BUILDING SCALABLE LINEAR ALGEBRA ALGORITHMS WITH STARPU

Numpex – retours d'expérience

Antoine Jégo

12 juin 2024

BACKGROUND

Application context

- PhD thesis through ANR project **Sol**vers for **H**eterogeneous **A**rchitectures over **R**untime systems, **I**nvestigating **S**calability.

- `qr_mumps` software package.
  - multifrontal methods: sparse problems, dense kernels

- Scalable dense linear algebra operations.
  - Matrix-matrix multiplication (`GEMM`, `SYMM`)
  - Matrix factorization (Cholesky `POTRF`, LU `GETRF`)

Scientific literature on scalable GEMM

L. E. Cannon. *"A Celluler Computer to Implement the Kalman Filter Algorithm"*. Phd Thesis, 1969.

→ Optimal algorithm when # of processors is a square number and matrices are square.

R. van de Geijn and J. Watts. *"SUMMA : Scalable Universal Matrix Multiplication Algorithm"*. Concurrency: Practice and Experience (1997).

→ 2D A/B/C-stationary using collective communication patterns.

E. Solomonik and J. Demmel. *"Communication-optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms"*. Europar (2011).

→ 3D Cannon algorithm.

M. D. Schatz, R van de Geijn and J. Poulson. *"Parallel Matrix Multiplication: a Systematic Journey"*. Journal on Scientific Computing (2016).

→ 3D A/B/C-stationary.

Scientific literature on scalable GEMM

L. E. Cannon. *"A Celluler Computer to Implement the Kalman Filter Algorithm"*. Phd Thesis, 1969.

→ Optimal algorithm when # of processors is a square number and matrices are square.

R. van de Geijn and J. Watts. *"SUMMA : Scalable Universal Matrix Multiplication Algorithm"*. Concurrency: Practice and Experience (1997).

→ 2D A/B/C-stationary using collective communication patterns.

E. Solomonik and J. Demmel. *"Communication-optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms"*. Europar (2011).

→ 3D Cannon algorithm.

M. D. Schatz, R van de Geijn and J. Poulson. *"Parallel Matrix Multiplication: a Systematic Journey"*. Journal on Scientific Computing (2016).

→ 3D A/B/C-stationary.

Scientific literature on scalable GEMM

L. E. Cannon. *"A Celluler Computer to Implement the Kalman Filter Algorithm"*. Phd Thesis, 1969.

→ Optimal algorithm when # of processors is a square number and matrices are square.

R. van de Geijn and J. Watts. *"SUMMA : Scalable Universal Matrix Multiplication Algorithm"*. Concurrency: Practice and Experience (1997).

→ 2D A/B/C-stationary using collective communication patterns.

E. Solomonik and J. Demmel. *"Communication-optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms"*. Europar (2011).

→ 3D Cannon algorithm.

M. D. Schatz, R van de Geijn and J. Poulson. *"Parallel Matrix Multiplication: a Systematic Journey"*. Journal on Scientific Computing (2016).

→ 3D A/B/C-stationary.

Scientific literature on scalable GEMM

L. E. Cannon. *"A Celluler Computer to Implement the Kalman Filter Algorithm"*. Phd Thesis, 1969.

→ Optimal algorithm when # of processors is a square number and matrices are square.

R. van de Geijn and J. Watts. *"SUMMA : Scalable Universal Matrix Multiplication Algorithm"*. Concurrency: Practice and Experience (1997).

→ 2D A/B/C-stationary using collective communication patterns.

E. Solomonik and J. Demmel. *"Communication-optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms"*. Europar (2011).

→ 3D Cannon algorithm.

M. D. Schatz, R van de Geijn and J. Poulson. *"Parallel Matrix Multiplication: a Systematic Journey"*. Journal on Scientific Computing (2016).

→ 3D A/B/C-stationary.

