Parallel matrix inversion for the revised simplex method - A study

Julian Hall

School of Mathematics

University of Edinburgh

June 15th 2006

Overview

- Nature of the challenge of matrix inversion for the revised simplex method

Overview

- Nature of the challenge of matrix inversion for the revised simplex method
- Identify dominant costs of serial matrix inversion

Overview

- Nature of the challenge of matrix inversion for the revised simplex method
- Identify dominant costs of serial matrix inversion
- Discuss parallel matrix inversion strategies

Overview

- Nature of the challenge of matrix inversion for the revised simplex method
- Identify dominant costs of serial matrix inversion
- Discuss parallel matrix inversion strategies
- Describe a parallel triangularisation scheme

Overview

- Nature of the challenge of matrix inversion for the revised simplex method
- Identify dominant costs of serial matrix inversion
- Discuss parallel matrix inversion strategies
- Describe a parallel triangularisation scheme
- Present results using a serial simulation

Solving LP problems

$$
\begin{array}{rll}
\text{minimize} & f = \boldsymbol{c}^T \boldsymbol{x} & \\
\text{subject to} & A\boldsymbol{x} = \boldsymbol{b} & \\
 & \boldsymbol{x} \geq \boldsymbol{0} & \\
\text{where} & \boldsymbol{x} \in I\!R^n \quad \text{and} & \boldsymbol{b} \in I\!R^m
\end{array}
$$

Solving LP problems

$$\begin{aligned} \text{minimize} \quad & f = \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ \text{where} \quad & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m \end{aligned}$$

- At any vertex the variables may be partitioned into index sets
  - $\mathcal{B}$ of m basic variables $\boldsymbol{x}_B \geq \boldsymbol{0}$
  - $\mathcal{N}$ of $n - m$ nonbasic variables $\boldsymbol{x}_N = \boldsymbol{0}$

Solving LP problems

$$\begin{aligned} \text{minimize} \quad & f = \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ \text{where} \quad & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m \end{aligned}$$

- At any vertex the variables may be partitioned into index sets
  - $\mathcal{B}$ of m basic variables $\boldsymbol{x}_B \geq \boldsymbol{0}$
  - $\mathcal{N}$ of $n - m$ nonbasic variables $\boldsymbol{x}_N = \boldsymbol{0}$
- Columns of A are partitioned correspondingly into
  - the **basis matrix** B
  - the matrix N

Equation-solving in the revised simplex method

- Each iteration of the revised simplex method requires the solution of

$$B\hat{\boldsymbol{a}}_q = \boldsymbol{a}_q$$

and

$$B^T\boldsymbol{\pi} = \boldsymbol{e}_p$$

- Initial basis matrix is inverted directly

Equation-solving in the revised simplex method

- Each iteration of the revised simplex method requires the solution of

$$B\hat{\boldsymbol{a}}_q = \boldsymbol{a}_q$$

and

$$B^T\boldsymbol{\pi} = \boldsymbol{e}_p$$

- Initial basis matrix is inverted directly
- Successive basis matrices are given by

$$B := B + (\boldsymbol{a}_q - \boldsymbol{a}_p)\boldsymbol{e}_p^T$$

allowing representation of B^{-1} to be updated

Equation-solving in the revised simplex method

- Each iteration of the revised simplex method requires the solution of

$$B\hat{\boldsymbol{a}}_q = \boldsymbol{a}_q$$

and

$$B^T\boldsymbol{\pi} = \boldsymbol{e}_p$$

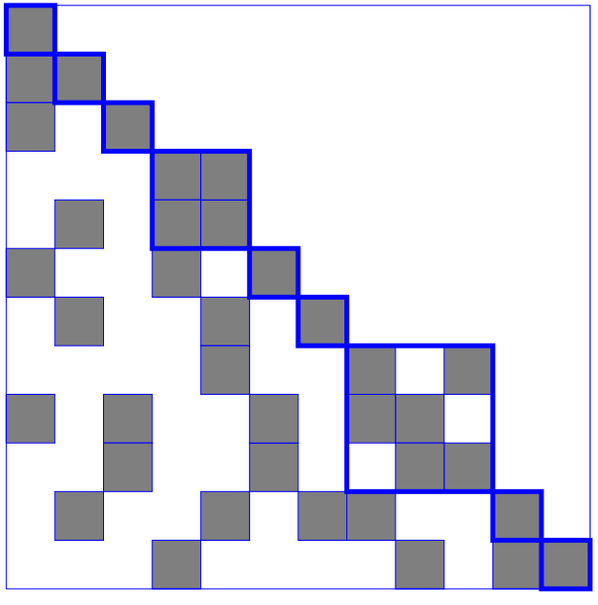
- Initial basis matrix is inverted directly
- Successive basis matrices are given by

$$B := B + (\boldsymbol{a}_q - \boldsymbol{a}_p)\boldsymbol{e}_p^T$$

allowing representation of B^{-1} to be updated

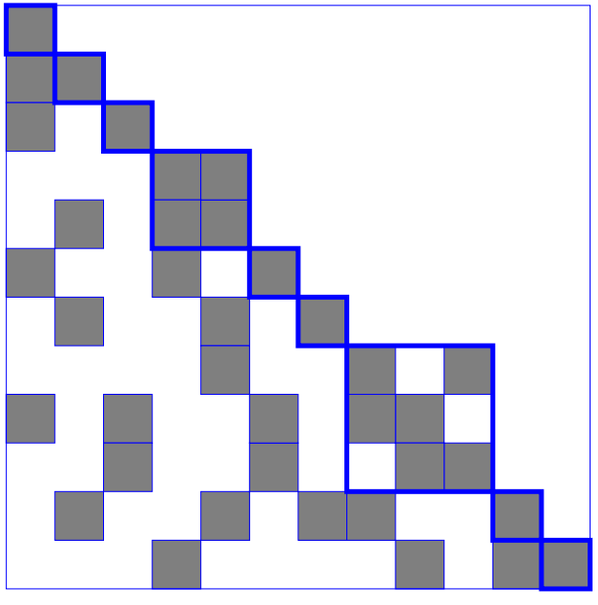
- Periodically it is necessary to re-invert the basis matrix directly

