Distributed sparse matrix factorization on top of tasks based runtime systems

SOLHAR 2014, Bordeaux

X. Lacoste

X. Lacoste, M. Faverge,
P. Ramet, S. Thibault
LabRI – Inria Bordeaux Sud-Ouest
G. Bosilca
ICL – University of Tennessee

Guideline

From PaStiX dedicated scheduler to generic runtimes

Shared memory results
- Matrices and Machines
- Multicore results
- Heterogeneous results
- Memory study

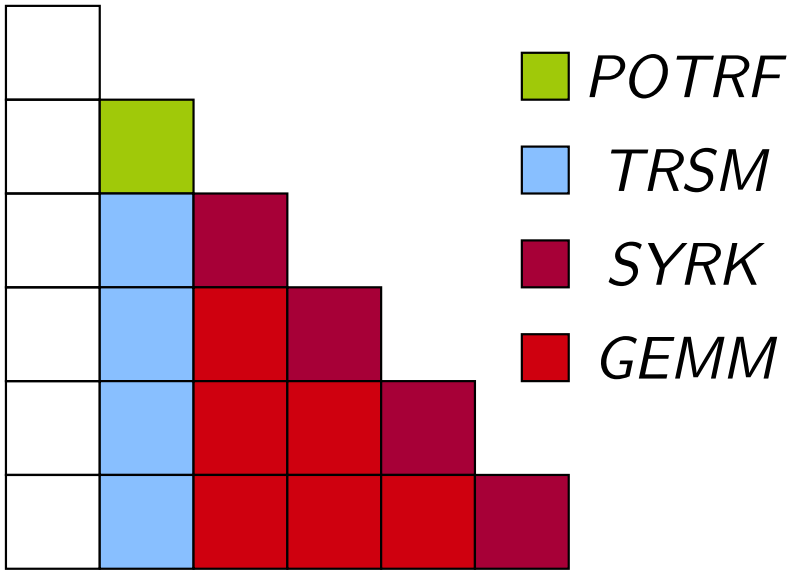
DAG runtimes and distributed architectures
- Distributed factorization algorithms

Distributed results

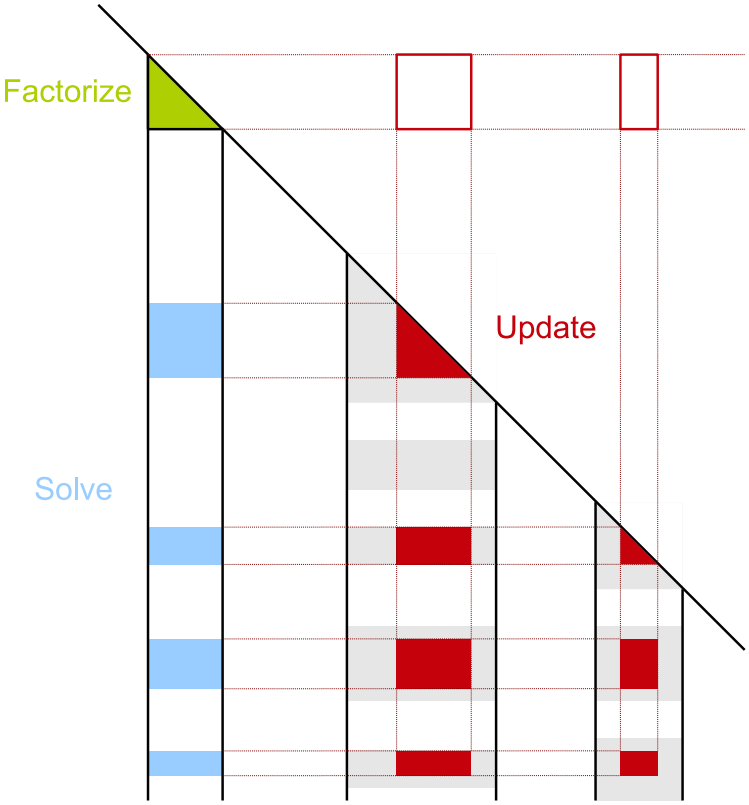
Conclusion and future works

1
From PaStiX dedicated scheduler to generic runtimes

Tasks structure

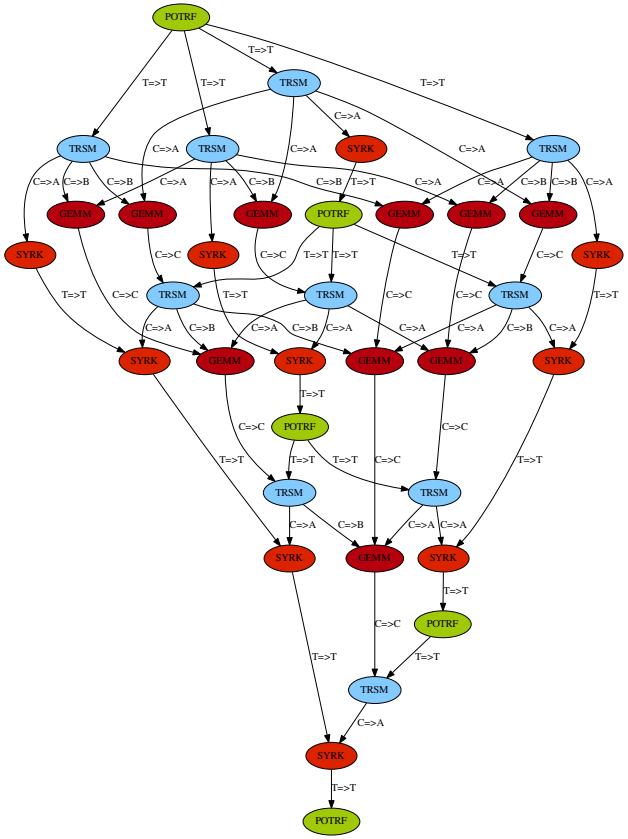


(a) Dense tile task decomposition

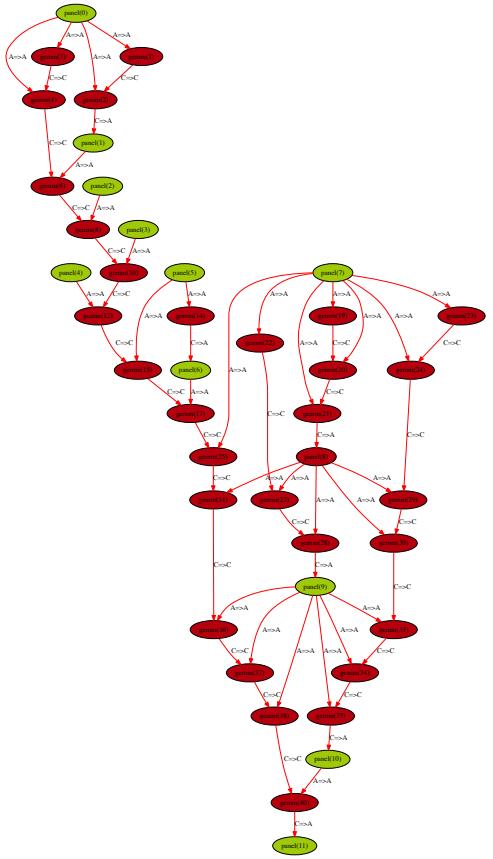


(b) Decomposition of the task applied while processing one panel

DAG representation



(c) Dense DAG



(d) Sparse DAG representation of a sparse LDL^T factorization

Task-based Runtimes

- Task-based programming model;
- Tasks scheduled on computing units (CPUs, GPUs, ...);
- Data transfers management;
- Dynamically build models for kernels;
- Add new scheduling strategies with plugins;
- Get information on idle times and load balances.
- PaRSEC, StarPU, Quark, StarSS, XKaapi...

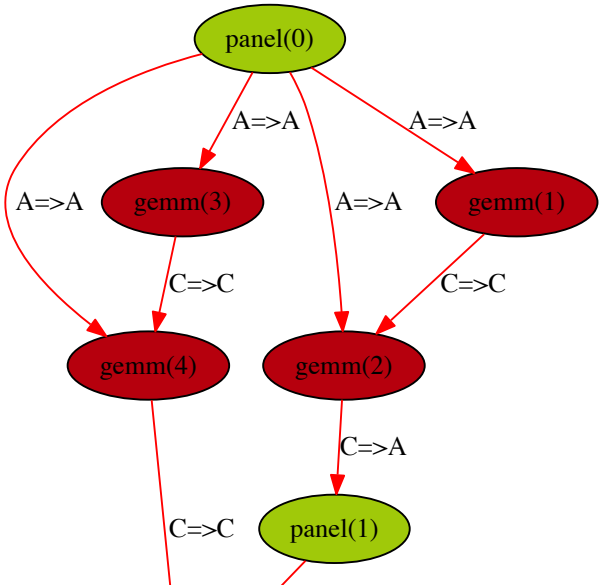
Supernodal sequential algorithm

forall the *Supernode* S_1 **do**
  `panel` (S_1);
  `/* update of the panel */`
  forall the *extra diagonal block* B_i *of* S_1 **do**
    $S_2 \leftarrow$ `supernode_in_front_of` (B_i);
    `gemm` (S_1, S_2);
    `/* sparse` GEMM $B_{k,k \geq i} \times B_i^T$ `substracted from` S_2 `*/`
  end
end

StarPU Tasks submission

forall the *Supernode* S_1 **do**
  `submit_panel` (S_1);
  `/* update of the panel */`
  forall the *extra diagonal block* B_i *of* S_1 **do**
    $S_2 \leftarrow$ `supernode_in_front_of` (B_i);
    `submit_gemm` (S_1, S_2);
    `/* sparse` GEMM $B_{k,k \geq i} \times B_i^T$ `subtracted from` S_2 `*/`
  end
end

PaRSEC's parameterized task graph



Task Graph

```
1   panel(j)
2
3   /* Execution Space */
4   j = 0 .. cblknbr-1
5
6   /* Task Locality (Owner Compute) */
7   :A(j)
8
9   /* Data dependencies */
10  RW A <- ( leaf ) ? A(j) : C gemm( lastbrow )
11       -> A gemm( firstblock+1 .. lastblock )
12       -> A(j)
```

