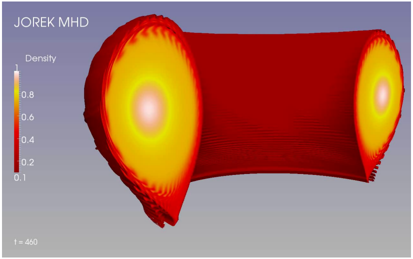
Enhancing sparse direct solver scalability through runtime system automatic data partition.

A. Lisito, M. Faverge, G. Pichon, P. Ramet

Applications

- Computational fluid dynamics
- Electromagnetism
- Nuclear fusion



ITER project

Sparse Direct Linear Algebra Solvers:

- Solve $Ax = b$
- A a sparse matrix
- x and b two vectors

How ?

- Permute A: $A = PA_pP^t$
- **Factorize $\mathbf{A_p}$: $\mathbf{A_p} = \mathbf{LU}$** (or $A_p = LL^t$ or $A_p = LDL^t$)
- Solve $Ly = (Pb)$
- Solve $U(Px) = y$

The operation

- $\mathbf{A_p} = \mathbf{LU}$ or $\mathbf{A_p} = \mathbf{LL^t}$ or $\mathbf{A_p} = \mathbf{LDL^t}$

The goal

- Have enough parallelism to feed all the computing cores
- Good task efficiency

How ?

- Have large enough task size
- Have a reasonable number of tasks
- Have few data dependencies between the tasks

Factorization algorithm

For cblk_k in cblks

· · · · · · FACTO(diag_k)

· · · · · · **For** $\text{blok}_{k,m}$ in odb_k

· · · · · · · · · · TRSM($\text{blok}_{k,m}$)

· · · · · **EndFor**

· · · · · · **For** $\text{blok}_{k,n}$ in odb_k

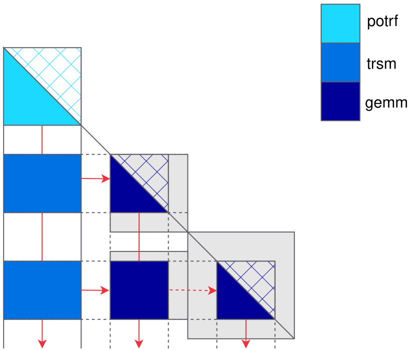
· · · · · · · · · · **For** $\text{blok}_{k,m}$ in odb_k ($m > n$)

· · · · · · · · · · · · · · GEMM($\text{blok}_{k,m}$, $\text{blok}_{k,n}$)

· · · · · · · · · · **EndFor**

· · · · · · **EndFor**

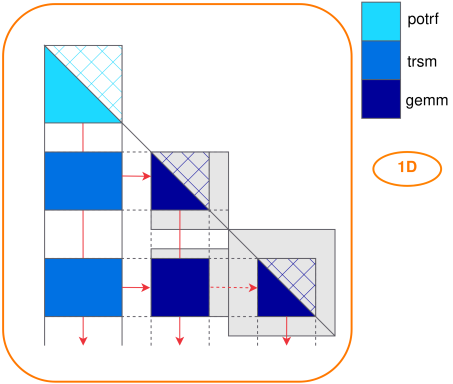
EndFor



- Three nested loops
- Task sizes ?

Factorization algorithm: 1D task level

```
For cblk_k in cblks
    FACTO( diag_k )
    For blok_{k,m} in odb_k
        TRSM( blok_{k,m} )
    EndFor
    For blok_{k,n} in odb_k
        For blok_{k,m} in odb_k (m > n)
            GEMM( blok_{k,m}, blok_{k,n} )
        EndFor
    EndFor
EndFor
```



- Few but very large tasks
- Lots of data dependencies
- Blas parallelism
- Synchronisation

