Management of Deep Memory Hierarchies - Recursive Blocking and Hybrid Data Structures for Dense Matrix Computations

Bo Kågström
Dept of Computing Science & HPC2N
Umeå University, Sweden

5th Workshop on Linux Clusters for Supercomputing, October 18-19, 2004, Linköping University

HPC2N - “HPC to North”

° National center for Scientific and Parallel Computing

Sarek - new super cluster (HP, installed 2004-06-07):
• 384 proc (64 bit, AMD Opteron 2.2 GHz)
• 1.5 TB memory (8 GB per node)
• Myrinet-2000 (HPI with 250 MB/s)
• > 1.3 Tflops/s HP-Linpack (~79% of peak)
• Most powerful computer in Sweden
• Funded by the Wallenberg Foundation (KAW)

° Funded by the Swedish Research Council and its meta-center SNIC
Matrix Computations

- Fundamental and ubiquitous in computational science and its vast application areas
- Library software - optimized building blocks for fundamental operations
  - BLAS, (Sca)LAPACK, SLICOT (see also NETLIB)
  - ESSL and other vendors
  - Portability and efficiency
- Continuing demand for new and improved algorithms and software along with the computer evolution

"Data transport" in memory hierarchies

- of today's computer systems
  - PC - cluster - supercomputer
  - Small, Fast, Expensive
  - Large, Slow, less Expensive
Management of deep memory hierarchies

- **Architecture evolution**: HPC systems with multiple SMP nodes, several levels of caches, more functional units per CPU
- **Key to performance**: understand the algorithm and architecture interaction
- **Hierarchical blocking**

The fundamental AHC triangle

```
1  2  5  6
3  4  7  8
9 10 13 14
11 12 15 16
```
Outline

- Hierarchical blocking: motivation and implications
- Recursive blocked templates and algorithms
- Recursive blocked data structures
- Case studies:
  - General matrix multiply and add (GEMM)
  - Packed Cholesky factorization
  - QR factorization and linear systems
  - Triangular matrix equations and condition estimation
- Some related and complementary work
- Work in progress: periodic matrix equations
- Concluding remarks

Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software

Joint work with:

Erik Elmroth
Fred Gustavson
Isak Jonsson
Bo Kågström

Abstract. Matrix computations are both fundamental and ubiquitous in computational science and its vast application areas. Along with the development of more advanced computer systems with complex memory hierarchies, there is a continuing demand for new algorithms and library software that efficiently utilize and adapt to new architecture features. This article reviews and details some of the recent advances made by applying the paradigm of recursion to dense matrix computations on today's memory-tiered computer systems. Recursion
Blocking for a memory hierarchy

Explicit multi-level blocking

Recursion leads to automatic variable blocking

Fits L1 Cache Stopping criteria controlled by parameter (blksz).

Fits low level in memory hierarchy

Fits high level in memory hierarchy

Recursive blocking
Splittings defining independent and dependent tasks

Critical path of subtasks: (1), (2), (3)

TRSM Operation: \( AX = C \),
A mxm upper triangular, \( C/X \) mxn

\[
A \begin{bmatrix} X_1 & X_2 \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \end{bmatrix}
\]

\[
AX_1 = C_1, \\
AX_2 = C_2.
\]

\[
\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}
\]

\[
A_{11}X_1 = C_1 - A_{12}X_2, \\
A_{22}X_2 = C_2.
\]
Case Study 1

General matrix multiply and add (GEMM)

Recursive splittings for GEMM:

\[ C \leftarrow \beta \text{op}(C) + \alpha \text{op}(A)\text{op}(B) \]

