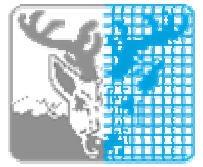
Parallel ScaLAPACK-style Algorithms for Solving Continous-Time Sylvester Matrix Equations

Robert Granat, Bo Kågström and Peter Poromaa

Dept of Computing Science and HPC2N
Umeå University, Sweden

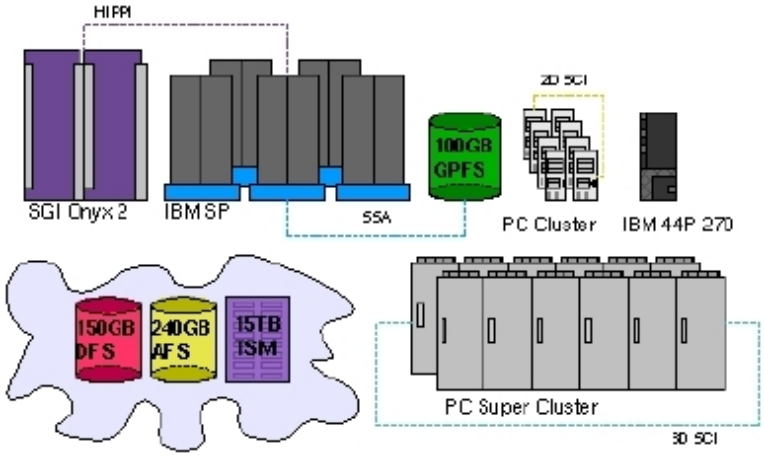


Outline

- Solving the CT Sylvester Matrix Equation
- Serial/Parallel Block Algorithms
- Application: Condition Estimation
- Implementation Issues
- Experiments
- Ongoing Work
- Conclusions
- Some References

Preliminaries

- *People:*
  - Robert Granat, PhD student, Dept. Computing Science, UmU
  - Prof. Bo Kågström, Dept. Computing Science and HPC2N, UmU
  - PhD Peter Poromaa, Dept. Computing Science and HPC2N, UmU
- *Sponsors:*
  - *The Swedish Research Council*
  - *The Swedish Foundation for Strategic Research*
- *Computing resources:*
  - High Performance Computing Center North (HPC2N)



Solving the CT Sylvester Matrix Equation (1)

We want to solve

$$op(A)X - Xop(B) = C$$

where $A \in R^{M\times M},\ B \in R^{N\times N},\ C \in R^{M\times N}$

$$op(A) \in \{ A\ ,\ A^T \}$$

The solution $X \in R^{M\times N}$ is unique iff

$$\lambda(A) \bigcap \lambda(B) = \varnothing \Leftrightarrow sep(A,B) \neq 0$$

Solving the CT Sylvester Matrix Equation (2)

Bartels-Stewart (1972)

1. Transform *A* and *B* to real Schur form:
$$T_A = Q^T A Q, T_B = P^T B P$$
2. Transform *C* with respect to the ”new” looks on *A* and *B*: $\tilde{C} = Q^T C P$
3. Solve the triangular problem
$$op(T_A)\tilde{X} - \tilde{X} op(T_B) = \tilde{C}$$
4. Transform back to the original coordinate system: $C \leftarrow X = Q\tilde{X}P^T$

Solving the CT Sylvester Matrix Equation (3)

- Bartels-Stewart in practice:
  1. Hessenberg reduction + the QR-algorithm
  2. GEMM-operation $C \leftarrow \beta \cdot C + \alpha \cdot op(A)op(B)$
  3. Triangular solver $O(M^2N + MN^2)$
  4. GEMM-operations
- If we use explicit blocking, these four steps can be parallelized.
- Our focus is on step 3.

Serial/Parallel Block Algorithms (1)

- After blocking our transformed matrices, the four triangle matrix equation variants can be expressed as

$$AX - XB = C \Leftrightarrow A_{ii}X_{ij} - X_{ij}B_{ij} = C_{ij} - (\sum_{k=i+1}^{D_a} A_{ik}X_{kj} - \sum_{k=1}^{j-1} X_{ik}B_{kj})$$

$$A^TX - XB^T = C \Leftrightarrow A_{ii}{}^TX_{ij} - X_{ij}B_{ij}{}^T = C_{ij} - (\sum_{k=1}^{i-1} A_{ki}{}^TX_{kj} - \sum_{k=j-1}^{D_b} X_{ik}B_{jk}{}^T)$$

$$A^TX - XB = C \Leftrightarrow A_{ii}{}^TX_{ij} - X_{ij}B_{ij} = C_{ij} - (\sum_{k=1}^{i-1} A_{ki}{}^TX_{kj} - \sum_{k=1}^{j-1} X_{ik}B_{kj})$$

$$AX - XB^T = C \Leftrightarrow A_{ii}X_{ij} - X_{ij}B_{ij}{}^T = C_{ij} - (\sum_{k=i+1}^{D_a} A_{ik}X_{kj} - \sum_{k=j+1}^{D_b} X_{ik}B_{jk}{}^T)$$

Serial/Parallel Block Algorithms (2)

- *A* is blocked by the factor *MB*, *B* is blocked by the factor *NB* ⇒ *C*/*X* have row-block size *MB* and column-block size *NB*.
- # diagonal blocks of *A* and *B*:

$$D_a = \lceil M / MB \rceil, \quad D_b = \lceil N / NB \rceil$$

- These summation formulas lead to serial block algorithms for solving the triangular reduced problems.