Software implementations

Literature recap

X-stationary means *Don't move X which is the biggest matrix.*
3D means *replicate biggest matrix if memory is cheap.*

- `ScaLAPACK`, 1998 : 2D A,B,C-stationary, no GPU
- `Elemental`, 2015 : 2D A,B,C-stationary, no GPU
- `ParSEC`, 2011 : 2D C-stat. w/ GPU, A,B partial GPU support
- `Chameleon`, 2013 : 2D C-stationary, GPU-aware
- `Slate`, 2020 : 2D A,C-stationary, GPU-aware

Modern software packages don't implement all 3D algorithms.

Software implementations

Literature recap

X-stationary means *Don't move X which is the biggest matrix.*
3D means *replicate biggest matrix if memory is cheap.*

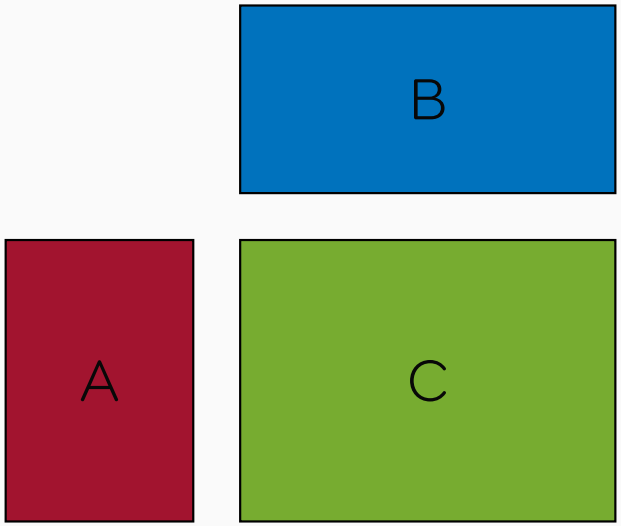
- `ScaLAPACK`, 1998 : 2D A,B,C-stationary, no GPU
- `Elemental`, 2015 : 2D A,B,C-stationary, no GPU
- `ParSEC`, 2011 : 2D C-stat. w/ GPU, A,B partial GPU support
- `Chameleon`, 2013 : 2D C-stationary, GPU-aware
- `Slate`, 2020 : 2D A,C-stationary, GPU-aware

Modern software packages don't implement all 3D algorithms.

Writing matrix-matrix multiplication

C is $m \times n$, A is $m \times k$, B is $k \times n$

```
for (int l = 0; l < k; l++)
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            C[i,j] += A[i,l] * B[l,j];
```

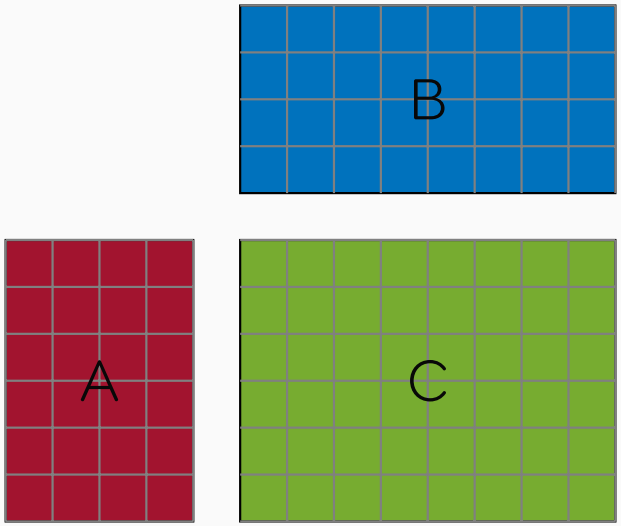


This is the plain sequential implementation.

Writing matrix-matrix multiplication

C is $m \times n$, A is $m \times k$, B is $k \times n$

```
for (int l = 0; l < k/b; l++)
    for (int i = 0; i < m/b; i++)
        for (int j = 0; j < n/b; j++)
            gemm(C[i,j], A[i,l], B[l,j]);
```



Matrices are tiled and BLAS is invoked.

