Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

New Class of Block Matrix Orderings for the Parallel Two-Sided Jacobi SVD Algorithm

Gabriel Okša [1], Ondrej Sýkora [2], Marián Vajteršic [3]

[1]Institute of Mathematics
Slovak Academy of Sciences
Bratislava, Slovakia

[2]Department of Computer Science
Loughborough University
Loughborough, United Kingdom

[3]Institute of Scientific Computing
University of Salzburg
Salzburg, Austria

Outline

1 Problem formulation

2 Parallel Two-Sided Block-Jacobi Algorithm

3 Known Types of Block Ordering

4 New Clique-Based Block Ordering

5 First numerical results

Our task

Compute in parallel the Singular Value Decomposition (SVD) of a complex matrix A of the size $m \times n$, $m \geq n$:

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^H.,$$

where $U(m \times m)$ and $V(n \times n)$ are orthogonal and $\Sigma = \mathrm{diag}(\sigma_i)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$.
Numerically stable way of computation:

- one- or two-sided block-Jacobi methods;
- large degree of parallelism.

Target architecture:

- distributed memory machines (parallel supercomputers and clusters) with Message Passing Interface (MPI).

Our task

Compute in parallel the Singular Value Decomposition (SVD) of a complex matrix A of the size $m \times n$, $m \geq n$:

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^H.,$$

where $U(m \times m)$ and $V(n \times n)$ are orthogonal and $\Sigma = \mathrm{diag}(\sigma_i)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$.
Numerically stable way of computation:

- one- or two-sided block-Jacobi methods;
- large degree of parallelism.

Target architecture:

- distributed memory machines (parallel supercomputers and clusters) with Message Passing Interface (MPI).

Our task

Compute in parallel the Singular Value Decomposition (SVD) of a complex matrix A of the size $m \times n$, $m \geq n$:

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^H.,$$

where $U(m \times m)$ and $V(n \times n)$ are orthogonal and $\Sigma = \mathrm{diag}(\sigma_i)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$.
Numerically stable way of computation:

- one- or two-sided block-Jacobi methods;
- large degree of parallelism.

Target architecture:

- distributed memory machines (parallel supercomputers and clusters) with Message Passing Interface (MPI).

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Main structure of algorithm

- Implemented on p processors, an even blocking factor $\ell = 2p$ is used together with some ordering of subproblems.
- Each processor contains 2 block columns of A, U, V.
- Algorithm is based on the SVDs of 2×2-block subproblems:

$$S_{ij} = \begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix} .$$

- Global termination criterion:

$$F(A,\ell) = \sqrt{\sum_{i,j=1,\, i\neq j}^{\ell} \|A_{ij}\|_F^2} < \epsilon\,, \quad \epsilon \equiv prec \cdot \|A\|_F\,.$$

Main structure of algorithm

- Implemented on p processors, an even blocking factor $\ell = 2p$ is used together with some ordering of subproblems.
- Each processor contains 2 block columns of A, U, V.
- Algorithm is based on the SVDs of 2×2-block subproblems:

$$S_{ij} = \begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix} .$$

- Global termination criterion:

$$F(A,\ell) = \sqrt{\sum_{i,j=1,\, i\neq j}^{\ell} \|A_{ij}\|_F^2} < \epsilon\,, \quad \epsilon = prec \cdot \|A\|_F\,.$$

Main structure of algorithm

- Implemented on p processors, an even blocking factor $\ell = 2p$ is used together with some ordering of subproblems.
- Each processor contains 2 block columns of A, U, V.
- Algorithm is based on the SVDs of 2×2-block subproblems:

$$S_{ij} = \begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix} .$$

- Global termination criterion:

$$F(A,\ell) = \sqrt{\sum_{i,j=1,\, i\neq j}^{\ell} \|A_{ij}\|_F^2} < \epsilon\,, \quad \epsilon \equiv prec \cdot \|A\|_F\,.$$

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Main structure of algorithm

- Implemented on p processors, an even blocking factor $\ell = 2p$ is used together with some ordering of subproblems.
- Each processor contains 2 block columns of A, U, V.
- Algorithm is based on the SVDs of 2×2-block subproblems:

$$S_{ij} = \begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix} .$$

- Global termination criterion:

$$F(A, \ell) = \sqrt{\sum_{i,j=1, i \neq j}^{\ell} \|A_{ij}\|_{\mathrm{F}}^2} < \epsilon \,, \quad \epsilon \equiv prec \cdot \|A\|_{\mathrm{F}} \,.$$

Main structure of algorithm (cont.)