Serial/Parallel Block Algorithms (3)

```
for j=1, D_b
    for i=D_a, 1, -1
        {Solve the (i, j)th subsystem}
        A_ii X_ij − X_ij B_jj = C_ij
        for k=1, i − 1
            {Update block column j of C}
            C_kj = C_kj − A_ki X_ij
        end
        for k=j + 1, D_b
            {Update block row i of C}
            C_ik = C_ik + X_ij B_jk
        end
    end
end
```

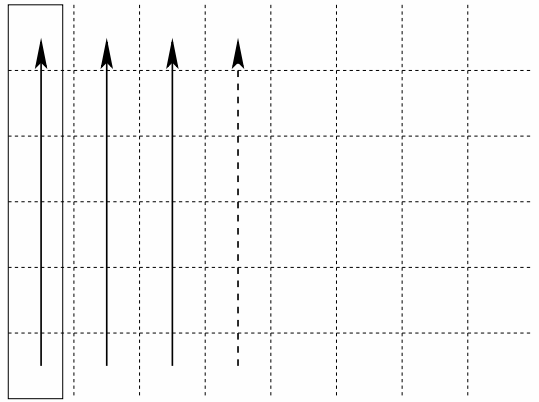


Fig. 1. Block algorithm for solving $AX - XB = C$, A and B in upper Schur form.

Serial/Parallel Block Algorithms (4)

- ScaLAPACK environment:
  - rectangular process grid *Pr x Pc*
  - 2D block cyclic data distribution
- Parallelizing the serial algorithm:
  1. Traverse the matrix ***C/X*** along its block diagonals
  2. Solve for every C_{ij} the small subsystems associated with the currect block diagonal in parallel
  3. Broadcast each computed X_{ij} in block row *i* and block column *j*
  4. Update block row *i* and block column *j* in parallel
- Communicate for needed blocks ”on demand”
- The same approach can be used for any of the transpose cases of SYCT

Serial/Parallel Block Algorithms (5)

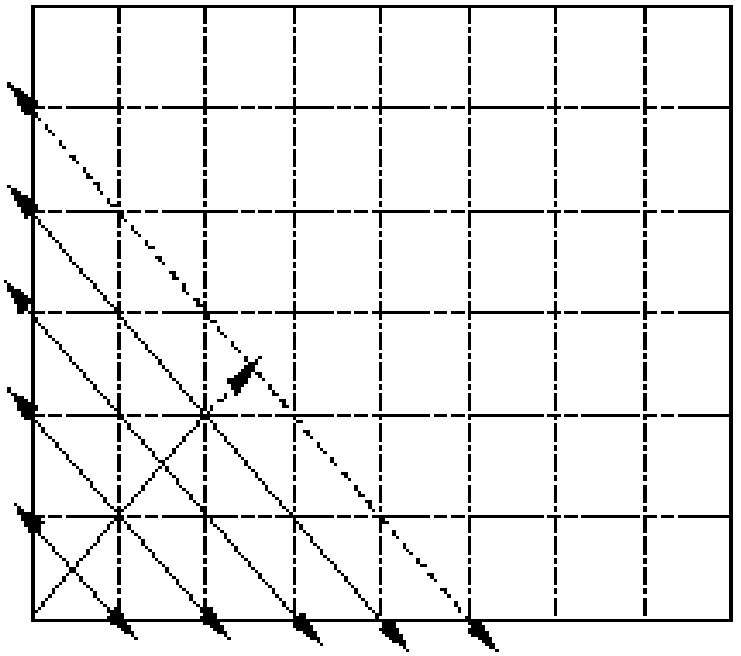


Fig. 3. Traversing the matrix C/X when solving $AX - XB = C$.

Serial/Parallel Block Algorithms (6)