Panel Factorization in JDF Format

2
Shared memory results

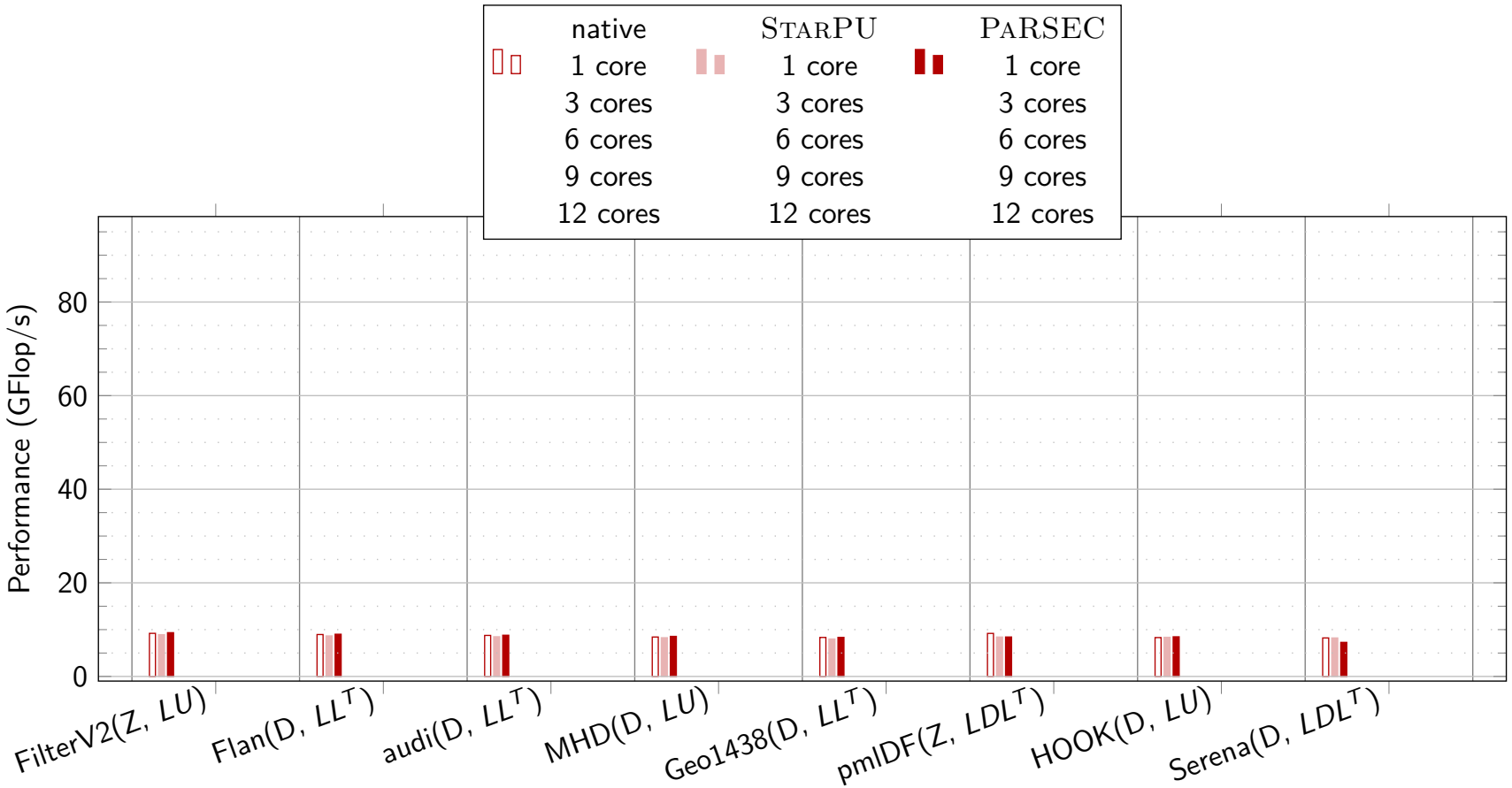
Matrices and Machines

Matrix	Prec	Method	Size	nnz_A	nnz_L	TFlop
FilterV2	Z	LU	0.6e+6	12e+6	536e+6	3.6
Flan	D	LL^T	1.6e+6	59e+6	1712e+6	5.3
Audi	D	LL^T	0.9e+6	39e+6	1325e+6	6.5
MHD	D	LU	0.5e+6	24e+6	1133e+6	6.6
Geo1438	D	LL^T	1.4e+6	32e+6	2768e+6	23
Pmldf	Z	LDL^T	1.0e+6	8e+6	1105e+6	28
Hook	D	LU	1.5e+6	31e+6	4168e+6	35
Serena	D	LDL^T	1.4e+6	32e+6	3365e+6	47

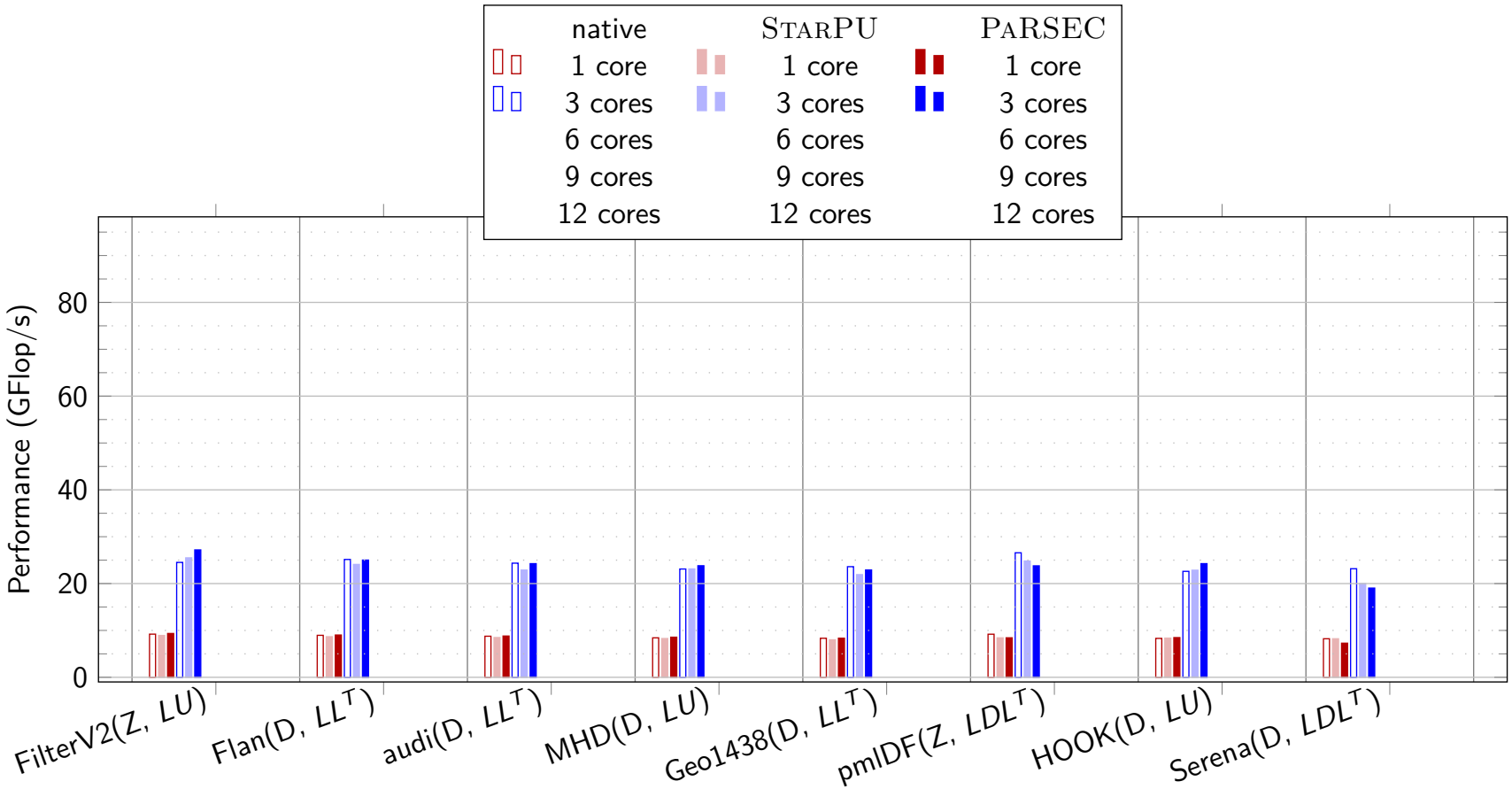
Table: Matrix description (Z: double complex, D: double).

Machine	**Processors**	**Frequency**	GPUs	**RAM**
Mirage	Westmere Intel Xeon X5650 (2×6)	2.67 GHz	Tesla M2070 ($\times 3$)	36 GB

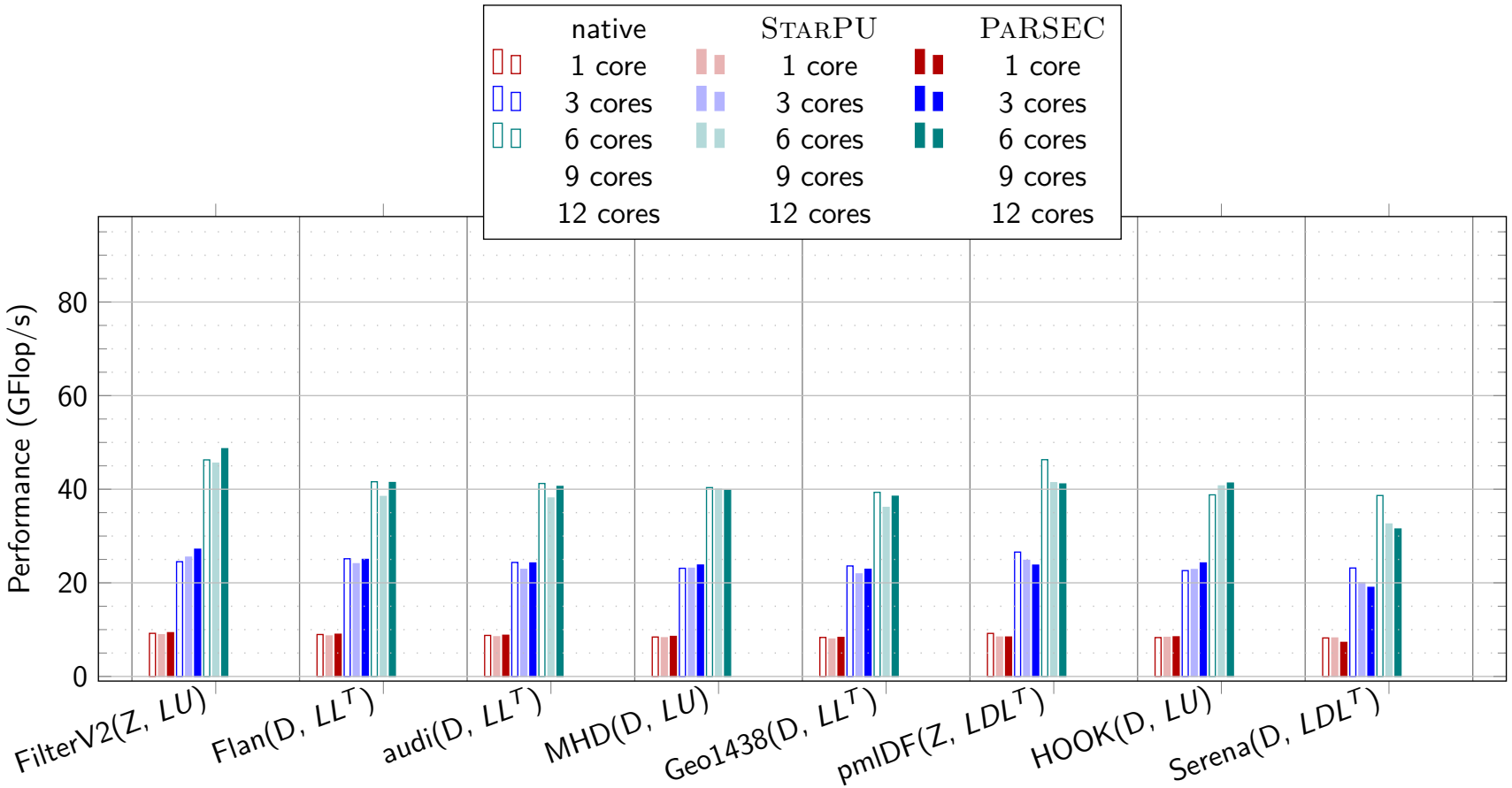
CPU scling study: GFlop/s during numerical factorization