Writing matrix-matrix multiplication

C is $m \times n$, A is $m \times k$, B is $k \times n$

```
register_handles(handles, matrix, M, N) {
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            register_handle(handles[i,j], matrix[i,j]);
};
register_handles(hA,A,m/b,k/b);
register_handles(hB,B,k/b,n/b);
register_handles(hC,C,m/b,n/b);
register_codelet(gemm_cl, cpu:blas_gemm, gpu:cublas_gemm);

for (int l = 0; l < k/b; l++)
    for (int i = 0; i < m/b; i++)
        for (int j = 0; j < n/b; j++)
            insert_task(gemm_cl,
                        hC[i,j]:RW, hA[i,l]:R, hB[l,j]:R);
wait_for_all();
```

Using StarPU requires

- Handing matrices out as *handles*
- Handing (cu)BLAS out as *codelets*
- Inserting tasks with *access modes*

This leads to a GPU-aware code on a single node

Writing matrix-matrix multiplication

C is $m \times n$, A is $m \times k$, B is $k \times n$

```
register_handles(handles, matrix, M, N) {
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            register_handle(handles[i,j], matrix[i,j],
                            owner=2dbc(i,j,p,q));
};
register_handles(hA,A,m/b,k/b);
register_handles(hB,B,k/b,n/b);
register_handles(hC,C,m/b,n/b);
register_codelet(gemm_cl, cpu:blas_gemm, gpu:cublas_gemm);

for (int l = 0; l < k/b; l++)
    for (int i = 0; i < m/b; i++)
        for (int j = 0; j < n/b; j++)
            if (pruning(i,j,l,me))
                insert_task(gemm_cl, hC[i,j]:RW, hA[i,l]:R, hB[l,j]:R);
wait_for_all();
```

- To use StarPU in a distributed-memory setting each handle should have an owner described by its (MPI) rank.
- 2D Block-Cyclic distribution is a standard distribution for linear algebra operations where # of nodes is $p \times q$.

WHY STARPU ?

These specific features fit requirements of linear algebra operations.

- `1.3` Dynamic collective communication detection

A. Denis, E. Jeannot, P. Swartvagher and S. Thibault. *"Using Dynamic Broadcasts to improve Task-Based Runtime Performances"*. Europar (2020).

```
export NMAD_MCAST_TREE=binomial|binary|chain
```

- `1.4` Automatic distributed-memory reduction patterns

These recent features are **transparent** during graph submission.

Scheduling, profiling, out-of-core, etc. are also available.

Writing **scalable** matrix-matrix multiplication

```
register_handles(handles, matrix, M, N) {
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            register_handle(handles[i,j], matrix[i,j],
                            owner=2dbc(i,j,p,q));
};
register_handles(hA,A,m/b,k/b);
register_handles(hB,B,k/b,n/b);
register_handles(hC,C,m/b,n/b);
register_codelet(gemm_cl, cpu:blas_gemm, gpu:cublas_gemm);
register_reduction_codelets(hC, init_cl, add_cl);

for (int l = 0; l < k/b; l++)
    for (int i = 0; i < m/b; i++)
        for (int j = 0; j < n/b; j++)
            if (pruning(i,j,l,me))
                insert_task(gemm_cl, hC[i,j]:RANK_REDUX,
                            hA[i,l]:R, hB[l,j]:R,
                           mapping(i,j,l):EXECUTING_RANK);
wait_for_all();
```

E. Agullo, A. Buttari, A. Guermouche, J. Herrmann and A. Jego. *"Task-based parallel programming for scalable matrix product algorithms"*. ACM TOMS (2023).

