Parallel Implementations of the One–Sided Indefinite Block Jacobi Methods

Sanja Singer

Faculty of Mechanical Engineering and Naval Architecture,
University of Zagreb, Croatia

e-mail: ssinger@math.hr, ssinger@fsb.hr

joint work with: Saša Singer, Vjeran Hari,
Krešimir Bokulić, Davor Davidović,
Marijan Jurešić and Aleksandar Uščumlić

IWASEP VII
Dubrovnik, Croatia
June 9–12, 2008

Motivation

Known facts:

- indefinite Jacobi algorithm (HSVD) computes eigenvalues (hyperbolic singular values) with high relative accuracy, when possible,
- Jacobi algorithm is easy to parallelize, especially if the one-sided strategy is used.

Goals

Our goals are:

- development of an efficient parallel algorithm, which locally (inside each processor) uses blocking,
- speedup on the single processor computer with two-level memory hierarchy, and on the cluster of processors, with multiple-level memory hierarchy.
- balance processor's jobs and reuse data inside each processor whenever it is possible.

Outline of the talk

Outline of the talk:

- brief description of the different Jacobi algorithms,
- blocking algorithms, some theoretical results,
- parallelization of the algorithm,
- some implementation details,
- numerical results,
- future work.

J–Jacobi algorithm

If Hermitian, indefinite H is given, the method consists of the following steps:

- Factorize H using symmetric indefinite factorization (with pivoting): $H = MDM^*$, D block diagonal. Additional diagonalization of D and scaling of columns of M yields

$$H = GJG^*, \quad J = \mathrm{diag}(j_{11}, \ldots, j_{nn}), \quad j_{ii} \in \{-1, 1\}.$$

- Note that multiplication of the eigenvalue problem

$$GJG^*x = \lambda x$$

from the left by G^*, and notation $z = JG^*x$ gives generalized eigenvalue problem

$$G^*Gz = \lambda Jz.$$

J–Jacobi algorithm (continued)

The pair (G^*G, J) is diagonalized by a sequence of J–unitary congruences (trigonometric and hyperbolic rotations)

- either explicitly (two-sided algorithm on $A := G^*G$),
- or implicitly (one-sided algorithm on G).

One-sided algorithm is equivalent to the hyperbolic SVD of G

$$G = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^*, \quad V^* J V = J.$$

Usually, we apply sequence of J–unitary congruences W_1, W_2, $\ldots$, W_z on the right-hand side of G, i.e.,

$$V^{-*} = W_1 \cdot W_2 \cdots W_z.$$

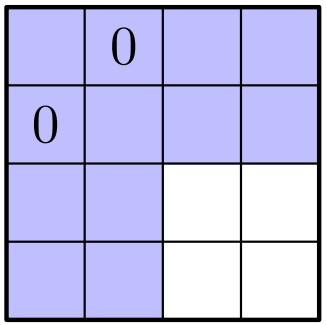
J–Jacobi algorithm (continued)

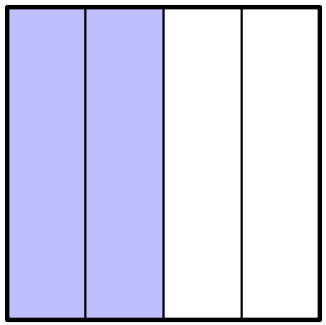
For example, if

$$J = \mathrm{diag}(1, -1, 1, -1)$$

and we choose row–cyclic strategy, we have

two-sided alg. on G^*G: one-sided alg. on G:





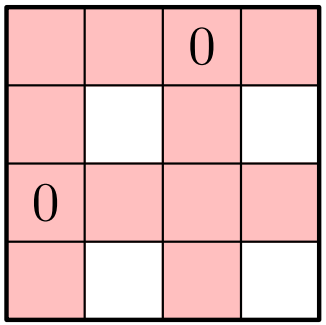
J–Jacobi algorithm (continued)

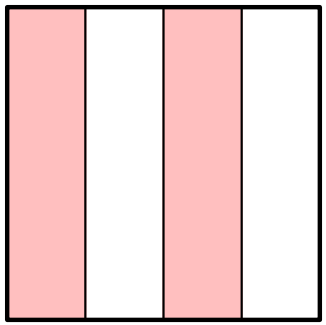
For example, if

$$J = \mathrm{diag}(1, -1, 1, -1)$$

and we choose row–cyclic strategy, we have

two-sided alg. on G^*G: one-sided alg. on G:





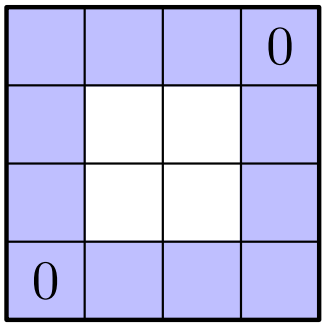
J–Jacobi algorithm (continued)

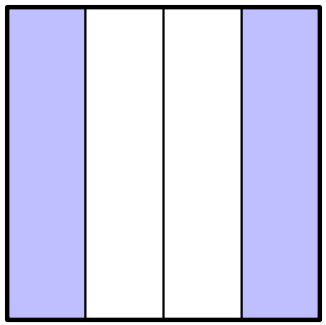
For example, if

$$J = \mathrm{diag}(1, -1, 1, -1)$$

and we choose row–cyclic strategy, we have

two-sided alg. on G^*G: one-sided alg. on G:





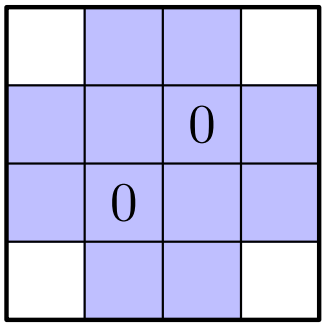
J–Jacobi algorithm (continued)

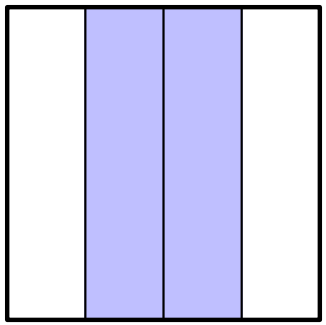
For example, if

$$J = \mathrm{diag}(1, -1, 1, -1)$$

and we choose row–cyclic strategy, we have

two-sided alg. on G^*G: one-sided alg. on G:





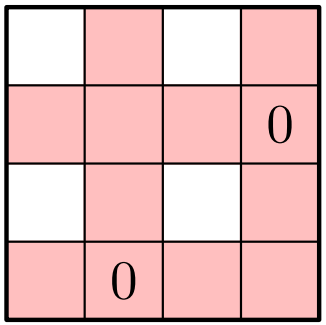
J–Jacobi algorithm (continued)

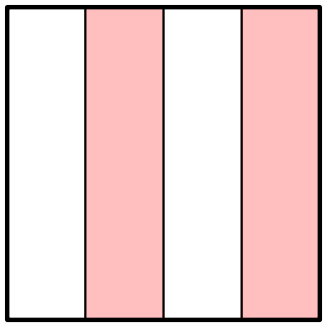
For example, if

$$J = \mathrm{diag}(1, -1, 1, -1)$$

and we choose row–cyclic strategy, we have

two-sided alg. on G^*G: one-sided alg. on G:





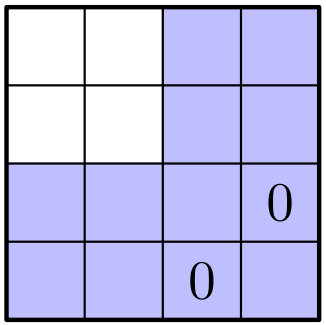
J–Jacobi algorithm (continued)

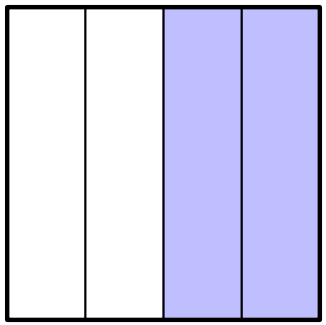
For example, if

$$J = \mathrm{diag}(1, -1, 1, -1)$$

and we choose row–cyclic strategy, we have

two-sided alg. on G^*G: one-sided alg. on G:





J–Jacobi algorithm (continued)

Note that:

- update of two columns in the one-sided algorithm is independent – update routine is `xROTM` BLAS 1 routine,
- columns on the previous pages could be block-columns, and we obtain a block algorithm – update routine is BLAS 3 matrix-matrix multiplication routine.

One-sided or two-sided, that is the question

In practice, the answer is very simple – one-sided algorithm.