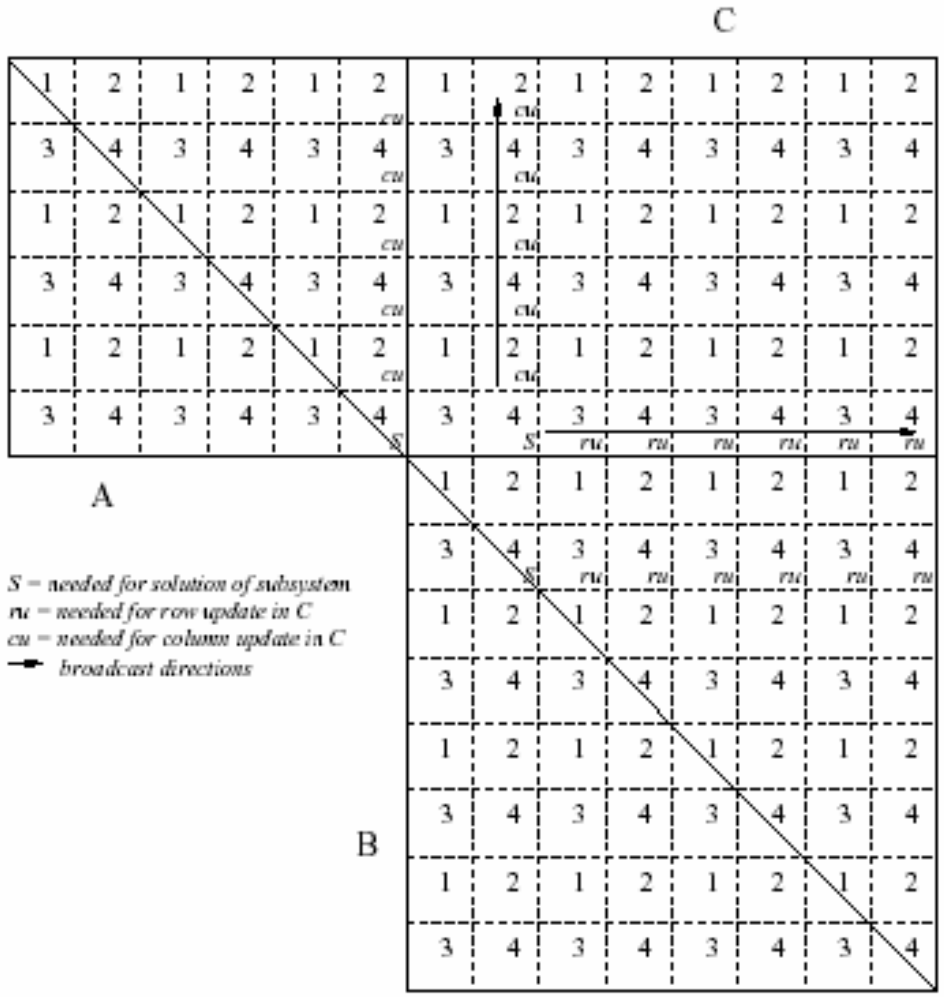


Fig. 4. Data dependencies and mapping for $AX - XB = C$ on a 2×2 processor grid.

Serial/Parallel Block Algorithms (7)

```
for k=1, # block diagonals in C
      {Solve subsystems on current block diagonal in parallel}
      if(mynode holds C_ij)
        if(mynode does not hold A_ii and/or B_jj)
          Communicate for A_ii and/or B_jj
        Solve for X_ij in op(A_ii)X_ij − X_ij op(B_ij) = C_ij
        Broadcast X_ij to processors that need X_ij for updates
      elseif(mynode needs X_ij)
        Receive X_ij
        if(mynode does not hold needed block in A for updating block column j)
          Communicate for requested block in A
        Update block column j of C in parallel
        if(mynode does not hold needed block in B for updating block row i)
          Communicate for requested block in B
        Update block row i of C in parallel
      endif
end
```

Fig. 5. Parallel block algorithm for $\mathrm{op}(A)X - X\mathrm{op}(B) = C$, A and B in Schur form.

Matrix equations as linear systems using Kronecker products

- All linear matrix equations can be written as a linear system of equations:

$$op(A)X - Xop(B) = C \quad \Leftrightarrow \quad Z_{SYCT}x = y$$

$$Z_{SYCT} = I_N \otimes op(A) - op(B)^T \otimes I_M$$

$$x = col(X), \;\; y = col(C)$$

In blocked algorithms, *Zx = y* representation is used in kernels for solving small-sized matrix equations.

Application: Condition estimation (1)

- An important quantity in the perturbation theory for Sylvester-type equations is the *separation between two matrices*:

$$sep(A,B) = \inf_{\|X\|_F=1} \|op(A)X - Xop(B)\|_F = \sigma_{\min}(Z_{SYCT}) = \|Z_{SYCT}^{-1}\|_2^{-1}$$

- To compute *sep(A,B)* exactly costs $O(M^3N^3)$ flops (only of interest in theory). We want to compute a reliable but low cost estimate (serially as well as in parallel).

- We apply a general method (Hager'84, Higham'88, Kågström-Poromaa'92) for estimating $\|A^{-1}\|_1$ which only uses matrix vectors products:

$$A^{-1}x \qquad A^{-T}x$$

- ⇒ we can estimate *sep(A,B)* for SYCT by solving the equation itself to an $O(M^2N + MN^2)$ cost.

Condition estimation (2)

- To use our condition estimator we only have to compute $Z_{SYCT}^{-1} y$, and $Z_{SYCT}^{-T} y$

  which corresponds to solving triangular

$$AX - XB = C$$

$$A^T X - XB^T = C$$

- Notice that when *sep(A, B)* is tiny, the SYCT equation is close to singular, i.e., ill-conditioned (compare with the scalar case x = c / (a – b)).