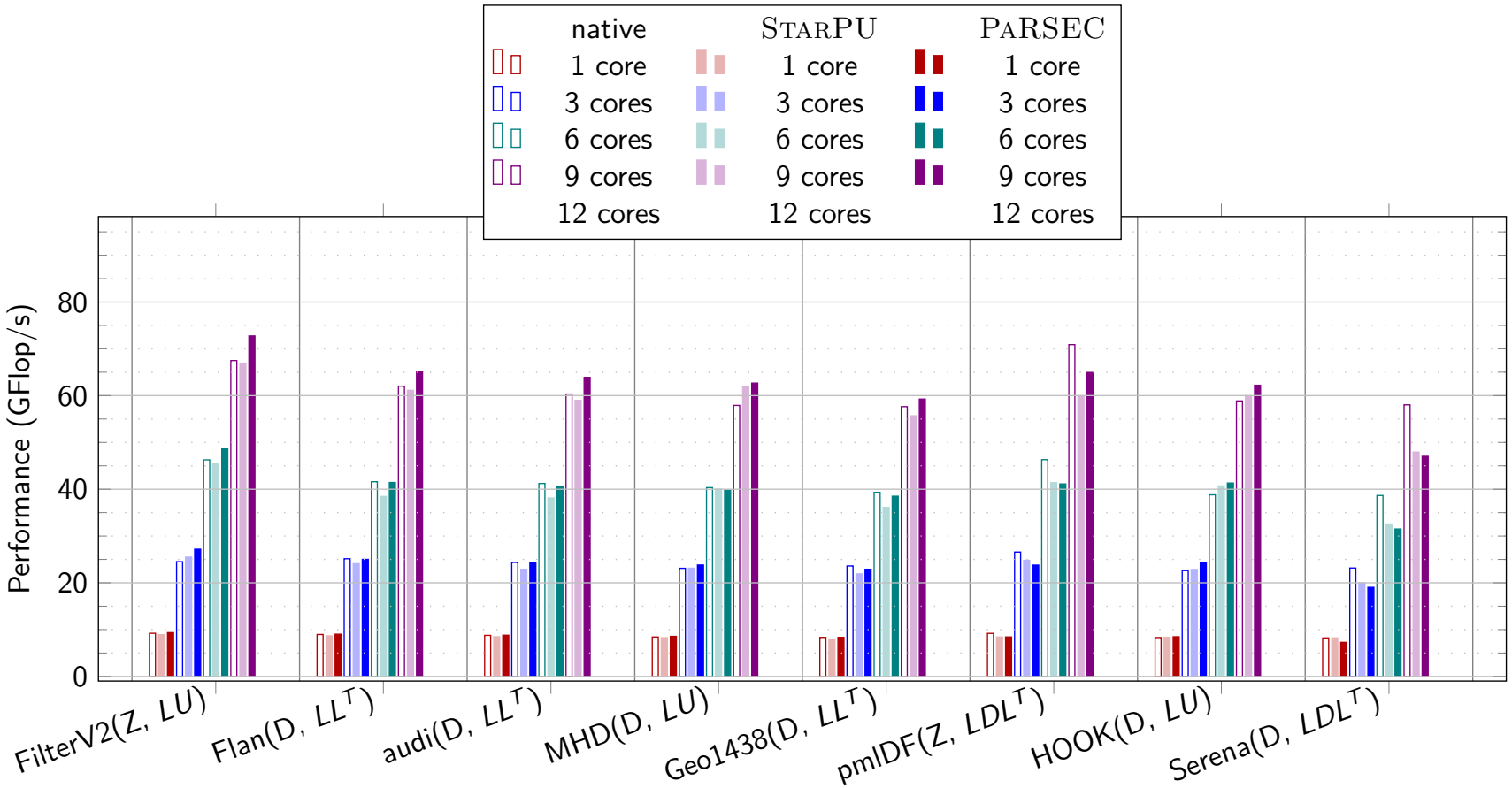
CPU scling study: GFlop/s during numerical factorization



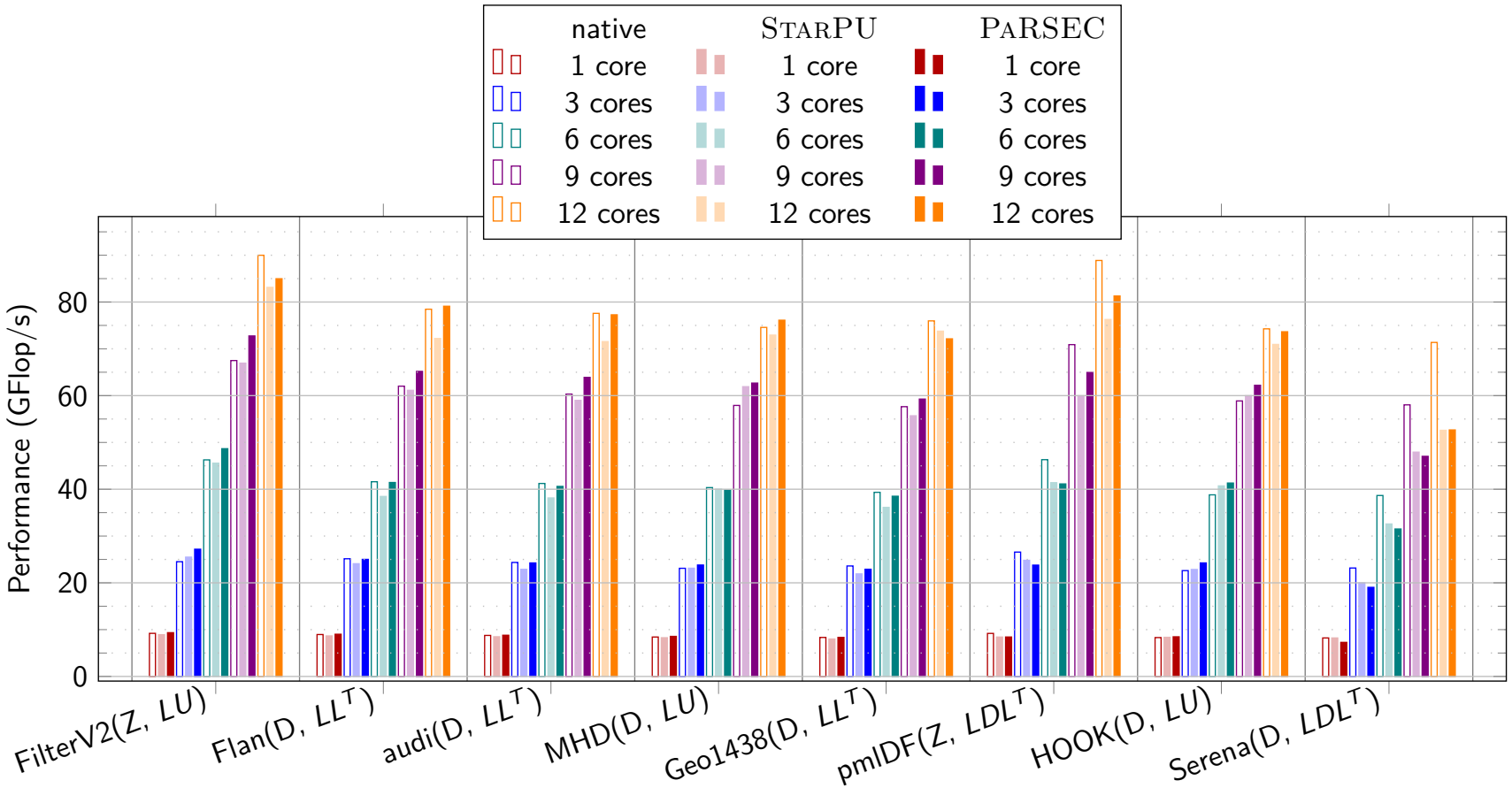
CPU scling study: GFlop/s during numerical factorization



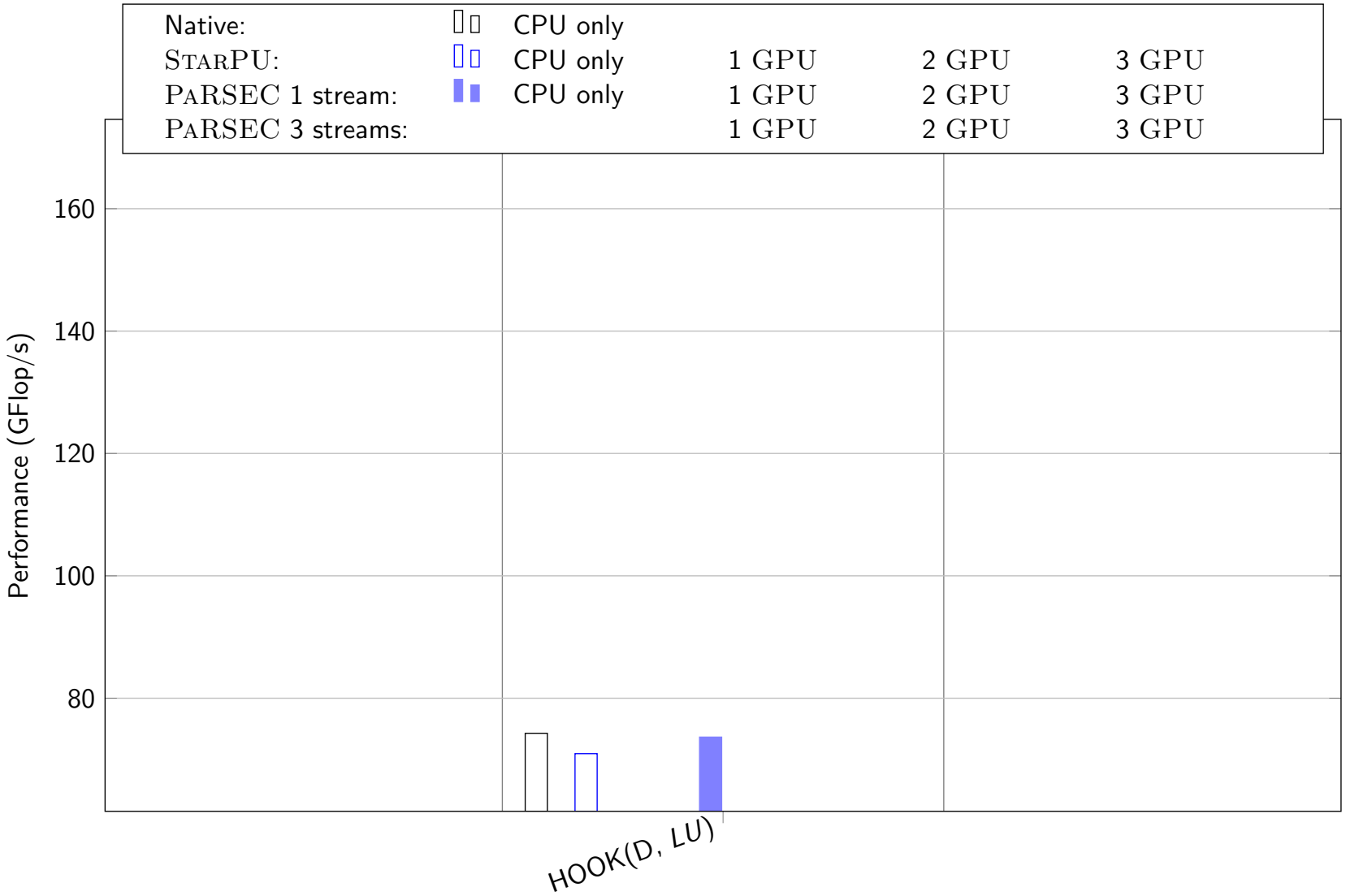
CPU scling study: GFlop/s during numerical factorization



