RECSY —
A High Performance Library for Sylvester-Type Matrix Equations

Isak Jonsson, Bo Kågström

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

{isak,bokg}@cs.umu.se

Euro-Par 2003, Klagenfurt, Austria
August 26--29, 2003

Outline

- Blocked algorithms and memory hierarchies.
- Sylvester-type matrix equations.
- Recursive blocked solvers for triangular Sylvester-type matrix e
- Implementation issues.
- The RECSY library.
- Questions.

Deep Memory Hierarchies

Architecture Evolution: HPC systems with multiple SMP
caches and more functional units per CPU.

Key to performance: Understand the algorithm and archite

Goal: Maintain 2-dim data locality at every level of the 1-dim ti

- Hierarchical blocking.
- Matching an algorithm and its data structure.

1–dim ↔ 2–dim: Blocking

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 d
- Recursive algorithms – divide and conquer style.
- Automatic hierarchical blocking – variable and "squaris
- Only tuning parameter is L1 cache.

Sylvester-Type Matrix Equati

Appear in different control theory applications: stability problems
balancing, H_∞ control.

One-sided:

- Sylvester (SYCT): $AX - XB = C$, A, B and C general
- Lyapunov (LYCT): $AX + XA^T = C$, A general, $C = C^T$ (sen
- Generalized (coupled) Sylvester (GCSY):

$$\begin{aligned} AX - YB &= C \\ DX - YE &= F \end{aligned}$$

Two-sided:

- Discrete Sylvester (SYDT): $AXB^T - X = C$
- Discrete Lyapunov (or Stein) (LYDT): $AXA^T - X = C$
- Generalized Sylvester (GSYL): $AXB^T - CXD^T = F$
- Generalized Lyapunov

$$\begin{aligned} &\text{(GLYCT):} \quad AXE^T + EXA^T = F \\ &\text{(GLYDT):} \quad AXA^T - EXE^T = F \end{aligned}$$

Sylvester-Type Matrix Equati

Bartels-Stewart-type of algorithms $\Longrightarrow$
second major step in the solution is to solve a TRIANGULAR MAT

Our blocked recursive technique works for all! Here

- TRIANGULAR DISCRETE-TIME SYLVESTER AND LYAPUNO
- TRIANGULAR GENERALIZED SYLVESTER AND LYAPUNOV

Great source of triangular matrix equation problems from:

- CONDITION ESTIMATION
  - of the matrix equations themselves,
  - and in various eigenspace problems including reordering of e
- COMPUTING FUNCTIONS OF MATRICES.

Recursive Triangular Sylvester S

$op(A) \cdot X \pm X \cdot op(B) = \beta \cdot C$, $C \leftarrow X$ $(M \times N)$, where $A(M \times M)$ and $B(N \times N)$ upper quasi-triangular.

$transA =$'N', $transB =$'N', $sign = -$, $\beta = 1$:

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

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

$$\begin{aligned} A_{11}X_1 - X_1B &= C_1 - A_{12}X_2 \\ A_{22}X_2 - X_2B &= C_2 \end{aligned}$$

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)

Recursive Triangular Sylvester S

Case 3 ($N/2 < M < 2N$): Split A, B and C

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

$$\begin{aligned} A_{11}X_{11} - X_{11}B_{11} &= C_{11} - A_{12}X_{21} \\ A_{11}X_{12} - X_{12}B_{22} &= C_{12} - A_{12}X_{22} + X_{11}B \\ A_{22}X_{21} - X_{21}B_{11} &= C_{21} \\ A_{22}X_{22} - X_{22}B_{22} &= C_{22} + X_{21}B_{12} \end{aligned}$$

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 Operatio

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 dimen
   factor 2 from each other.

2. $\Longrightarrow$ a shorter but wider recursion tree, which offers more "para

Use of BLAS routines

The recursive approach gives algorithms that call level 3 BLAS rou
SYRK, etc.) with square blocks. This enables the best performanc
BLAS routines.