- Local termination criterion:

$$F(S_{ij}, \ell) = \sqrt{\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2} < \delta\,, \quad \delta \equiv \epsilon \cdot \sqrt{\frac{2}{\ell\,(\ell-1)}}\,.$$

- Main question: How to order 2×2-block subproblems to be efficient?

Main structure of algorithm (cont.)

- Local termination criterion:

$$F(S_{ij},\ell) = \sqrt{\|A_{ij}\|_F^2 + \|A_{ji}\|_F^2} < \delta\,, \quad \delta \equiv \epsilon \cdot \sqrt{\frac{2}{\ell\,(\ell-1)}}\,.$$

- Main question: How to order 2×2-block subproblems to be efficient?

Cyclic block ordering

- Let the matrix A be cut into $\ell \times \ell$ block structure with $\ell = 2p$ where p is the number of processors..
- Cyclic block-row ordering define 2×2block subproblems by block rows: (A_{12}, A_{21}), $(A_{13}, A_{31}), \ldots, (A_{1\ell}, A_{\ell 1})$, (A_{23}, A_{32}), (A_{24}, A_{42}), $\ldots, (A_{2\ell}, A_{\ell 2})$, $\ldots, (A_{\ell-1,\ell}, A_{\ell,\ell-1})$.
- Cyclic bloc-column ordering is defined similarly.
- Parallel method: Take p consecutive pairs from the list, each pair defines one 2×2-block SVD subproblem for one processor.

Cyclic block ordering

- Let the matrix A be cut into $\ell \times \ell$ block structure with $\ell = 2p$ where p is the number of processors..
- Cyclic block-row ordering define 2×2block subproblems by block rows: (A_{12}, A_{21}), $(A_{13}, A_{31}), \ldots, (A_{1\ell}, A_{\ell 1})$, (A_{23}, A_{32}), (A_{24}, A_{42}), $\ldots, (A_{2\ell}, A_{\ell 2})$, $\ldots, (A_{\ell-1,\ell}, A_{\ell,\ell-1})$.
- Cyclic bloc-column ordering is defined similarly.
- Parallel method: Take p consecutive pairs from the list, each pair defines one 2×2-block SVD subproblem for one processor.

Cyclic block ordering

New Block Orderings

Gabriel Okša

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

- Let the matrix A be cut into $\ell \times \ell$ block structure with $\ell = 2p$ where p is the number of processors..
- Cyclic block-row ordering define 2×2block subproblems by block rows:
  (A_{12}, A_{21}), $(A_{13}, A_{31}), \ldots, (A_{1\ell}, A_{\ell 1})$,
  (A_{23}, A_{32}), (A_{24}, A_{42}), $\ldots, (A_{2\ell}, A_{\ell 2})$,
  $\ldots, (A_{\ell-1,\ell}, A_{\ell,\ell-1})$.
- Cyclic bloc-column ordering is defined similarly.
- Parallel method: Take p consecutive pairs from the list, each pair defines one 2×2-block SVD subproblem for one processor.

Cyclic block ordering

- Let the matrix A be cut into $\ell \times \ell$ block structure with $\ell = 2p$ where p is the number of processors..
- Cyclic block-row ordering define 2×2block subproblems by block rows: $(A_{12}, A_{21}),\ (A_{13}, A_{31}), \ldots, (A_{1\ell}, A_{\ell 1})$, $(A_{23}, A_{32}),\ (A_{24}, A_{42}),\ \ldots, (A_{2\ell}, A_{\ell 2})$, $\ldots, (A_{\ell-1,\ell}, A_{\ell,\ell-1})$.
- Cyclic bloc-column ordering is defined similarly.
- Parallel method: Take p consecutive pairs from the list, each pair defines one 2×2-block SVD subproblem for one processor.

Cyclic block ordering (cont.)

- During one sweep of the method each non-diagonal block of A is nullified exactly once.
- Main problem: The ordering is prescribed and fixed, there is no consideration about an *actual status* (e.g., Frobenius norm) of individual blocks—may be very unefficient in decreasing the off-norm.
- Significant amount of data (e.g., whole column blocks) need to be explicitly transferred among processors at the beginning of each parallel step.