For cblk_k in cblks

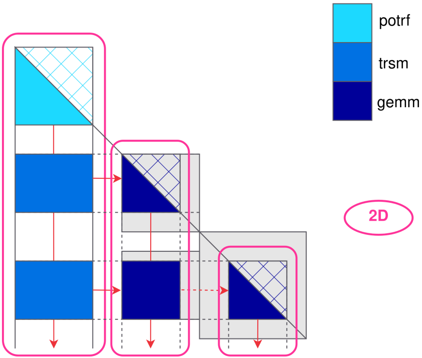
- FACTO(diag_k)
- **For** $\text{blok}_{k,m}$ in odb_k
  - TRSM($\text{blok}_{k,m}$)
- **EndFor**

For $\text{blok}_{k,n}$ in odb_k

- **For** $\text{blok}_{k,m}$ in odb_k ($m > n$)
  - GEMM($\text{blok}_{k,m}$, $\text{blok}_{k,n}$)
- **EndFor**

EndFor

EndFor



- Medium sized tasks
- Less data dependencies
- Panel + Blas parallelism
- Less synchronization

For $cblk_k$ in cblks

FACTO($diag_k$)

For $blok_{k,m}$ in odb_k

TRSM($blok_{k,m}$)

EndFor

For $blok_{k,n}$ in odb_k

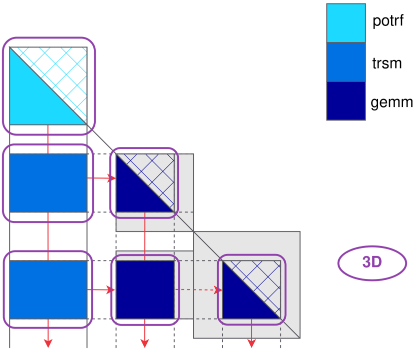
For $blok_{k,m}$ in odb_k ($m > n$)

GEMM($blok_{k,m}$, $blok_{k,n}$)

EndFor

EndFor

EndFor



- Lots of small tasks
- Few data dependencies
- Block parallelism
- Hard to handle without runtime

PaStiX

- Many variants to support multi-core systems
  - POSIX Threads:
    Single-thread, Multi-thread with static or **dynamic scheduling**
  - Use of external runtime systems:
    StarPU, PaRSEC
- Support of distributed architectures with MPI
- Numerical features
  - Low / Full rank
  - Mixed precision
  - Multi-DOF support (constant and variadic)

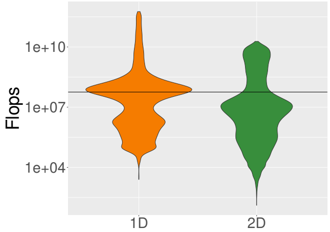
Tasks in PaStiX

- Dynamic scheduler: *1D* and *2D*
- StarPU runtime: *2D* and ***3D* new to PaStiX 6**

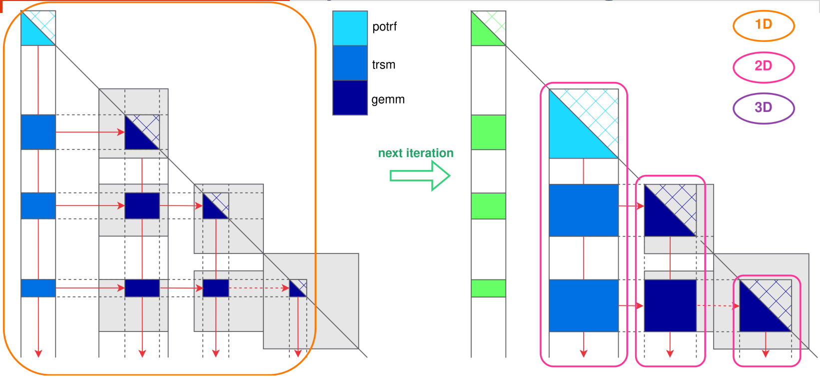
Dynamic Factorization

Scheduler	Task distribution	Number of			Mflop/t	Factorization	
		cblks	bloks	tasks		s.	GFlop/s
Dynamic	1D	17 307	193 887	17 307	5 976.6	66.07	1424.86
	2D			193 887	655.5	52.63	1801.16

- Target: dense gemm $384^3 \rightarrow 56.6 MFlops$
- *2D* tasks 10 times smaller than *1D* tasks
- Largest *1D* task takes 150ms!
- 1.26 speed-up with *2D*
- Is *2D* the best ?