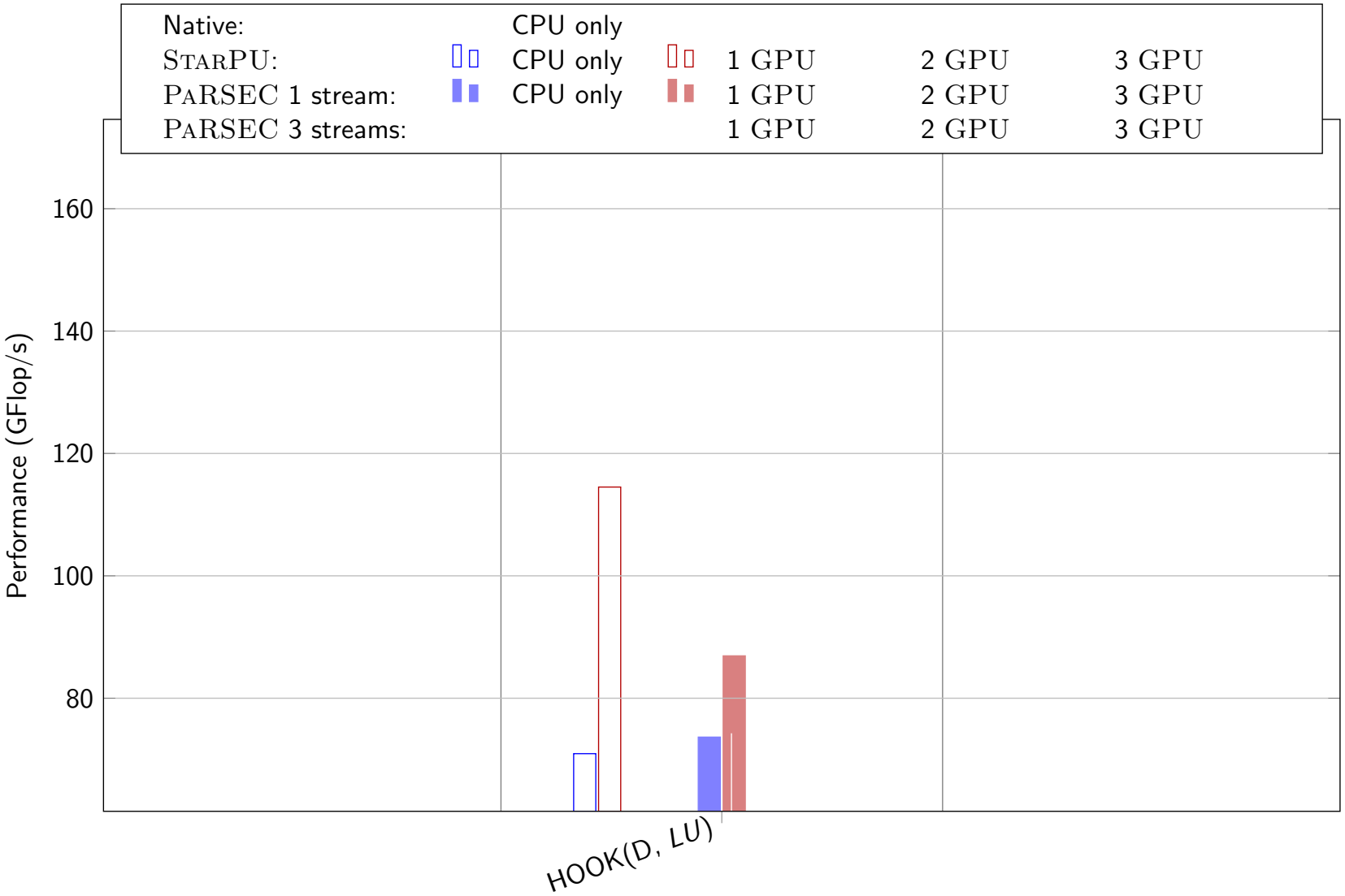
CPU scling study: GFlop/s during numerical factorization