<table>
<thead>
<tr>
<th>Split</th>
<th>( m \times n )</th>
<th>( m \times k )</th>
<th>( k \times n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m, n, k )</td>
<td>[ \begin{bmatrix} C_{11} &amp; C_{12} \ C_{21} &amp; C_{22} \end{bmatrix} ] + [ \begin{bmatrix} A_{11} &amp; A_{12} \ A_{21} &amp; A_{22} \end{bmatrix} ] [ \begin{bmatrix} B_{11} &amp; B_{12} \ B_{21} &amp; B_{22} \end{bmatrix} ] =</td>
<td>[ \begin{bmatrix} C_{11} &amp; C_{12} \ C_{21} &amp; C_{22} \end{bmatrix} ] + [ \begin{bmatrix} A_{11} &amp; A_{12} \ A_{21} &amp; A_{22} \end{bmatrix} ] [ \begin{bmatrix} B_{11} &amp; B_{12} \ B_{21} &amp; B_{22} \end{bmatrix} ] =</td>
<td>[ \begin{bmatrix} C_{11} &amp; C_{12} \ C_{21} &amp; C_{22} \end{bmatrix} ] + [ \begin{bmatrix} A_{11} &amp; A_{12} \ A_{21} &amp; A_{22} \end{bmatrix} ] [ \begin{bmatrix} B_{11} &amp; B_{12} \ B_{21} &amp; B_{22} \end{bmatrix} ] + [ \begin{bmatrix} A_{11} &amp; A_{12} \ A_{21} &amp; A_{22} \end{bmatrix} ] [ \begin{bmatrix} B_{11} &amp; B_{12} \ B_{21} &amp; B_{22} \end{bmatrix} ] =</td>
</tr>
</tbody>
</table>
Recursive splitting - by breadth or by depth

GEMM recursive blocked template - splitting by depth

\[ C = \text{rgemm}(A, B, C, \text{blksz}) \]
If \( m, n, \) and \( k \leq \text{blksz} \)
\[ C = \text{opt\_gemm}(A, B, C) \% \text{optimized GEMM kernel!} \]

```
elseif \( m = \max(m, n, k) \% \text{split } m: m2 = m/2\)
C(1:m2, :) = \text{rgemm}(A(1:m2, :), B, C(1:m2, :), \text{blksz})
C(m2+1:m, :) = \text{rgemm}(A(m2+1:m, :), B, C(m2+1:m, :), \text{blksz})
```

```
elseif \( n = \max(n, k) \% \text{split } n: n2 = n/2, k\)
C(:,1:n2) = \text{rgemm}(A, B(:,1:n2), C(:,1:n2), \text{blksz})
C(:,n2+1:n) = \text{rgemm}(A, B(:,n2+1:n), C(:,n2+1:n), \text{blksz})
```

```
else \% \text{split } k: k2 = k/2, C = \text{rgemm}(A(:,1:n2), B(1:m2, :), C, \text{blksz})
C = \text{rgemm}(A(:,n2+1:n), B(m2+1:m, :), C, \text{blksz})
```

When to end the recursive splitting?
Locality of reference

- Recursive blocked algorithms mainly improve on the **temporal locality**
- Further performance improvements by matching the data structure with the algorithm (and vice versa)
- Recursive blocked data structures improve on the **spatial locality**

Blocked data formats

Blocks $A_{ij}$ of size $mb \times nb$ can be ordered in $(pq)!$ different ways
Recursive blocked row format

\[
\begin{bmatrix}
    A_{11} & A_{12} & \cdots & A_{18} \\
    A_{21} & \cdots & \cdots & A_{28} \\
    \vdots & \vdots & \ddots & \vdots \\
    A_{q1} & A_{q2} & \cdots & A_{q8}
\end{bmatrix}
\]

\( q = 8 \)

\( p = 8 \)

Recursive ordering: a 1-dim tour through a 2-dim object (Hilbert space filling heuristics)

RBR <-> Z-Morton ordering

---

Recursive GEMM: multi-level vs. recursive blocking

IBM PPC604, 112 MHz

---

CGEMM: Using conventional and recursive blocking

RGEMM

ATLAS
Recursive blocked GEMM and SMP parallelism via threads

IBM PPC604, 4 proc

Recursion template for one-sided matrix factorization

1. Partition
2. Factor left hand side
3. Update right hand side
4. Factor right hand side

Fits low level in memory hierarchy
Fits high level in memory hierarchy

Factorization completed
Update completed
Case Study 2

Cholesky factorization for matrices in packed format

Packed Cholesky factorization

\[
A = \begin{bmatrix} A_{11} & A_{21} \\ A_{21}^T & A_{22} \end{bmatrix} = L L^T = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22} \end{bmatrix}
\]

