Very Fast FEM Poisson Solvers on Lower Precision Accelerator Hardware

Dustin Ruda, Stefan Turek, Dirk Ribbrock, Peter Zajac

Institute for Applied Mathematics (LS 3), TU Dortmund University

ECCOMAS Congress 2022, 5–9 June 2022, Oslo, Norway

8 June 2022
Motivation

<table>
<thead>
<tr>
<th>Prec.</th>
<th>double</th>
<th>double + TC</th>
<th>single</th>
<th>single + TC</th>
<th>half</th>
<th>half + TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100</td>
<td>7.8</td>
<td>-</td>
<td>15.7</td>
<td>-</td>
<td>31.4</td>
<td>125</td>
</tr>
<tr>
<td>A100</td>
<td>9.7</td>
<td>19.5</td>
<td>19.5</td>
<td>156</td>
<td>-</td>
<td>312</td>
</tr>
<tr>
<td>H100</td>
<td>30</td>
<td>60</td>
<td>60</td>
<td>500</td>
<td>120</td>
<td>1,000</td>
</tr>
</tbody>
</table>

TFlop/s peak rates for **NVIDIA V100** (2017), **A100** (2020) and **H100** (Q3 2022) (similar: AMD Matrix Core)

- 100+ TFlop/s only achievable in **lower precision** by **Tensor Cores** (TC)
- Peak rates only achievable with **dense** matrix operations
- **Aim**: Profitable use of this hardware for linear systems in FE simulations (CFD)
- Consider **Poisson’s equation**: Global-in-time Navier–Stokes solver allows for solving Pressure Poisson problems for all time steps at once $\rightarrow$ many right hand sides (RHS)
- Standard FEM solvers (MG) require double precision (DP) and include large, sparse matrices $\rightarrow$ **Prehandling** and new **Schur complement-based methods**
How to handle ill-conditioned Poisson-like Problems

• **Split the error:**
  \[ u - \tilde{u}_h = (u - u_h) + (u_h - \tilde{u}_h) \]

• **Discr. Error:**
  \[ \|u - u_h\| = O(h^{p+1}) \]
  - depending on **FEM space** and smoothness
  - Here for simplicity: \(p = 1\)

• **Comp. Error:**
  \[ \|u_h - \tilde{u}_h\| \approx \text{cond} \cdot \text{“data error”} \]
  - Data error at least TOL of respective precision
  - Poisson: \(\text{cond}(A_h) = O(h^{-2})\)

• Critical \(h\) at intersection of both errors: Discr. Error \(\approx\) Comp. Error \(\Rightarrow h \approx (\text{cond} \cdot \text{TOL})^{\frac{1}{2}}\)

Source: Ruda et al, 2022
How to handle ill-conditioned Poisson-like Problems

- Critical grid width: $h \approx (\text{cond} \cdot \text{TOL})^{\frac{1}{2}}$
- Poisson $-\Delta u = f$: Substitute $\text{cond} = O(h^{-2}) \Rightarrow h \approx \text{TOL}^{\frac{1}{4}}$
- 1D example: $(\text{TOL}_{SP})^{\frac{1}{4}} = 2^{-5.75} \quad (\text{TOL}_{DP})^{\frac{1}{4}} = 2^{-13}$
- Wish: $\text{cond} = O(1) \Rightarrow h \approx \text{TOL}^{\frac{1}{2}} \rightarrow \text{SP (and even HP?)}$ possible

$L^2$-error with standard FEM in 1D, $h = 2^{-\text{level}}$

Source: Ruda et al., 2022
Concept of Prehandling of Linear Systems

Preconditioning: \[ x^{k+1} = x^k - C^{-1}(Ax^k - b) \]

Prehandling: \[ x^{k+1} = x^k - (C^{-1}Ax^k - C^{-1}b) = x^k - (\tilde{A}x^k - \tilde{b}) \]

- Yields same solution (if converged and with “infinite precision”) and same iteration numbers, but \( \text{cond}(A) \leq \text{cond}(\tilde{A}) \) since different linear systems
- **Central idea:** Explicitly transforming \( Ax = b \) into equivalent \( \tilde{A}\tilde{x} = \tilde{b} \), \( B\tilde{x} = x \) with:
  1) \( \text{cond}(\tilde{A}) \ll \text{cond}(A) \)
  2) \( \tilde{A} \) only moderately less sparse than \( A \)
  3) Transformation to \( \tilde{A}, \tilde{b} \) (resp. \( x \) via \( B \)) fast (i.e. \( O(N \log N) \))
HFEM: Ideas, Realization & Properties

