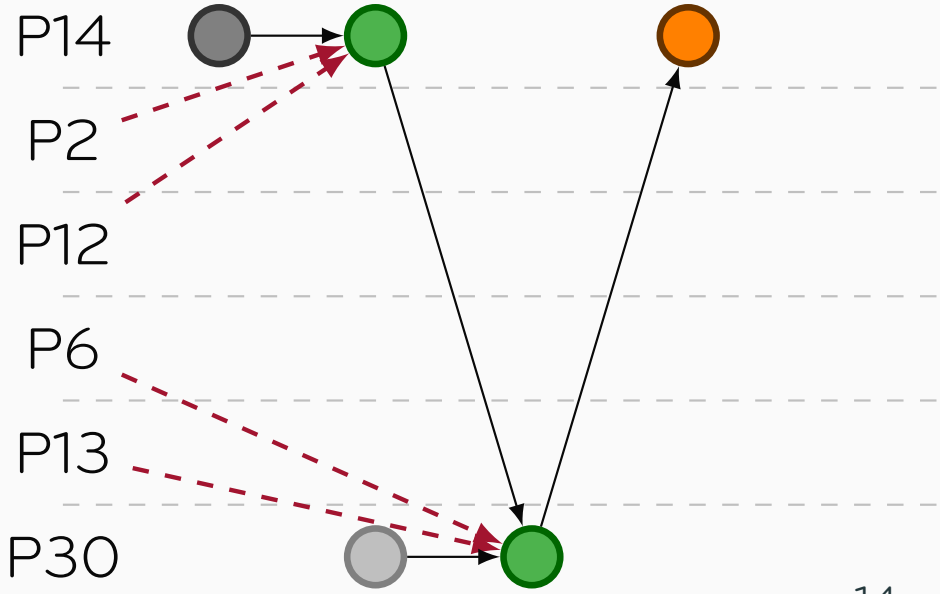
Key 2 : Reduction pattern

```
!data_map: A,B,C distributed on one layer
!task_map: executing node of A_{i,l}B_{l,j}
!
call instantiate_local(C)
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(gemm,
                       A_{i,l}:R,
                       B_{l,j}:R,
                       C_{i,j}:RW,
                       task_map[i,j,l])
    end do
  end do
end do
call reduce_to_initial_layer(C)
```



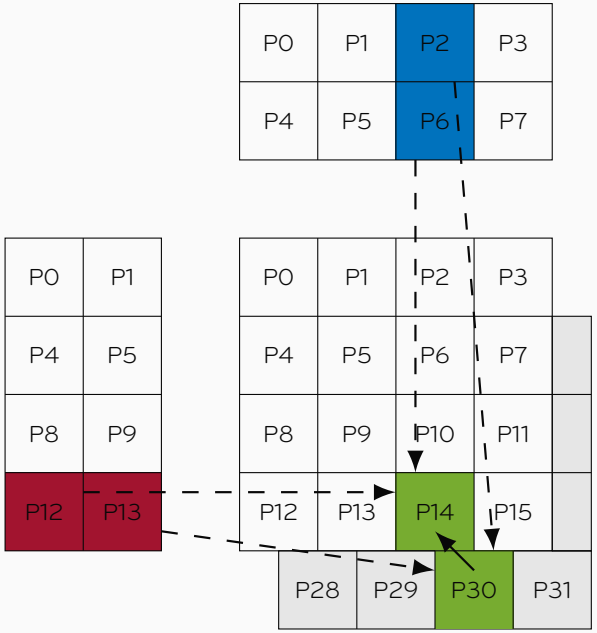
Advanced model MX

- MO-0 Data mapping
- MX-1 Explicit task mapping
- MO-2 Comms. inferred
- MO-3 Explicit reduction pattern



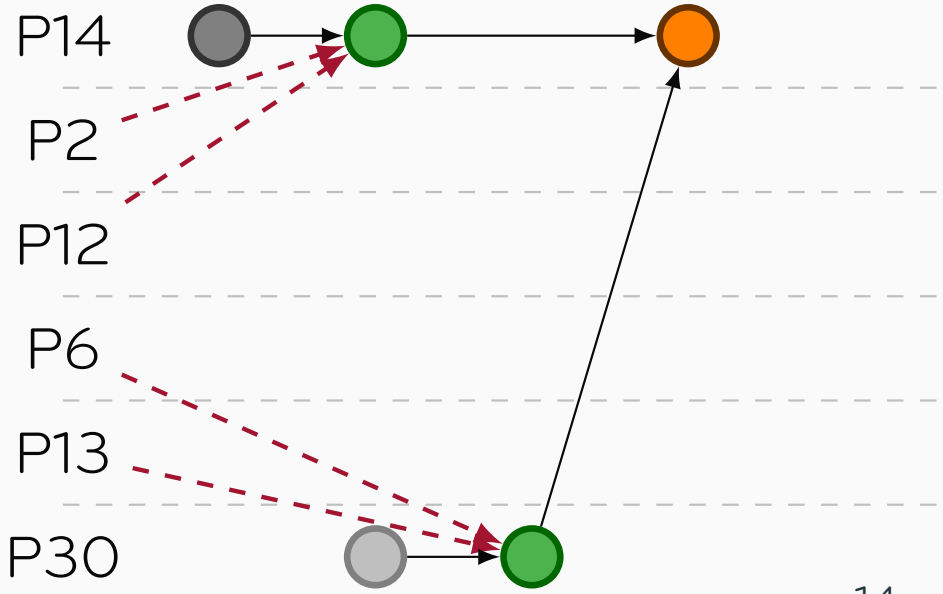
Key 2 : Reduction pattern

```
!data_map: A,B,C distributed on one layer
!task_map: executing node of A_{i,l}B_{l,j}
!
call instantiate_local(C)
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(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
call reduce_to_initial_layer(C)
```



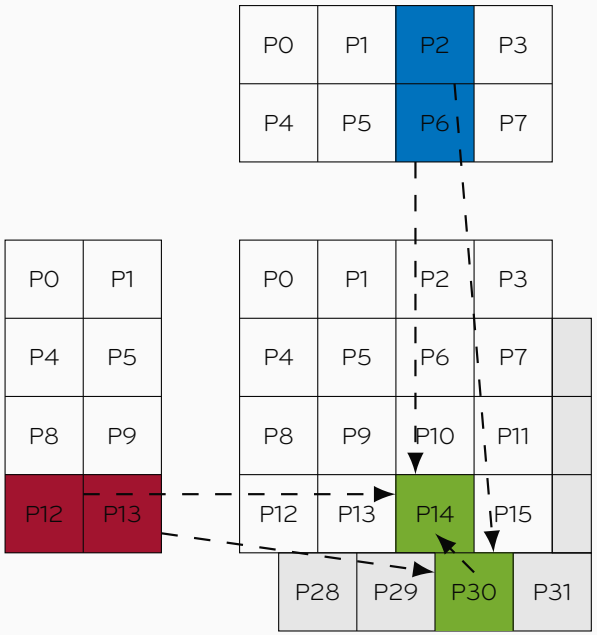
Advanced model MX

- MO-0 Data mapping
- MX-1 Explicit task mapping
- MO-2 Comms. inferred
- MO-3 Explicit reduction pattern



