Recursion in High-Performance Matrix Computations

Isak Jonsson

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

isak@cs.umu.se

Based on material from Bo Kågström, bokg@cs.umu.se

NGSSC: Lectures 17-18
June 10, 2003
Acknowledgements

Joint project with IBM T.J. Watson Research Center.

People involved in the work presented:

- Erik Elmroth, Umeå University
- Fred Gustavson, IBM T.J. Watson Research Center
- Isak Jonsson, Umeå University
- Bo Kågström, Umeå University

Earlier contributions from

- Andre Henriksson, Umeå University
- Per Ling, Umeå University
- Charles Van Loan, Cornell University

We also collaborate with UNI·C, DTU, Lyngby, Denmark.
Outline

Overview of recent progress.

- Motivation and background – managing deep memory hierarchies
- Recursive blocked algorithms (GEMM) and data structures
- Packed Cholesky factorization
- Triangular Sylvester-type matrix equations
- QR factorization and the linear least squares problem
- Conclusions

Performance results throughout the presentation.
MANAGEMENT OF
DEEP MEMORY HIERARCHIES

ARCHITECTURE EVOLUTION: HPC systems with multiple SMP nodes, several levels of caches and more functional units per CPU.

KEY TO PERFORMANCE: Understand the algorithm and architecture interaction.

GOAL: Maintain 2-dim data locality at every level of the 1-dim tiered memory structure.

- Hierarchical blocking.
- Matching an algorithm and its data structure.
**Blocking for Deep Memory Hierarchies**

1. **Explicit multi-level blocking**
   - Each loop set matches a specific level of the memory hierarchy.
   - Deep knowledge of architecture characteristics needed.
   - Needs a blocking parameter for each level.
   - Two-level blocked matrix multiply (tuned for L1 and L2 cache).

2. **Automatic blocking via recursion**
   - **Recursion**: key concept for matching an algorithm and its data structure.
   - Recursive algorithms – divide and conquer style.
   - Automatic **Hierarchical Blocking** – variable and “squarish”.
   - Only tuning parameter is L1 cache.
Recursive Blocked Algorithms

GEMM: $C \leftarrow \beta C + \alpha A \cdot B$

We can split the blocked matrix multiply and add in any of the three matrix dimensions: $M$, $N$, and $K$.

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} +
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
= \begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} +
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
= \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

Always split the largest dimension: problem is kept “squarish”.

Ratio between #flops made on subblocks and #subblocks is maintained as high as possible.
**Recursive Blocked GEMM**

If $M \leq \Gamma$ and $N \leq \Gamma$ and $K \leq \Gamma$

solve problem using optimized GEMM kernel.

Otherwise,

If $M = \max(M, N, K)$,

two recursive calls: $M' = M/2, N' = N, K' = K$.

else, if $N = \max(N, K)$,

two recursive calls: $M' = M, N' = N/2, K' = K$.

else,

two recursive calls: $M' = M, N' = N, K' = K/2$.

The split ordering is not so crucial when 2 or 3 of the dimensions are equal. Less than optimal ordering will only impair performance in one or two levels of the recursion.

Why $\Gamma > 1$?

For very small problems, the overhead for the recursion becomes too expensive – use **optimized kernel**. “Conquer” part is trivial: addition of the results are made implicitly by the leaf kernels.

The recursive technique *automatically blocks* for every level of cache.

**This gives temporal locality!**
Recursive Blocked Data Formats

Objective: Match the recursive algorithm with a recursively blocked data format.

- A new set of data formats for storing block-partitioned matrices.
- Hybrid of two addressing techniques.
- At block level each submatrix is stored in standard column major (or row-major) order.
- Block size constrained so that a few of them fit in L1 cache.
- Blocks are stored in two recursive matrix formats: Rectangle and Isosceles triangle.

Recursive blocked formats allow for maintaining the 2-dim data locality at every level of the 1-dim layered memory structure. Claim!

Block row and block column orderings only maintain data locality at a submatrix level.

This gives spatial locality!
Rectangular Recursive Blocked Data Formats

- Always split the largest dimension of rectangular submatrix.
- If the dimensions are equal, we have two choices: split rows (RBR), or split columns (RBC).
- Odd number of rows: middle row assigned to block at the bottom.
- Odd number of columns: middle column assigned to block to the right.
  Blocks to the right or at the bottom may contain submatrices that are not entirely filled. Strategy “minimizes” the difference in number of used elements between the two blocks after the splitting.
- The mutual ordering of the blocks are assigned recursively.