Dynamic factorization algorithm

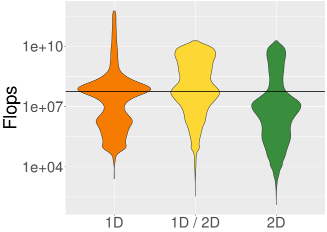


- PaStiX + dynamic = mixed *1D* and *2D*
- Smaller tasks grouped in *1D* and larger split in *2D*
- The *2D* tasks free other tasks faster

Dynamic Factorization

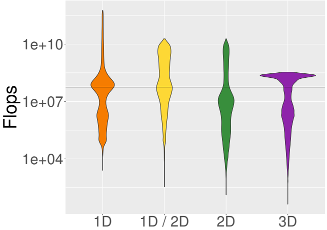
Scheduler	Task distribution	Number of cblks	Number of bloks	Number of tasks	Mflop/t	Factorization s.	Factorization GFlop/s
Dynamic	1D	17 307	193 887	17 307	5 976.6	66.07	1424.86
	1D / 2D			85 532	1 414.7	**51.27**	**1851.64**
	2D			193 887	655.5	52.63	1801.16

- Mixed *1D* and *2D* doubles the average task size from only *2D*
- 1.29 speed-up with mixed *1D* and *2D*

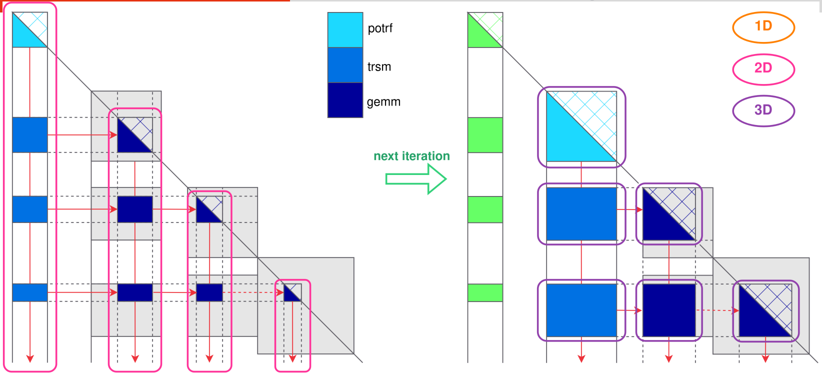


Scheduler	Task distribution	Number of cblks	Number of bloks	Number of tasks	Mflop/t	Factorization s.	Factorization GFlop/s
Dynamic	1D	17 307	193 887	17 307	5 976.6	66.07	1424.86
	1D / 2D			85 532	1 414.7	**51.27**	**1851.64**
	2D			193 887	655.5	52.63	1801.16
StarPU	2D	17 307	193 887	193 887	655.5	45.89	2050.21
	3D			1 114 228	81.4	24.99	3767.27

- StarPU *2D* better than Dynamic (speed-up of 1.15)
- 2.64 speed-up with *3D*
- Lots of very small tasks in *3D*
- What about mixed *2D* / *3D* ?



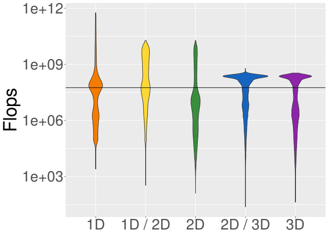
StarPU factorization algorithm



- PaStiX + StarPU = mixed *2D* and *3D*
- Smaller tasks grouped in *2D* and larger split in *3D*
- The *3D* tasks free other tasks faster