Cyclic block ordering (cont.)

- During one sweep of the method each non-diagonal block of A is nullified exactly once.
- Main problem: The ordering is prescribed and fixed, there is no consideration about an *actual status* (e.g., Frobenius norm) of individual blocks—may be very unefficient in decreasing the off-norm.
- Significant amount of data (e.g., whole column blocks) need to be explicitly transferred among processors at the beginning of each parallel step.

Cyclic block ordering (cont.)

- During one sweep of the method each non-diagonal block of A is nullified exactly once.
- Main problem: The ordering is prescribed and fixed, there is no consideration about an *actual status* (e.g., Frobenius norm) of individual blocks—may be very unefficient in decreasing the off-norm.
- Significant amount of data (e.g., whole column blocks) need to be explicitly transferred among processors at the beginning of each parallel step.

Dynamic Block Ordering

- Main idea: Take into consideration the actual status of a matrix and, in one parallel iterations step (p subproblems) *decrease the off-norm as much as possible*.
- This task can be translated into the language of graph theory:
  1. Construct the weighted complete graph $\mathcal{G}$ with ℓ vertices ($\ell = 2p$ is the blocking factor), where the edge (i, j) has the weight $\|A_{ij}\|_F^2 + \|A_{ji}\|_F^2$.
  2. At the beginning of each parallel iteration step, find the maximum-weight perfect matching on $\mathcal{G}$, which defines the p SVD subproblems—only *polynomial* complexity $O(\ell^3)$.

Dynamic Block Ordering

- Main idea: Take into consideration the actual status of a matrix and, in one parallel iterations step (p subproblems) *decrease the off-norm as much as possible*.
- This task can be translated into the language of graph theory:
  1. Construct the weighted complete graph $\mathcal{G}$ with ℓ vertices ($\ell = 2p$ is the blocking factor), where the edge (i, j) has the weight $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.
  2. At the beginning of each parallel iteration step, find the maximum-weight perfect matching on $\mathcal{G}$, which defines the p SVD subproblems—only *polynomial* complexity $O(\ell^3)$.

Dynamic Block Ordering

New Block Orderings

Gabriel Okša

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

- Main idea: Take into consideration the actual status of a matrix and, in one parallel iterations step (p subproblems) *decrease the off-norm as much as possible*.
- This task can be translated into the language of graph theory:
  1. Construct the weighted complete graph $\mathcal{G}$ with ℓ vertices ($\ell = 2p$ is the blocking factor), where the edge (i, j) has the weight $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.
  2. At the beginning of each parallel iteration step, find the maximum-weight perfect matching on $\mathcal{G}$, which defines the p SVD subproblems—only *polynomial* complexity $O(\ell^3)$.

Dynamic Block Ordering

- Main idea: Take into consideration the actual status of a matrix and, in one parallel iterations step (p subproblems) *decrease the off-norm as much as possible*.
- This task can be translated into the language of graph theory:
  1. Construct the weighted complete graph $\mathcal{G}$ with ℓ vertices ($\ell = 2p$ is the blocking factor), where the edge (i, j) has the weight $\|A_{ij}\|_\mathrm{F}^2 + \|A_{ji}\|_\mathrm{F}^2$.
  2. At the beginning of each parallel iteration step, find the maximum-weight perfect matching on $\mathcal{G}$, which defines the p SVD subproblems—only *polynomial* complexity $O(\ell^3)$.