Basis matrix inversion: The nature of the challenge



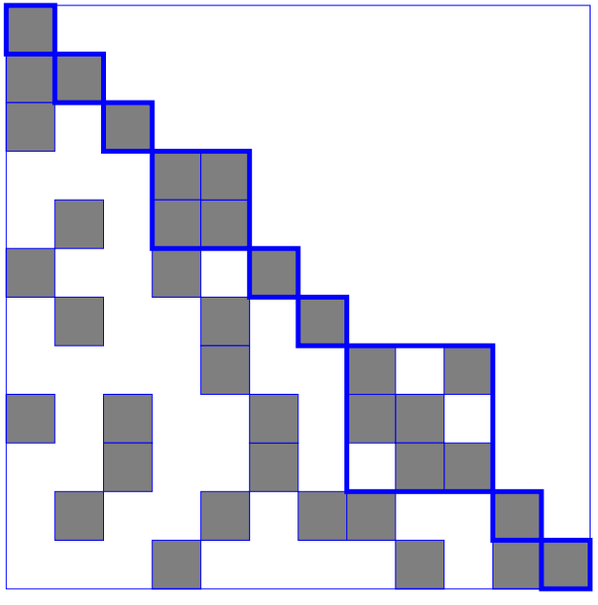
- Consider optimal block triangular form (Tarjan) of B
  - All diagonal blocks are 1×1 (trivial) or irreducible
  - For LU factors of B, elimination is restricted to diagonal blocks

Basis matrix inversion: The nature of the challenge



- Consider optimal block triangular form (Tarjan) of B
  - All diagonal blocks are 1×1 (trivial) or irreducible
  - For LU factors of B, elimination is restricted to diagonal blocks
- Total cost of inverting B depends on
  - Cost of finding irreducible blocks
  - Cost of factorising irreducible blocks

Basis matrix inversion: The nature of the challenge



- Consider optimal block triangular form (Tarjan) of B
  - All diagonal blocks are 1×1 (trivial) or irreducible
  - For LU factors of B, elimination is restricted to diagonal blocks
- Total cost of inverting B depends on
  - Cost of finding irreducible blocks
  - Cost of factorising irreducible blocks
- Relative number/size of 1×1 and irreducible Tarjan blocks is closely related to **hyper-sparsity** of the LP

Revised simplex method and hyper-sparsity

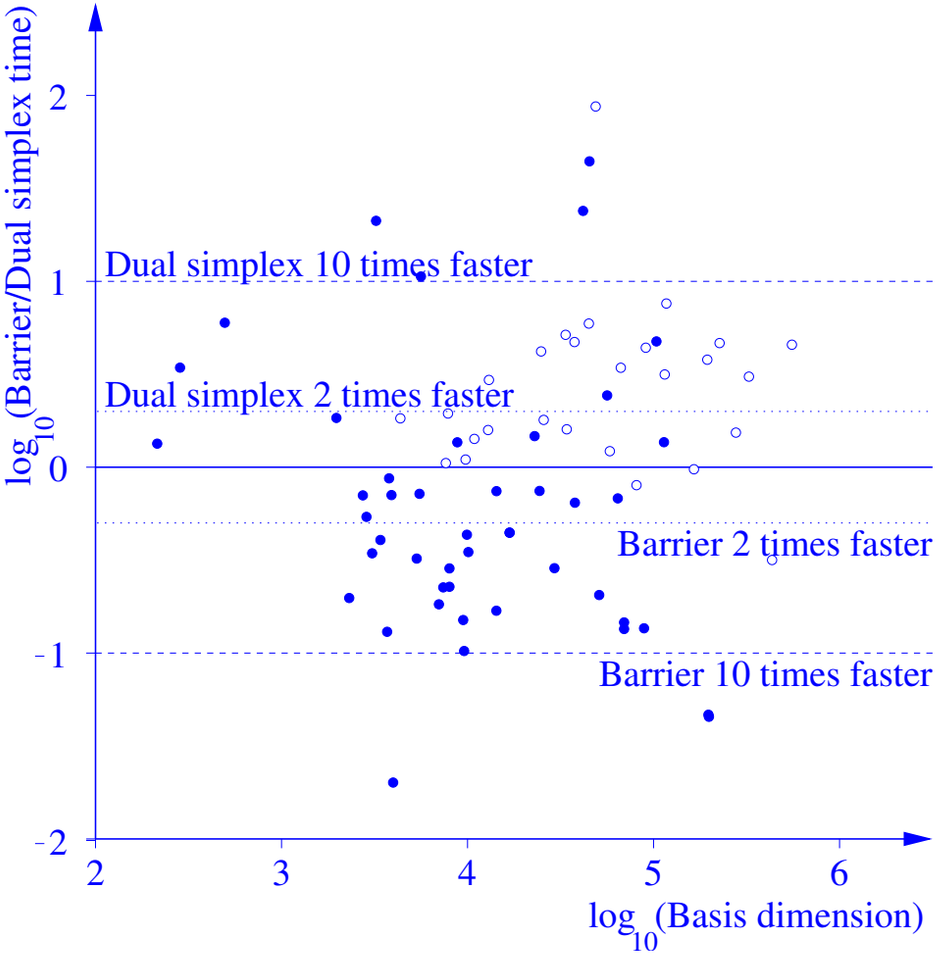
- Major computational cost each (primal) revised simplex iteration
  - Solve $B\hat{\boldsymbol{a}}_q = \boldsymbol{a}_q$ (FTRAN)
  - Solve $B^T\boldsymbol{\pi} = \boldsymbol{e}_p$ (BTRAN)
  - Form $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$ (PRICE)
- RHS vectors $\boldsymbol{a}_q$ and $\boldsymbol{e}_p$ are sparse