One-sided algorithm is:

- more accurate,
- more than two times faster if vectorized routines are used (either compiler vectorization or BLAS from Math Kernel Library).

Note:

It is easier to describe algorithms as two-sided, and implementation is one-sided.

Block partitions

Suppose that G has the following block-column partition:

$$G = [G_1, G_2, \ldots, G_p],$$

where the number of columns (block size) in G_i is n_i, and J diagonal matrix such that

$$J = \mathrm{diag}(J_1, J_2, \ldots, J_p),$$

where J_i is of order n_i. This block partition naturally induces a "square block" partition of $A = G^*G$, with blocks

$$A_{ij} = G_i^* G_j.$$

One block step

In one step of the ordinary Jacobi algorithm, we choose a pivot matrix

$$\widehat{A} = \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ij}^* & a_{jj} \end{bmatrix}$$

and annihilate the off-diagonal element a_{ij}.

In one block step, we choose a pivot block matrix

$$\widehat{A} = \begin{bmatrix} A_{ii} & A_{ij} \\ A_{ij}^* & A_{jj} \end{bmatrix}.$$

and transform it as

$$A' = W^* \widehat{A} W.$$

The purpose is to make A' more diagonal than $\widehat{A}$.

Block algorithms

We distinguish two types of algorithms/strategies.

Block–oriented algorithms

- the norm of the off-diagonal block A_{ij} is only reduced.

Full block algorithms

- the off-diagonal block A_{ij} is annihilated.

We have two levels of pivot strategies:

- block level or macro strategies,
- micro level strategies inside each block.

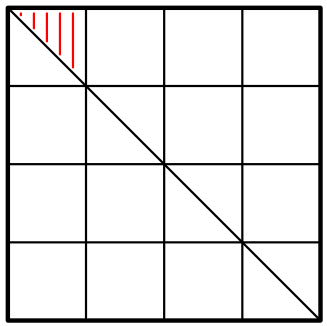
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



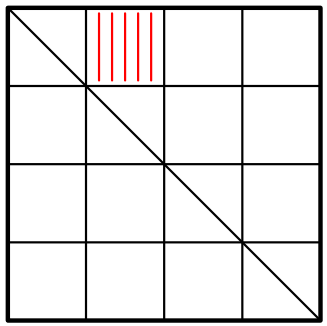
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



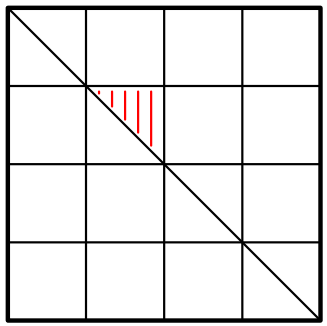
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



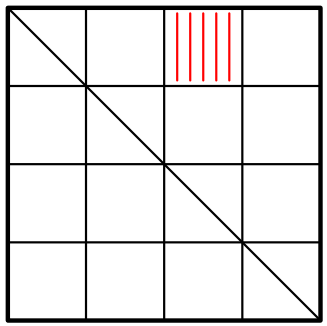
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



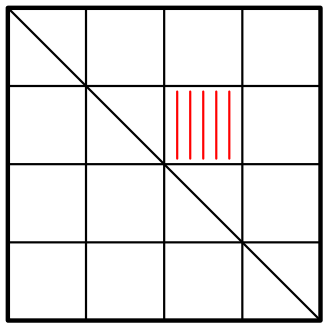
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



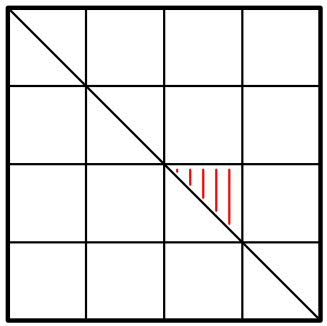
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



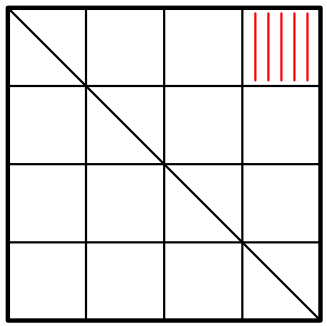
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



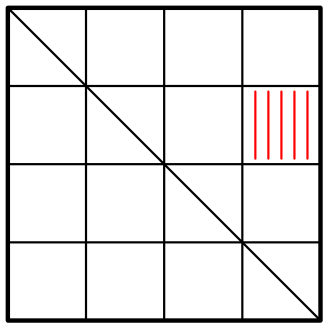
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