- Only candidate for prehandling so far: HFEM
- **Idea:** Use of *hierarchical* instead of *nodal basis* starting from a coarse grid
- Transform linear system $\tilde{A} = S^T A S$, $\tilde{b} = S^T b$, $x = S \tilde{x}$

$$\text{cond} \left( \tilde{A} \right) = O \left( \left( \log \frac{1}{h} \right)^2 \right) \text{ in 1D, 2D; FEM: } \text{cond}(A) = O \left( \left( \frac{1}{h} \right)^2 \right)$$

- Add. partial Cholesky prehandling: $$\begin{pmatrix} \tilde{A}_0 & 0 \\ 0 & \tilde{D} \end{pmatrix} = L^T L \rightarrow L^{-1} \tilde{A} L^{-T}$$
- Remark: in 3D $\text{cond} \left( \tilde{A} \right) = O \left( \frac{1}{h} \log \frac{1}{h} \right)$ resp. $O \left( \frac{1}{h} \right) \rightarrow$ Possible in SP

Source: Deuflhard et al., 1989
HFEM: Numerical results (errors)

$L^2$-errors for different levels in 2D in DP, SP, HP without (left) and with (right) prehandling via HFEM for “smooth” solution. Source: Ruda et al., 2020
• Fine $h$ for tolerance of $\approx 1\%$ for complex problems
  → large problems
  → requires HPC

$L^2$-errors for different levels in 2D in DP, SP, HP without and with **prehandling via HFEM** for **strongly oscillating** solution

Source: Ruda et al., 2022
• **Objective:** Construct solver consisting as much as possible on multiplications with dense, well-conditioned matrices

• **Starting Point:** Linear system after prehandling via HFEM+Cholesky $Ax = b$

• Subdivide nodes into 3 types ($C, E, I$) and renumber $A$ accordingly

Source: Ruda et al., 2022
Direct SC Methods

Idea

\[
\begin{pmatrix}
I & B & 0 \\
B^T & E & D \\
0 & D^T & C
\end{pmatrix}
\begin{pmatrix}
\chi_C \\
\chi_E \\
\chi_I
\end{pmatrix}
=
\begin{pmatrix}
b_C \\
b_E \\
b_I
\end{pmatrix}
\]

- \(D, E\) are sparse
- \(B\) is moderately dense
- \(C\) decomposes into \textit{independent} blocks (as many as macro cells)

- Blocks \(C_i\) of \(C\) are equal if they correspond to similar macro cells
- Only \(C\) grows like \(N (= \#\text{Dof})\)
- Applying \textbf{Schur complement} \(\rightarrow\) semi-iterative method
- Applying further Schur complement \(\rightarrow\) completely direct method

Source: Ruda et al., 2022
### Semi-iterative Method

\[ \Lambda = E - DC^{-1}D^T \]

Use conjugate gradient method to solve

\[
\begin{pmatrix}
I & B \\
B^T & \Lambda
\end{pmatrix}
\begin{pmatrix}
x_C \\
x_\varepsilon
\end{pmatrix}
= 
\begin{pmatrix}
b_C \\
b_\varepsilon - DC^{-1}b_I
\end{pmatrix}
\]

\[ x_\Sigma = \Sigma^{-1} \left( b_\Sigma - D^T x_\varepsilon \right) \]

- Matrices \( \Sigma, \Lambda, \Pi, C \) well-cond. (5–50 on unit square with Q1)
- Block structure of \( C \): **Only \( C_i^{-1} \) computed and stored**
- Semi-iterative: Less storage consuming
- Direct: More storage consuming but even higher potential for TC, especially in case of many RHS

### Direct Method

\[ \Lambda = E - DC^{-1}D^T \]

\[ \Pi = \Lambda - B^TB \]

\[ x_\varepsilon = \Pi^{-1} \left( b_\varepsilon - B^T b_C - DC^{-1}b_I \right) \]

\[ x_C = b_C - B x_\varepsilon \]

\[ x_\Sigma = C^{-1} \left( b_\Sigma - D^T x_\varepsilon \right) \]
Multiplication with $C^{-1}$

- Both methods require 2 multiplications with $C^{-1}$
- Efficient implementation by transforming into dense matrix product (also if $\#\text{RHS} = 1$):

  \[
  \mathcal{O}(N^{3/2}) \text{ but fast calculation by TC}
  \]
**Storage and Computational Cost of the Direct Method**