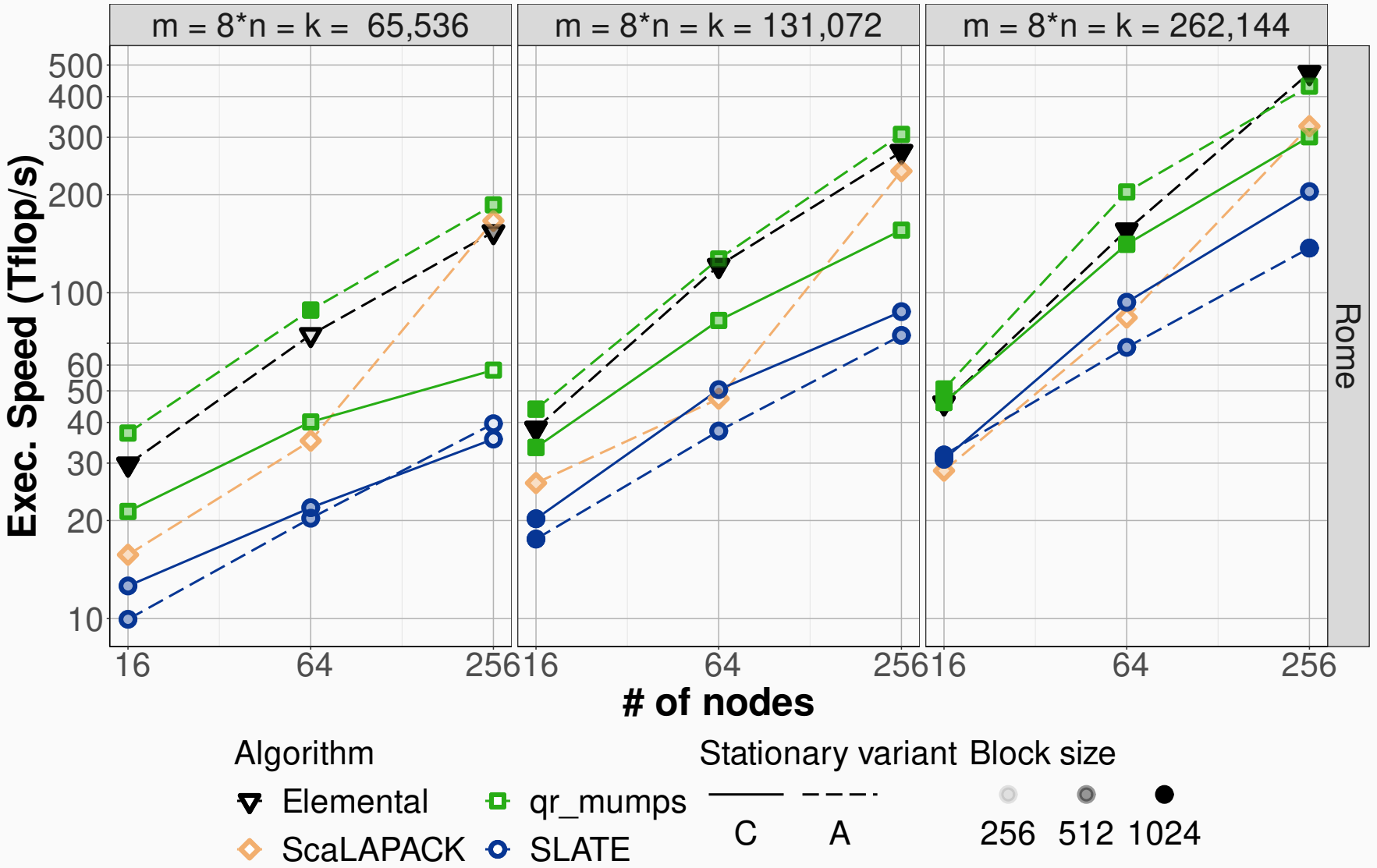
Writing **scalable** matrix-matrix multiplication

```
register_handles(handles, matrix, M, N) {
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            register_handle(handles[i,j], matrix[i,j],
                            owner=2dbc(i,j,p,q));
};
register_handles(hA,A,m/b,k/b);
register_handles(hB,B,k/b,n/b);
register_handles(hC,C,m/b,n/b);
register_codelet(gemm_cl, cpu:blas_gemm, gpu:cublas_gemm);
register_reduction_codelets(hC, init_cl, add_cl);

/* reordering loops helps with parallelism ; A:jil, B:ilj, C:lij*/
for (int l = 0; l < k/b; l++)
    for (int i = 0; i < m/b; i++)
        for (int j = 0; j < n/b; j++)
            if (pruning(i,j,l,me))
                insert_task(gemm_cl, hC[i,j]:RANK_REDUX,
                            hA[i,l]:R, hB[l,j]:R,
                            /* 3D A-stationary */
                            hA[i,l].owner+P/h*j\%h:EXECUTING_RANK);
                            /* 3D B-stationary */
                            hB[l,j].owner+P/h*i\%h:EXECUTING_RANK);
                            /* 3D C-stationary */
                            hC[i,j].owner+P/h*l\%h:EXECUTING_RANK);
wait_for_all();
```