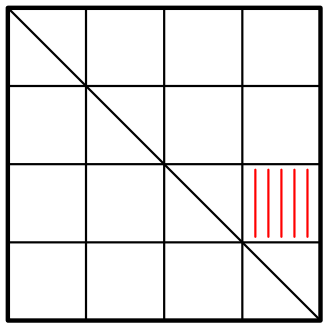
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



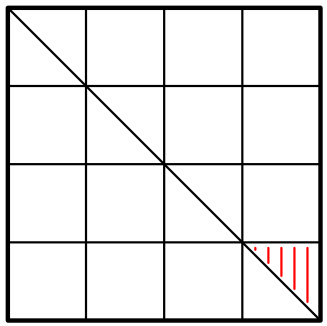
Cyclic block strategies

The simplest block–oriented algorithm is to rearrange one sweep of the ordinary cyclic Jacobi in a "block aware" manner.

For example, we can use

- the column-cyclic strategy on macro level
- and single sweep of column-cyclic strategy on micro level (left to right, top to bottom).

This strategy belongs to the family of wave-front orderings, which are equivalent to the column-cyclic strategy.



Full block algorithms

Full block algorithms annihilate the off-diagonal block A_{ij} in $\widehat{A}$ in each block–step.

- Annihilation of just A_{ij} is linearly slow.
- Solution: we should diagonalize whole pivot sub-matrices $\widehat{A}$.
- After certain number of steps (at worst after the full sweep) all diagonal blocks A_{ii} will be diagonal.
- This suggests the following preprocessing step: diagonalization of all A_{ii}, the diagonal can be stored in a separate vector, and updated after each sweep

$$\begin{bmatrix} \Lambda'_{ii} & 0 \\ 0 & \Lambda'_{jj} \end{bmatrix} = \begin{bmatrix} W_{ii} & W_{ij} \\ W_{ji} & W_{jj} \end{bmatrix}^* \begin{bmatrix} \Lambda_{ii} & A_{ij} \\ A^*_{ij} & \Lambda_{jj} \end{bmatrix} \begin{bmatrix} W_{ii} & W_{ij} \\ W_{ji} & W_{jj} \end{bmatrix}.$$

Practical tricks

Computational details:

- J in partitioned form is used, $J = I \oplus (-I)$,
- (normwise) column sorting of G after each sweep, with respect to I and $-I$,
- fast "quadratic convergence" stopping criterion,
- threshold annihilation strategy.

Theoretical results

Accuracy and convergence of block–oriented algorithms

All algorithms are accurate in the relative sense.
Convergence:

- is easy to prove.

Accuracy and convergence of full–block algorithms

Algorithms are accurate in the relative sense.

- we believe that global convergence can be proved (work in progress).

Modulus pivot strategy

Blocked variant of the modulus pivot strategy is an ideal choice as parallel pivot strategy.

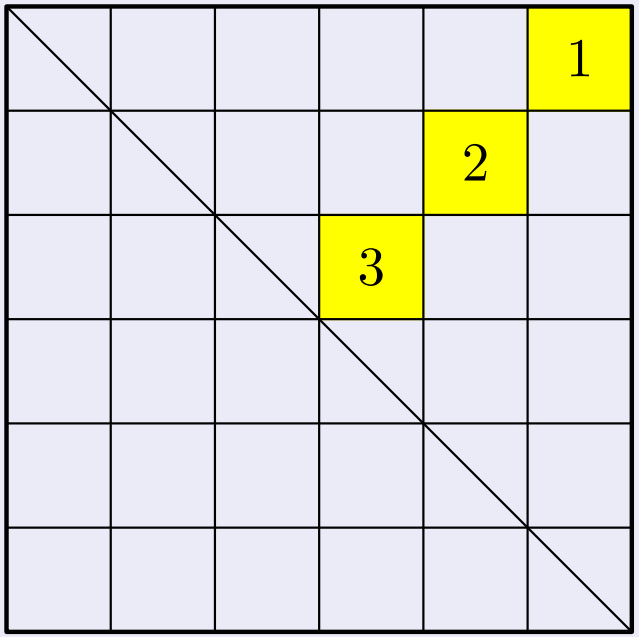
Partition of G:

- complete diagonaliztion job is divided in p tasks (p need not be the number of processors/cores – usually p is 1.5–2 times bigger),
- columns of G are partitioned in $2p-1$ (easier to implement, natural path is through the ring of processors) or $2p$ block–columns (probably a bit faster) such that:
  - block-columns 1 and last belong to task 1,
  - block-columns 2 and penultimate belong to task 2,
  - $\vdots$
  - one/two middle block-columns belong to task p.