Sylvester Kernel

For problems smaller than the block size, the dimension is split in
is of size 2×2 – 4×4, when subsystems are solved using

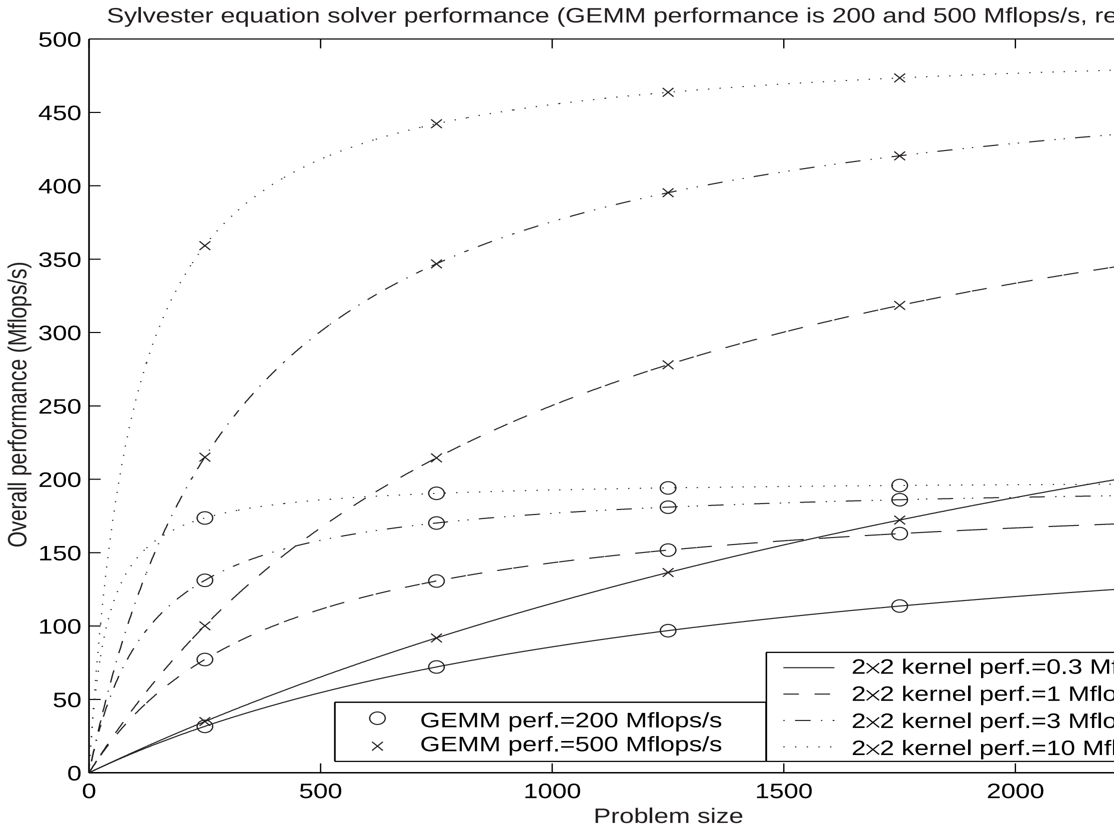
$$(B \otimes A - I_n \otimes I_m)\mathsf{vec}(X) = \mathsf{vec}(C)$$

Impact of Kernel Solvers

Kernel solvers execute $O(N^2)$ flops out of the total $O(N^3)$.
Model of the *overall computation rate*:

$$S = \frac{\text{Total number of flops}}{\text{GEMM time} + \text{Kernel time}} = \frac{2N^3}{\frac{2N^3-4N^2}{G} + \cdots}$$

where G = DGEMM perf., and K = kernel solver perf.



Impact of Kernel Solvers

Modelling results:

- N = 2500; G = 500, K = 0.3 Mflops/s $\Longrightarrow$
  Overall performance is at most approaching 50% of GEMM pe
- N = 2500; G = 500, K = 3.0 or 10 Mflops/s $\Longrightarrow$
  Overall performance rather quickly approaches 80–90% of GEM
- N = 750; G = 500, K = 0.3 Mflops/s $\Longrightarrow$
  Kernel flops = 0.3%, Kernel time = 80%
  (e.g. LAPACK DTGSY2 on modern RISC/CISC processors)
  DTGSY2 designed with the primary goal of producing high-acc
  signaling ill-conditioning (complete pivoting, overflow guarding

Trade-off between ROBUSTNESS, RELIABILITY AND SPEED.

Kernel performance decisive for the overall performance of matrix

Optimized Superscalar Kerne

Our design approach:

- One single routine to solve a kernel problem using Kronecker p
  representation $\Longrightarrow$
  great potential for register reuse.

- LU with partial pivoting and overflow guarding, forward and b
  using complete loop unrolling.

- Matrix multiply kernel "lite" using register blocking techniques
  unrolling and fusion).

Facilitates for the compiler to look ahead and schedule the resultin
optimally.

Two-sided Matrix Equations
Cost and Execution Order

Recursive blocked algorithms require both extra workspace (two-s
and more flops compared to the standard elementwise algorithms.