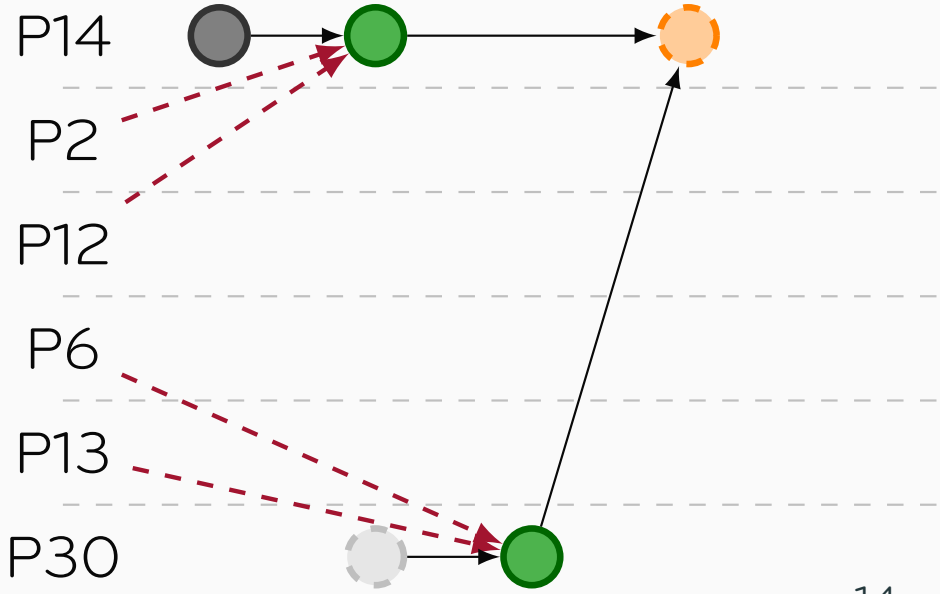
Key 2 : Reduction pattern

```
!data_map: A,B,C distributed on one layer
!task_map: executing node of A_{i,l}B_{l,j}
!
call bind_methods(C, init, sum)
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(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
```



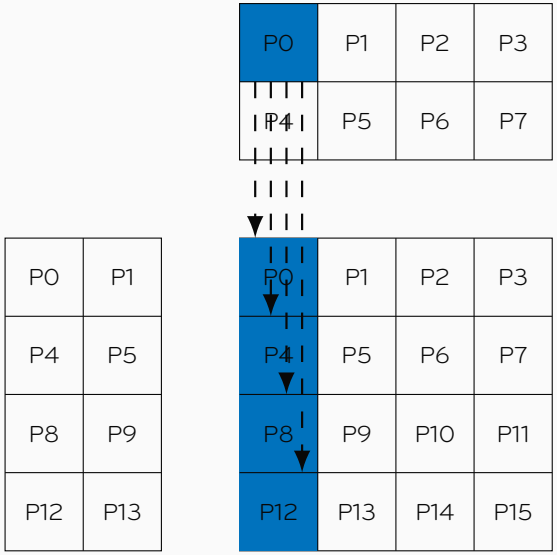
Advanced model MX

- MO-0 Data mapping
- MX-1 Explicit task mapping
- MO-2 Comms. inferred
- MX-3 **Implicit** reduction pattern



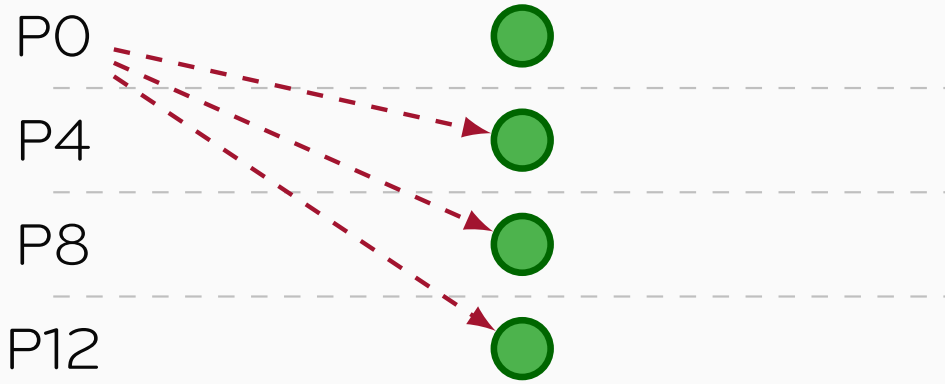
Key 3 : Communications

```
!data_map: A,B,C distributed on one layer
!task_map: executing node of A_{i,l}B_{l,j}
!
call bind_methods(C, init, sum)
do i=1,m
  do j=1,n
    do l=1,k
      call insert_task(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
```



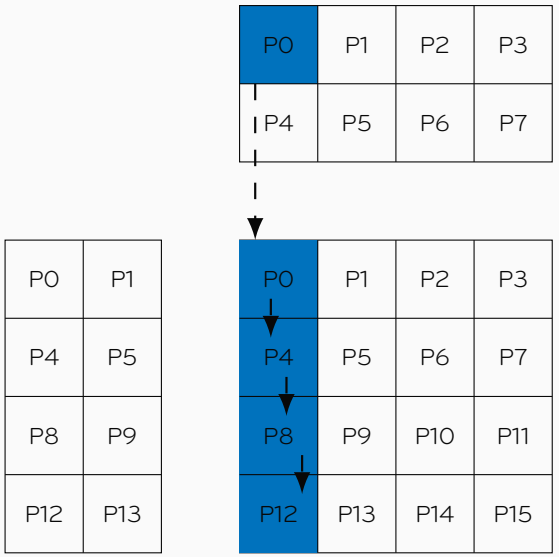
Advanced model MX

- MO-0 Data mapping
- MX-1 Explicit task mapping
- MO-2 Comms. inferred
- MX-3 Implicit reduction pattern



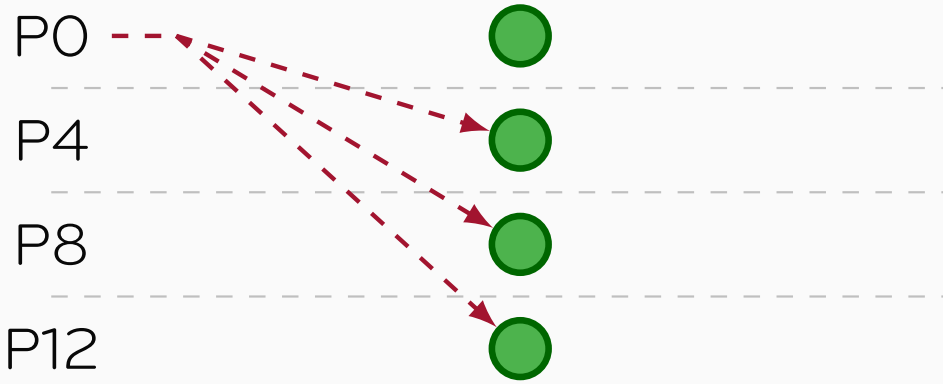
Key 3 : Communications

```
!data_map: A,B,C distributed on one layer
!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 insert_task(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
```



Advanced model MX

- MO-0 Data mapping
- MX-1 Explicit task mapping
- MX-2 **Coll. comms.** inferred[a]
- MX-3 Implicit reduction pattern



[a] Denis, Jeannot, Swartvagher, and Thibault, 2020.

Ideal expression

```
!data_map: A,B,C distributed on one layer
!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 insert_task(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
```

Advanced model MX

- MO-0 Data mapping
- MX-1 **Explicit** task mapping
- MX-2 **Collective** communications **inferred** from task mapping
- MX-3 **Implicit** reduction pattern

Ideal expression - task mappings

- 2D-C-Stat owner of C computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `RW`

- 2D-A-Stat owner of A computes
  - `task_map[i,j,l]` $= l \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `MPI_REDUX`

- 2.5D-C-Stat several nodes computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r) + rc * \frac{l}{k/h}$
  - `access_mode` $=$ `MPI_REDUX`

The advanced model allows to cover all SUMMA variants within three nested loops.