Standard approach (typified by LAPACK):
- Packed storage -> cannot use standard level 3 BLAS (e.g., DGEMM)
- 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.
- Minimum storage: \(1/2n(n+1)\) elements
### Packed recursive blocked data

<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>3</td>
<td>5</td>
<td>8</td>
<td>12</td>
<td>17</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>13</td>
<td>18</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>19</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Divide into two isosceles triangles $T_1$, $T_2$ and rectangle $R$.
- Divide triangles recursively down to element level.
- Store in order: $T_1$, $R$, $T_2$.
- Rectangles stored in full format $\Rightarrow$ Possible to use full storage level 3 BLAS.

![Diagram showing packed recursive blocked data]

**Fig. 3.1.** Memory indices for $7 \times 7$ upper triangular matrix stored in traditional packed format and recursive packed format.

### Cholesky recursive blocked template

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

**Factor:** $A_{11} = L_{11}L_{11}^T$.  \tag{2}

**TRSM:** $L_{21}L_{11}^T = A_{21}$.  \tag{3}

**SYRK:** $\tilde{A}_{22} = A_{22} - L_{21}L_{21}^T$.  \tag{4}

**Factor:** $\tilde{A}_{22} = L_{22}L_{22}^T$.  \tag{5}
Packed recursive blocked
Cholesky highlights

- Recursive blocked algorithm + recursive packed data layout => can make use of high performance level 3 BLAS routines (e.g., DGEMM)
- Use minimal storage for matrix A
- Temporary workspace = 1/8n^2 elements (~25%)
- Leaf problems (< blksz) are solved using superscalar kernels (Cholesky, TRSM, SYRK)

---

Recursive blocked Cholesky vs. LAPACK - (rec.) packed format

Runs at level 3 performance - at least!
Case Study 3
QR factorization and linear systems

1. Divide $A_{mxn}$ in two parts (left & right)

2. Factorize left hand side by a recursive call

3. Update right hand side

4. Factorize by a recursive call

Stopping criteria:
if $n < 4$ use standard algorithm

#flops grows cubically with Householder transformations being aggregated (compact WY)!
Recursive blocked QR highlights

- Recursive splitting controlled by \( nb \) (splitting point = \( \min(nb, n/2) \), \( nb = 32-64 \))
- Level 3 algorithm for generating \( Q = I - YTYT \) (compact WY) within the recursive blocked algorithm (T triangular of size \( \leq nb \))
- Replaces LAPACK level 2 and 3 algorithms

Recursive QR vs. LAPACK

- \( m = n \)
  - Performance results in MTops/s for square matrices (left) and performance ratio for tall, thin matrices (right) for the recursive algorithms RGEQRFP and DGEQRF of LAPACK on the 200 MHz IBM Power3.
  - 1.2-1.3x
  - Up to 1.95x
Case Study 4

Triangular matrix equations and condition estimation

Matrix equations

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix equation</th>
<th>Acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Sylvester (CT)</td>
<td>$AX - XB = C$</td>
<td>SYCT</td>
</tr>
<tr>
<td>Standard Lyapunov (CT)</td>
<td>$AX + XA^T = C$</td>
<td>LYCT</td>
</tr>
<tr>
<td>Generalized coupled Sylvester</td>
<td>$(AX - YB, DX - YE) = (C, F)$</td>
<td>GCSY</td>
</tr>
<tr>
<td>Standard Sylvester (DT)</td>
<td>$AXB^T - X = C$</td>
<td>SYDT</td>
</tr>
<tr>
<td>Standard Lyapunov (DT)</td>
<td>$AXA^T - X = C$</td>
<td>LYDT</td>
</tr>
<tr>
<td>Generalized Sylvester</td>
<td>$AXB^T - CXD^T = E$</td>
<td>GSYL</td>
</tr>
<tr>
<td>Generalized Lyapunov (CT)</td>
<td>$AXE^T + EXA^T = C$</td>
<td>GLYCT</td>
</tr>
<tr>
<td>Generalized Lyapunov (DT)</td>
<td>$AXA^T - EXE^T = C$</td>
<td>GLYDT</td>
</tr>
</tbody>
</table>