Revised simplex method and hyper-sparsity

- Major computational cost each (primal) revised simplex iteration
  - Solve $B\hat{\boldsymbol{a}}_q = \boldsymbol{a}_q$ (FTRAN)
  - Solve $B^T\boldsymbol{\pi} = \boldsymbol{e}_p$ (BTRAN)
  - Form $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$ (PRICE)
- RHS vectors $\boldsymbol{a}_q$ and $\boldsymbol{e}_p$ are sparse
- If the results of these operations are (usually) sparse then LP is said to be **hyper-sparse**

Revised simplex method and hyper-sparsity

- Major computational cost each (primal) revised simplex iteration
  - Solve $B\hat{\boldsymbol{a}}_q = \boldsymbol{a}_q$ (FTRAN)
  - Solve $B^T\boldsymbol{\pi} = \boldsymbol{e}_p$ (BTRAN)
  - Form $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$ (PRICE)
- RHS vectors $\boldsymbol{a}_q$ and $\boldsymbol{e}_p$ are sparse
- If the results of these operations are (usually) sparse then LP is said to be **hyper-sparse**
- For hyper-sparse LP problems (○), the (dual) revised simplex method is typically better than barrier methods
- For non hyper-sparse LPs (●), barrier methods are frequently better than the revised simplex method



Tomlin INVERT (1972)

- Matrix inversion procedure designed for the revised simplex method

Tomlin INVERT (1972)

- Matrix inversion procedure designed for the revised simplex method
- Operates in two phases

  Triangularisation:
  - Active row/column **singleton** entries are identified as pivots
    until all active rows/columns have at least *two* active nonzeros
  - Corresponds to Markowitz pivot selection so long as count is zero
  - Yields approximate Tarjan form at *much* lower cost

Tomlin INVERT (1972)

- Matrix inversion procedure designed for the revised simplex method
- Operates in two phases

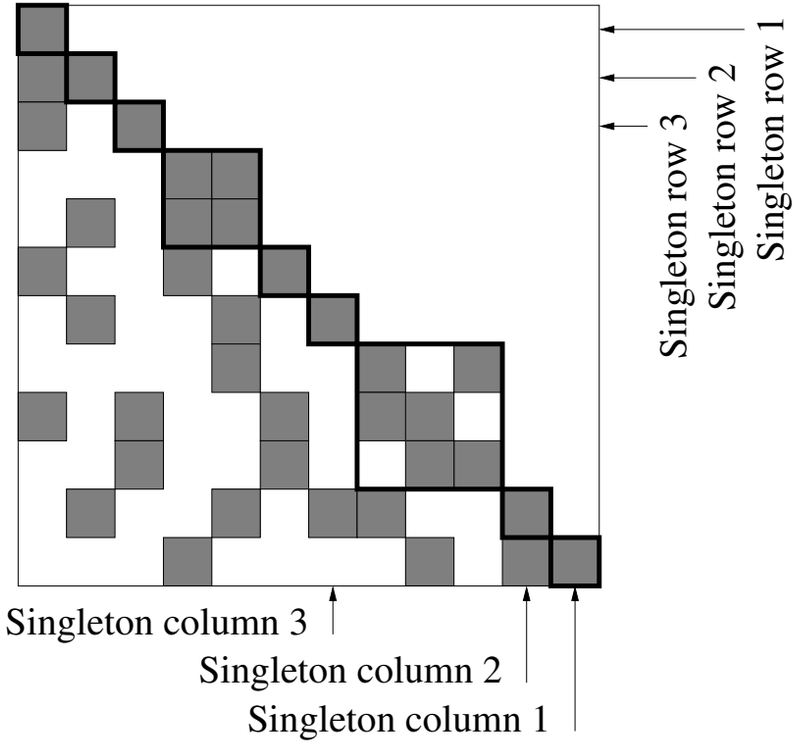
  Triangularisation:

  - Active row/column **singleton** entries are identified as pivots until all active rows/columns have at least *two* active nonzeros
  - Corresponds to Markowitz pivot selection so long as count is zero
  - Yields approximate Tarjan form at *much* lower cost

  Factorisation:

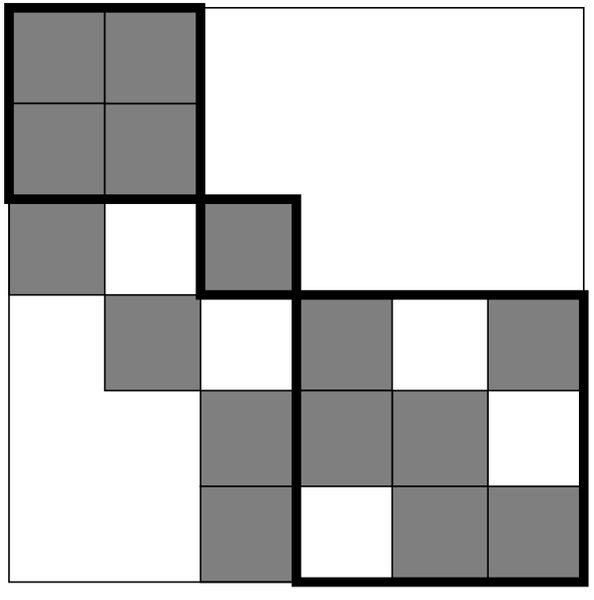
  - Gaussian elimination applied the residual **bump**