Condition estimation (3)

$kase = iter = 0$
while($kase \neq 0$ or $iter = 0$) do
    {Compute a new estimate using PDLACON}
    Call PDLACON(MN, c, est, $kase$)
    {Check if the last estimate was the final one}
    if($kase \neq 0$)
        {Update iteration counter}
        $iter = iter + 1$
        {Build the right-hand-side C from the vector c}
        $C = reshape(c, M, N)$
        if($kase = 1$)
            Compute $Z_{SYCT}^{-1} c$, i.e., solve $AX - XB = C$ for the matrix X
        elseif($kase = 2$)
            Compute $Z_{SYCT}^{-T} c$, i.e., solve $A^T X - XB^T = C$ for the matrix X
        end
        {Build the vector c of the columns of X}
        $c = reshape(X, MN, 1)$
    end
end
$est = est / \sqrt{M \cdot N}$

Figure 7. Brief outline of PZSYCTCOND

Condition estimation (4)

- We use ScaLAPACK routine PDLACON
- Implemented parallel impl. (”*reshape*”) of the vector operator *col* and its inverse
- Typically the while-loop will be executed about 5 times
- PZSYCTCOND computes a lower bound 1-norm-based estimate of $sep^{-1}(A,B)$, which can be transformed to spectral norm: $\|A\|_1 / \sqrt{N} \leq \|A\|_2, A \in R^{N \times N}$

Implementation Issues (1)
Parallel SYCT software developed

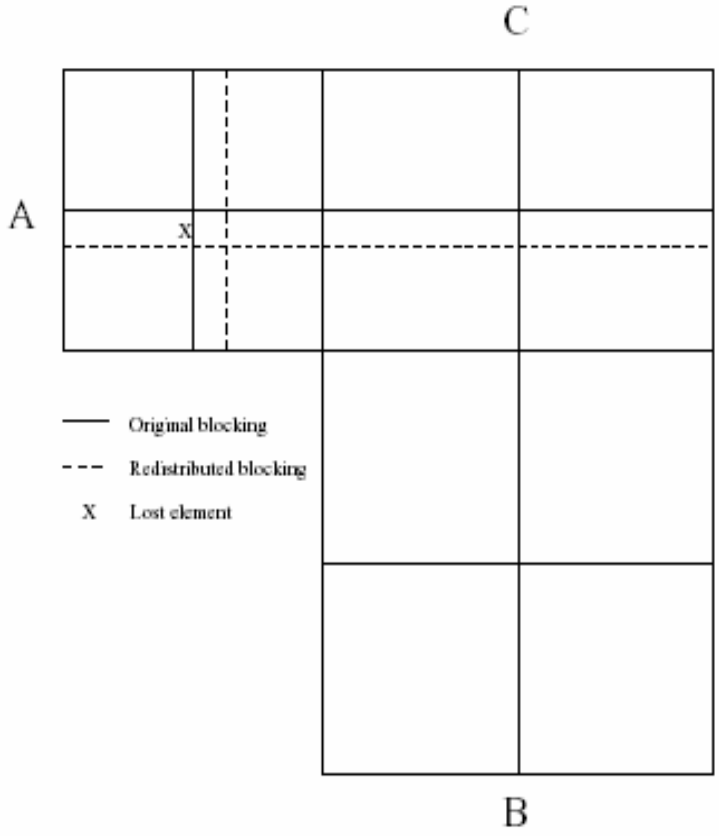
- PDGESY – Parallel Bartels-Stewart
  - Solves all four transpose cases
  - Uses ScaLAPACK/PBLAS routines PDGEHRD, PDLAHQR, PDGEMM
  - Uses our PDTRSY
- PDTRSY – Parallel triangular SYCT solver
  - Uses LAPACK/BLAS routines DTRSYL, DGEMM
  - Handles the ”2x2 diagonal block split” problem
- PZSYCTCOND – Parallel condition estimator for SYCT
  - Uses ScaLAPACK routine PDLACON and our PDTRSY
  - Works with parallel *col*-operators (”*reshape*”)

Implementation Issues (2)
2x2 diagonal block split problem

- The explicit blocking can be in conflict with 2x2 blocks (conj. pairs of eigenvalues) on the diagonals of *A* and *B*

- Elements must be redistributed in the grid

- Implicit redistribution approach:
  - Local workarrays are filled with elements from neighbouring processes
  - Controlled by a global object (similar to the ScaLAPACK descriptors)



Implementation Issues (3)
Data objects for 2x2 diagonal block split

- The global object is an array of integers describing the implicit redistribution.
- *A* and *B* are assigned one global array each of size D_a and D_b , respectively.
- Using this global object, the ”correct” submatrices can be formed during the solution process.

$$INFO\_ARRAY\_A(i) = \begin{cases} 0 \Leftrightarrow A_{ii} \text{ unchanged} \\ 1 \Leftrightarrow A_{ii} \text{ extended} \\ 2 \Leftrightarrow A_{ii} \text{ diminished} \\ 3 \Leftrightarrow A_{ii} \text{extended and diminished} \end{cases}$$

Experiments (1)

- The implementations are tested on the IBM Scalable POWERparallel (SP) system at HPC2N.
- The execution time dominated (~80%) by the reduction to Schur form. PDGEMM operations take at most 17% and the triangular solver below 10%.
- PDGESY and PDTRSY shows good accuracy for quite ill-conditioned random problems and delivered speedup typically around $\sqrt{p}$ and $2\sqrt{p}$.