One-sided (top) and two-sided (bottom)
Separation of two matrices

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

where \[ Z = I_n \otimes A - B^T \otimes I_m. \]

Computing \( \text{Sep}[A,B] \) costs \( O(m^3n^3) \) - impractical!

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

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

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

Matrix equation Sep-functions

<table>
<thead>
<tr>
<th>( Z )-matrix</th>
<th>Sep-function = ( \sigma_{\text{min}}(Z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z_{\text{SYCT}} = I_n \otimes A - B^T \otimes I_m )</td>
<td>( \inf_{|X|_F = 1} |AX - XB|_F )</td>
</tr>
<tr>
<td>( Z_{\text{LYCT}} = I_n \otimes A + A \otimes I_n )</td>
<td>( \inf_{|X|_F = 1} |AX - X(-A^T)|_F )</td>
</tr>
<tr>
<td>( Z_{\text{GCSTY}} = \begin{bmatrix} I_n \otimes A &amp; -B^T \otimes I_m \ I_n \otimes D &amp; -E^T \otimes I_n \end{bmatrix} )</td>
<td>( \inf_{|X,Y|_F = 1} |(AX - YB, DX - YE)|_F )</td>
</tr>
<tr>
<td>( Z_{\text{SYDT}} = B \otimes A - I_n \otimes I_n )</td>
<td>( \inf_{|X|_F = 1} |AB^T - X|_F )</td>
</tr>
<tr>
<td>( Z_{\text{LYDT}} = A \otimes A - I_n \otimes I_n )</td>
<td>( \inf_{|X|_F = 1} |AX^T - X|_F )</td>
</tr>
<tr>
<td>( Z_{\text{GSCPY}} = B \otimes A - D \otimes C )</td>
<td>( \inf_{|X|_F = 1} |AXB^T - CXD^T|_F )</td>
</tr>
<tr>
<td>( Z_{\text{GCPYT}} = E \otimes A + A \otimes E )</td>
<td>( \inf_{|X|_F = 1} |AXE^T - EX(-A^T)|_F )</td>
</tr>
<tr>
<td>( Z_{\text{GCPYD}} = A \otimes A - E \otimes E )</td>
<td>( \inf_{|X|_F = 1} |AXA^T - EXB^T|_F )</td>
</tr>
</tbody>
</table>

\( Z \times = b, Z \) is a Kronecker product representation

Sep-function = smallest singular value of \( Z \)
Recursive SYCT - Case 3

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

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

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

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

\[
A_{22}X_{22} - X_{22}B_{22} = C_{22} + X_{21}B_{12}
\]
SYCT and matrix functions

- $A$ triangular $\Rightarrow F := f(A)$ triangular
- $f$ analytic $\Rightarrow$ exists series expansion $\Rightarrow A F - F A = 0$
- recursive template:

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
F_{11} & F_{12} \\
F_{21} & F_{22}
\end{bmatrix}
- 
\begin{bmatrix}
F_{11} & F_{12} \\
F_{21} & F_{22}
\end{bmatrix}
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
= 
\begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}
\]

\[
A_{11}F_{11} - F_{11}A_{11} = 0 \\
A_{11}F_{12} - F_{12}A_{22} = F_{11}A_{12} - A_{12}F_{22}, \\
A_{22}F_{22} - F_{22}A_{22} = 0
\]