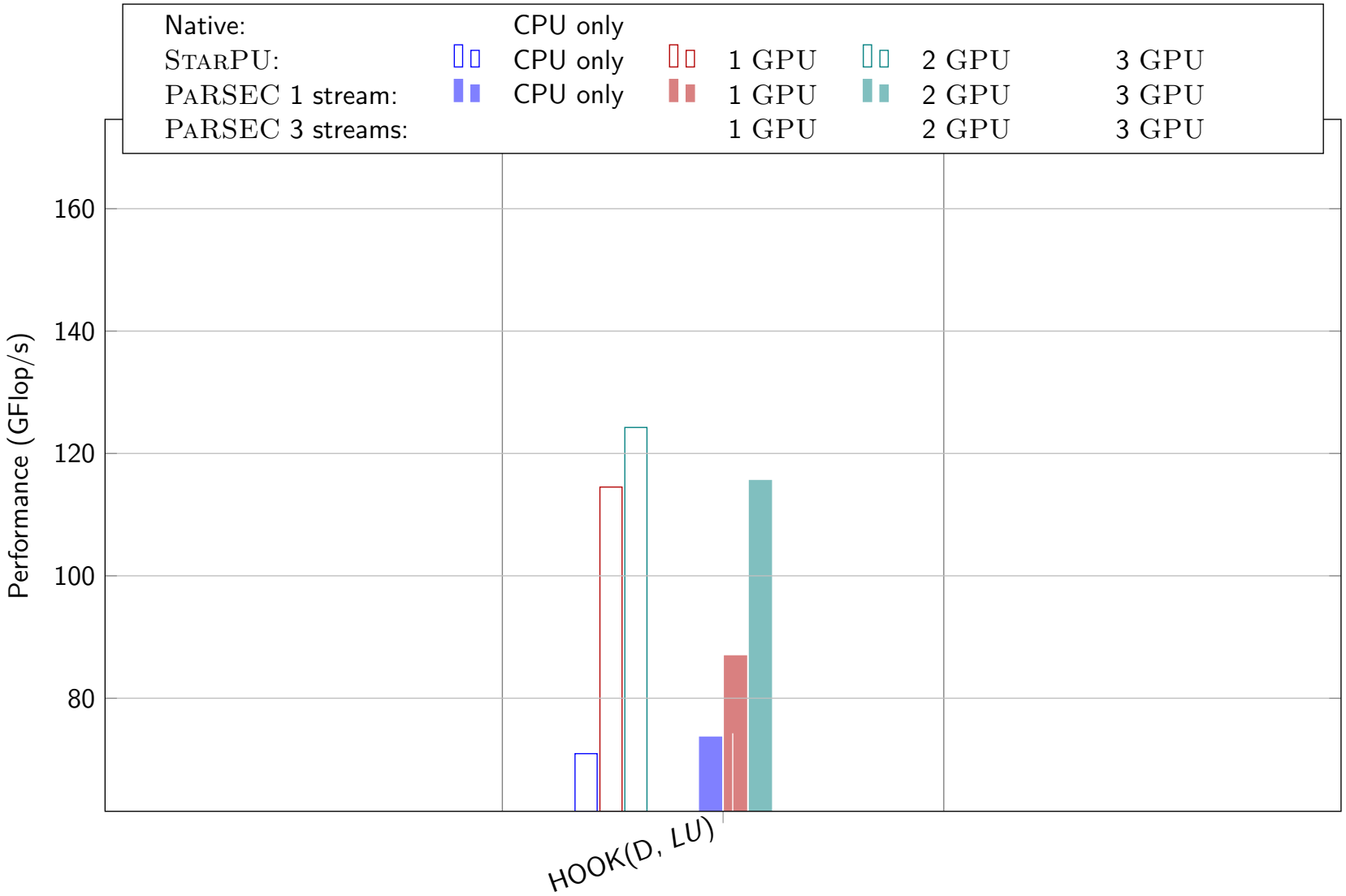
GPU scaling study



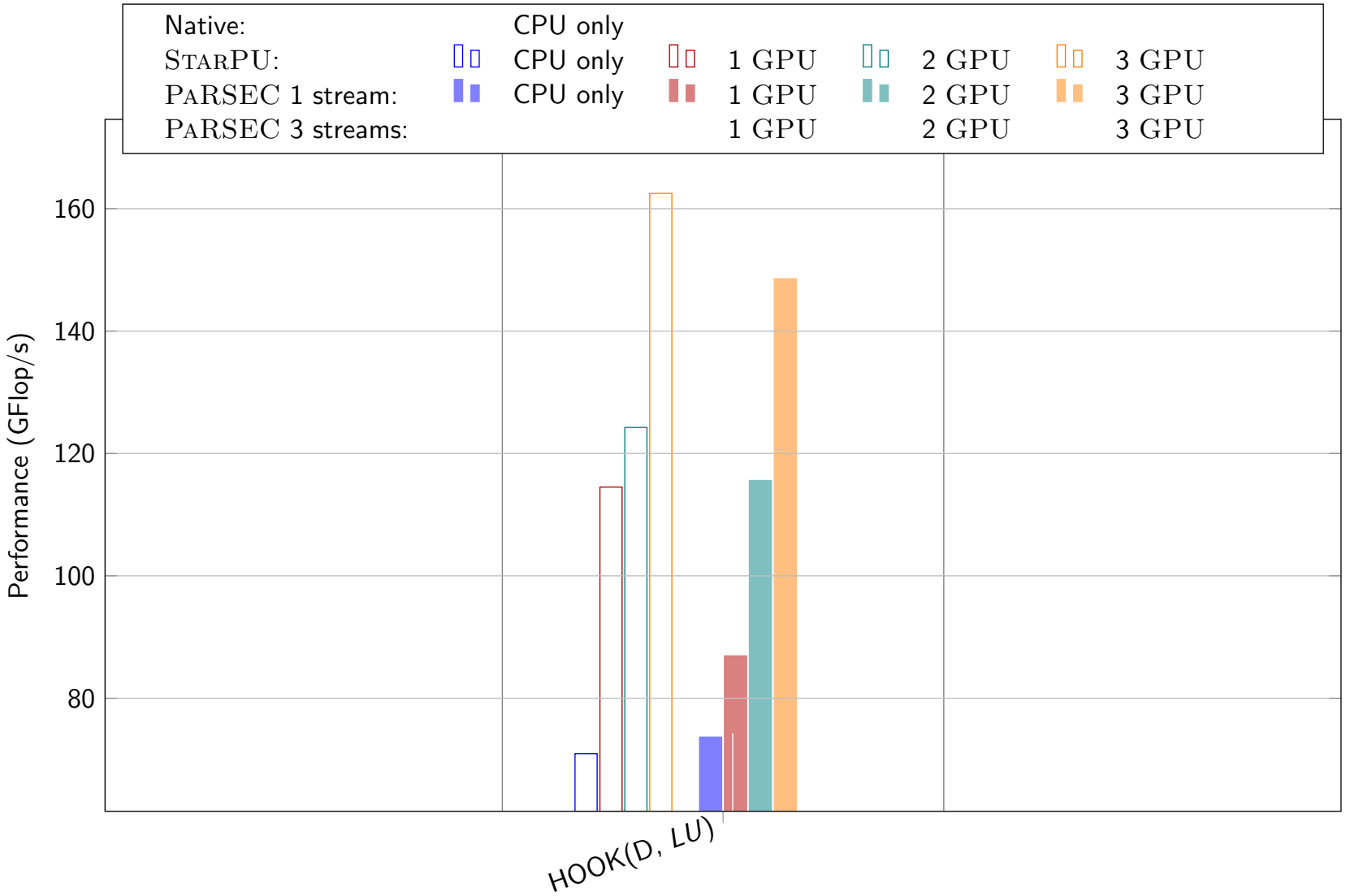
GPU scaling study



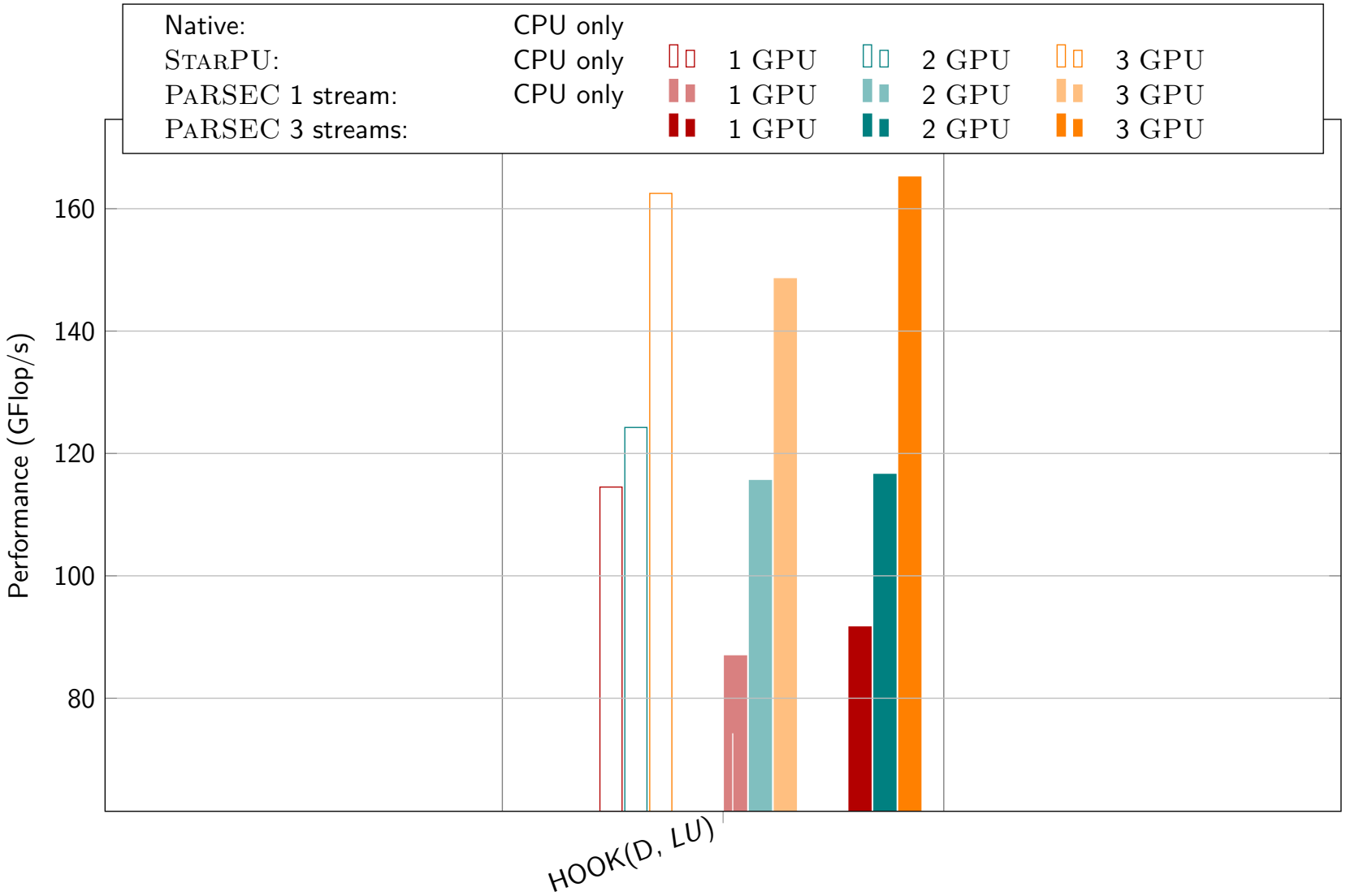
GPU scaling study



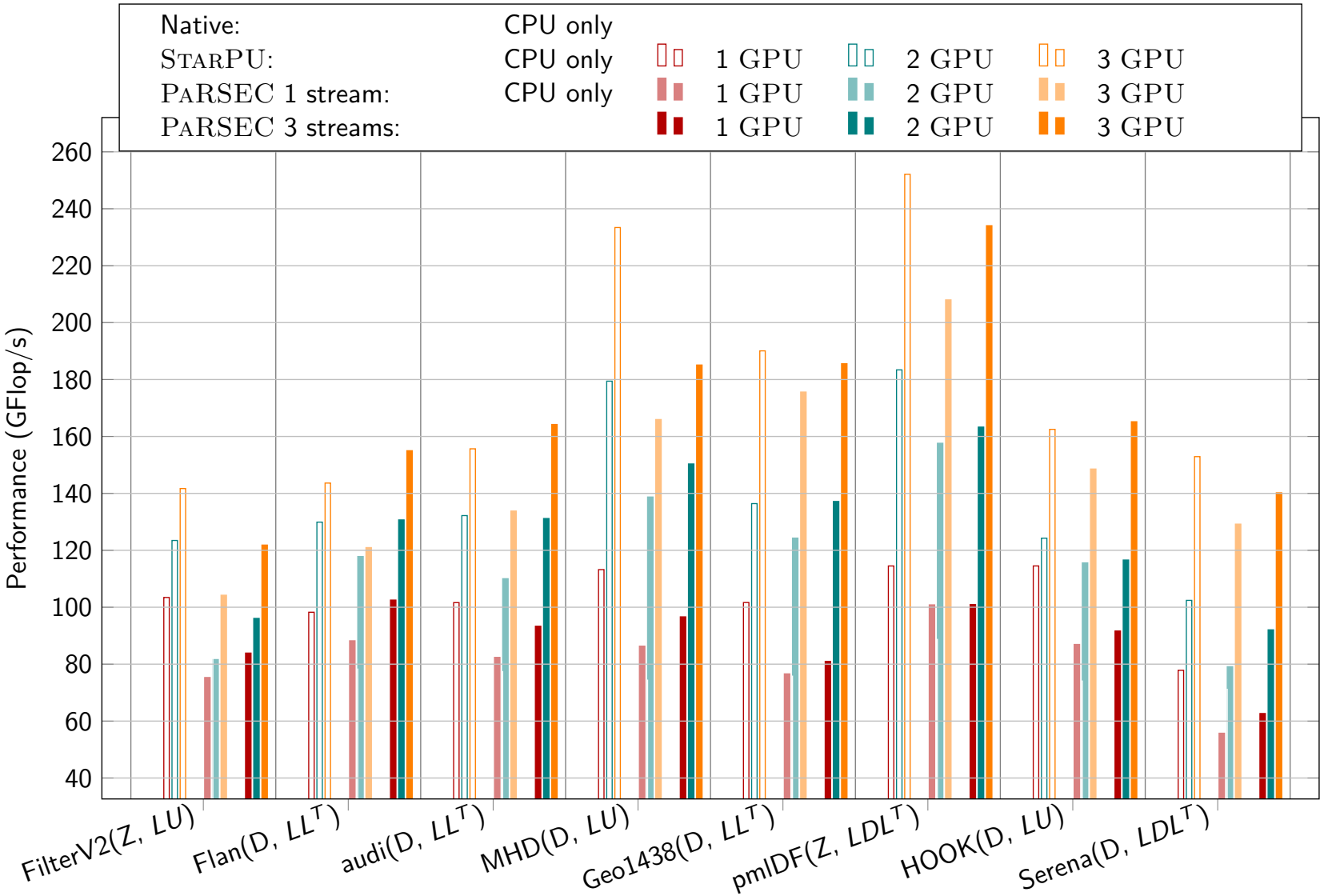
GPU scaling study



GPU scaling study



GPU scaling study



Memory consumption

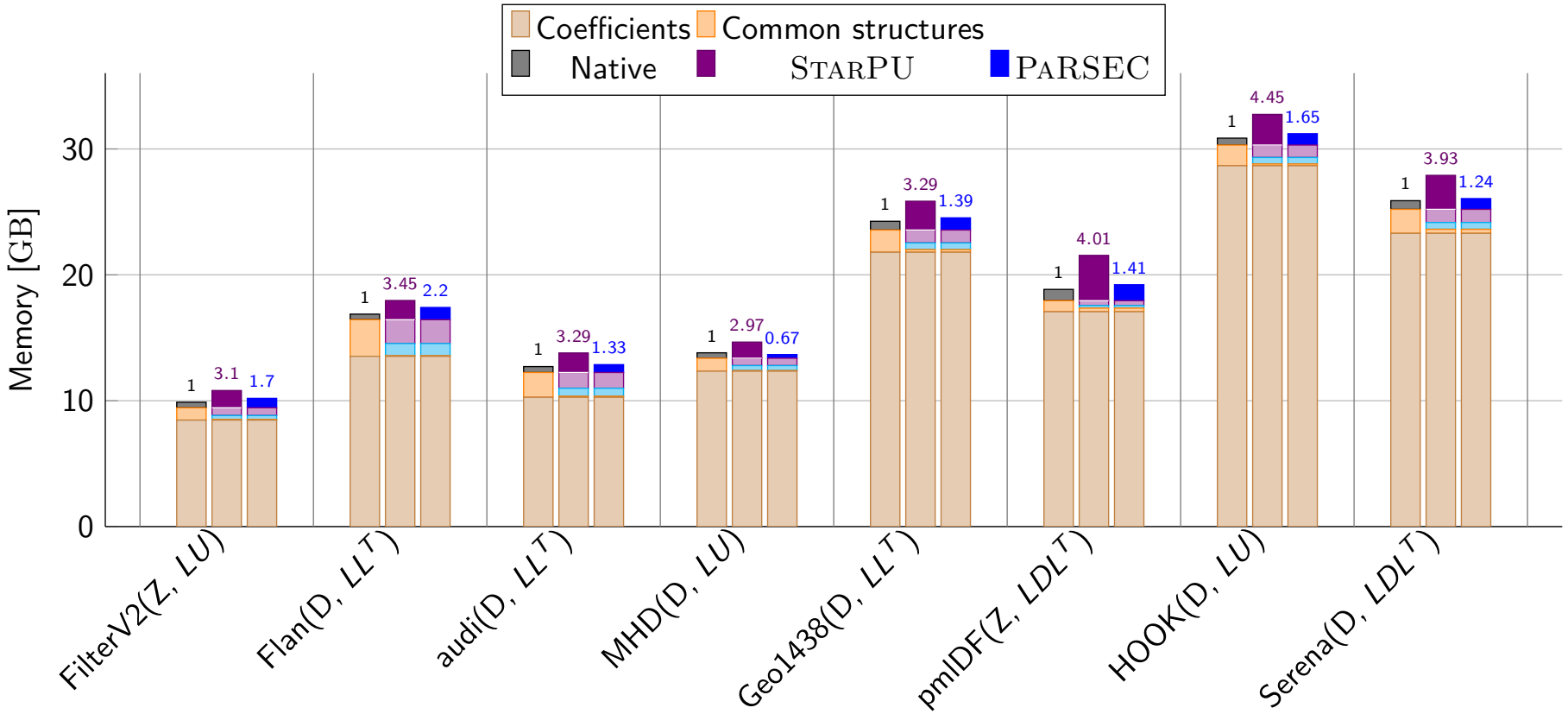


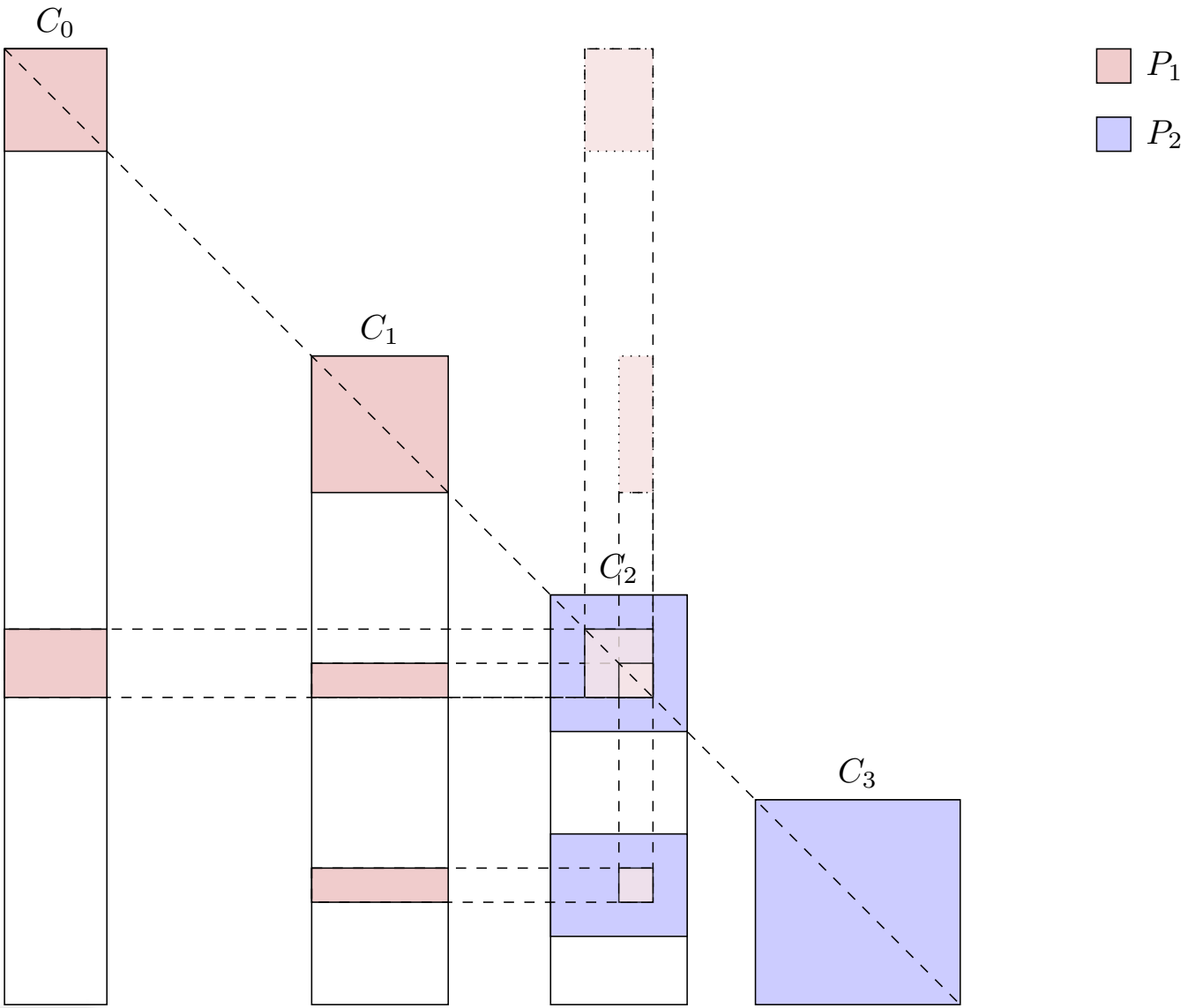
Figure: Memory consumption, common data structures are detailed, overhead ratio on top of bar chart

3
DAG runtimes and distributed architectures

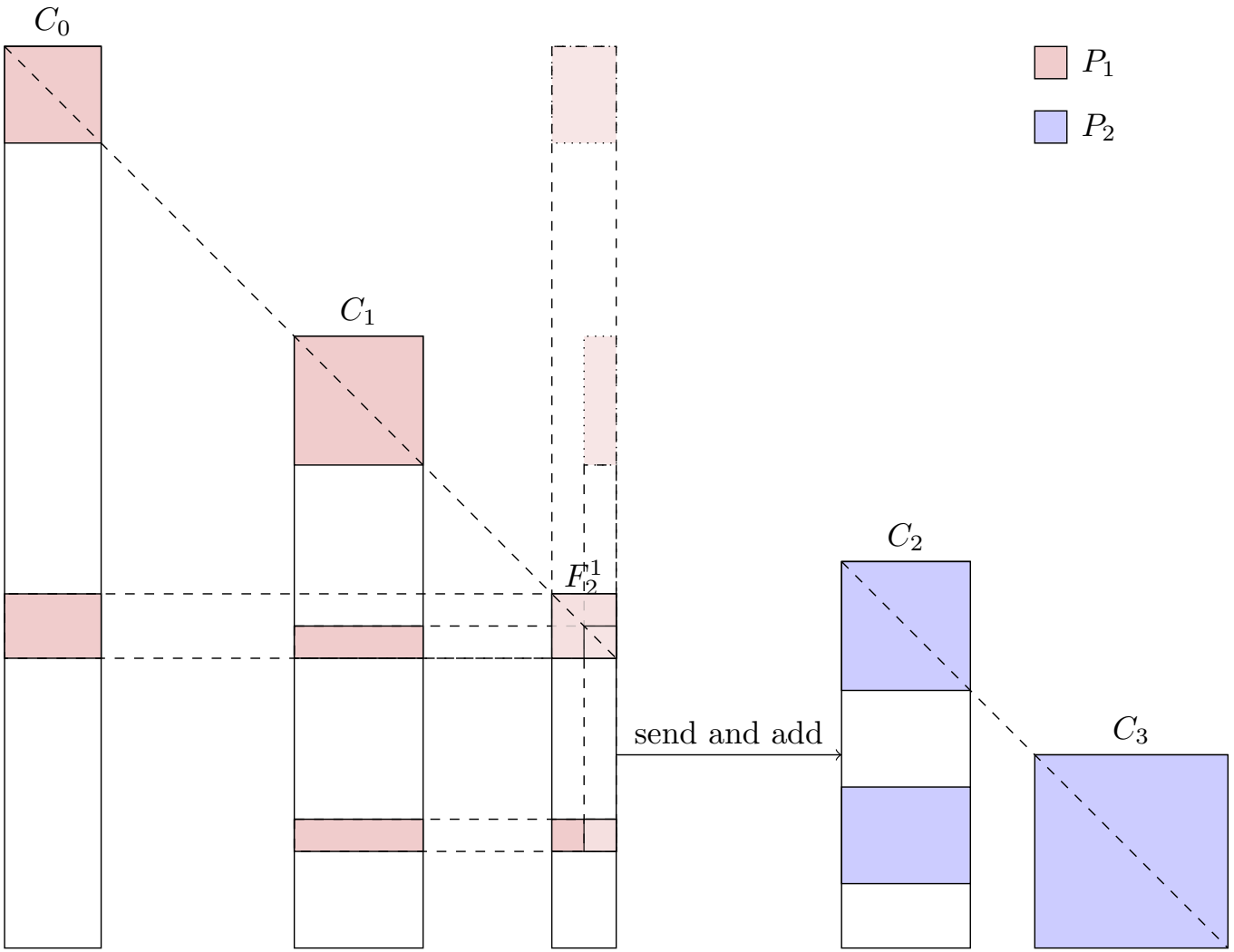
Two algorithms

- Fan-out: column blocks are sent to target blocks owner which performs the product;