Ideal expression - task mappings

- 2D-C-Stat owner of C computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `RW.and.COMMUTE`

- 2D-A-Stat owner of A computes
  - `task_map[i,j,l]` $= l \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `MPI_REDUX`

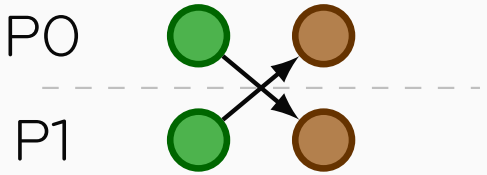
- 2.5D-C-Stat several nodes computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r) + rc * \frac{l}{k/h}$
  - `access_mode` $=$ `MPI_REDUX`

The advanced model allows to cover all SUMMA variants within three nested loops.

Ideal expression - task mappings

- 2D-C-Stat owner of C computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `RW.and.COMMUTE`

- 2D-A-Stat owner of A computes
  - `task_map[i,j,l]` $= l \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `MPI_REDUX`



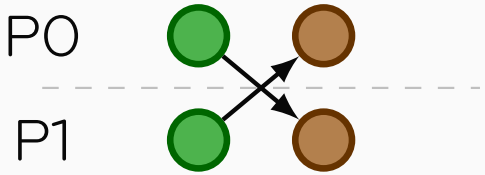
- 2.5D-C-Stat several nodes computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r) + rc * \frac{l}{k/h}$
  - `access_mode` $=$ `MPI_REDUX`

The advanced model allows to cover all SUMMA variants within three nested loops.

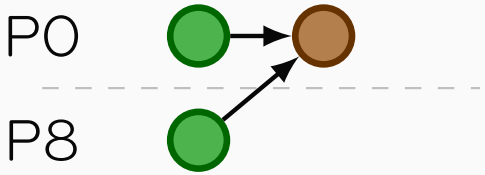
Ideal expression - task mappings

- 2D-C-Stat owner of C computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `RW.and.COMMUTE`

- 2D-A-Stat owner of A computes
  - `task_map[i,j,l]` $= l \,\%\, c + c * (i \,\%\, r)$
  - `access_mode` $=$ `MPI_REDUX`



- 2.5D-C-Stat several nodes computes
  - `task_map[i,j,l]` $= j \,\%\, c + c * (i \,\%\, r) + rc * \frac{l}{k/h}$
  - `access_mode` $=$ `MPI_REDUX`



The advanced model allows to cover all SUMMA variants within three nested loops.

EXPERIMENTAL RESULTS

Implementation details

- StarPU runtime
  - Support STF programming model
  - Developed by Storm Team at INRIA - Bordeaux
  - NewMadeleine communications backend (TaDaaM Team)

`M0-0` Data mapping is provided.

`MX-1` Explicit task mapping is provided.

`MX-2` Dynamic collective communications are available in an experimental branch if compiled with newMadeleine.

`MX-3` Reduction pattern exists in shared memory, **we have adapted it to distributed memory**.

Implementation details

- StarPU runtime
  - Support STF programming model
  - Developed by Storm Team at INRIA - Bordeaux
  - NewMadeleine communications backend (TaDaaM Team)

`M0-0` Data mapping is provided.

`MX-1` Explicit task mapping is provided.

`MX-2` Dynamic collective communications are available in an experimental branch if compiled with newMadeleine.

`MX-3` Reduction pattern exists in shared memory, **we have adapted it to distributed memory**.

- StarPU runtime
  - Support STF programming model
  - Developed by Storm Team at INRIA - Bordeaux
  - NewMadeleine communications backend (TaDaaM Team)

`M0-0` Data mapping is provided.

`MX-1` Explicit task mapping is provided.

`MX-2` Dynamic collective communications are available in an experimental branch if compiled with newMadeleine.

`MX-3` Reduction pattern exists in shared memory, **we have adapted it to distributed memory**.

- StarPU runtime
  - Support STF programming model
  - Developed by Storm Team at INRIA - Bordeaux
  - NewMadeleine communications backend (TaDaaM Team)

`M0-0` Data mapping is provided.

`MX-1` Explicit task mapping is provided.

`MX-2` Dynamic collective communications are available in an experimental branch if compiled with newMadeleine.

`MX-3` Reduction pattern exists in shared memory, **we have adapted it to distributed memory**.

- StarPU runtime
  - Support STF programming model
  - Developed by Storm Team at INRIA - Bordeaux
  - NewMadeleine communications backend (TaDaaM Team)

`M0-0` Data mapping is provided.

`MX-1` Explicit task mapping is provided.

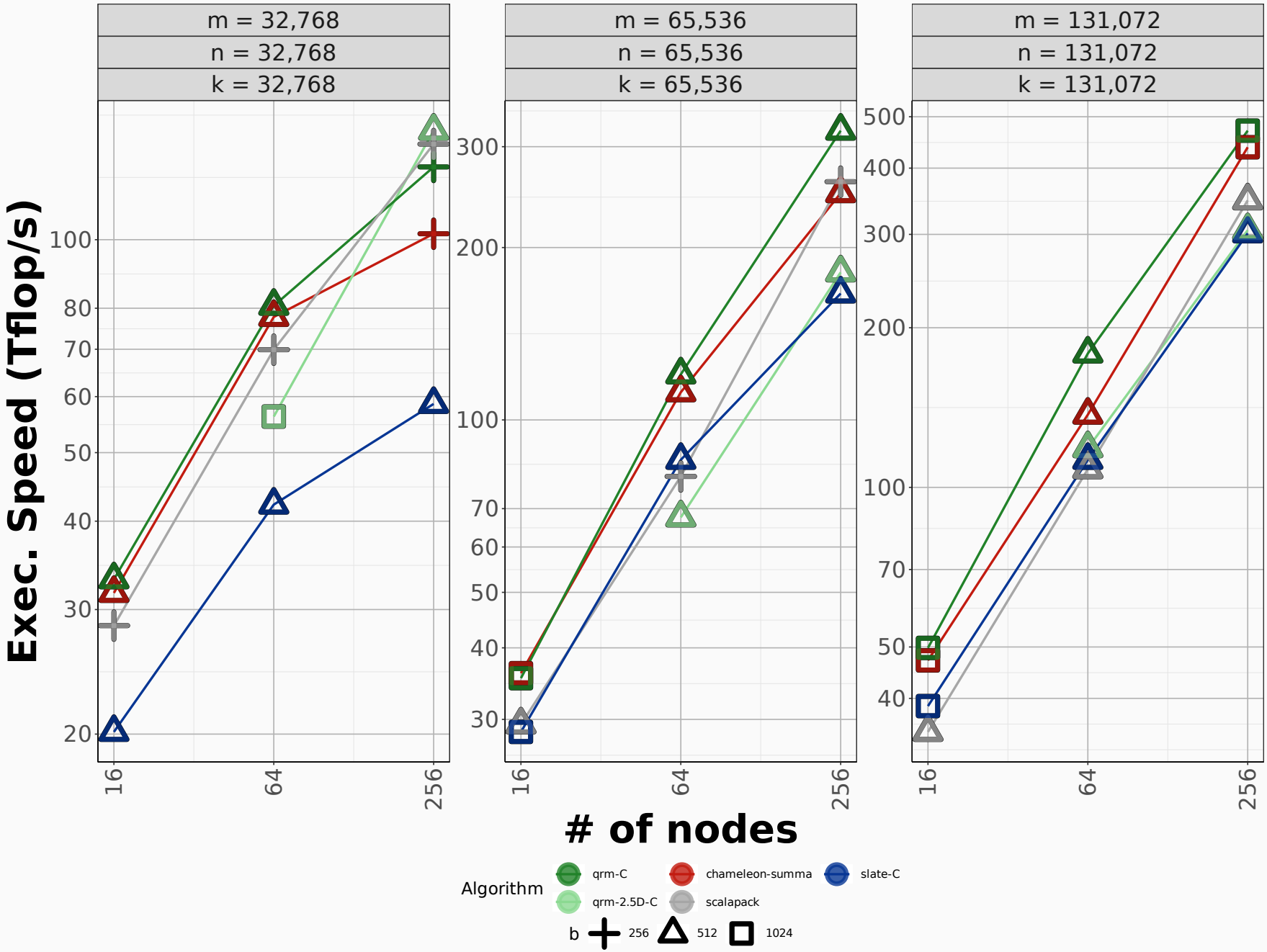
`MX-2` Dynamic collective communications are available in an experimental branch if compiled with newMadeleine.