Tomlin INVERT: Illustrative example



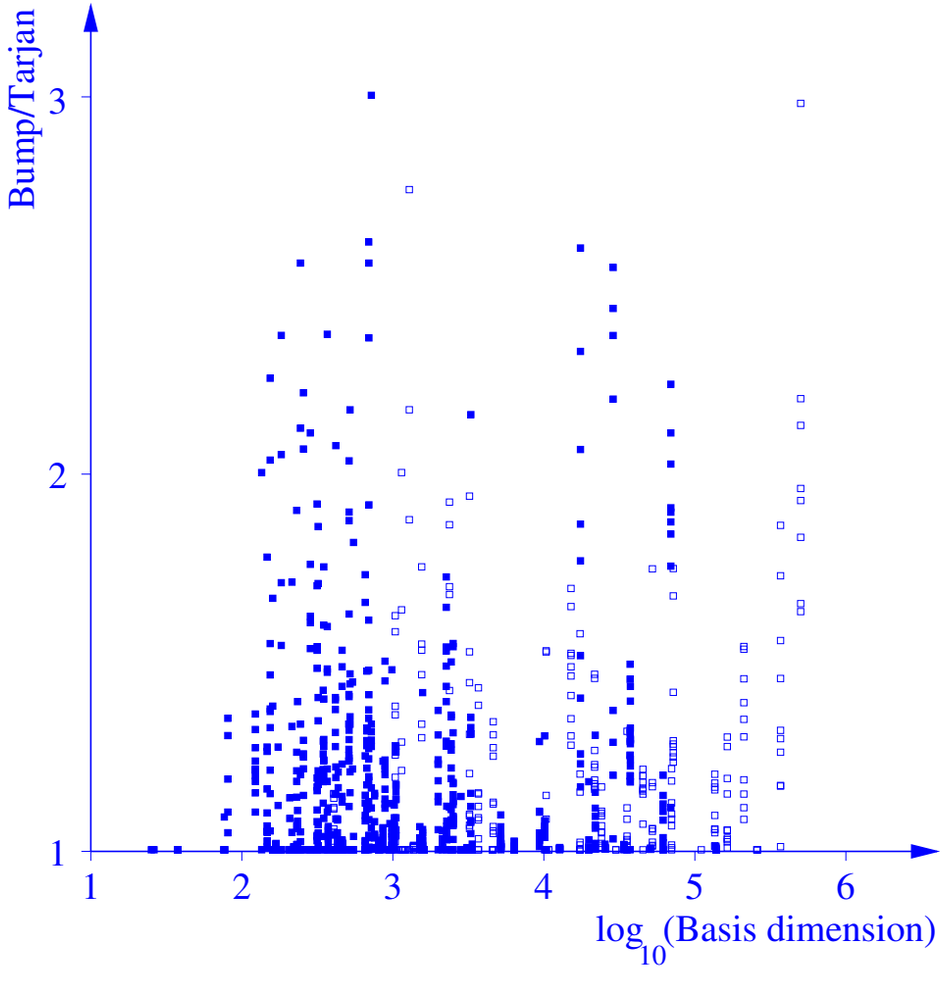
- Triangularisation identifies 3 singleton rows and 3 singleton columns

Tomlin INVERT: Bump



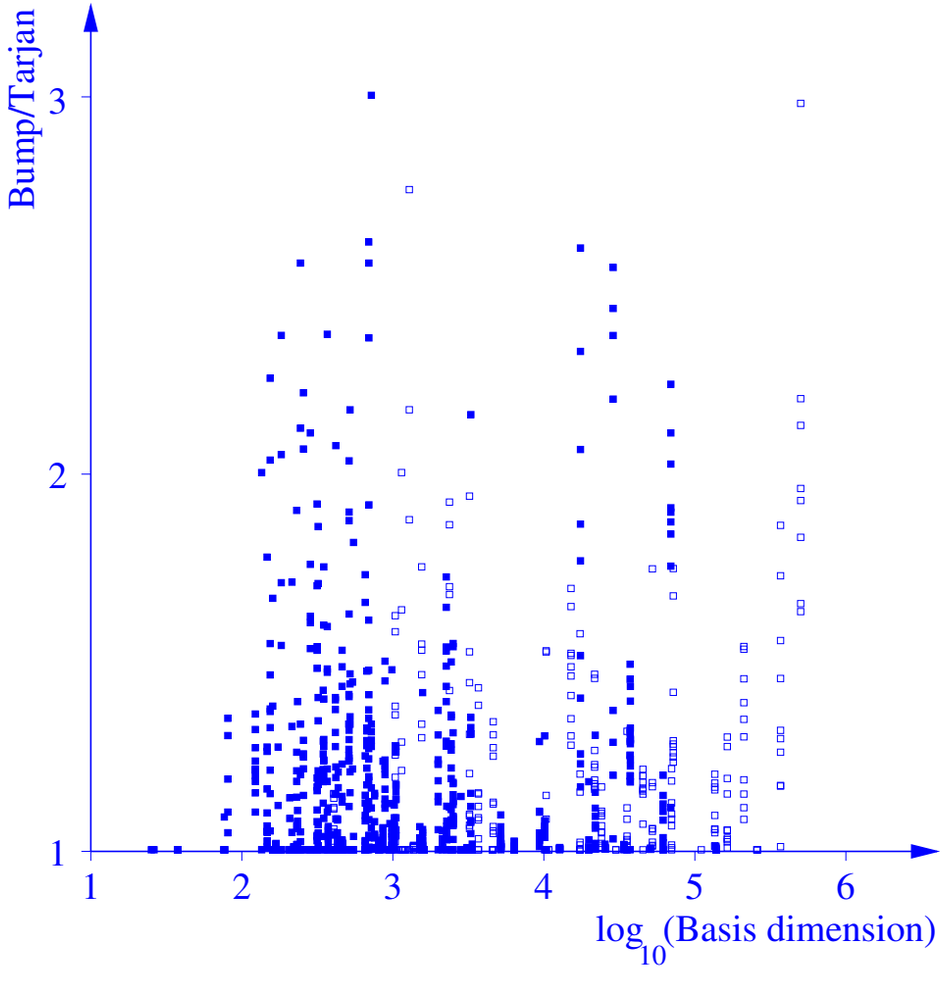
- Every row and column has a count of at least 2
- One of the trivial Tarjan blocks cannot be placed in Tomlin triangular form