**Example:** A of size $496 \times 380$ and $mb = nb = 100$, giving $m = 5$ and $n = 4$. Assign the numbers 0–19 (contiguous blocks in memory) to the blocks of $A$. 
**Rectangular RBR and RBC Formats**

**RBR**: First split is horizontal \( m = 5 > n = 4 \). Assign 0–7 to upper part and 8–19 to lower part.

Since \#block-rows in upper part \( 2 \times 2 < n = 4 \), second split is vertical, assigning 0–3 to the left hand side. This submatrix is square so next split is by row. Etc.

\[
\begin{array}{cccc}
A_{11} & A_{12} & A_{13} & A_{14} & 0 & 5 & 10 & 15 \\
A_{21} & A_{22} & A_{23} & A_{24} & 1 & 6 & 11 & 16 \\
A_{31} & A_{32} & A_{33} & A_{34} & 2 & 7 & 12 & 17 \\
A_{41} & A_{42} & A_{43} & A_{44} & \sim & 3 & 8 & 13 & 18 & \sim \\
A_{51} & A_{52} & A_{53} & A_{54} & 4 & 9 & 14 & 19 \\
\end{array}
\]

Block matrix

\[
\begin{array}{cccc}
0 & 1 & 4 & 5 \\
2 & 3 & 6 & 7 \\
8 & 9 & 14 & 15 \\
\sim & 10 & 11 & 16 & 17 & \sim \\
12 & 13 & 18 & 19 \\
\end{array}
\]

Recursive block split row (RBR)

\[
\begin{array}{cccc}
0 & 2 & 4 & 6 \\
1 & 3 & 5 & 7 \\
8 & 9 & 14 & 15 \\
\sim & 10 & 12 & 16 & 18 \\
11 & 13 & 17 & 19 \\
\end{array}
\]

Recursive block split column (RBC)

**This gives spatial locality!**
**Multi-level vs. Recursive Blocking**

Uniprocessor Performance–IBM PowerPC 604, 112 MHz

Optimized GEMM Kernel with Level 3 Prefetching: Technique which enables data to be brought to registers and cache ahead of its use, so when it is needed, it is immediately available. Embedded in the level 3 kernel; during a subblock GEMM computation, the next set of subblocks are prefetched.
SMP Parallelization Using Threads

Dynamic distribution of tasks – good load balancing on a non-dedicated SMP. A virtual recursion tree is maintained throughout the execution, which is divided into subtrees. Different processes or threads execute on different subtrees.

Performance of Recursive GEMM

DGEMM on a non-dedicated 4-processor PowerPC 604 node, 112 MHz.
Level 3 BLAS

References


Also implicit in other references!
Packed Cholesky Factorization

Current approach (typified by LAPACK):

- Cannot use high performance level 3 BLAS routines (e.g. \texttt{DGEMM}) due to packed storage.
  
  – Possible to produce packed level 3 BLAS routines at a great programming cost.

- Run at level 2 performance, i.e., much below full storage routines.

- Uses minimum storage $= \frac{1}{2} n(n + 1)$ elements.

Recursive algorithm + Recursive packed data layout:

- Make use of high performance level 3 kernel routines (e.g. \texttt{DGEMM}).

- Runs at level 3 performance – at least!

- Used storage for matrix $A = \frac{1}{2} n(n + 1)$ elements.

- Required temporary workspace $= \frac{1}{8} n^2$ elements (25%).
Recursive Cholesky Factorization

\[ A = \begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = LL^T = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22} \end{pmatrix} \] (1)

**Factor**: \( A_{11} = L_{11} L_{11}^T \). (2)

**TRSM**: \( L_{21} L_{11}^T = A_{21} \). (3)

**SYRK**: \( \tilde{A}_{22} = A_{22} - L_{21} L_{21}^T \) (4)

**Factor**: \( \tilde{A}_{22} = L_{22} L_{22}^T \) (5)
If we break Equation (6) into its component pieces we get

\[ X A^T = B \quad \text{or} \quad m \left\{ \begin{pmatrix} k_1 X_1 & k_2 X_2 \end{pmatrix} \begin{pmatrix} A_{11}^T & A_{21}^T \\ 0 & A_{22}^T \end{pmatrix} \right\} = m \left\{ \begin{pmatrix} k_1 \tilde{B}_1 & k_2 \tilde{B}_2 \end{pmatrix} \right\} = B \]

\[ \text{TRSM} : X_1 A_{11}^T = B_1 \quad \text{TRSM} : X_2 A_{22}^T = \tilde{B}_2 \]

\[ \text{GEMM} : \tilde{B}_2 = B_2 - X_1 A_{21}^T \]

\[ \text{GEMM} : \tilde{B}_2 = B_2 - X_1 A_{21}^T \]

**Similar formulation for SYRK**
Recursive packed format

1. Divide triangle into two isosceles triangles, $T_1$ and $T_2$ and a rectangle $S$.
2. Divide triangles recursively, down to element level.
3. Store in order $T_1$, $S$, $T_2$.