`MX-3` Reduction pattern exists in shared memory, **we have adapted it to distributed memory**.

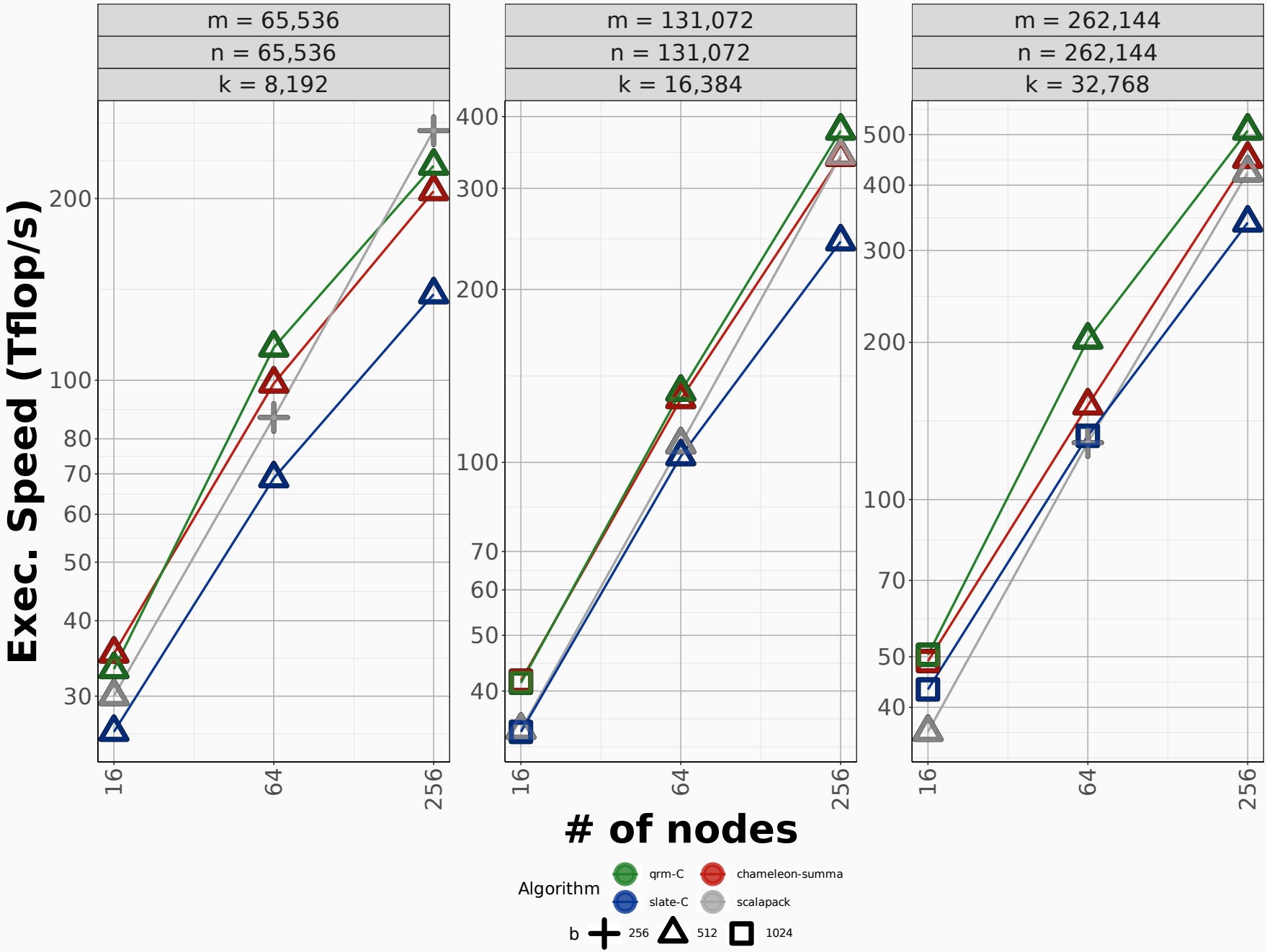
Experimental Setup

- **Hardware**: Très Grand Centre de Calcul (TGCC)
  - **Irène - Skylake** Infiniband EDR
    - Dual-processor Intel Skylake 8168 @ 2.7 GHz
    - 24 cores per processor, 192 GB DDR4 memory
  - **Irène - Rome** Infiniband HDR100
    - Dual-processor AMD Rome (Epyc) @ 2.6 GHz
    - 64 cores per processor, 256 GB DDR4 memory
- **Software**: GNU 9.3.0, MKL 21.3.0
  - `scalapack` – OpenMPI 4.0.5, 2 cores / MPI
    - Synchronous approach
    - 2D A,B,C-stationary hand-made routines
  - `slate` – OpenMPI 4.0.5, 1 MPI / node
    - Dynamic approach
    - 2D A,C-stationary based on OpenMP **`task`**
  - `chameleon` – nMad, 1 MPI / node
    - StarPU with explicit communications management
    - 2D C-stationary pipelined version
  - `qr_mumps` **ours** – nMad, 1 MPI / node
    - Based on StarPU with advanced STF model
    - 2.5D A,B,C-stationary version