Tomlin INVERT: Size of bump relative to non-trivial Tarjan blocks



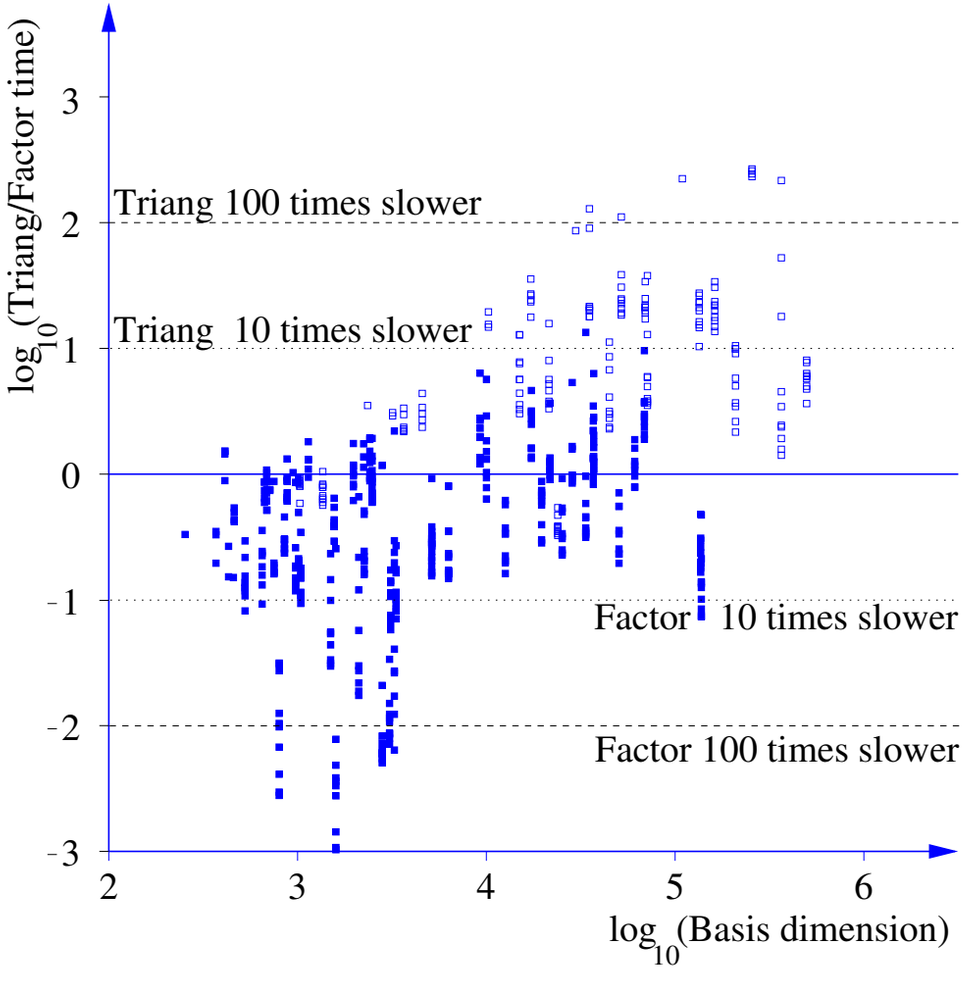
- Results for 10 representative basis matrices from each of the test LPs
  - Matrices from hyper-sparse LPs
  - Matrices from non hyper-sparse LPs