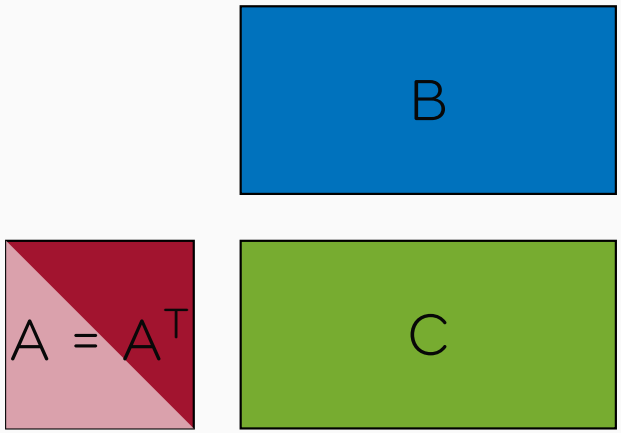
E. Agullo, A. Buttari, A. Guermouche, J. Herrmann and A. Jego. *"Task-based parallel programming for scalable matrix product algorithms"*. ACM TOMS (2023).

Writing matrix-matrix multiplication – some results



Scalable **symmetric** matrix-matrix multiplication

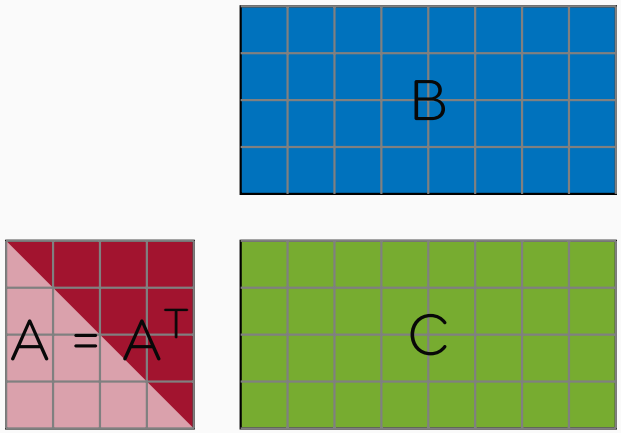
```
for (int j = 0; j < n; j++)
    for (int i = 0; i < m; i++) {
        for (int l = 0; l < i; l++)
            C[i,j] += A[i,l] * B[l,j];
        C[i,j] += A[i,i] * B[i,j];
        for (int l = i+1; l < m; l++)
            C[i,j] += A[l,i] * B[l,j];
    }
```



General matrix-matrix multiplication is not everything ...
The same ideas can be applied to symmetric matrix-matrix multiplication.

Scalable **symmetric** matrix-matrix multiplication

```
for (int j = 0; j < n/b; j++)
    for (int i = 0; i < m/b; i++) {
        for (int l = 0; l < i; l++)
            gemm(C[i,j], A[i,l], B[l,j]);
        symm(C[i,j], A[i,i], B[i,j]);
        for (int l = i+1; l < m/b; l++)
            gemm(C[i,j], A[l,i]^T, B[l,j]);
    }
```



SYMM is part of BLAS – which is part of the reason we may wish to implement it in distributed-memory in the first place.