Matrix equation	Overall cost in flops	Flop r (M =
SYDT	$\frac{6}{4}M^2N + 2MN^2 \quad (M \leq N)$ $2M^2N + \frac{6}{4}MN^2 \quad (M > N)$	1.166
LYDT	$\frac{25}{12}N^3$	1.562
GSYL	$3M^2N + 4MN^2 \quad (M \leq N)$ $4M^2N + 3MN^2 \quad (M > N)$	1.166
GLYCT	$\frac{21}{6}N^3$	1.312
GLYDT	$\frac{25}{6}N^3$	1.562

Outperform the standard algorithms for large enough problems—
LOCALITY.

DATA REFERENCE PATTERNS: Order in which they access data
and how many times the data is moved in the memory hierarchy.

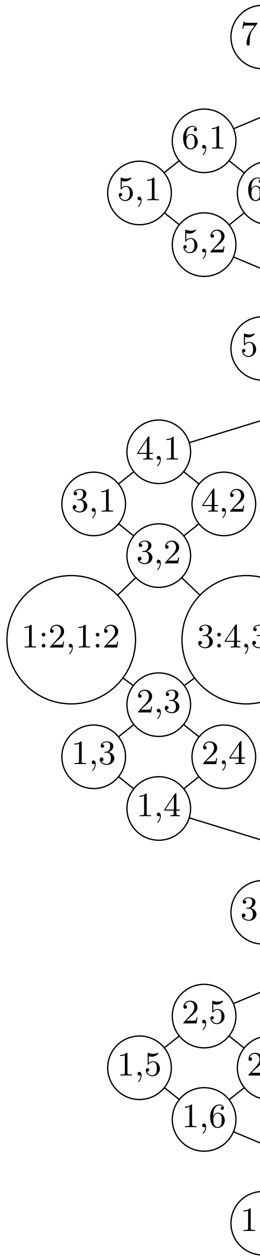
SMP Parallelism

SMP parallelism is obtained by solving independent equations as different OpenMP sections.

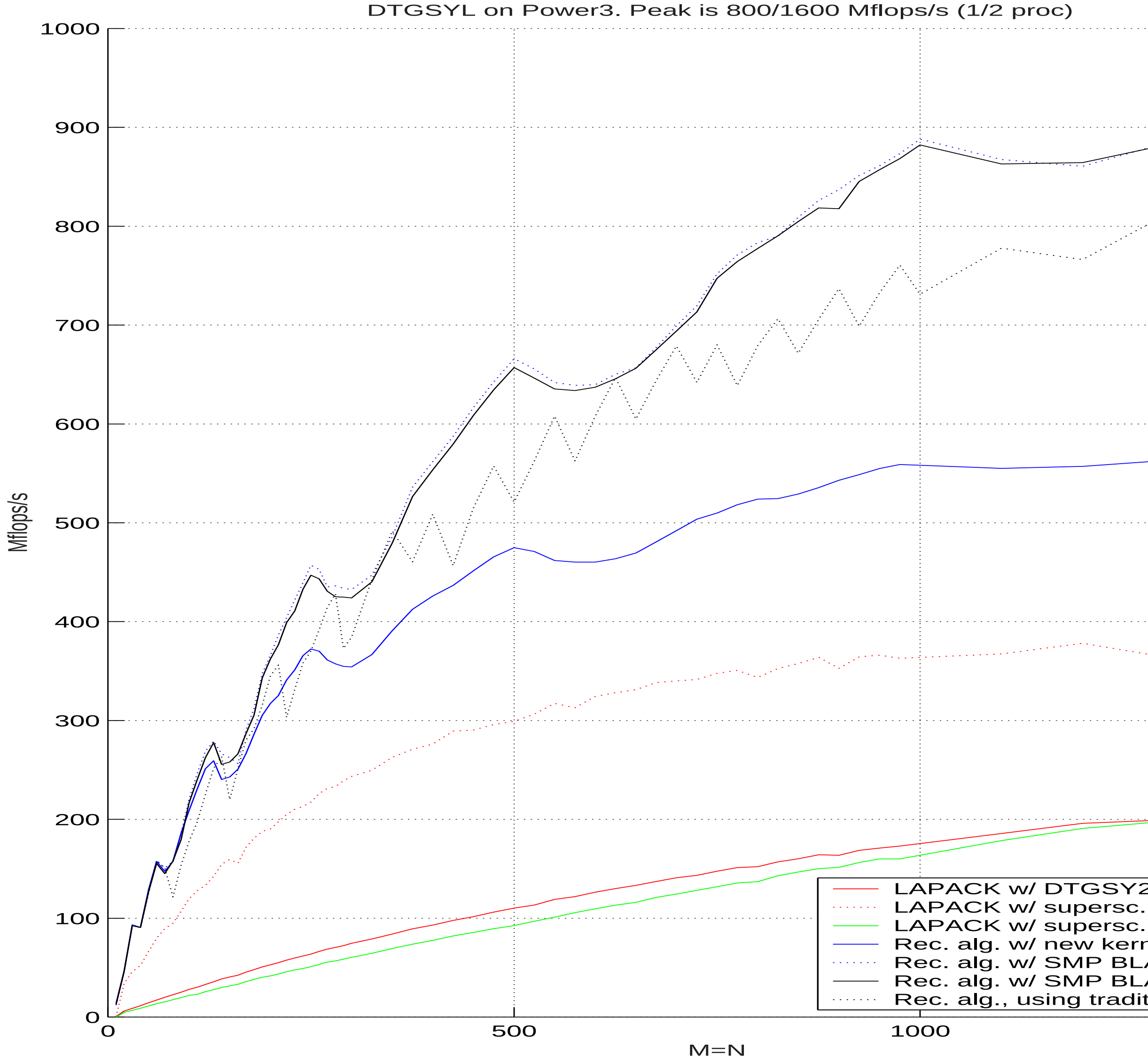
Recursion:

- Shows what parts can be solved in parallel.
- Creates problems that are large ⇒ coarse granularity.

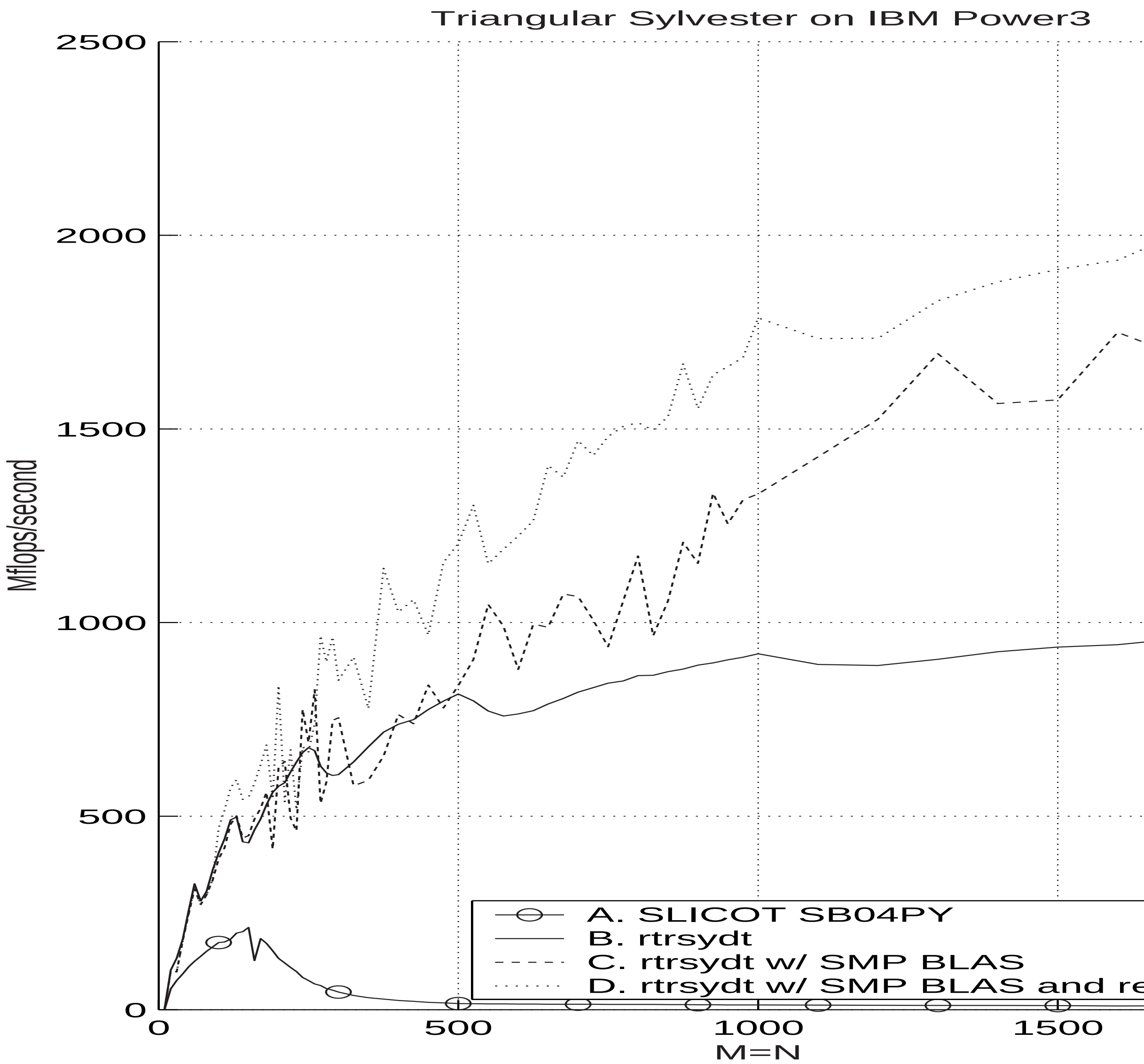
Also, due to the coarse granularity, SMP versions of DGEMM run well.