- Consider equidistantly refined unit square, \( Q_1 \)
- Let \( h_0 = 2^\ell \sqrt{h} \), \( \ell = \ldots, -1, 0, 1, 2, \ldots \)
- Relevant for storage: \( \Pi^{-1}, C_i^{-1} \)
- Relevant for FLOP: \( \Pi^{-1}, C^{-1}(2\times) \)

\[
\Lambda = E - DC^{-1}D^T, \quad \Pi = \Lambda - B^T B
\]

\[
x_E = \Pi^{-1} \left( b_E - B^T b_C - DC^{-1} b_I \right)
\]

\[
x_C = b_C - Bx_E
\]

\[
x_I = C^{-1} \left( b_I - D^T x_E \right)
\]

\[
h = \frac{1}{1024}:
\]

<table>
<thead>
<tr>
<th>( \ell )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total NNZ/( N )</td>
<td>16,400</td>
<td>14,100</td>
<td>1,000</td>
<td>500</td>
<td>4,200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \ell )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>#FLOP/( N^{3/2} )</td>
<td>33</td>
<td>12</td>
<td>18</td>
<td>65</td>
<td>256</td>
</tr>
</tbody>
</table>

- Best choice in terms of complexity: \( h_0 = \sqrt{h} \) (or \( \ell = 1 \) considering storage)
- \( \approx 12N^{3/2} \rightarrow 12,000 \) (SC) vs. \( 1,000 \) FLOP (MG) for \( h = \frac{1}{1024} \)
Direct SC Methods  Numerical Results

Direct Method: Unit Square on A100

GFLOP/s (left) and MDof/s (right) with direct method on A100 with one and many RHS depending on $h^{-1}$ in DP, SP and HP (left, middle and right 3 columns, respectively)

→ Up to 60 TFlop/s (for problems with many RHS)

→ More arithmetic work ($\times 12$), but still much faster than standard MG solver on x64 AMD CPU (8 MDof/s for many rhs)
Unstructured Coarse Grids

- So far: Analysis and numerical tests on unit square
- **Direct method** also applicable to “arbitrary” P1 grids
- Coarse grids with many similar cells are advantageous → few different $C_i$
- Example “flow around a square”
  - 3 Groups of similar cells → $C_1^{-1}, C_2^{-1}, C_3^{-1}$ must be calculated and stored

<table>
<thead>
<tr>
<th>$L$ ($L_0$)</th>
<th>#FLOP $N^{3/2}$</th>
<th>NNZ $N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (1)</td>
<td>10.5</td>
<td>410</td>
</tr>
<tr>
<td>7 (2)</td>
<td>10.9</td>
<td>1,750</td>
</tr>
<tr>
<td>8 (2)</td>
<td>11.6</td>
<td>1,810</td>
</tr>
</tbody>
</table>

Dustin Ruda
Direct Method: Flow around a Square on A100

GFLOP/s (left) and MDof/s (right) with direct method on A100 with one and many RHS depending on $h^{-1}$ in DP, SP and HP (left, middle and right 3 columns, respectively)
Limitations of the Direct Method

- High storage requirement of $\mathcal{O}\left(N^{\frac{3}{2}}\right)$ due to $\Pi^{-1}$
- Limit of fine grid width in our tests: $h = \frac{1}{1024}$ (on one GPU)
- Hardly applicable to 3D because storage requirement of $\mathcal{O}\left(N^{\frac{5}{3}}\right)$
- Requirement for simple form of the direct method: No coupling between nodes in $\mathcal{C}$ and $\mathcal{I}$ (coarse grid and interior nodes)
  - Rectangular Q1 grids
  - “arbitrary” P1 grids
- Less memory consuming, more versatile but also less performant variant: Semi-iterative method
Semi-iterative Method: Storage Requirement

<table>
<thead>
<tr>
<th>$\frac{1}{h}$</th>
<th>$\frac{N}{10^6}$</th>
<th>$\frac{1}{h_0}$</th>
<th>$\Sigma$</th>
<th>$C_i^{-1}$</th>
<th>$D$</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1.05</td>
<td>16</td>
<td>15</td>
<td>15.1</td>
<td>1.0</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>25</td>
<td>0.9</td>
<td>1.6</td>
<td>27</td>
</tr>
<tr>
<td>2048</td>
<td>4.19</td>
<td>32</td>
<td>19</td>
<td>3.8</td>
<td>1.0</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>40</td>
<td>0.2</td>
<td>1.6</td>
<td>42</td>
</tr>
<tr>
<td>4096</td>
<td>16.77</td>
<td>32</td>
<td>16</td>
<td>15.5</td>
<td>0.7</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>27</td>
<td>0.9</td>
<td>1.0</td>
<td>29</td>
</tr>
</tbody>
</table>