Same memory requirement! ($= \frac{1}{2} n(n + 1)$)

Reorganization to recursive format is an $O(n^2)$ algorithm, requiring a temporary buffer of approx. $\frac{1}{8}n^2$ elements.
Recursive packed format

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>4</th>
<th>7</th>
<th>11</th>
<th>16</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>15</td>
<td>19</td>
<td>10</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>16</td>
<td>20</td>
<td>23</td>
<td>15</td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>17</td>
<td>21</td>
<td>24</td>
<td>26</td>
<td>21</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>7</td>
<td>18</td>
<td>25</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>

Packed lower

<table>
<thead>
<tr>
<th>1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>7</th>
<th>10</th>
<th>13</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>11</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>15</td>
<td>19</td>
<td></td>
<td></td>
<td>19</td>
<td>20</td>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>21</td>
<td></td>
<td>21</td>
<td>23</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>17</td>
<td>22</td>
<td>24</td>
<td>26</td>
<td></td>
<td>26</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>18</td>
<td>23</td>
<td>25</td>
<td>27</td>
<td>28</td>
<td></td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

Packed recursive col. lower

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th></th>
<th></th>
<th></th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>19</td>
<td></td>
<td></td>
<td>19</td>
<td>20</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>20</td>
<td>21</td>
<td></td>
<td>21</td>
<td>24</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>22</td>
<td>23</td>
<td>26</td>
<td></td>
<td>26</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>24</td>
<td>25</td>
<td>27</td>
<td>28</td>
<td></td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

Packed recursive row lower

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>7</th>
<th>10</th>
<th>13</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>11</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>15</td>
<td>19</td>
<td>19</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>21</td>
<td>21</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>17</td>
<td>22</td>
<td>24</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>18</td>
<td>23</td>
<td>25</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>

Packed recursive col. upper

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>19</td>
<td>19</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>20</td>
<td>21</td>
<td>21</td>
<td>24</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>22</td>
<td>23</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>24</td>
<td>25</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>

Packed recursive row upper

Can these properties be exploited?
GEMM operation in the recursive TRSM formulation:

\[ \tilde{B}_2 = B_2 - X_1 A_{21}^T \]

Here we identify a bad memory reference pattern—both \( X_1 \) and \( A_{21}^T \) referenced by stride \( = n \).

**Cure:**

- Transpose the equation:

\[ \tilde{B}_2^T = B_2^T - A_{21} \cdot X_1^T. \]

- Store matrix \( L \) in packed recursive row lower storage \( \Rightarrow \) can use a \texttt{DGEMM('T', 'N')} call with stride \( = 1 \).
Pruning enables High Performance

Why pruning the recursion tree?

- Overhead of recursion is very large for small problems
  - Serious performance degradation.

Kernel Routine characteristics

- Optimized for superscalar processors.
- Operates on recursively stored data.
- Elements accessed in “non-linear” storage:
  - Addressing maps—constructed before factorization starts.
  - Copy buffers used for large rectangular operands in TRSM and SYRK.
- Only tuning parameter: L1 cache.
UNIPROCESSOR PERFORMANCE

Factorization on IBM POWER3

- BC, without data transformation
- BC, including time for data transformation
- LAPACK DPOTRF
- LAPACK DPPTRF

Mflops/s vs. N

N = 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
Notice: The flops of Algorithm $BC$ follows a geometric progression. Hence, the $C$-matrix of DGEMM can easily be broken into large pieces for SMP-DGEMM to work on.
Packed Cholesky Factorization

References


Sylvester-Type Matrix Equations

Appear in control theory applications: stability problems, model reduction, balancing, $H_\infty$ control.