Block–oriented algorithm

Single cycle in parallel

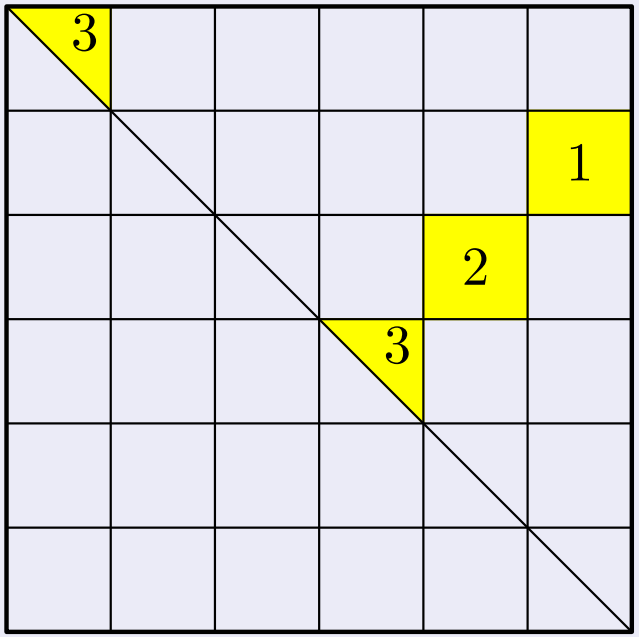
Suppose that A has 6 block–columns divided in 3 tasks.



Block–oriented algorithm

Single cycle in parallel

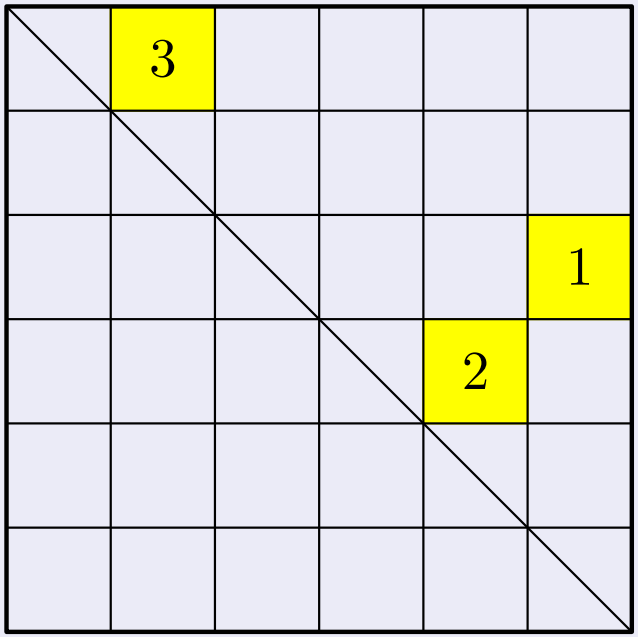
Suppose that A has 6 block–columns divided in 3 tasks.



Block–oriented algorithm

Single cycle in parallel

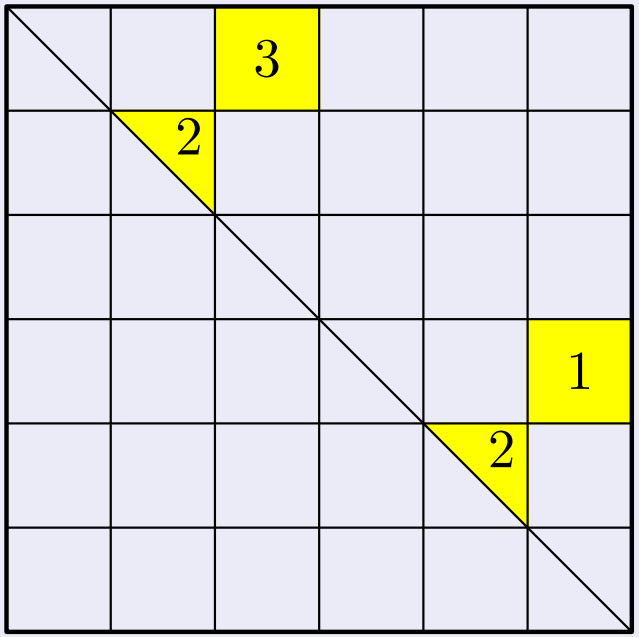
Suppose that A has 6 block–columns divided in 3 tasks.