- Fan-in: products computed locally and results stored in a temporary buffer corresponding to the destination column block and is sent when all local contributions have been performed.

Distributed matrix example



Distributed matrix example (Fanin)



StarPU fan-out tasks submission

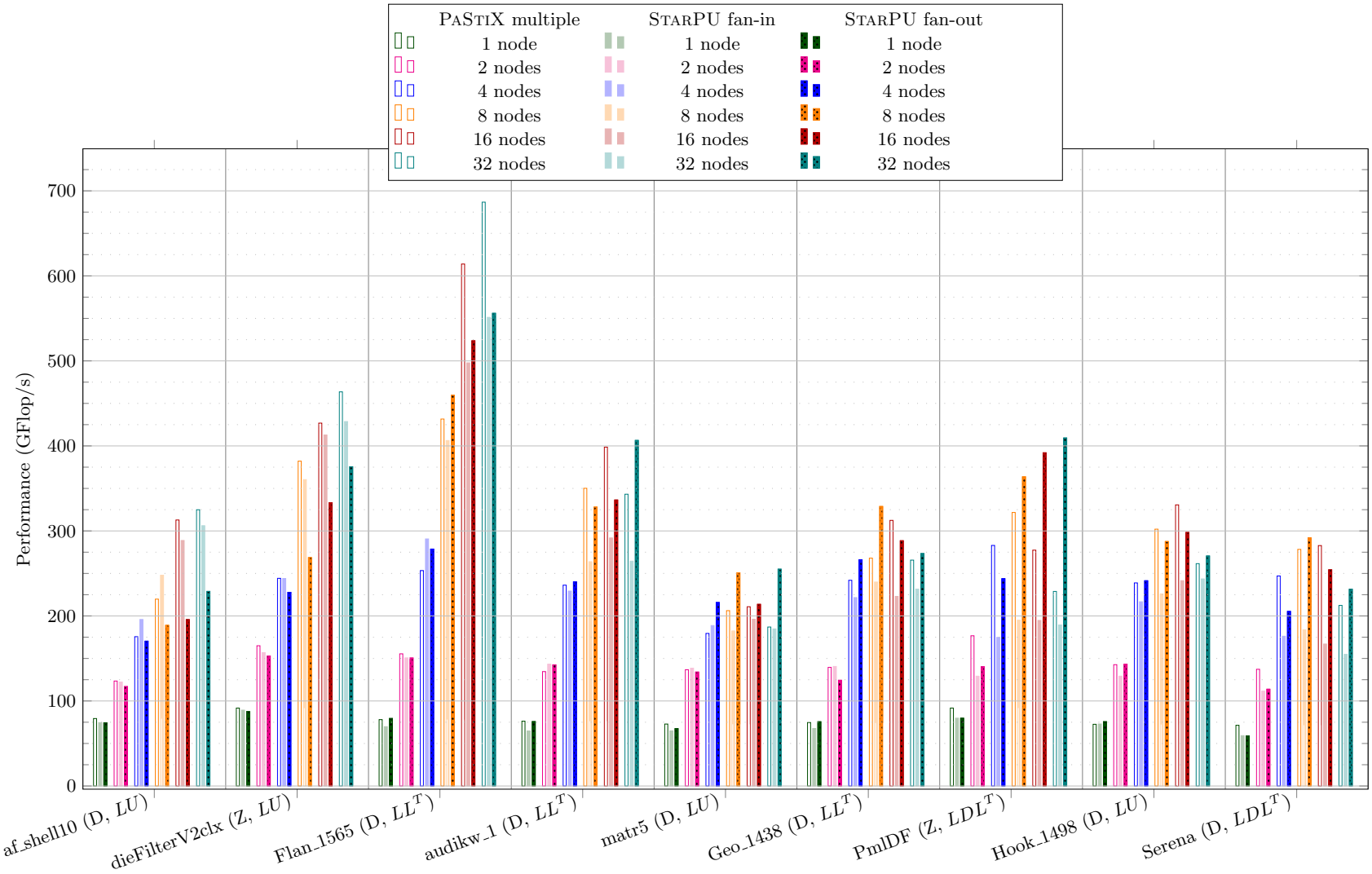
```
forall the local Supernode S1 do
    forall the remote Supernode S2 updating S1 do
        if S2 has not been received then
            forall the local update from S2 on S3 do
                submit_gemm (S2,S3);
            end
            flush_cache (S2);
        end
    end
    submit_panel (S1);
    /* update of the panel                                    */
    forall the extra diagonal block Bi of S1 do
        S2 ← supernode_in_front_of (Bi);
        submit_gemm (S1,S2);
    end
    flush_cache (S1);
end
```