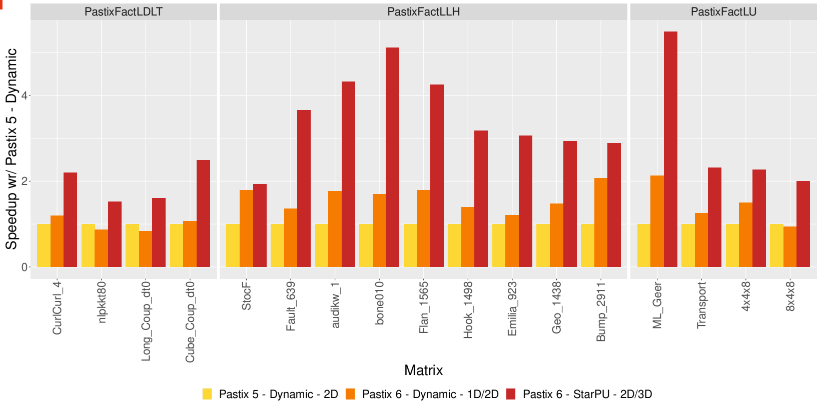
Scheduler	Task distribution	cblks	Number of bloks	tasks	Mflop/t	Factorization s.	Factorization GFlop/s
	1D			17 307	5 976.6	66.07	1424.86
Dynamic	1D / 2D	17 307	193 887	85 532	1 414.7	**51.27**	**1851.64**
	2D			193 887	655.5	52.63	1801.16
	2D			193 887	655.5	45.89	2050.21
StarPU	2D / 3D	17 307	193 887	908 256	105.5	**23.64**	**3980.67**
	3D			1 114 228	81.4	24.99	3767.27

- Mixed *3D* and *3D* increases the average task size from *2D*
- 1.94 speed-up with mixed from *2D* StarPU
- 2.79 speed-up with mixed overall



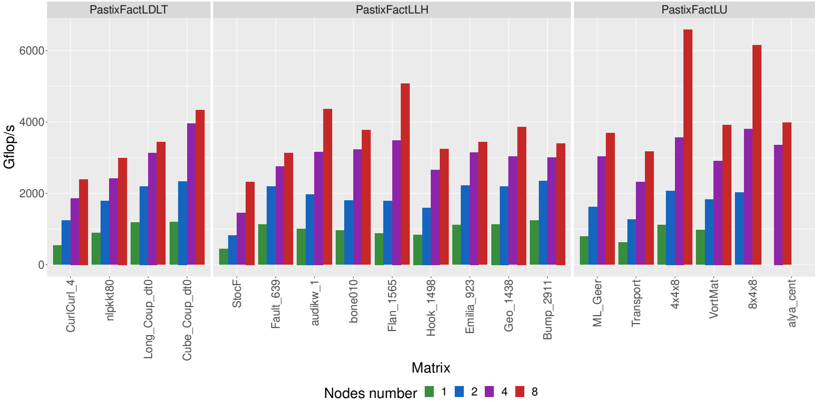
- The matrices:
  - Taken from the *SuiteSparse Matrix Collection*
  - Size: from $600K$ to $10M$ non zero elements
  - Reals and pattern symmetrics, numerical symmetrics or positives definites
- The machines:
  - Inria HPC platform Plafrim
  - Bora: 2 CPU with 18 cores Intel CascadeLake
  - 1 MPI process per node and 36 threads per MPI process
- The tools handled with guix:
  - mkl 2020
  - gcc 11.2
  - hwloc 2.9.0
  - scotch 7.0.1
  - starpu 1.4.3 (lws scheduler)
  - openmpi 4.1.5

Factorization - 4 MPI x 36 thr (1 MPI/node)



- PaStiX 6 faster than PaStiX 5
- Big speedup for StarPU

StarPU Factorization



- Speedup of 2.01 to 3.94 on 4 nodes

Conclusion

- Good management of the task size and number of tasks
- Improve performance scalability

Future study

- Exploit StarPU recursive tasks for more modularity
- Exploit 3D task level on GPU
- Use StarPU to redistribute the end of the matrix

Thank you for your attention!