Block–oriented algorithm

Single cycle in parallel

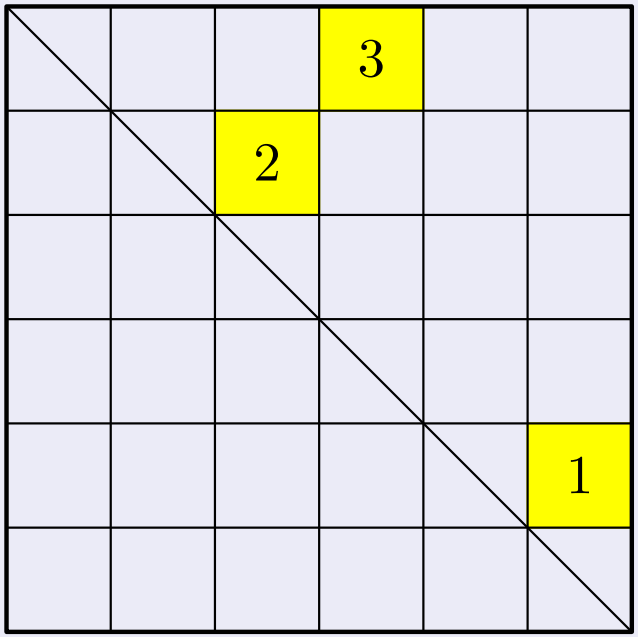
Suppose that A has 6 block–columns divided in 3 tasks.



Block–oriented algorithm

Single cycle in parallel

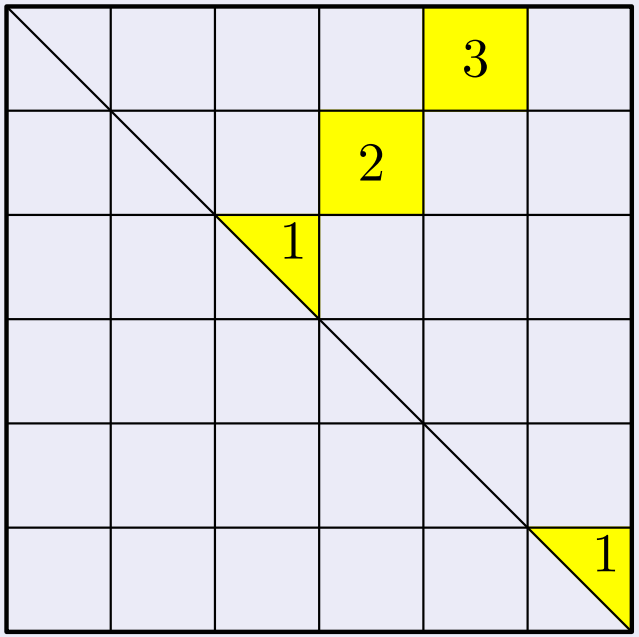
Suppose that A has 6 block–columns divided in 3 tasks.



Block–oriented algorithm

Single cycle in parallel

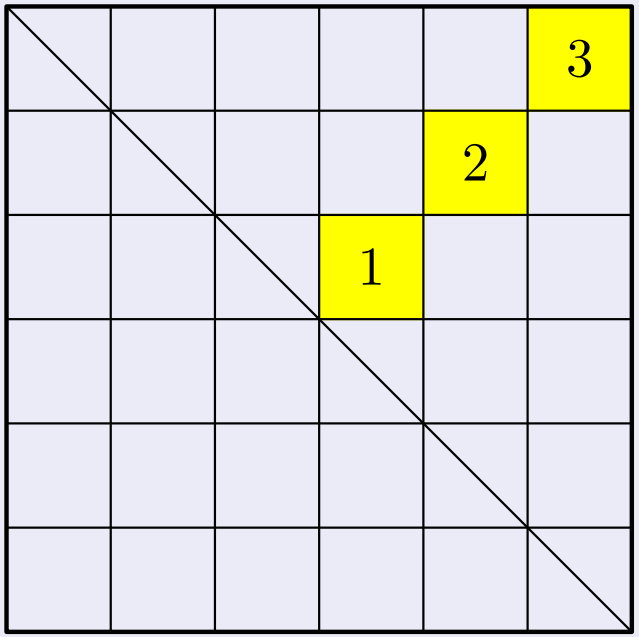
Suppose that A has 6 block–columns divided in 3 tasks.