StarPU fan-in tasks submission

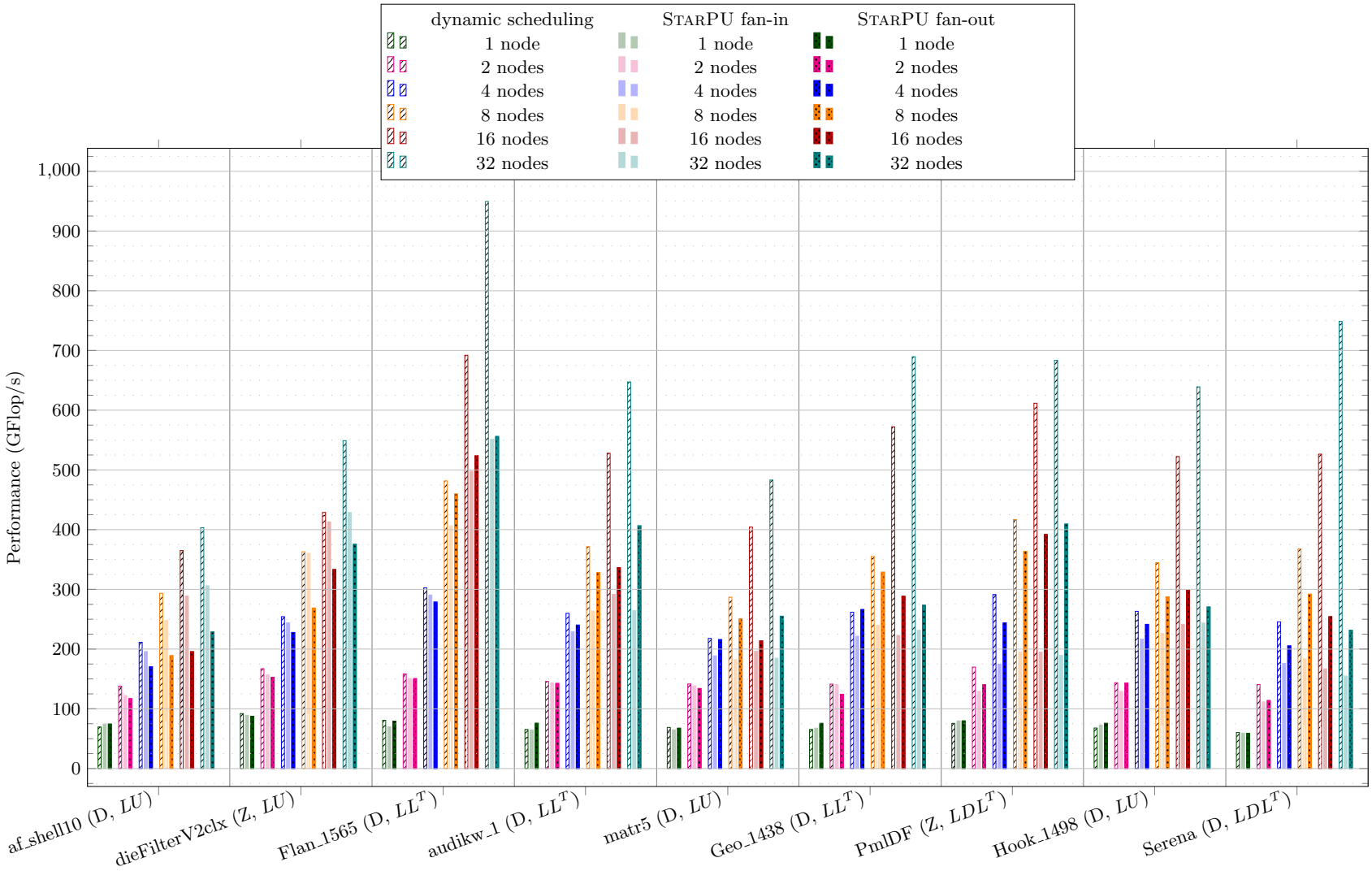
forall the *local Supernode* S_1 **do**
  forall the *fan-in buffer* S_2^P *updating* S_1 *from processor* P **do**
    `submit_add` (S_2^P, S_1);
    `flush_cache` (S_2^P);
  end
  `submit_panel` (S_1);
  forall the *extra diagonal block* B_i *of* S_1 **do**
    `submit_gemm` (S_1,`supernode_in_front_of` (B_i));
  end
  if S_2 *is a fan-in buffer and there is no contribution left on* S_2 **then**
    `submit_add` (S_2, S_2.dest);
    `flush_cache` (S_2);
  end
end

4
Distributed results

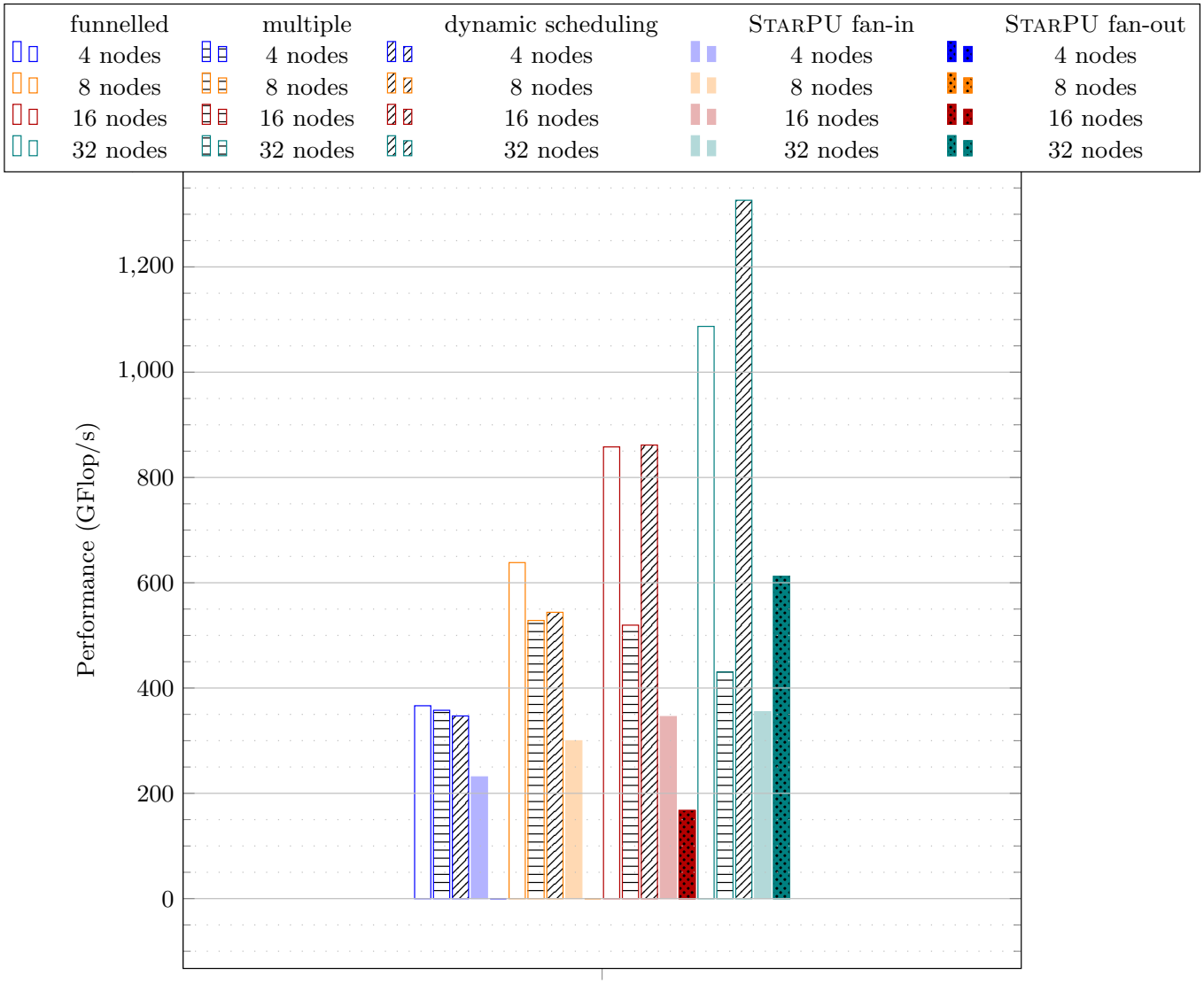
Distributed results



Distributed results vs dynsched



Distributed results (10Millions)



5
Conclusion and future works

Conclusion

- CPU Timing and scaling close to original PaStiX in shared memory, speedup with GPUs;
- Distributed implementation can outperform static scheduling in certain conditions, dynamic scheduling is still beyond;
- Scaling on larger problems is limited.

Future works

- Batched CUDA kernel submission;
- Improve results with heterogeneous distributed clusters;
- Submission window to reduce the runtime overhead ;
- Mix *fan-in/fan-out*;
- Finish my manuscript and celebrate.

Thanks !