- **Sylvester**: $AX - XB = C$, $A$, $B$ and $C$ general.
- **Lyapunov**: $AX + XA^T = C$.
- **Stein (or discrete Lyapunov)**: $AX^T + X = C$, $A$ general, $C = CT$ (semi)definite.
- **Generalized (coupled) Sylvester**: \[
\begin{aligned}
AX - YB &= C \\
DX - YE &= F
\end{aligned}
\]
- **Generalized Sylvester**: $AXB^T - CXD^T = F$.

Second major step in their solution is to solve a **TRIANGULAR MATRIX EQUATION**.

Our blocked recursive technique works for all! Here

- **TRIANGULAR STANDARD** and
- **TRIANGULAR GENERALIZED Sylvester equations**

They also appear naturally in estimating condition numbers of matrix equations and different eigenspace problems [5, 3, 4].
Block diagonalization and Spectral projectors

\[ S = \begin{bmatrix} A & -C \\ 0 & B \end{bmatrix}, \quad \text{in Schur form.} \]

- \( S \) block diagonalized by \textbf{similarity} transformation:
  \[
  \begin{bmatrix}
    I_m & -R \\
    0 & I_n
  \end{bmatrix}
  S
  \begin{bmatrix}
    I_m & R \\
    0 & I_n
  \end{bmatrix}
  =
  \begin{bmatrix}
    A & 0 \\
    0 & B
  \end{bmatrix},
  \]

  where \( R \) satisfies \( AR - RB = C \).

- \textbf{Spectral projector} associated with \((1, 1)\)-block \( A \):
  \[
  P = \begin{bmatrix}
    I_m & R \\
    0 & 0
  \end{bmatrix}
  \]

  Important quantity in error bounds for invariant subspaces and clusters of eigenvalues.

- Large \( \|P\|_2 = (1 + \|R\|_2^2)^{1/2} \), signals \textbf{ILL-CONDITIONING}.

- Computed estimate: \( s = 1/\|P\|_F \)
**Standard Sylvester Eq.–Application 2:**

Separation of two matrices

\[ \text{Sep}[A, B] = \inf_{\|X\|_F = 1} \|AX - XB\|_F = \sigma_{\text{min}}(Z), \]

where \( Z = I_n \otimes A - B^T \otimes I_m. \)

- \( \text{Sep}[A, B] = 0 \) if and only if \( A \) and \( B \) have a **common** eigenvalue.
- \( \text{Sep}[A, B] \) is **small** if there is small perturbation of \( A \) or \( B \) that makes them have a common eigenvalue.
- General case: \( \text{Sep} \) may be much smaller than min. distance between the eigenvalues of \( A \) and \( B \).

- **COMPUTING** \( \sigma_{\text{min}}(Z) \): \( O(m^3n^3) \) operation. Impractical!

- Reliable Sep **estimates** of cost \( O(mn^2 + m^2n) \):

\[
\frac{\|x\|_2}{\|y\|_2} = \frac{\|X\|_F}{\|C\|_F} \leq \|Z^{-1}\|_2 = \frac{1}{\sigma_{\text{min}}(Z)} = \text{Sep}^{-1},
\]

and

\[
(mn)^{-1/2}\|Z^{-1}\|_1 \leq \|Z^{-1}\|_2 \leq \sqrt{mn}\|Z^{-1}\|_1.
\]
Recursive Triangular Sylvester Solvers

\[ \text{op}(A) \cdot X \pm X \cdot \text{op}(B) = \beta \cdot C, \quad C \leftarrow X \ (M \times N) \], where
\[ A(M \times M) \text{ and } B(N \times N) \text{ upper quasi-triangular.} \]

\[ \text{trans}A = 'N', \ \text{trans}B = 'N', \ \text{sign} = -, \ \beta = 1: \]

Case 1 \(1 \leq N \leq M/2\): Split \(A\) and \(C\) (by rows)

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{22}
\end{bmatrix}
\begin{bmatrix}
X_1 \\
X_2
\end{bmatrix} - \begin{bmatrix}
X_1 \\
X_2
\end{bmatrix} B = \begin{bmatrix}
C_1 \\
C_2
\end{bmatrix}
\]

\[ A_{11}X_1 - X_1B = C_1 - A_{12}X_2 \]
\[ A_{22}X_2 - X_2B = C_2 \]

1. SYLV('N', 'N', \(A_{22}\), \(B\), \(C_2\))
2. GEMM('N', 'N', \(\alpha = -1\), \(A_{12}\), \(C_2\), \(C_1\))
3. SYLV('N', 'N', \(A_{11}\), \(B\), \(C_1\))