Coupled Sylvester – SMP Perfor



TRIANGULAR $AXB^T - X = C$ (SY
IBM POWER 3



RECSY

- A library that encompasses all eight mentioned matrix equatio
SMP parallel versions.
- Recursion is done using Fortran 90 RECURSIVE SUBROUTINES.
- Extra memory buffered are dynamically allocated, or can be pr
- SMP versions are available on all platforms with OpenMP com
- F77 Wrappers for SLICOT and LAPACK routines provided wi
recompile "legacy" code, only relink.
- Fall-back routines provide the same accuracy and stability as t
- Source publicly available at `http://www.cs.umu.se/~isak/`

RECSY – Uniprocessor Routin

- RECSYCT(UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, INFO
  (LAPACK: DTRSYL)
- RECLYCT(UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACHINE) (SLI
- RECGCSY(UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, D, L
  INFO, MACHINE) (LAPACK: DTGSYL)
- RECSYDT(UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, INFO
  WKSIZE) (SLICOT: SB04PY)
- RECLYDT(UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACHINE, WORK
  (SLICOT: SB03MX)
- RECGSYL(UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C, LDC, D, L
  MACHINE, WORKSPACE, WKSIZE) (No equivalent in SLICOT or LAPACK!)
- RECGLYDT(UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, INFO, MACH
  WKSIZE) (SLICOT: SG03AX)
- RECGLYCT(UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, INFO, MACH
  WKSIZE) (SLICOT: SG03AY)

RECSY – Multiprocessor Rout

- RECSYCT_P(PROCS, UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C,
- RECLYCT_P(PROCS, UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACH
- RECGCSY_P(PROCS, UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C,
  F, LDF, INFO, MACHINE)
- RECSYDT_P(PROCS, UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C,
  WORKSPACE, WKSIZE)
- RECLYDT_P(PROCS, UPLO, SCALE, M, A, LDA, C, LDC, INFO, MACH
  WKSIZE)
- RECGSYL_P(PROCS, UPLOSIGN, SCALE, M, N, A, LDA, B, LDB, C,
  INFO, MACHINE, WORKSPACE, WKSIZE)
- RECGLYDT_P(PROCS, UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, I
  WORKSPACE, WKSIZE)
- RECGLYCT_P(PROCS, UPLO, SCALE, M, A, LDA, E, LDE, C, LDC, I
  WORKSPACE, WKSIZE)

Unreduced Two-sided: $AXA^T - EX$
Solving and Sep[GLYDT]-estima

a) N	SG03AD using SG03AX Total time	Solver	SG03AD using REC Total time	Sol
50	0.0277	49.9 %	0.0185	20.
100	0.180	51.2 %	0.0967	9.0
250	2.89	46.8 %	1.62	4.7
500	59.0	42.3 %	34.5	1.5
750	303.4	42.0 %	177.5	0.9
1000	646.6	44.6 %	361.8	1.0
50	0.117	87.6 %	0.0263	45.6
100	0.709	87.3 %	0.152	40.6
250	9.98	84.5 %	2.08	25.4
500	178.6	80.9 %	37.8	9.4
750	924.1	80.9 %	184.4	4.5
1000	2076.6	82.7 %	391.8	8.4

We get another 2x speedup ($N > 500$) by replacing LAPACK rou
DHGEQZ by Dackland-Kågström's blocked Hessenberg-triangular
algorithms (ACM TOMS'99) for transforming (A, E) to generaliz

Conclusions – so far

- State-of-the-art HPC systems have **deep memory hierarchie**
- **Recursion** efficiently provides automatic variable blocking for hierarchy.
- Recursive blocking $\Longrightarrow$ Temporal locality
- Our recursive blocked implementations with optimized kernels
  - are GEMM-rich, and
  - show significant performance improvements (30% to 400+%)
- Code at `http://www.cs.umu.se/~isak/recsy`
  - Uses F90 for recursion, dynamic memory allocation
  - Uses (nested) OpenMP for SMP parallelism
  - Overloads LAPACK and SLICOT routines for Sylvester-typ