Block–oriented algorithm

Single cycle in parallel

Suppose that A has 6 block–columns divided in 3 tasks.

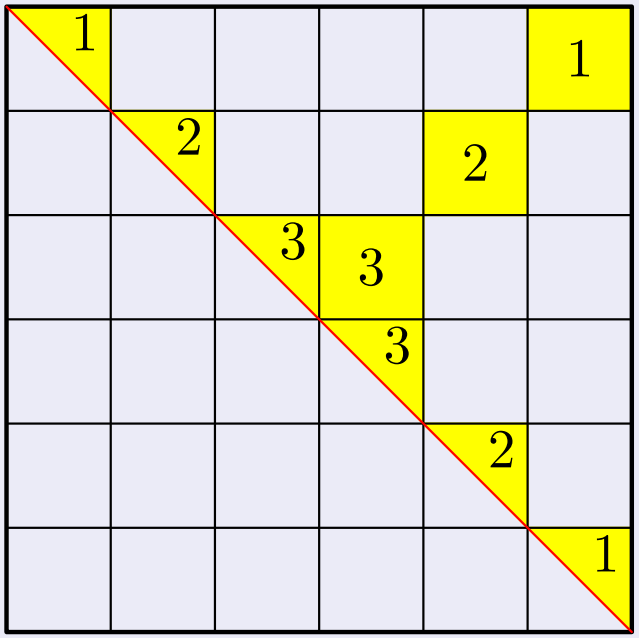


Note that in the end of the cycle, positions of block columns are "inverted".

Full–block algorithm

First step in parallel

Situation is similar to block–oriented algorithm, but each task is diagonalizing its own 2×2 block.



Note that diagonal blocks remain diagonal after the first step of the algorithm, and no additional perprocessing is needed.

Inner (micro level) strategies

Inside each task:

- we should use the block-oriented or the full block algorithm to obtain additional speedup.

Repartition of a local matrix

- Suppose that task k at time t is working on block–columns $G_{ij} = [G_i G_j]$.
- Single task is using block-oriented/full block algorithm on G_{ij}, i.e., G_{ij} should be repartitioned such that smaller block-columns fit well into the local cache memory.

Characteristics of the "Test computer"

18 physical PC computers connected via 100 Mbit switch

- Intel Core 2 Duo E6300 procesor @ 1.86 GHz,
- 1 GB DDR2 memory,
- 2 MB cache memory.

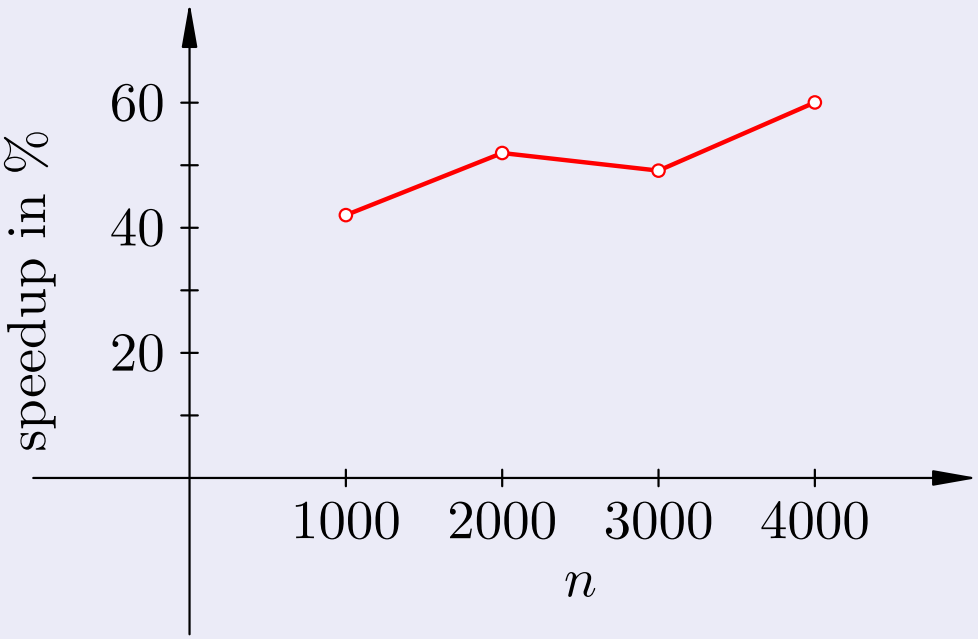
Used software:

- Linux Ubuntu 7.04, 32-bit,
- Intel Fortran 9.1.043,
- Intel Math Kernel Library 9.0,
- MPI, Open MP.