Scalable **symmetric** matrix-matrix multiplication

```
register_handles(handles, matrix, M, N) {
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            register_handle(handles[i,j], matrix[i,j],
                            owner=2dbc(i,j,p,q));
};
register_symmetric_handles(hA,A,m/b);
register_handles(hB,B,m/b,n/b);
register_handles(hC,C,m/b,n/b);
register_codelet(gemm_cl, cpu:blas_gemm, gpu:cublas_gemm);
register_codelet(symm_cl, cpu:blas_symm, gpu:cublas_symm);

for (int j = 0; j < n/b; j++)
    for (int i = 0; i < m/b; i++)
        for (int l = 0; l < m/b; l++)
            if (pruning(i,j,l,me))
                insert_task(i == l ? symm_cl : gemm_cl,
                            hC[i,j]:RANK_REDUX,
                            i <= l ? hA[i,l] : hA[l,i]^T:R, hB[i,j]:R,
                            mapping(i,j,i):EXECUTING_RANK);
wait_for_all();
```

StarPU helps with porting GEMM as well as SYMM.
But we can actually do even better this time!

Scalable **symmetric** matrix-matrix multiplication

```
register_handles(handles, matrix, M, N,
                 distribution = 2dbc(.,.,p,q)) {
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            register_handle(handles[i,j], matrix[i,j],
                            owner=distribution(i,j));
};
register_symmetric_handles(hA,A,m/b,
     symmetric_block_cyclic(.,.,r));
register_handles(hB,B,m/b,n/b);
register_handles(hC,C,m/b,n/b);
register_codelet(gemm_cl, cpu:blas_gemm, gpu:cublas_gemm);
register_codelet(symm_cl, cpu:blas_symm, gpu:cublas_symm);

for (int j = 0; j < n/b; j++)
    for (int i = 0; i < m/b; i++)
        for (int l = 0; l < m/b; l++)
            if (pruning(i,j,l,me))
                insert_task(i == l ? symm_cl : gemm_cl,
                            hC[i,j]:RANK_REDUX,
                            i <= l ? hA[i,l] : hA[l,i]^T:R, hB[i,j]:R,
                            mapping(i,j,i):EXECUTING_RANK);
wait_for_all();
```

Distribution doesn't need to follow the standard 2DBC.

The submission loop is **distribution-agnostic**.

Scalable **symmetric** matrix-matrix multiplication – Theory

When operations are symmetric, symmetric distributions should be invoked to minimize communication volume.

O. Beaumont, P. Duchon, L. Eyraud-Dubois, J. Langou, M. Vérité. *"Symmetric Block-Cyclic Distribution : Fewer Communications Leads to Faster Dense Cholesky Factorization"*. SC 2022.

This is **untractable** in software such as ScaLAPACK because input distribution is tightly tied to compute distribution.
This is **easy** when building software with a programming model like the one interfaced in StarPU.

- Symmetric Block-Cyclic distribution can be applied to SYMM.
- Triangular Block-Cyclic yields lowest communication volume.

E. Agullo, A. Buttari, O. Coulaud, L. Eyraud-Dubois, M. Faverge, A. Franc, A. Guermouche, A. Jego, R. Peressoni and F. Pruvost. *"On the arithmetic intensity of distributed-memory dense matrix multiplication involving a symmetric input matrix (symm)."*. IPDPS 2023.

Scalable **symmetric** matrix-matrix multiplication – Theory

When operations are symmetric, symmetric distributions should be invoked to minimize communication volume.

O. Beaumont, P. Duchon, L. Eyraud-Dubois, J. Langou, M. Vérité. *"Symmetric Block-Cyclic Distribution : Fewer Communications Leads to Faster Dense Cholesky Factorization"*. SC 2022.

This is **untractable** in software such as ScaLAPACK because input distribution is tightly tied to compute distribution.
This is **easy** when building software with a programming model like the one interfaced in StarPU.

- Symmetric Block-Cyclic distribution can be applied to SYMM.
- Triangular Block-Cyclic yields lowest communication volume.