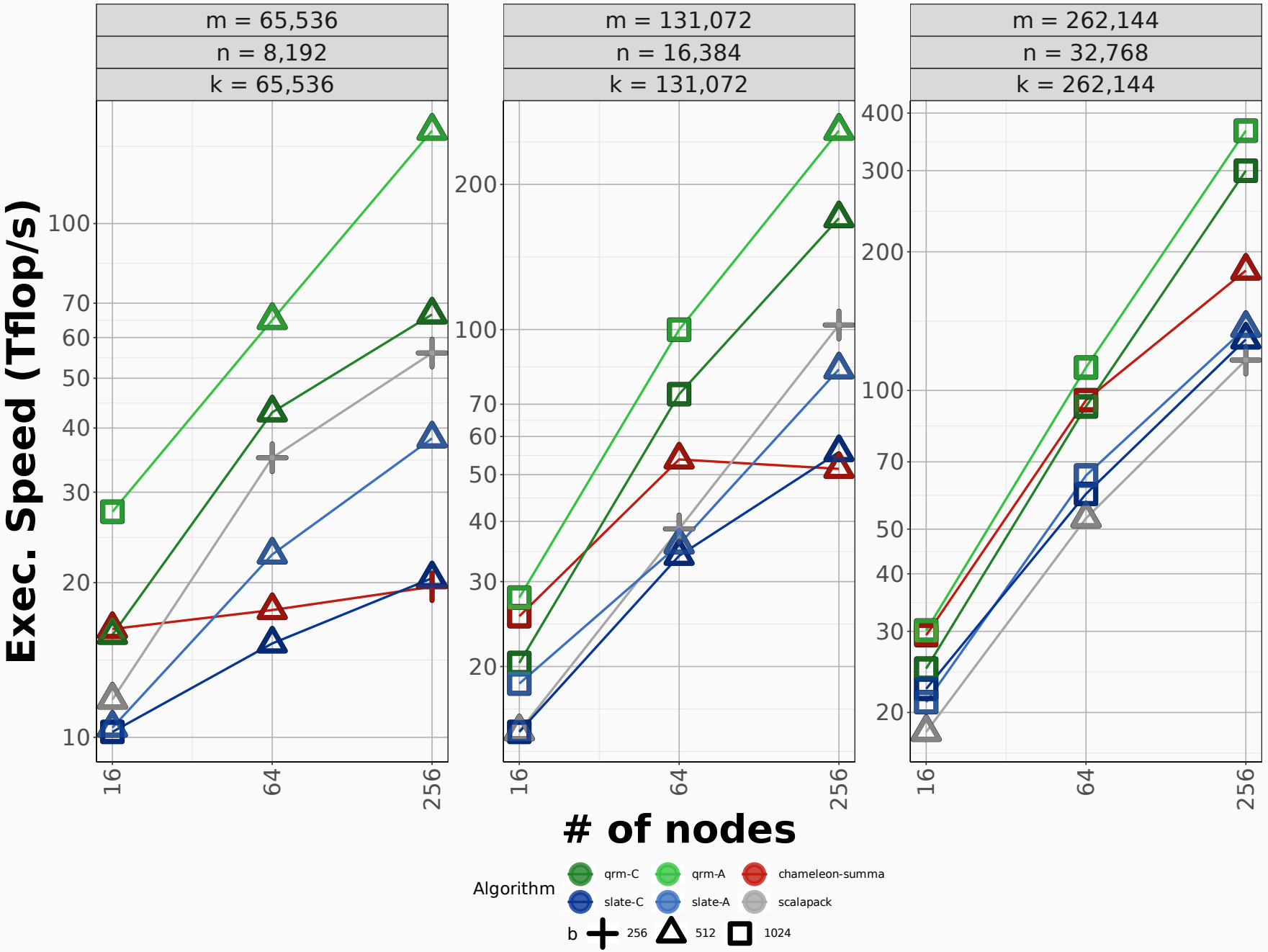
Rome nodes – square matrices, DGEMM



Rome nodes – large C, DGEMM



Skylake nodes – large A, DGEMM



CONCLUSIONS

Contributions

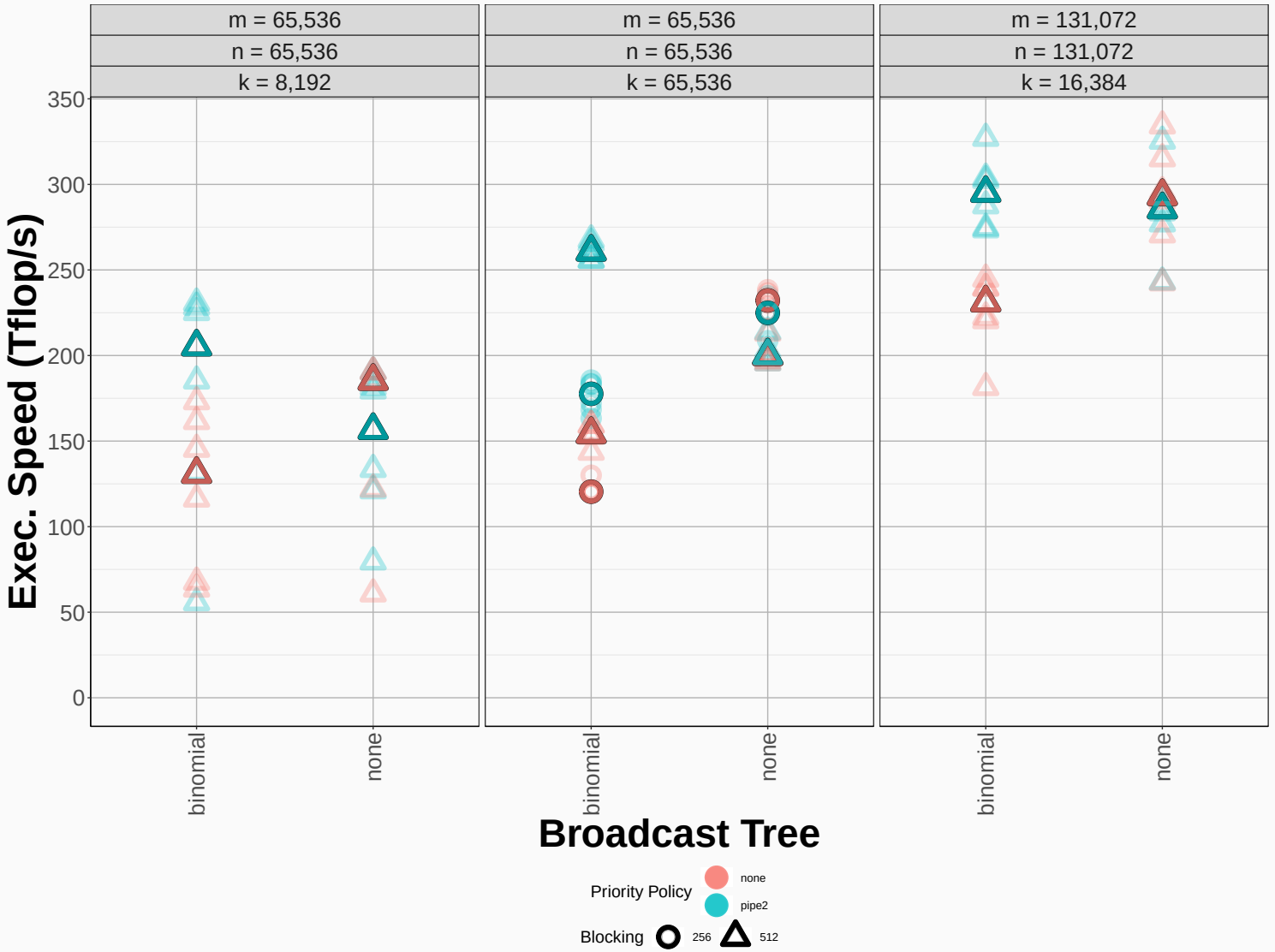
- **Show the STF programming model** allows for the expression of a complex distributed-memory algorithm such as SUMMA
- **Enhance StarPU programming interface** to cover this advanced model
- **Implement** our method and **compare it** with reference libraries, with interesting results

Future works

- Analyze and reliably **control memory consumption**
- **Analyze the impact of priorities** with communication patterns
- Use our approach to implement other numerical routines
  - **Dense** LU, Cholesky **factorization**
  - **Sparse factorization**

BACKUP SLIDES

Dynamic collective communications



Some common operations are done before switching on the **stat** parameter :

- Binding C to the **init** and **sum** codelets
- (submitting) the scaling C by β

```
do jh=1,n/h; do layer=1,h
  j = jh * layer
  do i=1,m
    do l=1,k
Ail => transa ? A_{l,i} : A_{i,l}
Blj => transb ? B_{j,l} : B_{l,j}
Cij => C_{i,j}
work_node = Ail%owner
          + r*c*layer
call insert_task( gemm,
      work_node:ON_NODE,
              Ail:R,Blj:R,
              Cij:MPI_REDUX
     )
    end do
    call mpi_redux_tree(
      C_{i,j})
  end do
end do; end do
```

2.5D-A-Stat

```
do ih=1,m/h; do layer=1,h
  i = ih * layer
  do j=1,n
    do l=1,k
Ail => transa ? A_{l,i} : A_{i,l}
Blj => transb ? B_{j,l} : B_{l,j}
Cij => C_{i,j}
work_node = Blj%owner
          + r*c*layer
call insert_task( gemm,
      work_node:ON_NODE,
              Ail:R,Blj:R,
              Cij:MPI_REDUX
     )
    end do
    call mpi_redux_tree(
      C_{i,j})
  end do
end do; end do
```