Tomlin INVERT: Size of bump relative to non-trivial Tarjan blocks

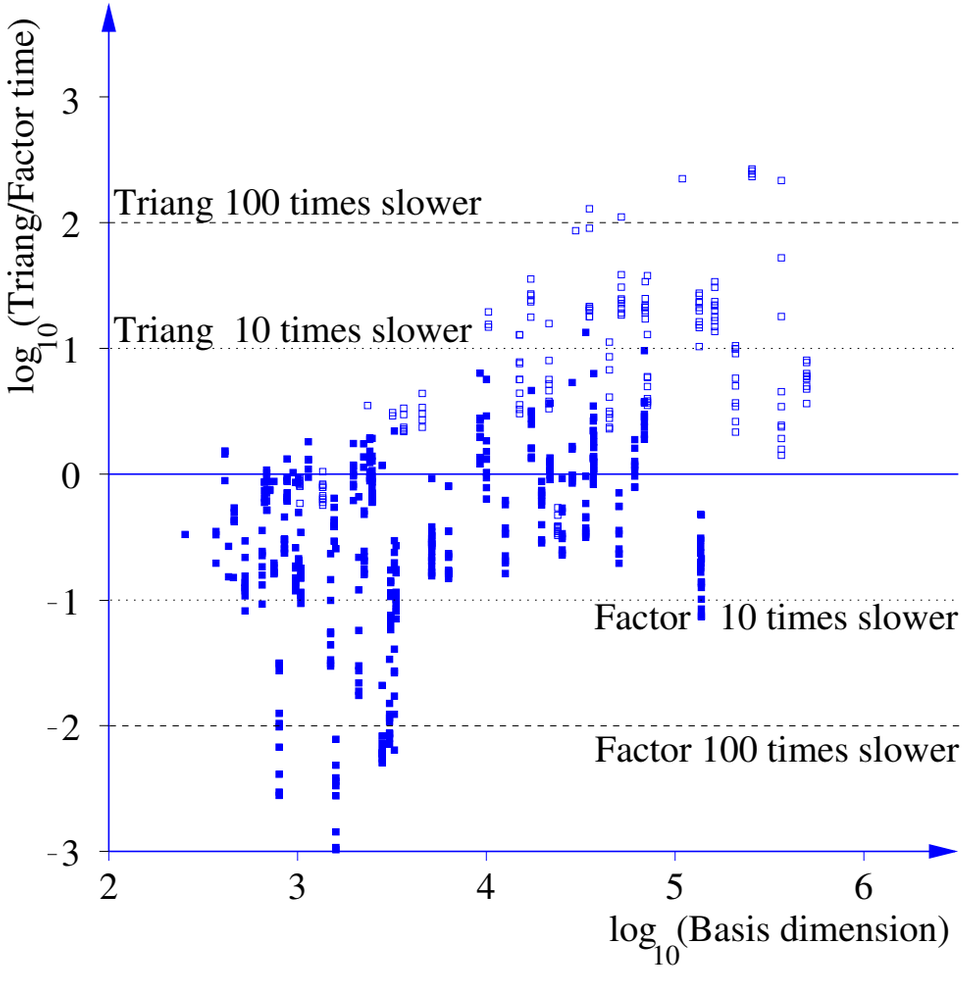


- Results for 10 representative basis matrices from each of the test LPs
  - Matrices from hyper-sparse LPs
  - Matrices from non hyper-sparse LPs
- Size of bump bounded below by sum of dimensions of non-trivial Tarjan blocks
  - Bump is frequently less than 50% larger
  - Bump is rarely more than 100% larger

Tomlin INVERT: Relative time for triangularisation and factorization



Tomlin INVERT: Relative time for triangularisation and factorization



- Triangularisation can be hugely dominant: particularly for large hyper-sparse LPs
- Bump factorization can be hugely dominant: particularly for smaller non hyper-sparse LPs

Parallel basis matrix inversion: Overview

- Both phases of Tomlin matrix inversion can dominate the computational cost
- Depends on the nature of the LP

Parallel basis matrix inversion: Overview

- Both phases of Tomlin matrix inversion can dominate the computational cost
- Depends on the nature of the LP
- Parallel bump factorisation (important for smaller, non hyper-sparse LPs)
  - Of lesser interest
  - Consider using existing codes such as MUMPS or SuperLU

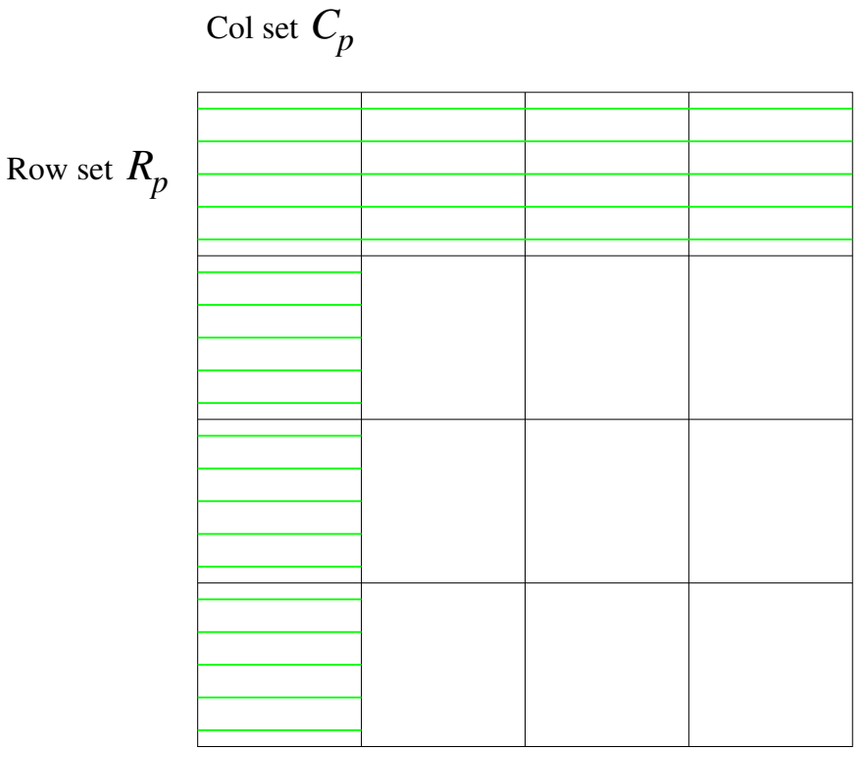
Parallel basis matrix inversion: Overview

- Both phases of Tomlin matrix inversion can dominate the computational cost
- Depends on the nature of the LP
- Parallel bump factorisation (important for smaller, non hyper-sparse LPs)
  - Of lesser interest
  - Consider using existing codes such as MUMPS or SuperLU
- Parallel triangularisation (important for larger, hyper-sparse LPs)
  - Of greater interest
  - The subject of the rest of this talk