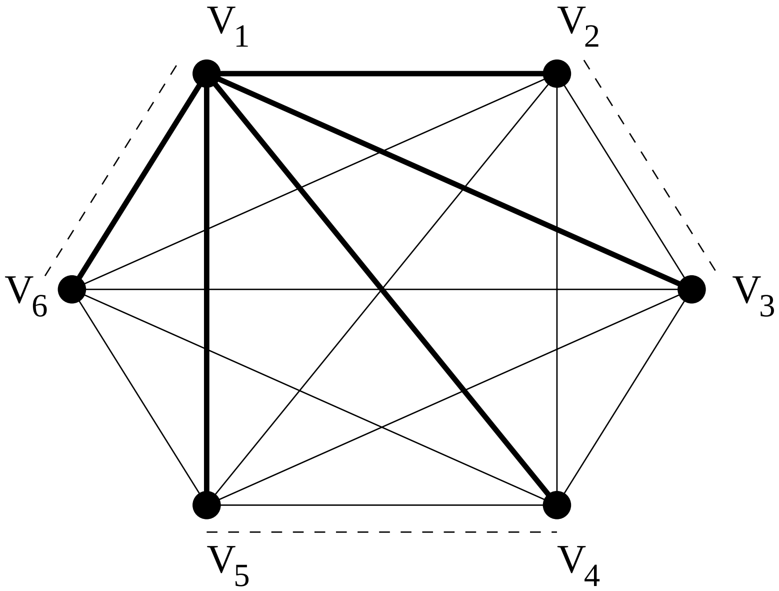


Figure: Maximum-weight perfect matching for $\ell = 6$ ($p = 3$).

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Clique-Based Block Ordering

- Physical blocking factor: for p processors, let A be partitioned into p block columns and block rows, each processor contains one block column.
- Logical blocking factor: $\ell = p/r$ for some integer r.
- In each parallel iteration step, ℓ local SVDs are computed in parallel where each SVD is of block-order $r \times r$ and comprises $2r$ off-diagonal blocks of A.
- Our aim: Decrease the off-diagonal Frobenius norm in each parallel iteration step as much as possible.

Clique-Based Block Ordering

- Physical blocking factor: for p processors, let A be partitioned into p block columns and block rows, each processor contains one block column.
- Logical blocking factor: $\ell = p/r$ for some integer r.
- In each parallel iteration step, ℓ local SVDs are computed in parallel where each SVD is of block-order $r \times r$ and comprises $2r$ off-diagonal blocks of A.
- Our aim: Decrease the off-diagonal Frobenius norm in each parallel iteration step as much as possible.

Clique-Based Block Ordering

- Physical blocking factor: for p processors, let A be partitioned into p block columns and block rows, each processor contains one block column.
- Logical blocking factor: $\ell = p/r$ for some integer r.
- In each parallel iteration step, ℓ local SVDs are computed in parallel where each SVD is of block-order $r \times r$ and comprises $2r$ off-diagonal blocks of A.
- Our aim: Decrease the off-diagonal Frobenius norm in each parallel iteration step as much as possible.

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Clique-Based Block Ordering

- Physical blocking factor: for p processors, let A be partitioned into p block columns and block rows, each processor contains one block column.
- Logical blocking factor: $\ell = p/r$ for some integer r.
- In each parallel iteration step, ℓ local SVDs are computed in parallel where each SVD is of block-order $r \times r$ and comprises $2r$ off-diagonal blocks of A.
- Our aim: Decrease the off-diagonal Frobenius norm in each parallel iteration step as much as possible.

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Graph-Theoretical Formulation

- Let $\mathcal{G}$ be a complete graph with p vertices ($=$ physical blocking factor). Let the edge (i,j) be weighted by $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.
- Task: Partition $\mathcal{G}$ into $\ell/2$ disjoint cliques of size r where the weight of cliques (i.e., the sum of weights through the edges belonging to chosen cliques) is maximized.
- For $r = 2$ this is just the maximum-weight perfect matching used in the Dynamic Ordering.
- Main problem: For $r \geq 3$, this task is NP-hard.

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Graph-Theoretical Formulation

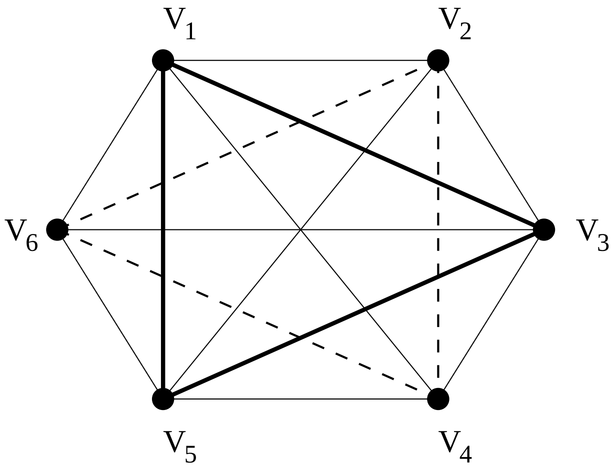
- Let $\mathcal{G}$ be a complete graph with p vertices ($=$ physical blocking factor). Let the edge (i, j) be weighted by $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.
- Task: Partition $\mathcal{G}$ into $\ell/2$ disjoint cliques of size r where the weight of cliques (i.e., the sum of weights through the edges belonging to chosen cliques) is maximized.
- For $r = 2$ this is just the maximum-weight perfect matching used in the Dynamic Ordering.
- Main problem: For $r \geq 3$, this task is NP-hard.