2.5D-B-Stat

```
do lh=1,k/h; do layer=1,h
  l = lh * layer
  do i=1,m
    do j=1,n
Ail => transa ? A_{l,i} : A_{i,l}
Blj => transb ? B_{j,l} : B_{l,j}
Cij => C_{i,j}
work_node = Cij%owner
          + r*c*layer
call insert_task( gemm,
      work_node:ON_NODE,
              Ail:R,Blj:R,
              Cij:MPI_REDUX
     )
    end do
  end do
end do; end do
do i=1,m; do j=1,n
  call mpi_redux_tree(C_{i,j})
end do; end do
```

2.5D-C-Stat

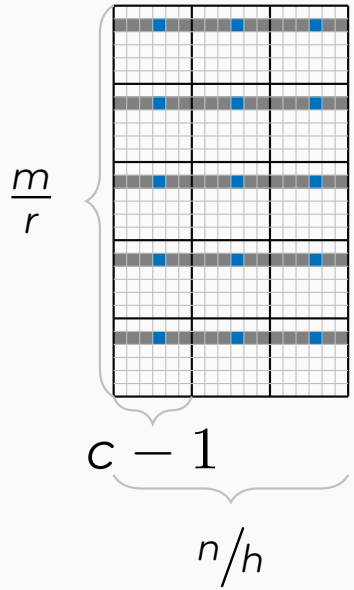
Memory Consumption Concern

Discarding owned blocks, Memory is consumed two ways

- by communication buffers
  - invalidated upon being used in all local contributions
- by reduction copies
  - freed after reduction pattern execution