Experiments (2)

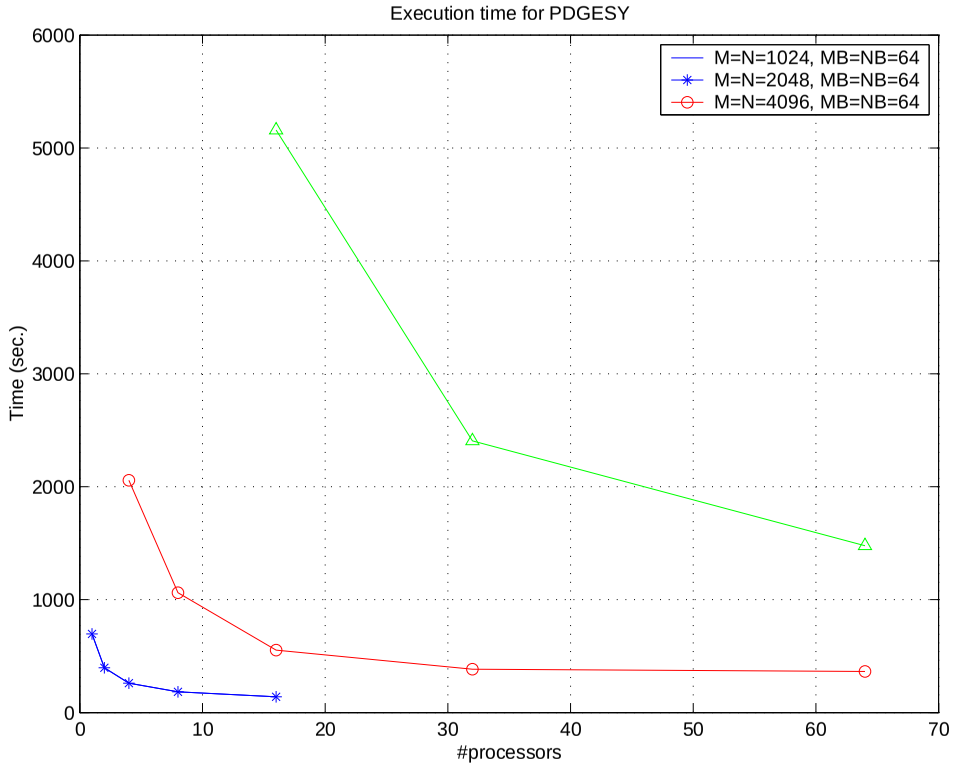
- PDTRSY is successfully used in parallel condition estimation for the SYCT equation.
- The implicit redistribution overhead caused by the 2x2 diagonal block split problem had no significant effect on the performance.
- The ”communicate on demand”-approach proved to be useful, even though a lot of data was communicated for throughout the computations. It was necessary to form the grid in such a way that a good loadbalance was acheived, otherwise the parallel performance was ruined.

Experiments (3)

Performance of PDGESY on IBM SP

$$AX - XB^T = C$$

M=N	MB	# proc	Sp
1024	64	1	1.0
1024	64	2	1.7
1024	64	4	2.7
1024	64	8	3.8
1024	64	16	5.0
2048	64	4	1.0
2048	64	8	1.9
2048	64	16	3.7
2048	64	32	5.4
2048	64	64	5.7
4096	64	16	1.0
4096	64	32	2.1
4096	64	64	3.5

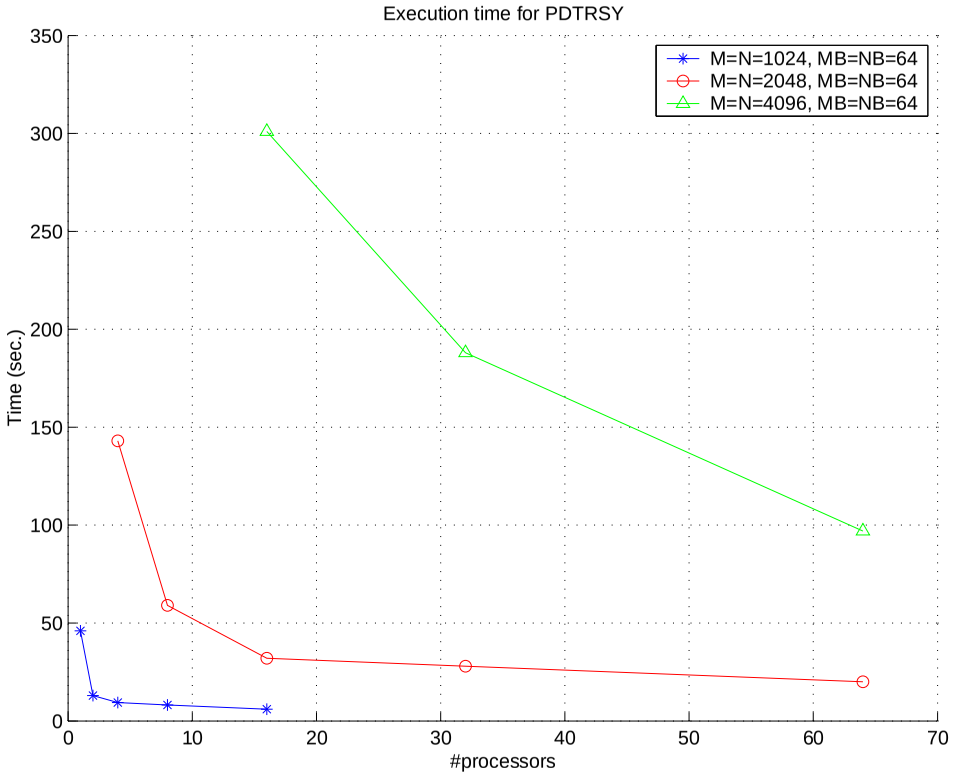


Experiments (4)