Graph-Theoretical Formulation

- Let $\mathcal{G}$ be a complete graph with p vertices ($=$ physical blocking factor). Let the edge (i,j) be weighted by $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.
- Task: Partition $\mathcal{G}$ into $\ell/2$ disjoint cliques of size r where the weight of cliques (i.e., the sum of weights through the edges belonging to chosen cliques) is maximized.
- For $r = 2$ this is just the maximum-weight perfect matching used in the Dynamic Ordering.
- Main problem: For $r \geq 3$, this task is NP-hard.

Graph-Theoretical Formulation

- Let $\mathcal{G}$ be a complete graph with p vertices ($=$ physical blocking factor). Let the edge (i,j) be weighted by $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.
- Task: Partition $\mathcal{G}$ into $\ell/2$ disjoint cliques of size r where the weight of cliques (i.e., the sum of weights through the edges belonging to chosen cliques) is maximized.
- For $r = 2$ this is just the maximum-weight perfect matching used in the Dynamic Ordering.
- Main problem: For $r \geq 3$, this task is NP-hard.



CLIQUE 1: INDICES (1, 3, 5) (blocks and processors)

CLIQUE 2: INDICES (2, 4, 6) (blocks and processors)

Figure: Clique-based block ordering for $p = 6$, $r = 3$.

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Example (cont.)

- In the example above there are 2 local SVDs defined by submatrices with block indices (1, 3, 5) and (2, 4, 6):

$$\mathrm{CL1} = \begin{pmatrix} A_{11} & A_{13} & A_{15} \\ A_{31} & A_{33} & A_{35} \\ A_{51} & A_{53} & A_{55} \end{pmatrix}, \quad \mathrm{CL2} = \begin{pmatrix} A_{22} & A_{24} & A_{26} \\ A_{42} & A_{44} & A_{46} \\ A_{62} & A_{64} & A_{66} \end{pmatrix}.$$

- MPI library: The SVD of $\mathrm{CL1}$ is computed in the context that consists of processors 1, 3 and 5.
  Similarly, the SVD of $\mathrm{CL2}$ is computed in the context that consists of processors 2, 4 and 6.
- In general, there are ℓ contexts, each with r processors.
- Important: The matrix data is not moved explicitly between processors at the beginning of a parallel iteration step—just new contexts are created.

Example (cont.)

- In the example above there are 2 local SVDs defined by submatrices with block indices (1, 3, 5) and (2, 4, 6):

$$\mathrm{CL1} = \begin{pmatrix} A_{11} & A_{13} & A_{15} \\ A_{31} & A_{33} & A_{35} \\ A_{51} & A_{53} & A_{55} \end{pmatrix}, \quad \mathrm{CL2} = \begin{pmatrix} A_{22} & A_{24} & A_{26} \\ A_{42} & A_{44} & A_{46} \\ A_{62} & A_{64} & A_{66} \end{pmatrix}.$$

- MPI library: The SVD of $\mathrm{CL1}$ is computed in the context that consists of processors 1, 3 and 5.
  Similarly, the SVD of $\mathrm{CL2}$ is computed in the context that consists of processors 2, 4 and 6.
- In general, there are ℓ contexts, each with r processors.
- Important: The matrix data is not moved explicitly between processors at the beginning of a parallel iteration step—just new contexts are created.

Example (cont.)

- In the example above there are 2 local SVDs defined by submatrices with block indices (1, 3, 5) and (2, 4, 6):

$$\mathrm{CL1} = \begin{pmatrix} A_{11} & A_{13} & A_{15} \\ A_{31} & A_{33} & A_{35} \\ A_{51} & A_{53} & A_{55} \end{pmatrix}, \quad \mathrm{CL2} = \begin{pmatrix} A_{22} & A_{24} & A_{26} \\ A_{42} & A_{44} & A_{46} \\ A_{62} & A_{64} & A_{66} \end{pmatrix}.$$

- MPI library: The SVD of $\mathrm{CL1}$ is computed in the context that consists of processors 1, 3 and 5.
  Similarly, the SVD of $\mathrm{CL2}$ is computed in the context that consists of processors 2, 4 and 6.
- In general, there are ℓ contexts, each with r processors.
- Important: The matrix data is not moved explicitly between processors at the beginning of a parallel iteration step—just new contexts are created.