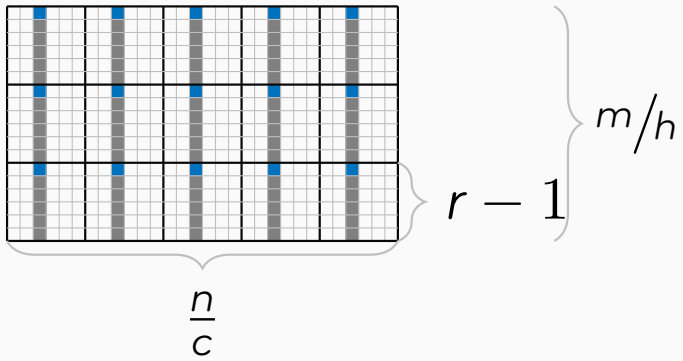
Temporary copies of C are limited

2.5D-A-Stat



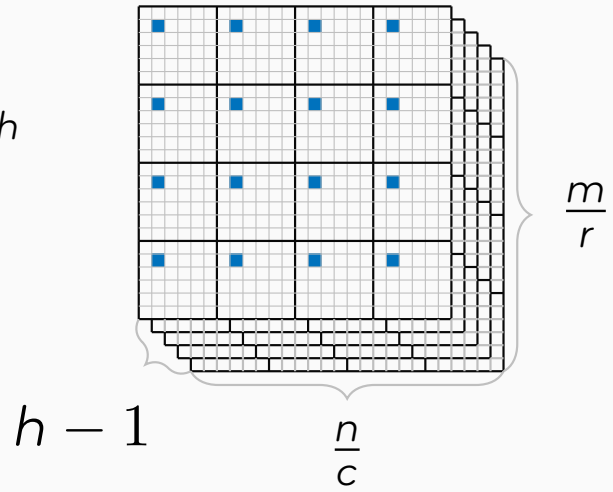
$$\frac{m}{r} n/h (c-1)$$

2.5D-B-Stat



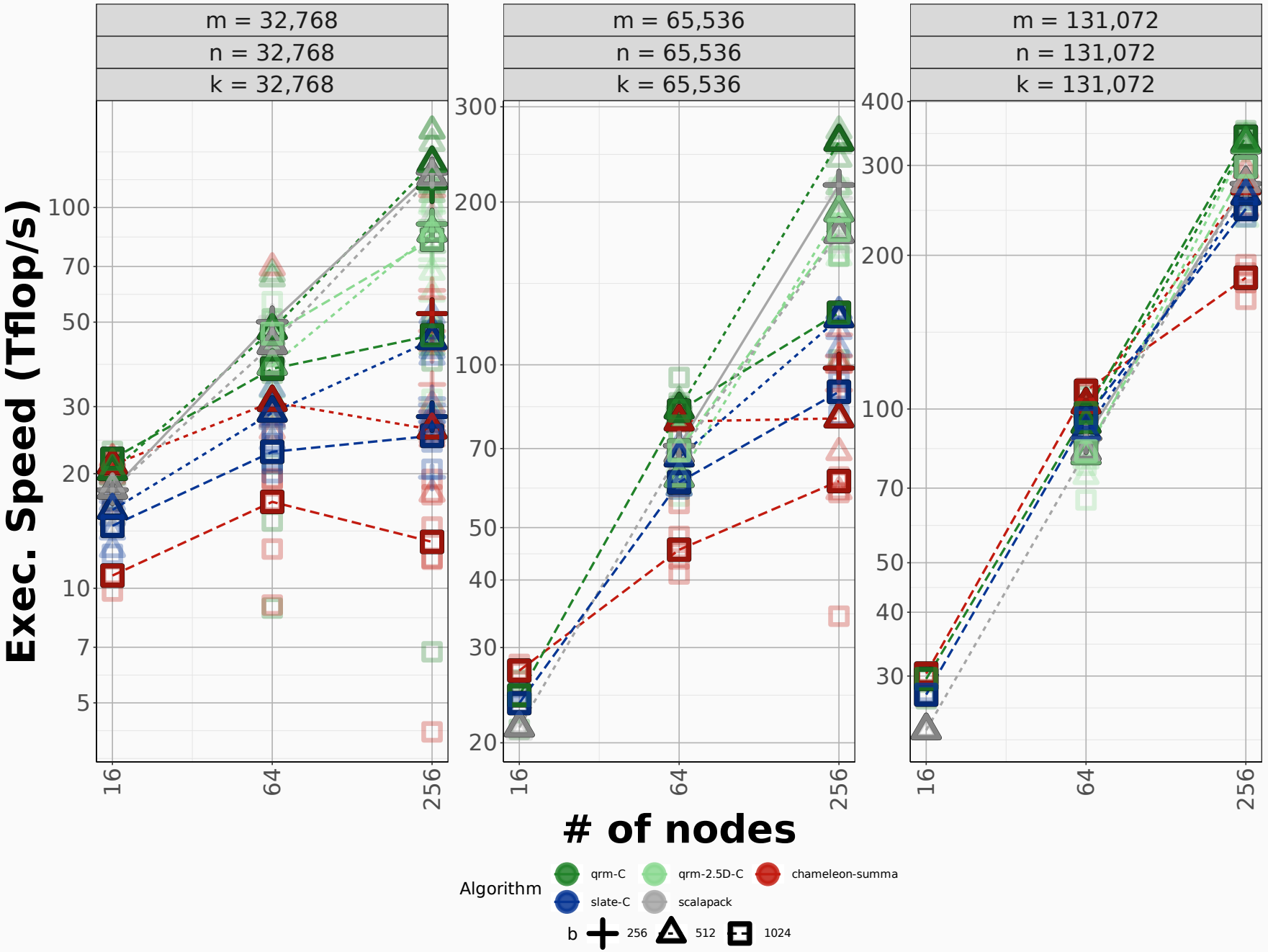
$$m/h \frac{n}{c} (r-1)$$

2.5D-C-Stat

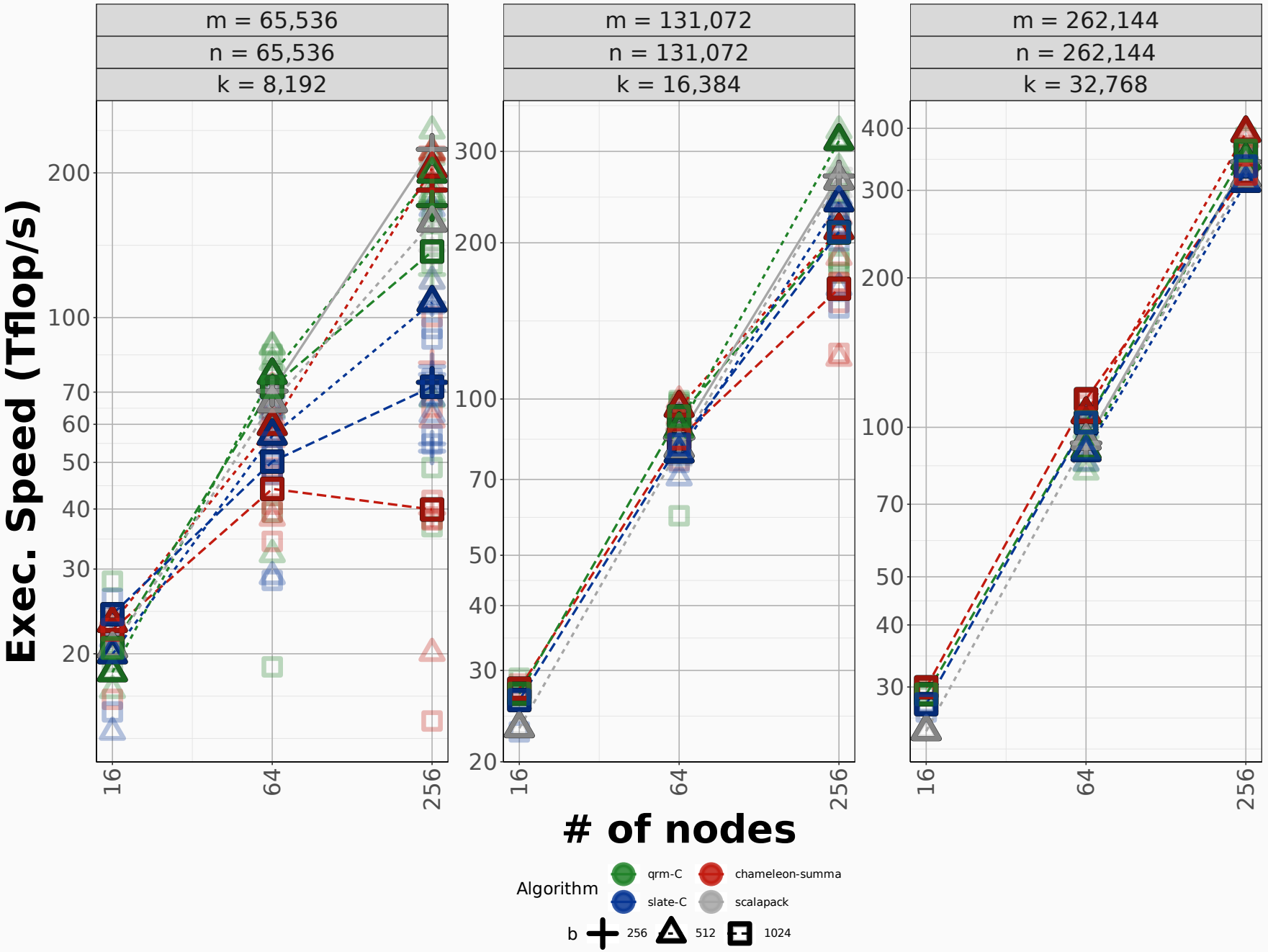


$$\frac{m}{r} \frac{n}{c} (h-1)$$

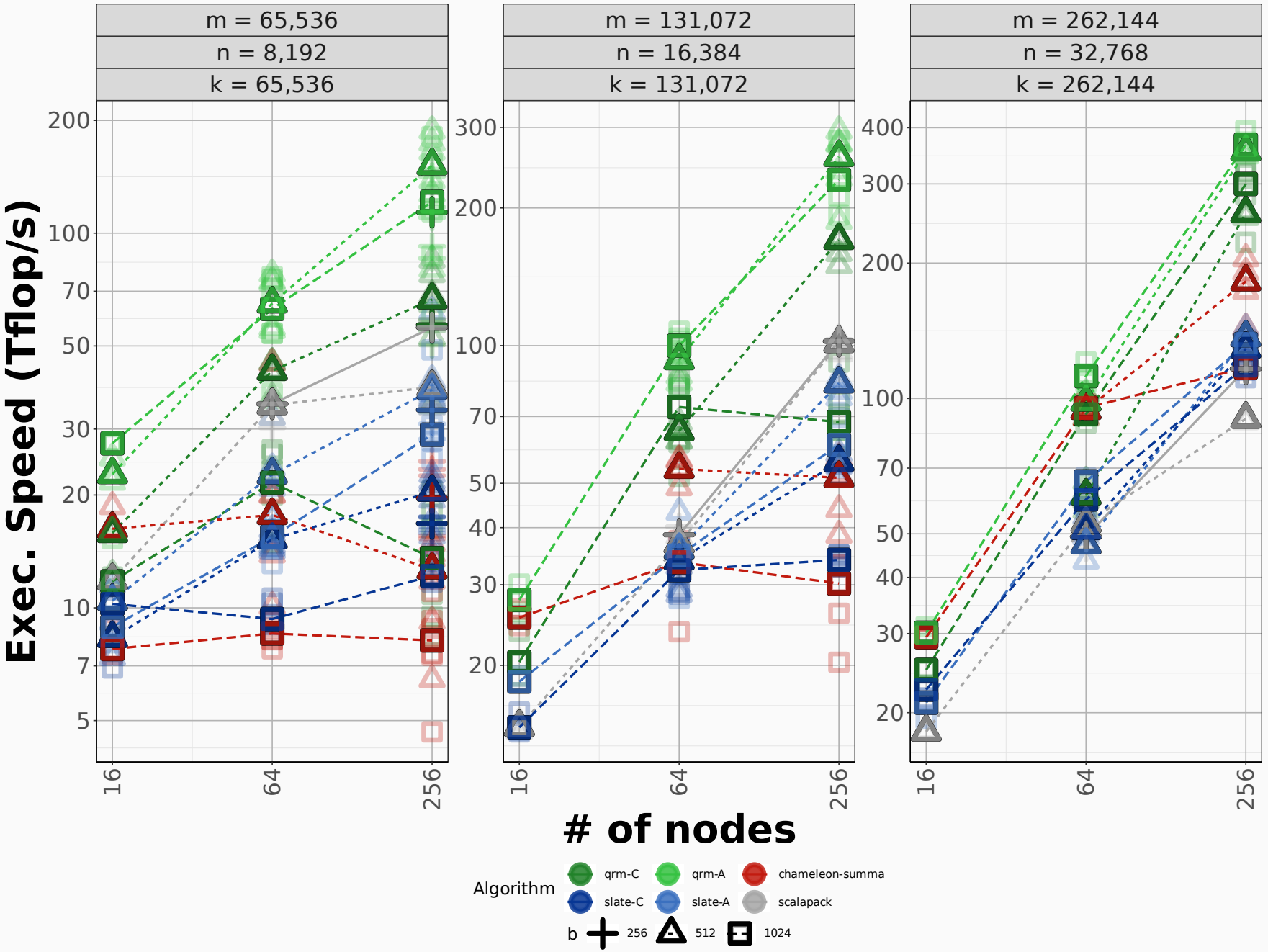
Skylake nodes – square matrices, DGEMM (full)