E. Agullo, A. Buttari, O. Coulaud, L. Eyraud-Dubois, M. Faverge, A. Franc, A. Guermouche, A. Jego, R. Peressoni and F. Pruvost. *"On the arithmetic intensity of distributed-memory dense matrix multiplication involving a symmetric input matrix (symm)."*. IPDPS 2023.

Scalable **symmetric** matrix-matrix multiplication – Theory

When operations are symmetric, symmetric distributions should be invoked to minimize communication volume.

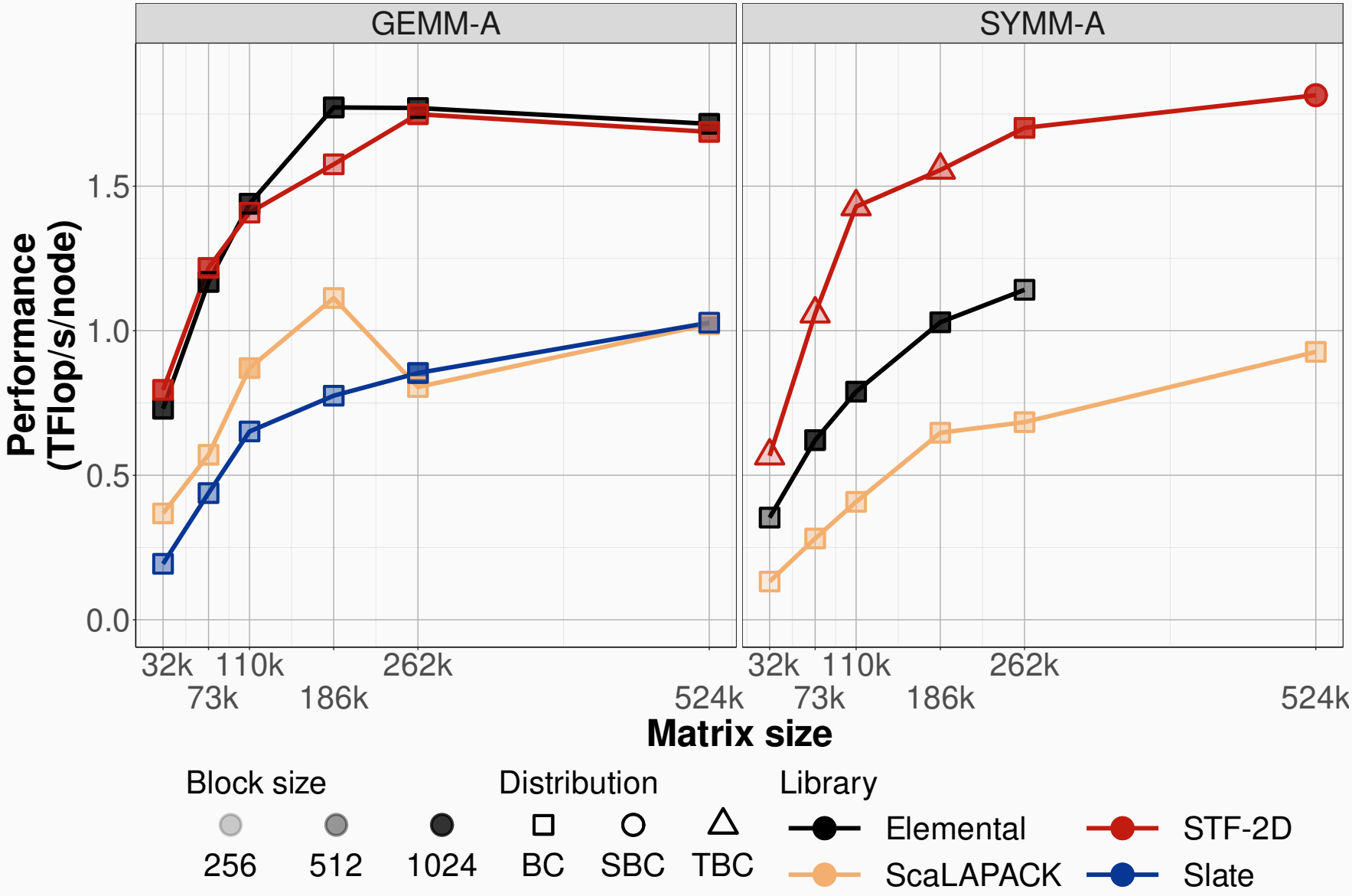
O. Beaumont, P. Duchon, L. Eyraud-Dubois, J. Langou, M. Vérité. *"Symmetric Block-Cyclic Distribution : Fewer Communications Leads to Faster Dense Cholesky Factorization"*. SC 2022.