Performance of PDTRSY on IBM SP

$$AX - XB^T = C$$

M=N	MB	# proc	Sp
1024	64	1	1.0
1024	64	2	3.4
1024	64	4	4.9
1024	64	8	5.6
1024	64	16	7.7
2048	64	4	1.0
2048	64	8	2.4
2048	64	16	4.5
2048	64	32	5.1
2048	64	64	7.2
4096	64	16	1.0
4096	64	32	1.6
4096	64	64	3.1



Experiments (5)

Execution time profile for IBM SP system:
The Schur decomposition is the obvious bottleneck

$M = N$	MB	P_r	P_c	Step 1 (%)	Steps 2+4 (%)	Step 3 (%)	Total time (sec.)
1024	64	1	1	83	12	4	696
1024	64	2	1	90	6	3	397
1024	64	2	2	92	4	4	260
1024	64	4	2	89	5	4	183
1024	64	4	4	89	2	4	140
2048	64	2	2	81	12	7	2057
2048	64	4	2	85	6	9	1061
2048	64	4	4	90	3	6	553
2048	64	4	8	87	4	7	384
2048	64	8	8	86	3	5	364
4096	64	4	4	75	17	8	5158
4096	64	8	4	79	12	8	2407
4096	64	8	8	89	4	7	1478

Table 3. Execution time profile of PDGESY solving $AX - XB^T = C$.

Experiments (6)

- We also tried our implementation on seth, the HPC2N Super Cluster
- Now it became obvious that we had to form our grid to achieve a homogenous loadbalance, since communication is more expensive on seth measured in ***time(communicated byte) / time(performed flop).***
- Moreover, there is no guarantee that the logical mesh is really mapped onto a physical mesh.
- Since our test problem was square we got best performance from square process grid.
- Especially the performance of PDTRSY was affected by this change of target machine.

Parameter	Value	Reference/Method/Comments
t_a	1.4×10^{-9} sec.	Computed $\frac{T_p}{M^2N+MN^2}$ from experiments, see Figure 8
t_s	3.7×10^{-6} sec.	seth network application level latency [7]
t_w	4.4×10^{-8} sec.	Pallas ping-pong benchmark [7] for a message of size $\sim 2^{15}$ bytes.
t_s/t_a	2.6×10^3	IBM SP system knut: 4.7×10^3
t_w/t_a	31.4	IBM SP system knut: 9.4

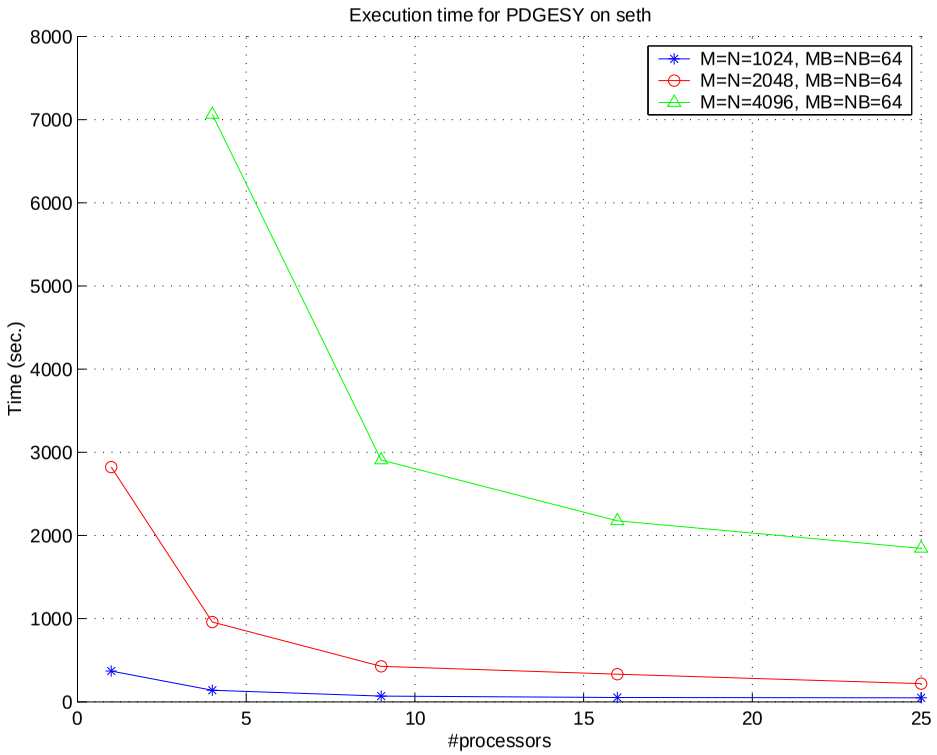
Table 2. Parallel execution modelling parameters on seth