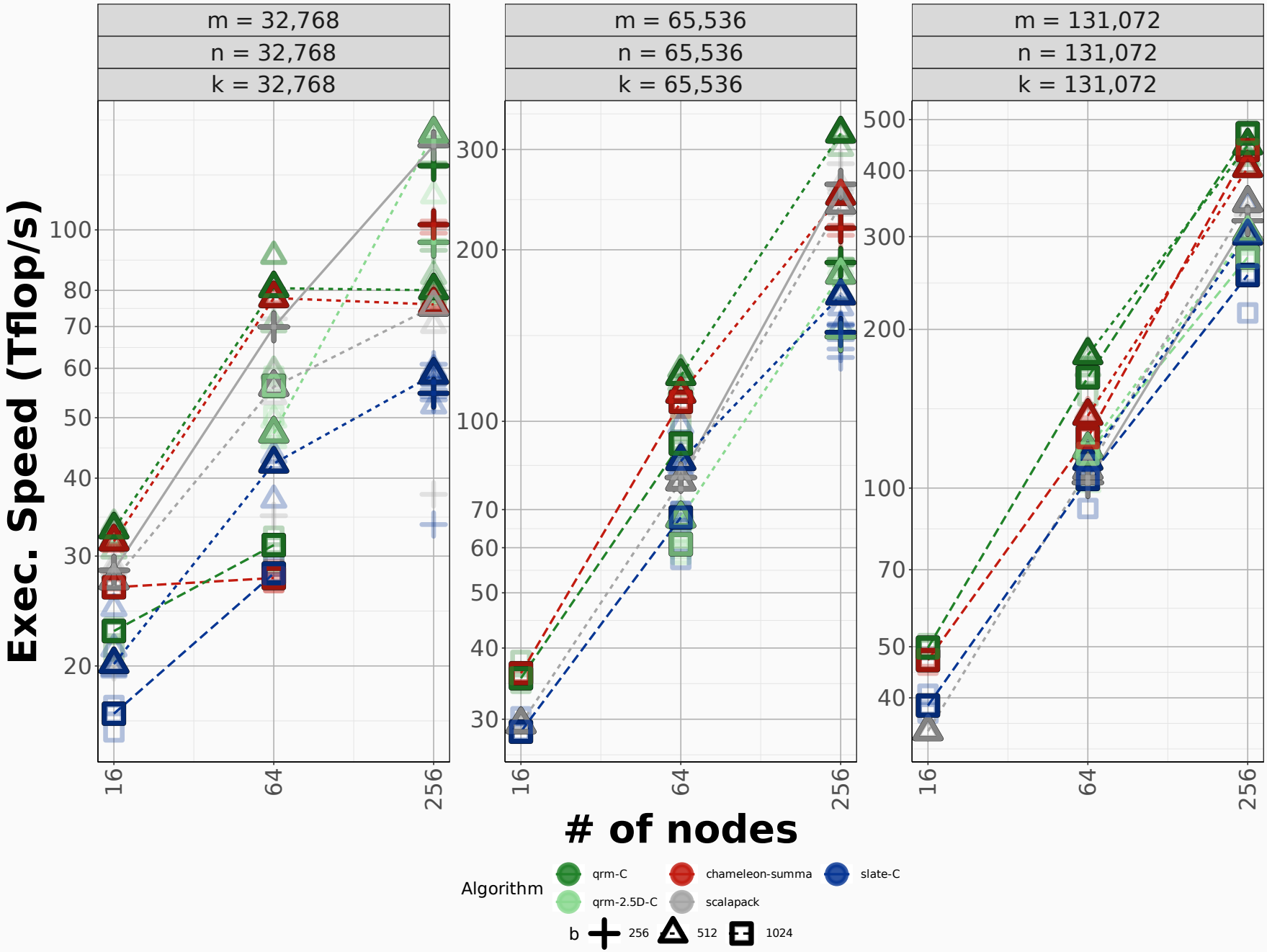
Skylake nodes – large C, DGEMM (full)



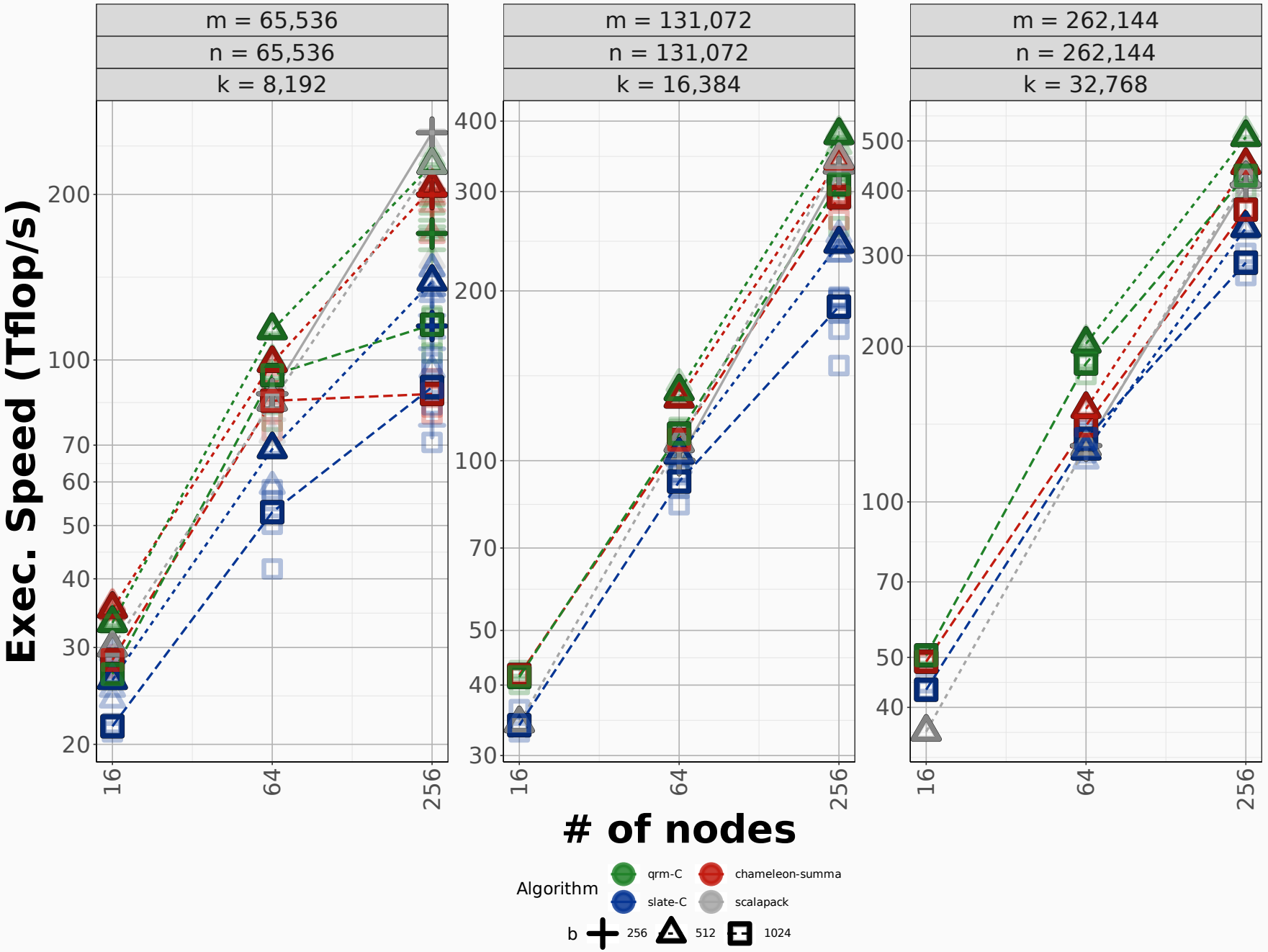
Skylake nodes – large A, DGEMM (full)



Rome nodes – square matrices, DGEMM (full)



Rome nodes – large C, DGEMM (full)



Rome nodes – large A, DGEMM (full)