Triangular generalized coupled Sylvester equation - GCSY

$AX - YB = C$
$DX - YE = F$

$(A, D)$ and $(B, E)$ in generalized Schur form

Solution $(X, Y)$ overwrites r.h.s. $(C, F)$
Two-sided matrix equation: GLYDT

- $AXA^T - EXE^T = C$
- $C = C^T n \times n$; $(A, E)$ $n \times n$ in gen. Schur form
- Unique sol'n $X = X^T \Leftrightarrow \lambda_i$ of $A - \lambda E$ satisfy $\lambda_i \lambda_j \neq 1$
- Recursive splitting:
  \[
  \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
  \end{bmatrix}
  \begin{bmatrix}
  X_{11} & X_{12} \\
  X_{21} & X_{22}
  \end{bmatrix}
  \begin{bmatrix}
  A_{11}^T & A_{12}^T \\
  A_{21}^T & A_{22}^T
  \end{bmatrix}
  =
  \begin{bmatrix}
  E_{11} & E_{12} \\
  E_{21} & E_{22}
  \end{bmatrix}
  \begin{bmatrix}
  X_{11} & X_{12} \\
  X_{21} & X_{22}
  \end{bmatrix}
  \begin{bmatrix}
  E_{11}^T & E_{12}^T \\
  E_{21}^T & E_{22}^T
  \end{bmatrix}
  =
  \begin{bmatrix}
  C_{11} & C_{12} \\
  C_{21} & C_{22}
  \end{bmatrix}
  \]

Two-sided matrix product

$$C = \beta C + \alpha \text{op}(A) \text{op}(B)^T$$

- $A$ and/or $B$ can be dense or triangular
- One or several of $A$, $B$ and $C$ can be symmetric
- Extra workspace - size of r.h.s.

Make use of symmetry, e.g., in GLYDT:

$$C_{11} = C_{11} - A_{12}X_{22}A_{12}^T$$
and
$$C_{11} = C_{11} + E_{12}X_{22}E_{12}^T$$
GLYDT performance with optional condition estimation

Table 5.3 Timings for solving unreduced two-sided matrix equations (GLYDT) with optional condition estimation. (Job = X, compute solution only; Job = X + Sep, compute solution and Sep-estimation.) Results from 375 MHz IBM Power5.

<table>
<thead>
<tr>
<th>n</th>
<th>SG03AD using SG03AX Total time</th>
<th>Solver part</th>
<th>SG03AD using rmglydt Total time</th>
<th>Solver part</th>
<th>Speedup</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.0277</td>
<td>49.9 %</td>
<td>0.0185</td>
<td>20.1 %</td>
<td>1.96</td>
<td>X</td>
</tr>
<tr>
<td>100</td>
<td>0.180</td>
<td>51.2 %</td>
<td>0.0067</td>
<td>9.0 %</td>
<td>1.86</td>
<td>X</td>
</tr>
<tr>
<td>250</td>
<td>2.89</td>
<td>46.8 %</td>
<td>1.62</td>
<td>4.7 %</td>
<td>1.79</td>
<td>X</td>
</tr>
<tr>
<td>500</td>
<td>59.0</td>
<td>42.3 %</td>
<td>34.5</td>
<td>1.5 %</td>
<td>1.71</td>
<td>X</td>
</tr>
<tr>
<td>750</td>
<td>363.4</td>
<td>42.0 %</td>
<td>177.5</td>
<td>0.9 %</td>
<td>1.71</td>
<td>X</td>
</tr>
<tr>
<td>1000</td>
<td>646.6</td>
<td>44.6 %</td>
<td>361.8</td>
<td>1.0 %</td>
<td>1.79</td>
<td>X</td>
</tr>
</tbody>
</table>

(a)

RECSY library

- Recursive blocked algorithms for solving reduced matrix equations
- Recursion implemented in F90
- SMP versions using OpenMP
- F77 wrappers for LAPACK and SLICOT routines
- www.cs.umu.se/research/parallel/recsy/
SCASY library

- **ScalAPACK-style** software package of matrix equation solvers for distributed memory machines.
- Triangular solvers are used in implementing parallel condition estimators for each matrix equation.
- With Robert Granat, PhD student