Experiments (7)

Performance of PDGESY on Super Cluster

M=N	MB	# proc	Sp
1024	64	1	1.0
1024	64	4	2.7
1024	64	9	5.3
1024	64	16	7.1
1024	64	25	7.7
2048	64	1	1.0
2048	64	4	2.9
2048	64	9	6.6
2048	64	16	8.5
2048	64	25	13.0
4096	64	4	1.0
4096	64	9	2.4
4096	64	16	3.2
4096	64	25	3.8

$$AX - XB = C$$

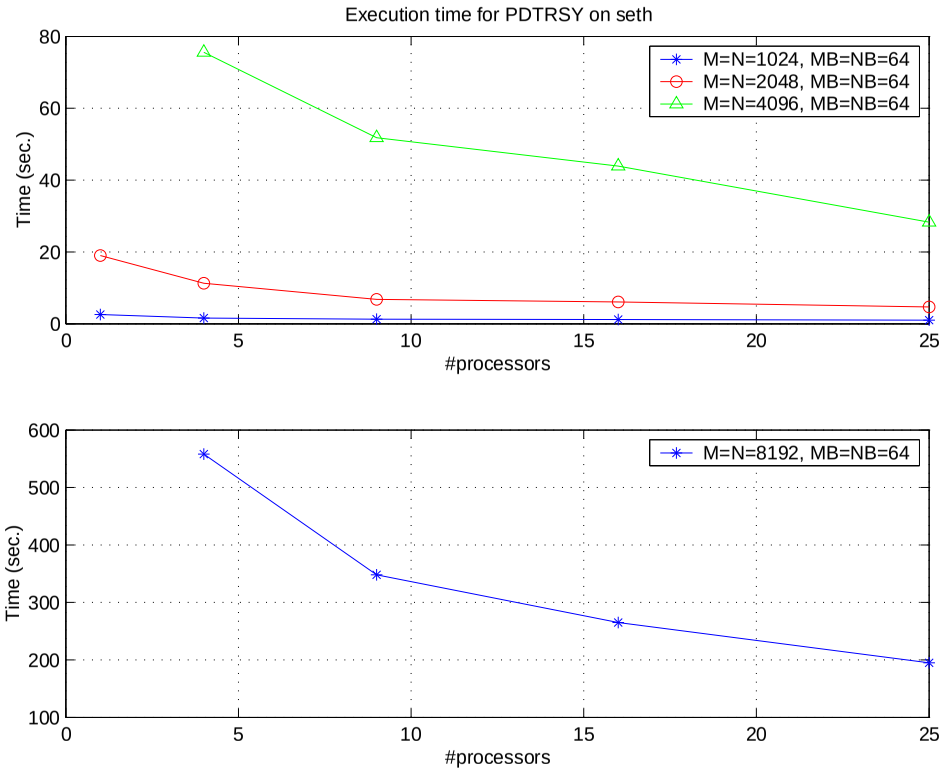


Experiments (8)

Performance of PDTRSY on Super Cluster

M=N	MB	# proc	Sp
1024	64	1	1.0
1024	64	4	1.6
1024	64	9	2.0
1024	64	16	2.2
1024	64	25	2.6
2048	64	1	1.0
2048	64	4	1.7
2048	64	9	2.8
2048	64	16	3.1
2048	64	25	4.0
4096	64	1	1.0
4096	64	4	1.7
4096	64	9	2.5
4096	64	16	2.9
4096	64	25	4.9
8192	64	4	1.0
8192	64	9	1.6
8192	64	16	2.1
8192	64	25	2.9

$$AX - XB = C$$



Obviously, it is not optimal for PDTRSY to use the same blocking factor for all problem sizes.

Experiments (6)

- We do condition estimation for SYCT on two examples:
  1. Well-conditioned: *A* and *B* have well-separated eigenvalues
  2. Ill-conditioned: *A* and *B* have some ”close” eigenvalues
- Each matrix is defined as *D* + *M*, where *D* is diagonal (holds all the eigenvalues) and *M* is strictly upper triangular.
- *M* is chosen as a random matrix with $|M(i,j)| \leq 1$
- Choice of *D*:
  1. 1st example: $\lambda_i(A) = i + 1000,\ \lambda_i(B) = i^{-1}$
  2. 2nd example: $\lambda_i(A) = i + \alpha,\ \lambda_i(B) = i^{-1}$
- The exact *Sep(A,B)* is computed using LAPACK routine DGESVD for evaluation of the estimate

Experiments (7): Condition estimation and accuracy

Ex.	*N*	*1/sep(A,B)*	*Est*	*Est*sep(A,B)*	*Iter*
1	1	1.0E-3	1.0E-3	1.0	1
1	2	1.0E-3	0.50E-3	0.50	4
1	4	1.0E-3	0.25E-3	0.25	5
1	8	1.0E-3	0.13E-3	0.13	5
1	16	1.0E-3	0.63E-4	0.063	5
1	32	1.0E-3	0.32E-4	0.032	5
1	64	1.0E-3	0.16E-4	0.016	5
2	1	1.0E14	1.0E13	0.10	1
2	2	2.1E14	9.0E12	0.043	4
2	4	2.1E14	9.0E13	0.43	5
2	8	4.0E14	6.2E13	0.16	5
2	16	7.1E14	1.6E14	0.23	5
2	32	9.6E14	2.3E15	2.4	5
2	64	4.9E15	8.4E15	1.7	5