Parallel triangularisation: Distribution of rows

Processor p operates using the following **row-wise** data

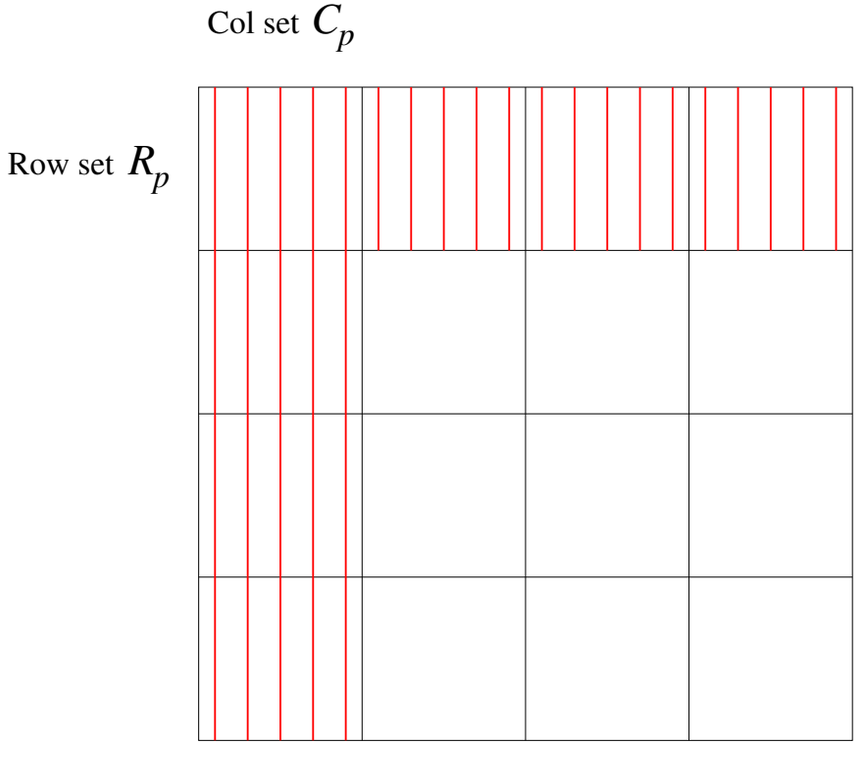
- All entries for rows in set R_p
- Entries in columns C_p for rows in set R'_p



Parallel triangularisation: Distribution of columns

Processor p operates using the following **column-wise** data

- All entries for columns in set C_p
- Entries in rows R_p for columns in set C'_p



Parallel triangularisation: Overview

For each processor $p = 1, \ldots, N$:

- **Initialisation:** Determine row and column count data for row set R_p and column set C_p

Parallel triangularisation: Overview

For each processor $p = 1, \ldots, N$:

- **Initialisation:** Determine row and column count data for row set R_p and column set C_p
- **Major iteration:** Repeat:
  - **Minor iteration:** Identify row (column) singletons in R_p (C_p) until it is "wise" to stop

Parallel triangularisation: Overview

For each processor $p = 1, \ldots, N$:

- **Initialisation:** Determine row and column count data for row set R_p and column set C_p
- **Major iteration:** Repeat:
  - **Minor iteration:** Identify row (column) singletons in R_p (C_p) until it is "wise" to stop
  - Broadcast pivot indices to all other processors

Parallel triangularisation: Overview

For each processor $p = 1, \ldots, N$:

- **Initialisation:** Determine row and column count data for row set R_p and column set C_p
- **Major iteration:** Repeat:
  - **Minor iteration:** Identify row (column) singletons in R_p (C_p) until it is "wise" to stop
  - Broadcast pivot indices to all other processors
  - Update row (column) count data using pivot indices from all other processors

Parallel triangularisation: Overview

For each processor $p = 1, \ldots, N$:

- **Initialisation:** Determine row and column count data for row set R_p and column set C_p
- **Major iteration:** Repeat:
  - **Minor iteration:** Identify row (column) singletons in R_p (C_p) until it is "wise" to stop
  - Broadcast pivot indices to all other processors
  - Update row (column) count data using pivot indices from all other processors
- Until there are no row or column singletons

Parallel triangularisation: Overview

For each processor $p = 1, \ldots, N$:

- **Initialisation:** Determine row and column count data for row set R_p and column set C_p
- **Major iteration:** Repeat:
  - **Minor iteration:** Identify row (column) singletons in R_p (C_p) until it is "wise" to stop
  - Broadcast pivot indices to all other processors
  - Update row (column) count data using pivot indices from all other processors
- Until there are no row or column singletons

Communication cost of $O(m)$ for computation cost $O(\tau)$

Parallel triangularisation: Minor iteration

Within each minor iteration

- Row and column counts are
  - initially global;
  - updated according to pivots determined locally;
  - become upper bounds on global counts

Parallel triangularisation: Minor iteration

Within each minor iteration

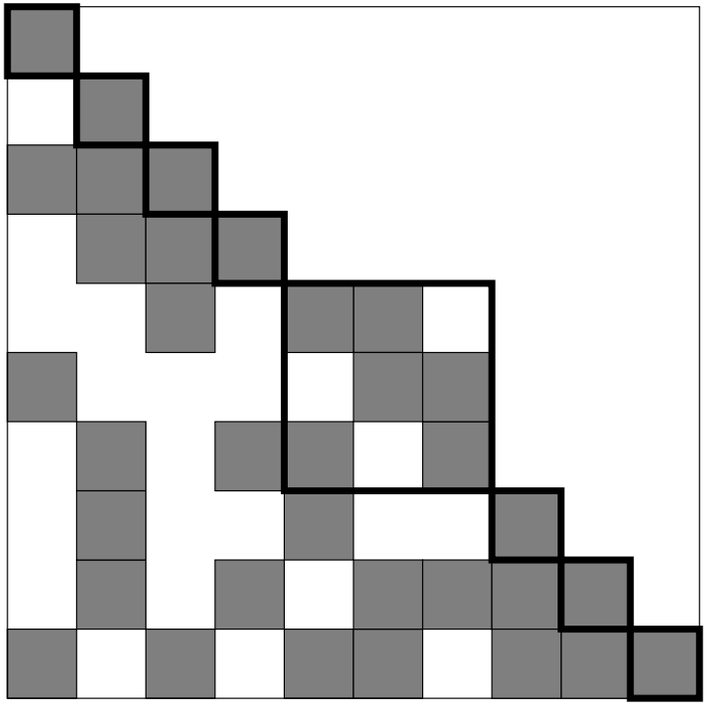
- Row and column counts are
  - initially global;
  - updated according to pivots determined locally;
  - become upper bounds on global counts
- When is it wise to stop performing minor iterations?
  - Too soon: communication overheads dominate
  - Too late: load imbalance