Two–level vs. three level cache

Speedup of blocked vs. non-blocked inner algorithm

Situation is similar to a single processor speedup, here 32 cores (16 processors) are used.



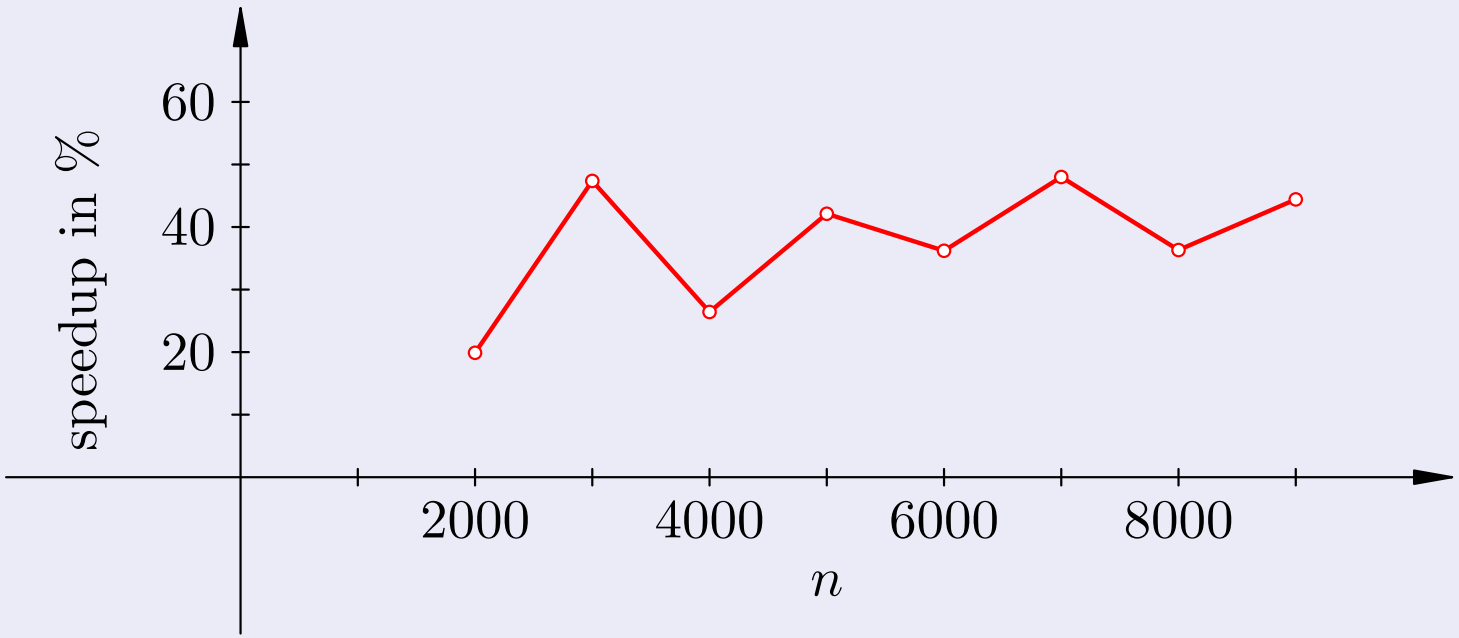
Speedup 8 vs. 4 processors

Testing on Isabella cluster

Characteristics of a cluster:

- $4 \times$ AMD Opteron (dual core),
- Infiniband connection (10 Gb/s).

Only few matrices are tested (very long waiting time to be scheduled for code execution).



Conclusion

Blocked algorithms are:

- more than 50% faster for suitably large dimensions,
- accurate in the relative sense.

Work in progress

- Efficient column sorting in parallel algorithms.
- Proof of asymptotic and quadratic convergence of parallel algorithms.
- Incorporation of QR factorization in three–level cache algorithms.
- Testing of various quasicyclic strategies.