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 $A_p$:** $A_p = LU$ (or $A_p = LL^t$ or $A_p = LDL^t$)
- Solve $Ly = (Pb)$
- Solve $U(Px) = y$
## Factorization algorithm: task granularity

### The operation
- \( A_p = LU \) or \( A_p = LL^t \) or \( A_p = LDL^t \)

### 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
For cblk<sub>k</sub> in cblks

... FACT0( diag<sub>k</sub> )

... For bloc<sub>k,m</sub> in odb<sub>k</sub>

... ... TRSM( bloc<sub>k,m</sub> )

... EndFor

... For bloc<sub>k,n</sub> in odb<sub>k</sub>

... ... For bloc<sub>k,m</sub> in odb<sub>k</sub> (m > n)

... ... ... GEMM( bloc<sub>k,m</sub>, bloc<sub>k,n</sub> )

... ... ... EndFor

... ... EndFor

EndFor

- Three nested loops
- Task sizes?
Factorization algorithm: 1D task level

For \( cblk_k \) in \( cblks \)

\[
\begin{align*}
\text{FACT0( diag}_k) \\
\text{For } \text{blok}_{k,m} \text{ in odb}_k \\
\text{TRSM( } \text{blok}_{k,m} ) \\
\text{EndFor} \\
\text{For } \text{blok}_{k,n} \text{ in odb}_k \\
\text{For } \text{blok}_{k,m} \text{ in odb}_k \ (m > n) \\
\text{GEMM( } \text{blok}_{k,m}, \text{blok}_{k,n} ) \\
\text{EndFor} \\
\text{EndFor}
\end{align*}
\]

• Few but very large tasks
• Lots of data dependencies
• Blas parallelism
• Synchronisation
For $cblk_k$ in $cblks$

\[
\begin{align*}
\text{FACTO}( \text{diag}_k ) \\
\text{For } & \text{blok}_{k,m} \text{ in } odb_k \\
\text{TRSM}( \text{blok}_{k,m} ) \\
\text{EndFor} \\
\text{For } & \text{blok}_{k,n} \text{ in } odb_k \\
\text{For } & \text{blok}_{k,m} \text{ in } odb_k (m > n) \\
\text{GEMM}( \text{blok}_{k,m}, \text{blok}_{k,n} ) \\
\text{EndFor} \\
\text{EndFor}
\end{align*}
\]

- Medium sized tasks
- Less data dependencies
- Panel + Blas parallelism
- Less synchronization
Factorization algorithm: 3D task level

For cblk\(_k\) in cblks

\[
\begin{cases}
\text{FACT0}( \text{diag}_k ) \\
\text{For } \text{blok}_{k,m} \text{ in } \text{odb}_k \\
\quad \text{TRSM}( \text{blok}_{k,m} ) \\
\text{EndFor} \\
\text{For } \text{blok}_{k,n} \text{ in } \text{odb}_k \\
\quad \text{For } \text{blok}_{k,m} \text{ in } \text{odb}_k (m > n) \\
\quad \quad \text{GEMM}( \text{blok}_{k,m}, \text{blok}_{k,n} ) \\
\quad \text{EndFor} \\
\text{EndFor}
\end{cases}
\]

- 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

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Task distribution</th>
<th>Number of</th>
<th>Mflop/t</th>
<th>Factorization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cblks</td>
<td>bloks</td>
<td>tasks</td>
</tr>
<tr>
<td>Dynamic</td>
<td>1D</td>
<td>17 307</td>
<td>193 887</td>
<td>17 307</td>
</tr>
<tr>
<td></td>
<td>2D</td>
<td></td>
<td></td>
<td>193 887</td>
</tr>
</tbody>
</table>

- **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?

---

Task granularity using StarPU with PaStiX - Alycia Lisito
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**

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Task distribution</th>
<th>Number of</th>
<th>Mflop/t</th>
<th>Factorization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cblks</td>
<td>bloks</td>
<td>tasks</td>
</tr>
<tr>
<td>1D</td>
<td></td>
<td></td>
<td></td>
<td>17 307</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>1D / 2D</td>
<td>17 307</td>
<td>193 887</td>
<td>85 532</td>
</tr>
<tr>
<td>2D</td>
<td></td>
<td>193 887</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

12 / 19 - Task granularity using StarPU with PaStiX - Alycia Lisito
## StarPU Factorization

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

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

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Task distribution</th>
<th>Number of</th>
<th>Mflop/t</th>
<th>Factorization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cblks</td>
<td>bloks</td>
<td>tasks</td>
</tr>
<tr>
<td>1D</td>
<td></td>
<td>17 307</td>
<td>5 976.6</td>
<td>66.07</td>
</tr>
<tr>
<td>Dynamic</td>
<td>1D / 2D</td>
<td>17 307</td>
<td>193 887</td>
<td>1 414.7</td>
</tr>
<tr>
<td>2D</td>
<td></td>
<td>193 887</td>
<td>655.5</td>
<td>52.63</td>
</tr>
<tr>
<td>2D</td>
<td></td>
<td>193 887</td>
<td>655.5</td>
<td>45.89</td>
</tr>
<tr>
<td>StarPU</td>
<td>2D / 3D</td>
<td>17 307</td>
<td>193 887</td>
<td>105.5</td>
</tr>
<tr>
<td>3D</td>
<td></td>
<td>1 114 228</td>
<td>81.4</td>
<td>24.99</td>
</tr>
</tbody>
</table>

- 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

15 / 19 - Task granularity using StarPU with PaStiX - Alycia Lisito
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
- PaStiX 6 faster than PaStiX 5
- Big speedup for StarPU
StarPU Factorization

- Speedup of 2.01 to 3.94 on 4 nodes

18 / 19 - Task granularity using StarPU with PaStiX - Alycia Lisito
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!