Example (cont.)

- In the example above there are 2 local SVDs defined by submatrices with block indices $(1, 3, 5)$ and $(2, 4, 6)$:

$$\mathrm{CL1} = \begin{pmatrix} A_{11} & A_{13} & A_{15} \\ A_{31} & A_{33} & A_{35} \\ A_{51} & A_{53} & A_{55} \end{pmatrix}, \quad \mathrm{CL2} = \begin{pmatrix} A_{22} & A_{24} & A_{26} \\ A_{42} & A_{44} & A_{46} \\ A_{62} & A_{64} & A_{66} \end{pmatrix}.$$

- MPI library: The SVD of $\mathrm{CL1}$ is computed in the context that consists of processors $1, 3$ and 5.
  Similarly, the SVD of $\mathrm{CL2}$ is computed in the context that consists of processors $2, 4$ and 6.
- In general, there are ℓ contexts, each with r processors.
- Important: The matrix data is not moved explicitly between processors at the beginning of a parallel iteration step—just new contexts are created.

Work in one parallel iteration step

1. Given p, r, ℓ and actual weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$, find ℓ maximum-weight disjoint cliques using a genetic serial algorithm in processor 1—this defines the new block ordering.
2. Broadcast the new block ordering to all processors.
3. Delete old contexts.
4. Create ℓ contexts based on the new ordering, each context of size r, and compute ℓ SVDs of block size $r \times r$ in parallel by the two-sided Jacobi method.
5. Update left and right singular vectors in all processors by matrix multiplications.
6. Update weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.

Work in one parallel iteration step

1. Given p, r, ℓ and actual weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$, find ℓ maximum-weight disjoint cliques using a genetic serial algorithm in processor 1—this defines the new block ordering.
2. Broadcast the new block ordering to all processors.
3. Delete old contexts.
4. Create ℓ contexts based on the new ordering, each context of size r, and compute ℓ SVDs of block size $r \times r$ in parallel by the two-sided Jacobi method.
5. Update left and right singular vectors in all processors by matrix multiplications.
6. Update weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.

Work in one parallel iteration step

1. Given p, r, ℓ and actual weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$, find ℓ maximum-weight disjoint cliques using a genetic serial algorithm in processor 1—this defines the new block ordering.
2. Broadcast the new block ordering to all processors.
3. Delete old contexts.
4. Create ℓ contexts based on the new ordering, each context of size r, and compute ℓ SVDs of block size $r \times r$ in parallel by the two-sided Jacobi method.
5. Update left and right singular vectors in all processors by matrix multiplications.
6. Update weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.

New Block Orderings

Gabriel Okša

Problem formulation

Parallel Two-Sided Block-Jacobi Algorithm

Known Types of Block Ordering

New Clique-Based Block Ordering

First numerical results

Work in one parallel iteration step

1. Given p, r, ℓ and actual weights $\|A_{ij}\|_F^2 + \|A_{ji}\|_F^2$, find ℓ maximum-weight disjoint cliques using a genetic serial algorithm in processor 1—this defines the new block ordering.
2. Broadcast the new block ordering to all processors.
3. Delete old contexts.
4. Create ℓ contexts based on the new ordering, each context of size r, and compute ℓ SVDs of block size $r \times r$ in parallel by the two-sided Jacobi method.
5. Update left and right singular vectors in all processors by matrix multiplications.
6. Update weights $\|A_{ij}\|_F^2 + \|A_{ji}\|_F^2$.

Work in one parallel iteration step

1. Given p, r, ℓ and actual weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$, find ℓ maximum-weight disjoint cliques using a genetic serial algorithm in processor 1—this defines the new block ordering.
2. Broadcast the new block ordering to all processors.
3. Delete old contexts.
4. Create ℓ contexts based on the new ordering, each context of size r, and compute ℓ SVDs of block size $r \times r$ in parallel by the two-sided Jacobi method.
5. Update left and right singular vectors in all processors by matrix multiplications.
6. Update weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.