Number of nonzero entries relative to $N$

- Storage requirement: $30N - 40N$ in SP (in comparison: $9N$ with standard FEM in DP)
Semi-iterative Method: Performance Estimate

- **Basic composition of the method:**
  - $1 \times D$ and $1 \times C^{-1}$ to compute RHS
  - **Iterative step:** $1 \times \Sigma$, 3 AXPY and 2 dot products per iteration
    $\rightarrow \#\text{iter} \left[ 2\text{NNZ}(\Sigma) + 6\#\text{rows} + 4\#\text{rows} \right]$
    $\#\text{rows} = O \left( N^3 \right)$
  - Intermediate step: $1 \times D^T$
  - **Direct step:** $1 \times C^{-1}$

- Entire method in SP on A100
- Majority of the work: Dense matrix operations; Small part: sparse $\times$ dense and BLAS1

\[
\Lambda = E - DC^{-1}D^T
\]

Use conjugate gradient method to solve

\[
\begin{pmatrix}
I & B \\
B^T & \Lambda
\end{pmatrix}
\begin{pmatrix}
x_C \\
x_\Sigma
\end{pmatrix} =
\begin{pmatrix}
b_C \\
b_\Sigma - DC^{-1}b_I
\end{pmatrix}
\]

\[
x_I = C^{-1}\left( b_I - D^Tx_\Sigma \right)
\]
Semi-iterative Method: Performance Estimate

<table>
<thead>
<tr>
<th>$\Sigma \ast$ dense</th>
<th>AXPY</th>
<th>dot product</th>
<th>dense $\ast$ dense</th>
<th>$D \ast$ dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,375</td>
<td>227</td>
<td>321</td>
<td>150,000</td>
<td>1,200</td>
</tr>
</tbody>
</table>

GFlop/s in benchmarks on A100 in SP

<table>
<thead>
<tr>
<th>$\frac{1}{h}$</th>
<th>$\frac{1}{h_0}$</th>
<th>#iter</th>
<th>FLOP/N it.</th>
<th>FLOP/N dir.</th>
<th>time it.+dir.</th>
<th>GFLOP/s</th>
<th>MDof/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>16</td>
<td>30</td>
<td>913</td>
<td>15,400</td>
<td>0.43 + 0.11</td>
<td>31,445</td>
<td>1,926</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>24</td>
<td>1,217</td>
<td>3,615</td>
<td>0.60 + 0.03</td>
<td>8,206</td>
<td>1,698</td>
</tr>
<tr>
<td>2048</td>
<td>32</td>
<td>28</td>
<td>1,085</td>
<td>15,400</td>
<td>2.04 + 0.43</td>
<td>27,919</td>
<td>1,694</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>23</td>
<td>1,881</td>
<td>3,611</td>
<td>3.53 + 0.10</td>
<td>6,333</td>
<td>1,153</td>
</tr>
<tr>
<td>4096</td>
<td>32</td>
<td>31</td>
<td>1,011</td>
<td>63,543</td>
<td>7.43 + 7.10</td>
<td>74,476</td>
<td>1,154</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>25</td>
<td>1,374</td>
<td>15,391</td>
<td>10.17 + 1.72</td>
<td>23,636</td>
<td>1,410</td>
</tr>
</tbody>
</table>

Performance estimate
Semi-iterative Method: Accuracy

Relative error with semi-iterative method in DP and SP for differently smooth solutions
Comparison of **MDof/s** for many RHS with **MG** in **DP** on **AMD CPU**, **direct** method in **HP** on **A100** and **semi-iterative** in **SP** on **A100** (estimate)
Outlook and Conclusion

- Implementation of the semi-iterative method on GPU
- Prehandling in 3D
- Analysis of suitable preconditioners for the iterative step and initial guesses for the solution vector to reduce number of iterations
- Testing semi-iterative Method for other FE spaces and in 3D
- Implementation into FEATFLOW software

**Conclusion:** It is possible to exploit Lower-Precision Accelerator Hardware for PDE computing (under certain conditions)
References