Discrete-time periodic systems

\[ x_{k+1} = A_k x_k + B_k u_k \]
\[ y_k = C_k x_k + D_k u_k \]

\[ A_k \in \mathbb{R}^{n_{k+1} \times n_k}, \quad B_k \in \mathbb{R}^{n_{k+1} \times m} \]
\[ C_k \in \mathbb{R}^{p \times n_k}, \quad D_k \in \mathbb{R}^{p \times m} \] - \( K \)-periodic

\( (A_{k+K} = A_k, B_{k+K} = B_k, \ldots) \)

From discretization of continuous-time periodic models: revolving satellite; helicopter in forward flight; multi-rate sampled control systems \( \rightarrow \) time-varying dimensions
**Periodic Sylvester equation**

- \( A(:,k) X(:,k) + X(:,k+1) B(:,k) = C(:,k+1) \)
- for \( k = 1 : K - 1 \)
- \( A(:,K) X(:,K) + X(:,1) B(:,K) = C(:,1) \)

**Script notations:**
- \( X_k \) \( K \)-periodic
- \( X_k := \text{diag} (X_{k}, X_{k+1}, \ldots, X_{k+K-1}) \)
- \( \sigma X_k := \text{diag} (X_{k+1}, \ldots, X_{k+K-1}, X_k) \)

---

**Recursive blocked periodic Sylvester equation**

**Some preliminary performance results**

![Graph showing performance results](image)
Recursive blocking ...

- creates new algorithms for linear algebra software
- expresses dense linear algebra algorithms entirely in terms of level~3 BLAS like matrix-matrix operations
- introduces an automatic variable blocking that targets every level of a deep memory hierarchy
- can also be used to define hybrid data formats for storing block-partitioned matrices (general, triangular, symmetric, packed) - L1, L2 and TLB misses are minimized for certain block sizes (Park-Hong-Prosana’03)

High-performance software

- implementations are based on data locality and superscalar optimization techniques
- recursive blocked algorithms improve on the temporal data locality
- hybrid data formats improve on the spatial data locality
- portable and generic superscalar kernels ensure that all functional units on the processor(s) are used efficiently
Acknowledgements

- Erik Elmroth, Isak Jonsson, Fred Gustavson (co-authors and co-workers)
- André Henriksson, Olov Gustavsson and Andreas Lindkvist (earlier MSc students)
- Bjarne Andersén, Jerzy Wasniewski (e.g., packed Cholesky)
- Robert Granat (PhD student)
- HPC and LA team at Umeå University
- Community that do related and complementary work! (see SIAM Rev. 2004)

Thanks for your attention!
Some related and complementary work

- Recursive algorithms and hybrid data structures
  - \textit{Winograd-Strassen'69}: Douglas et al'94, ESSL, Demmel-Higham'92 (stability)
  - Quad- and octtrees: Samet'84, Salman-Warner'94 (N-body, Barnes-Hut'84)
  - Cache oblivious algorithms: Leiserson et al'99 (sorting, FFT, $A^T$)
  - GEMM: Chatterjee et al'02, Valsalam and Skjellum'02, ATLAS-project
  - LU: Toledo'97(dense), Dongarra, Eijkhout Luszczek'01 (sparse)
  - QR: Rabani and Toledo'01 (out-of-core), Frens and Wise'03 (Givens-based)

Some related and complementary work

- Automated generation of library software and compiler technology
  - Empirical optimization:
    - PHiPAC - Bilmes, Demmel et al'97
    - ATLAS - Whaley, Petitet and Dongarra'00
    - Sparse kernels - Vuduc, Demmel et al'03
  - FLAME: Gunnels, Goto, Van de Geijn et al'01, '02
  - Compiler blockability: Wolf and Lam'91 (loop transformations), Carr and Lehoucq'97
  - Automatic generation of recursive codes:
    - Ahmed and Pingali'00 (iterative algorithms \rightarrow recursive)
    - Yi, Adve and Kennedy'00 (convert loop nests into recursive form)