Ex.	*N*	*Est*	*Iter*	$\|X-\tilde{X}\|_F / \|X\|_F$	$\|X-\tilde{X}\|_F$
1	512	2.3E-5	5	4.6E-16	1.3E-13
1	1024	1.3E-5	5	5.8E-16	3.4E-13
1	2048	9.3E-7	5	1.0E-15	1.2E-12
2	512	1.2E2	5	2.2E-12	6.5E-10
2	1024	2.9E5	5	6.9E-6	1.2E-8
2	2048	4.2E13	5	2.2E3	1.9

For the left table we used $\alpha = 1.0E-14$
and for the right we used $\alpha = 2.0$

The right table illustrates that a growing departure from normality in *A* and *B* makes the problem more ill-conditioned and finally singular.

Notice: Overestimating can occur as $\sigma_{\min}(Z_{SYCT})$ approaches ε_{mach}. This gives solutions of some subsystems with very large components and/or cancellation of terms in the update- and solving phase of the triangular solver.

Ongoing Work (1)

- The methods presented here can be applied to several similar problems.
- Our aim is to construct a ScaLAPACK-style software package of matrix equation solvers for distributed memory machines.
- The triangular solvers will be used in implementing parallel condition estimators for each matrix equation.

$op(A)X \pm Xop(B) = C$	SYCT	√
$op(A)X + Xop(A^T) = C$	LYCT	√
$op(A)Xop(B) \pm X = C$	SYDT	√
$op(A)Xop(A^T) - X = C$	LYDT	
$op(A)X \pm Yop(B) = C,$ $op(D)X \pm Yop(E) = F$	GCSY	
$op(A)Xop(B) \pm op(D)Xop(E) = C$	GSYL	
$op(A)Xop(A^T) - op(E)Xop(E^T) = C$	GLYCT	
$op(A)X(E^T) + op(E)Xop(A^T) = C$	GLYDT	

Ongoing Work (2)

- Derivation of communication probabilities and level of parallelism for the ”communicate on demand” approach.
- Theoretical scalability analysis and comparision with experimental results.
- Comparisions with other methods, e.g., iterative matrix sign-function based algorithms (Benner,Quintana-Orti ’99).
- Experiments on other // machines.
- Investigate // hybrid algorithms: explicit and recursive blocked (Jonsson –Kågström ’02) algorithms

Conclusions

- **Bartels-Stewart in practice:**
  1. *Hessenberg reduction + the QR-algorithm*
  2. *GEMM-operations*
  3. *Triangular solver*
  4. *GEMM-operations*
- If we use **explicit blocking**, these four operations can be parallelized. Our focus has been to develop and implement a parallel triangular solver.
- The parallel triangular solver is successfully used in parallel condition estimation of the SYCT equation
- The developed software is reliable and efficient
- Future research will focus on extending this work to other matrix equations as well as testing hybrid algorithms (recursive + explicit blocking) and comparing with other methods.

Some References

- R.H. Bartels and G.W. Stewart, Algorithm 432: Solution of the Equation AX+XB=C, *Comm. ACM*, 15(9):820-826, 1972.
- S. Blackford et al., *ScaLAPACK Users' Guide.* SIAM Publications, Philadelphia 1997.
- R. Granat, A Parallel ScaLAPACK-style Sylvester Solver, *Master Thesis,* UMNAD 435/03, Dept. Computing Science, Umeå University, Sweden, 2003.
- W.W. Hager, Condition Estimates, *SIAM J. Sci. Statist. Comput.* 5, pp. 311-316,1984.
- N. J. Higham, FORTAN codes for Estimating the One-Norm of a Real of Complex Matrix, *ACM Trans. Of Math. Software*, Vol. 14, No. 4, pp. 381-396, December 1988
- I. Jonsson and B. Kågström, Recursive Blocked Algorithms for Solving Triangular Matrix Equations - Part I: One-Sided and Coupled Sylvester-Type Equations, *ACM Trans. Math. Software*, Vol. 28, No. 4, pp 393-415, 2002.
- I. Jonsson and B. Kågström, Recursive Blocked Algorithms for Solving Triangular Matrix Equations - Part II: Two-Sided and Generalized Sylvester and Lyapunov Equations, *ACM Trans. Math. Software*, Vol. 28, No. 4, pp 416-435, 2002.
- B. Kågström and P. Poromaa, Distributed and shared memory block algorithms for the triangular Sylvester Equation with Sep^{-1} estimators, *SIAM J. Matrix Anal. Appl.,* 13 (1992), pp. 99-101.
- P. Poromaa, Parallel Algorithms for Triangular Sylvester Equations: Design, Scheduling and Scalability Issues. In Kågström et al. (eds), *Applied Parallel Computing. Large Scale Scientific and Industrial Problems*, Lecture Notes in Computer Science, Vol. 1541, pp. 438-446, Springer-Verlag, 1998.