This is **untractable** in software such as ScaLAPACK because input distribution is tightly tied to compute distribution.
This is **easy** when building software with a programming model like the one interfaced in StarPU.

- Symmetric Block-Cyclic distribution can be applied to SYMM.
- Triangular Block-Cyclic yields lowest communication volume.

E. Agullo, A. Buttari, O. Coulaud, L. Eyraud-Dubois, M. Faverge, A. Franc, A. Guermouche, A. Jego, R. Peressoni and F. Pruvost. *"On the arithmetic intensity of distributed-memory dense matrix multiplication involving a symmetric input matrix (symm)."*. IPDPS 2023.

Scalable **symmetric** matrix-matrix multiplication – Result



Some other programming models

- MPI+OpenMP+CUDA
  - Algorithms risk being rigid and hardly portable.
  - Developping new algorithms may require much effort.
  - As low-level as can practically be.
- SYCL, Kokkos
  - Lacks support for distributed-memory.
  - Celerity (SYCL), Remote Spaces (Kokkos) may help with the distributed side of things.
- ParSEC
  - Job Data Flow DSL may be deterring from a software development point-of-view.
  - It handles distributed-memory efficiently.

Pitfalls of StarPU

- Pruning of the DAG is not straightforward
  - Ensuring any rank inserts the minimal portion of the DAG may become difficult
  - May be alleviated with some analysis when describing matrices, may be helped by interfacing runtime and compile tools, ...
- Handles handling requires engineering
  - Ensuring ranks create the minimal amount of handles can get messy.
  - You can always wrap task insertion to include handle submission.
- Task priority meaning is sometimes unclear
  - Collective communication trees are built through the order of submission and the task priorities.
  - Simple heuristics on priority yield positive results on scalability. This is being studied further at Inria Bordeaux.

Pitfalls of StarPU – wishlist/perspectives

- Pruning of the DAG is not straightforward
  - Ensuring any rank inserts the minimal portion of the DAG may become difficult
  - May be alleviated with some analysis when describing matrices, may be helped by interfacing runtime and compile tools, ...
- Handles handling requires engineering
  - Ensuring ranks create the minimal amount of handles can get messy.
  - You can always wrap task insertion to include handle submission.
- Task priority meaning is sometimes unclear
  - Collective communication trees are built through the order of submission and the task priorities.
  - Simple heuristics on priority yield positive results on scalability. This is being studied further at Inria Bordeaux.

CONCLUSIONS

- Programming models taking both distributed memory and heterogeneity in consideration make writing algorithms simpler.
  - Make adaptating legacy algorithms a breeze.
  - Make further pushing the state-of-the-art possible.
- StarPU fits this description
  - ParSEC, *etc.* may fit the description *i.e.* YMMV
  - Distributed-memory programming models seem to be considered by Kokkos and the likes.

Questions ?

- Programming models taking both distributed memory and heterogeneity in consideration make writing algorithms simpler.
  - Make adaptating legacy algorithms a breeze.
  - Make further pushing the state-of-the-art possible.
- StarPU fits this description
  - ParSEC, *etc.* may fit the description *i.e.* YMMV
  - Distributed-memory programming models seem to be considered by Kokkos and the likes.

Questions ?