Case 2 \(1 \leq M \leq N/2\): Split \(B\) and \(C\) (by columns)

\[
A \begin{bmatrix}
X_1 \\
X_2
\end{bmatrix} - \begin{bmatrix}
X_1 \\
X_2
\end{bmatrix} \begin{bmatrix}
B_{11} & B_{12} \\
B_{22}
\end{bmatrix} = \begin{bmatrix}
C_1 \\
C_2
\end{bmatrix}
\]

\[ AX_1 - X_1B_{11} = C_1 \]
\[ AX_2 - X_2B_{22} = C_2 + X_1B_{12} \]
Case 3 ($N/2 < M < 2N$): Split $A$, $B$ and $C$

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{22}
\end{bmatrix}
\begin{bmatrix}
X_{11} & X_{12} \\
X_{21} & X_{22}
\end{bmatrix}
- 
\begin{bmatrix}
X_{11} & X_{12} \\
X_{21} & X_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{22}
\end{bmatrix}
= 
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
\]

\[
A_{11}X_{11} - X_{11}B_{11} = C_{11} - A_{12}X_{21}
\]
\[
A_{11}X_{12} - X_{12}B_{22} = C_{12} - A_{12}X_{22} + X_{11}B_{12}
\]
\[
A_{22}X_{21} - X_{21}B_{11} = C_{21}
\]
\[
A_{22}X_{22} - X_{22}B_{22} = C_{22} + X_{21}B_{12}
\]

1. SYLV('N', 'N', $A_{22}$, $B_{11}$, $C_{21}$)
2a. GEMM('N', 'N', $\alpha = +1$, $C_{21}$, $B_{12}$, $C_{22}$)
2b. GEMM('N', 'N', $\alpha = -1$, $A_{12}$, $C_{21}$, $C_{11}$)
3a. SYLV('N', 'N', $A_{22}$, $B_{22}$, $C_{22}$)
3b. SYLV('N', 'N', $A_{11}$, $B_{11}$, $C_{11}$)
4. GEMM('N', 'N', $\alpha = -1$, $A_{12}$, $C_{22}$, $C_{12}$)
5. GEMM('N', 'N', $\alpha = +1$, $C_{11}$, $B_{12}$, $C_{12}$)
6. SYLV('N', 'N', $A_{11}$, $B_{22}$, $C_{12}$)

Operations 2a, 2b can be executed in parallel, as well as Operations 3a, 3b.
IMPLEMENTATION ISSUES

Two alternatives for doing the recursive splits:

1. Always split the largest dimension in two (Cases 1 and 2).
2. Split both dimensions simultaneously (Case 3) when the dimensions are within a factor 2 from each other.

2. \implies a shorter but wider recursion tree, which offers more “parallel tasks”.

SYLVESTER KERNEL

For problems smaller than the block size, the dimension is split in two until the problem is of size $2 \times 2 - 4 \times 4$, when subsystems are solved using

\[(I_n \otimes A - B^T \otimes I_m)\text{vec}(X) = \text{vec}(C)\]

The system is permuted in order to make the problem more upper triangular and solved (LU with row pivoting).

The Sylvester kernel is solved using unrolled code with procedure inlining!
Performance Standard Sylvester

Results for DTRSYL-variants on IBM 604e, 332 MHz
Recursive Generalized Sylvester Solvers

\[
AX - YB = \beta C, \quad C \leftarrow X(M \times N)
\]
\[
DX - YE = \beta F, \quad F \leftarrow Y(M \times N)
\]

\((A, D)\) and \((B, E)\) in generalized Schur form with \(A, B\) quasi-triangular and \(D, E\) triangular.

Case 1 \((1 \leq N \leq M/2)\): Split \((A, D)\) and \((C, F)\) (by rows)

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{22}
\end{bmatrix}
\begin{bmatrix}
X_1 \\
X_2
\end{bmatrix}
- \begin{bmatrix}
Y_1 \\
Y_2
\end{bmatrix}
B = \begin{bmatrix}
C_1 \\
C_2
\end{bmatrix}
\]

\[
A_{11}X_1 - Y_1B = C_1 - A_{12}X_2
\]
\[
A_{22}X_2 - Y_2B = C_2
\]

\[
\begin{bmatrix}
D_{11} & D_{12} \\
D_{22}
\end{bmatrix}
\begin{bmatrix}
X_1 \\
X_2
\end{bmatrix}
- \begin{bmatrix}
Y_1 \\
Y_2
\end{bmatrix}
E = \begin{bmatrix}
F_1 \\
F_2
\end{bmatrix}
\]

\[
D_{11}X_1 - Y_1E = F_1 - D_{12}X_2
\]
\[
D_{22}X_2 - Y_2E = F_2
\]