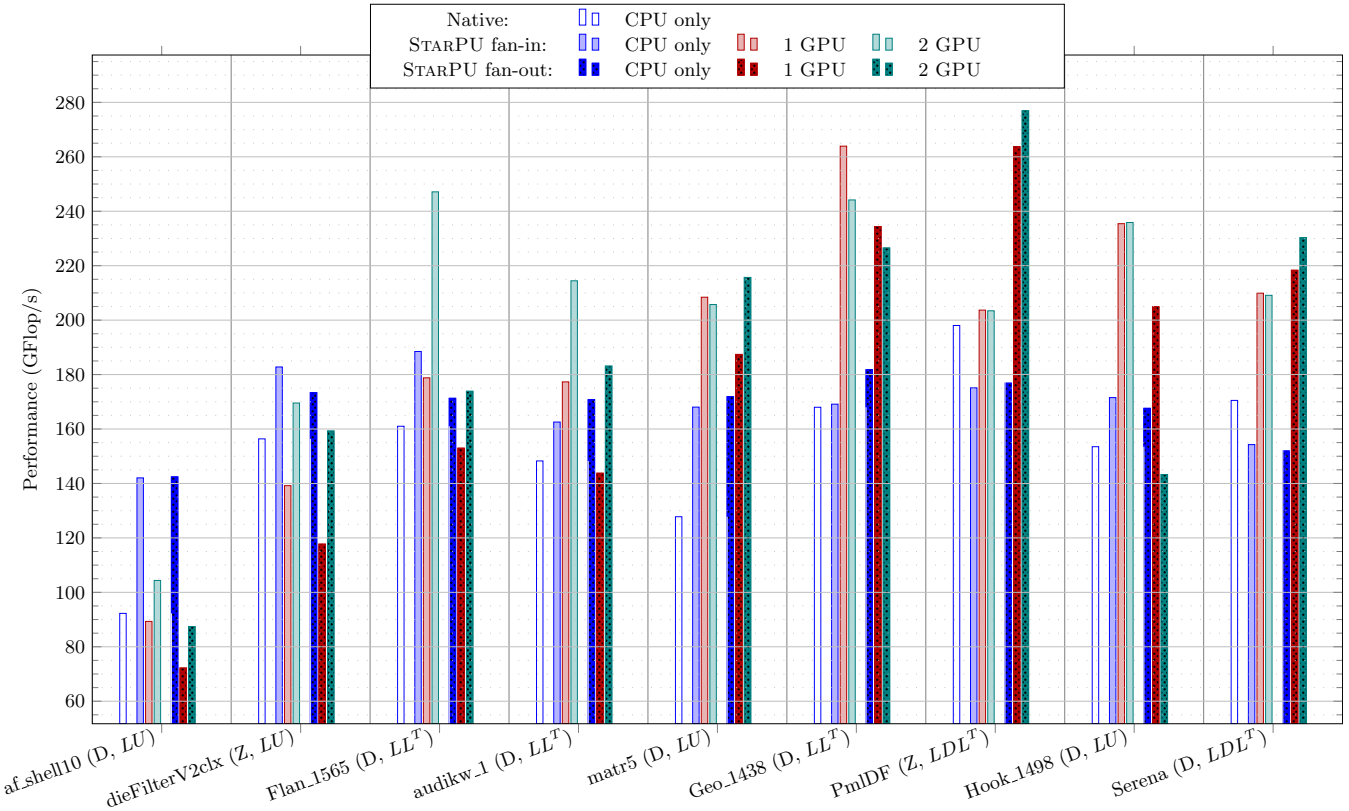
Xavier LACOSTE

INRIA HiePACS team

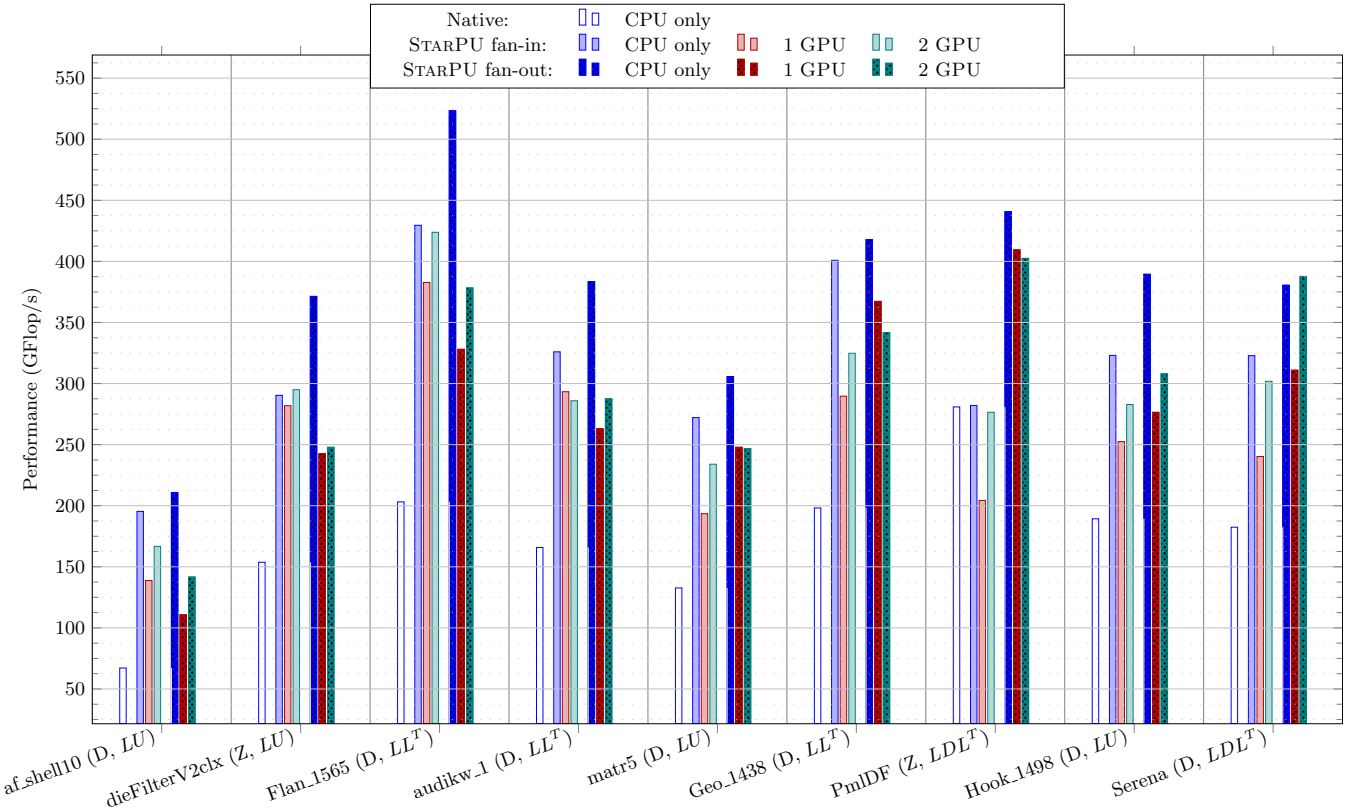
ANR SOLHAR - November 28, 2014

7
More

Distributed heterogeneous results (4 MPI Curie)



Distributed heterogeneous results (16 MPI Curie)

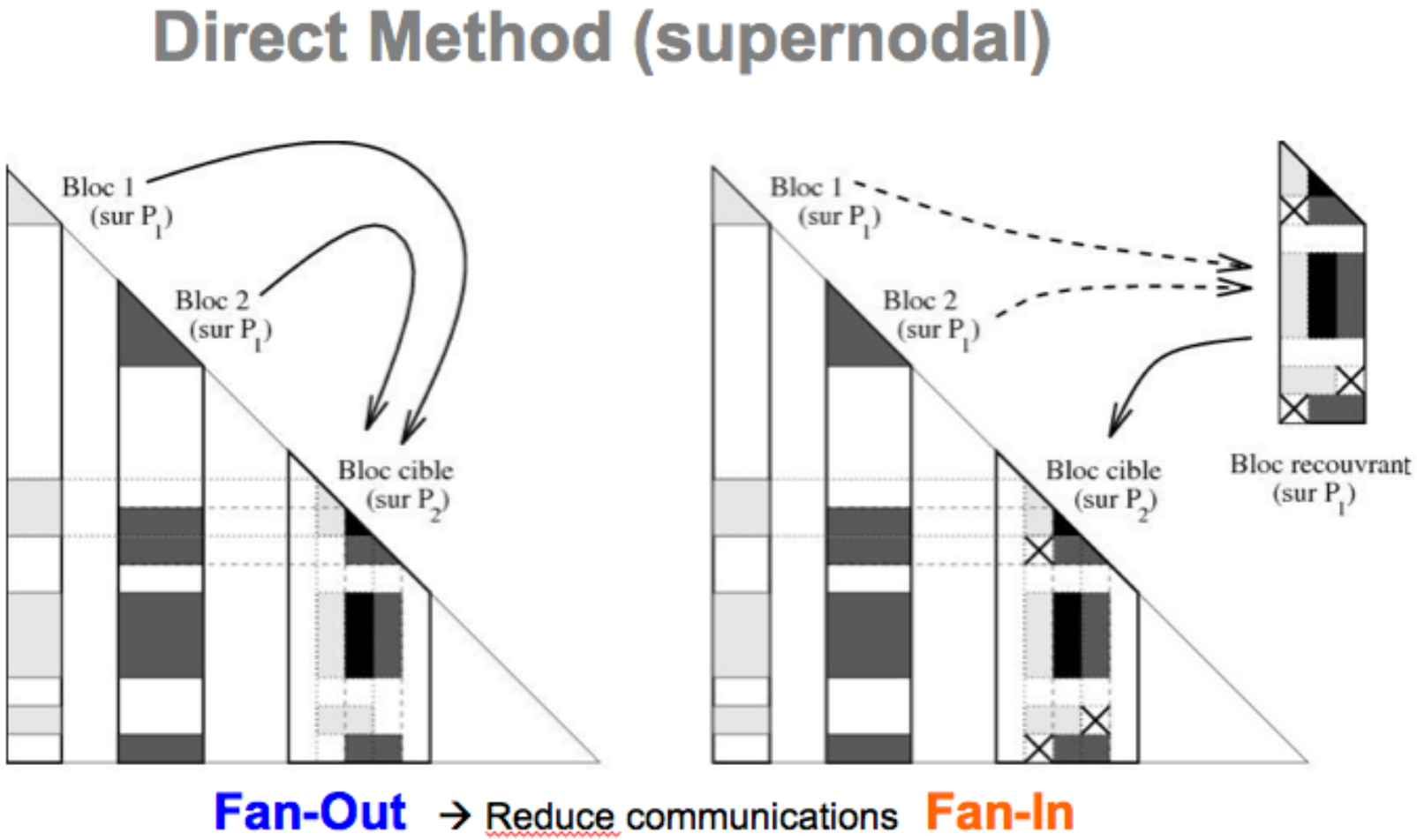


Two methods

Fan-out: Column block are sent before performing updates and GEMM are performed on target column block owner;

Fan-in: GEMM are performed on source column block owner, in a temporary buffer called fanin. Once all local contribution have been performed fanin buffers are sent and added on target.

Communication graph for supernodal factorization



Fan-out

Algorithm 1: Fan-out Cholesky implementation with StarPU.

```
foreach local column block c do
    foreach update from a column block h on a remote node do
        starpu_mpi_insert_task (gemm, h, c);
    end
    starpu_mpi_insert_task (potrf_trsm, c);
    foreach block b ∈ c do
        fc ← facing_cblk (b);
        starpu_mpi_insert_task (gemm, c, fc);
        /* This task be local or not, depending on fc          */
    end
end
```

Fan-in

Algorithm 2: Fan-in Cholesky implementation with StarPU.

```
foreach local column block c do
    foreach update from a fanin column block f on a remote node do
        starpu_mpi_insert_task (add, f, c);
    end
    starpu_mpi_insert_task (potrf_trsm, c);
    foreach block b ∈ c do
        fc ← facing_cblk (b) /* fc can be a local or fanin column
            block                                                  */
        ;
        starpu_mpi_insert_task (gemm, c, fc);
        if is_fanin (fc) and update_count (fc) = 0 then
            starpu_mpi_insert_task (add, fc, target (fc)) /* This task
                will be executed on target (fc) owner node         */
            ;
        end
    end
end
```

Preliminary results on audi with four 8 cores MPI nodes

Method	Factorization Time	Factorization FLOPS
pastix	39 s	124 GFlops
fanin	75 s	64 GFlops
fanout	79 s	61 GFlops

Still a lot of tuning work required to get performance.

DAG schedulers considered

StarPU

- RunTime Team – Inria Bordeaux Sud-Ouest
- C. Augonnet, R. Namyst, S. Thibault.
- Dynamic Task Discovery
- Computes cost models on the fly
- Multiple kernels on the accelerators
- Heterogeneous First-Time strategy

PaRSEC (formerly DAGuE)

- ICL – University of Tennessee, Knoxville
- G. Bosilca, A. Bouteiller, A. Danalys, T. Herault
- Parameterized Task Graph
- Only the most compute intensive kernel on accelerators
- Simple scheduling strategy based on computing capabilities
- GPU multi-stream enabled