Work in one parallel iteration step

1. Given p, r, ℓ and actual weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$, find ℓ maximum-weight disjoint cliques using a genetic serial algorithm in processor 1—this defines the new block ordering.
2. Broadcast the new block ordering to all processors.
3. Delete old contexts.
4. Create ℓ contexts based on the new ordering, each context of size r, and compute ℓ SVDs of block size $r \times r$ in parallel by the two-sided Jacobi method.
5. Update left and right singular vectors in all processors by matrix multiplications.
6. Update weights $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$.

Genetic algorithm

- The serial genetic algorithm for the maximum-weight clique partition has been implemented in `C++` using the free available library `GALIB (v.2.4.6)` from MIT, USA.
- The genome is the $2D$ binary string of dimensions $\ell \times p$ that represents one partition into disjoint cliques.
- The crossover function creates two new genomes (son and daughter) from two old genomes (parents) by choosing the heavier individual cliques (better parent is repeated).
- The objective function, which should be maximized, is the weight of a genome.
- Typical run:
  `population size` $= 20$,
  `number of generations` $= 10^4$.

Genetic algorithm

- The serial genetic algorithm for the maximum-weight clique partition has been implemented in `C++` using the free available library `GALIB (v.2.4.6)` from MIT, USA.
- The genome is the *2D* binary string of dimensions $\ell \times p$ that represents one partition into disjoint cliques.
- The crossover function creates two new genomes (son and daughter) from two old genomes (parents) by choosing the heavier individual cliques (better parent is repeated).
- The objective function, which should be maximized, is the weight of a genome.
- Typical run:
  `population size` $= 20$,
  `number of generations` $= 10^4$.

Genetic algorithm

- The serial genetic algorithm for the maximum-weight clique partition has been implemented in `C++` using the free available library `GALIB (v.2.4.6)` from MIT, USA.
- The genome is the *2D* binary string of dimensions $\ell \times p$ that represents one partition into disjoint cliques.
- The crossover function creates two new genomes (son and daughter) from two old genomes (parents) by choosing the heavier individual cliques (better parent is repeated).
- The objective function, which should be maximized, is the weight of a genome.
- Typical run:
  `population size` $= 20$,
  `number of generations` $= 10^4$.

Genetic algorithm

- The serial genetic algorithm for the maximum-weight clique partition has been implemented in `C++` using the free available library `GALIB (v.2.4.6)` from MIT, USA.
- The genome is the *2D* binary string of dimensions $\ell \times p$ that represents one partition into disjoint cliques.
- The crossover function creates two new genomes (son and daughter) from two old genomes (parents) by choosing the heavier individual cliques (better parent is repeated).
- The objective function, which should be maximized, is the weight of a genome.
- Typical run:
  `population size` $= 20$,
  `number of generations` $= 10^4$.

Genetic algorithm

- The serial genetic algorithm for the maximum-weight clique partition has been implemented in `C++` using the free available library `GALIB (v.2.4.6)` from MIT, USA.
- The genome is the 2*D* binary string of dimensions $\ell \times p$ that represents one partition into disjoint cliques.
- The crossover function creates two new genomes (son and daughter) from two old genomes (parents) by choosing the heavier individual cliques (better parent is repeated).
- The objective function, which should be maximized, is the weight of a genome.
- Typical run:
  `population size` $= 20$,
  `number of generations` $= 10^4$.

Table: Performance for $p = 12$, $prec = 10^{-13}$, $\kappa = 10$, multiple minimal SV. T_p is in seconds, $R_{GA} = (T_{GA}/T_p) * 100$.

	2 cliques			6 cliques		
n	n_{iter}	T_p	R_{GA}	n_{iter}	T_p	R_{GA}
1000	37	98.8	4.3	409	866.1	7.5
2000	39	463.5	1.0	416	1544.8	4.2
3000	38	1393.3	0.3	406	3922.6	1.6
4000	36	3084.8	0.1	402	8054.4	0.8
5000	37	6144.0	0.1	403	15101.6	0.4
6000	37	9994.2	< 0.1	427	25951.0	0.2
7000	37	15926.5	< 0.1	412	39925.8	0.2
8000	35	23757.1	< 0.1	423	61874.0	0.1
9000	37	34040.0	< 0.1	417	85072.0	< 0.1
10000	38	56532.5	< 0.1	431	125658.6	< 0.1