1. TGSYL(’N’, \(A_{22}\), \(B\), \(C_2\), \(D_{22}\), \(E\), \(F_2\))
2. GEMM(’N’, ’N’, \(\alpha = -1\), \(A_{12}\), \(C_2\), \(C_1\))
3. GEMM(’N’, ’N’, \(\alpha = -1\), \(D_{12}\), \(C_2\), \(F_1\))
4. TGSYL(’N’, \(A_{11}\), \(B\), \(C_1\), \(D_{11}\), \(E\), \(F_1\))
Recursive Generalized Sylvester Solvers

\[ AX - YB = \beta C, \quad C \leftarrow X(M \times N) \]
\[ DX - YE = \beta F, \quad F \leftarrow Y(M \times N) \]

Case 2 (1 ≤ M ≤ N/2): Split (B, E) and (C, F) (by cols)

\[
\begin{align*}
A \begin{bmatrix} X_1 & X_2 \end{bmatrix} & - \begin{bmatrix} Y_1 & Y_2 \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{22} \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \end{bmatrix} & AX_1 - Y_1B_{11} &= C_1 \\
D \begin{bmatrix} X_1 & X_2 \end{bmatrix} & - \begin{bmatrix} Y_1 & Y_2 \end{bmatrix} \begin{bmatrix} E_{11} & E_{12} \\ E_{22} \end{bmatrix} = \begin{bmatrix} F_1 & F_2 \end{bmatrix} & AX_2 - Y_2B_{22} &= C_2 + Y_1B_{12} \\
& & DX_1 - Y_1E_{11} &= F_1 \\
& & DX_2 - Y_2E_{22} &= F_2 + Y_1E_{12}
\end{align*}
\]

1. TGSYL(’N’, A, B_{11}, C_1, D, E_{11}, F_1)
2. GEMM(’N’, ’N’, \alpha = 1, F_1, B_{12}, C_2)
3. GEMM(’N’, ’N’, \alpha = 1, F_1, D_{12}, F_2)
4. TGSYL(’N’, A, B_{22}, C_2, D, E_{22}, F_2)
IMPLEMENTATION ISSUES

As for the standard case it is also possible to split both dimensions simultaneously (Case 3).

**Recursive Kernel:** Recursion is done down to $2 \times 2 - 4 \times 4$ blocks.

**Recursive SS-PP:** Small generalized Sylvester equations are solved by LU and row pivoting on the Kronecker product representation [5]. This kernel is then tuned for superscalar architectures. If kernel detects near-singularity, the kernel aborts and tries to solve the problem using complete pivoting and scaling.

Recursive **CP-S** outperforms the LAPACK DTGSYL subroutine [4] for the problem sizes tested.

DTGSYL implements a blocked algorithm, and it is necessary to carefully tune the algorithm with the correct blocking parameters (as with all LAPACK routines).

This is not necessary with the recursive algorithms!

Complete library available at [http://www.cs.umu.se/~isak/repsy](http://www.cs.umu.se/~isak/repsy)!
CP-S: Complete pivoting kernel with scaling to avoid over/underflow.
SS-PP: Superscalar partial pivoting kernel.
CP-S: Complete pivoting kernel with scaling to avoid over/underflow.
SS-PP: Superscalar partial pivoting kernel.
Triangular Sylvester-type Matrix Equations

References


QR Factorization and Least Squares

See separate pile of OH pictures, including references (PowerPoint presentation by Elmroth).
Conclusions

• State-of-the-art HPC systems have **deep memory hierarchies**.

• **Recursion** efficiently provides **automatic variable blocking** for all levels of the memory hierarchy.

• Recursive blocking $\implies$ **Temporal locality**:
  – Level 3 BLAS
  – Packed Cholesky factorization
  – Triangular Sylvester-type matrix equations
  – QR factorization (hybrid algorithm) and least squares

• Recursive data formats $\implies$ **Spatial locality**:
  – Level 3 BLAS (GEMM, TRSM) - more underway!
  – Packed Cholesky factorization

• Our recursive blocked implementations with **optimized kernels**
  – are GEMM-rich, and
  – show **significant performance improvements**.