Parallel triangularisation: Minor iteration

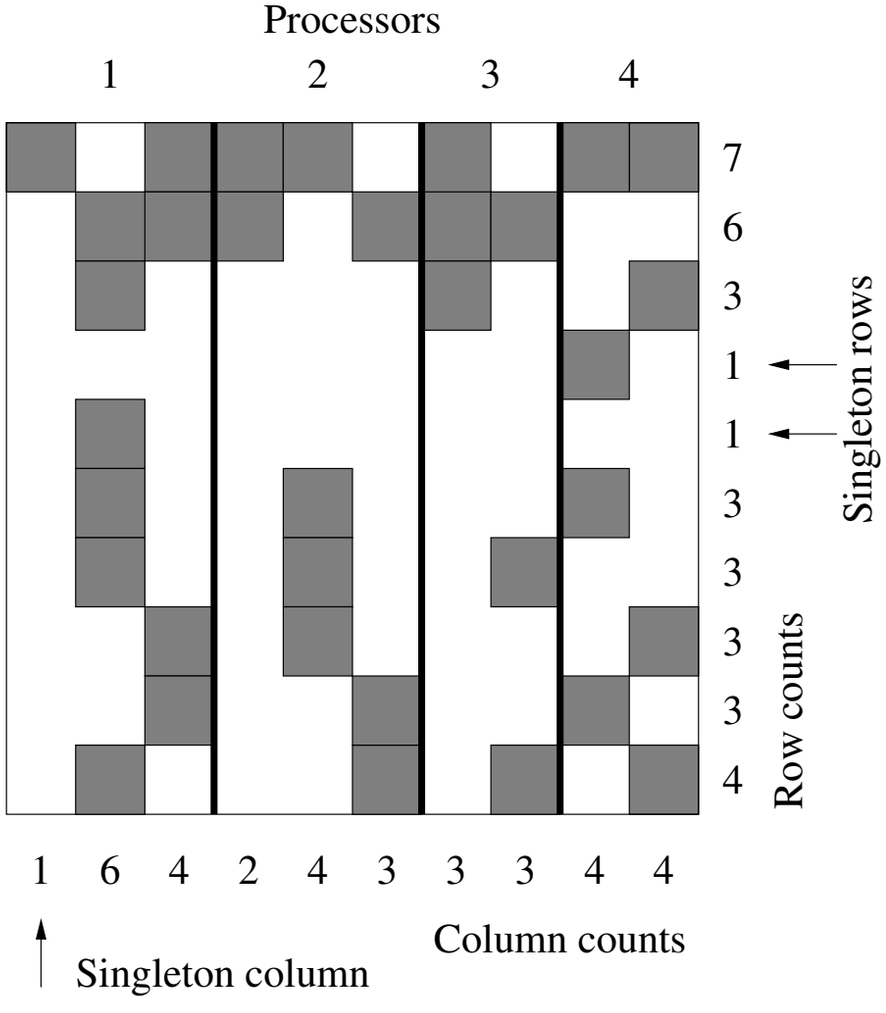
Within each minor iteration

- Row and column counts are
  - initially global;
  - updated according to pivots determined locally;
  - become upper bounds on global counts
- When is it wise to stop performing minor iterations?
  - Too soon: communication overheads dominate
  - Too late: load imbalance
- Aim to find a particular proportion of the pivots in each major iteration

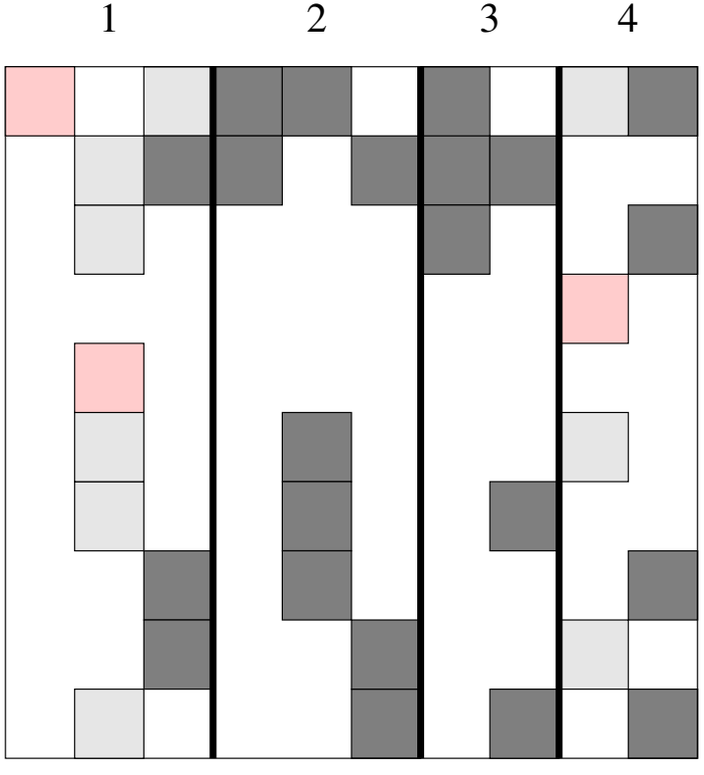
Parallel triangularisation: Example



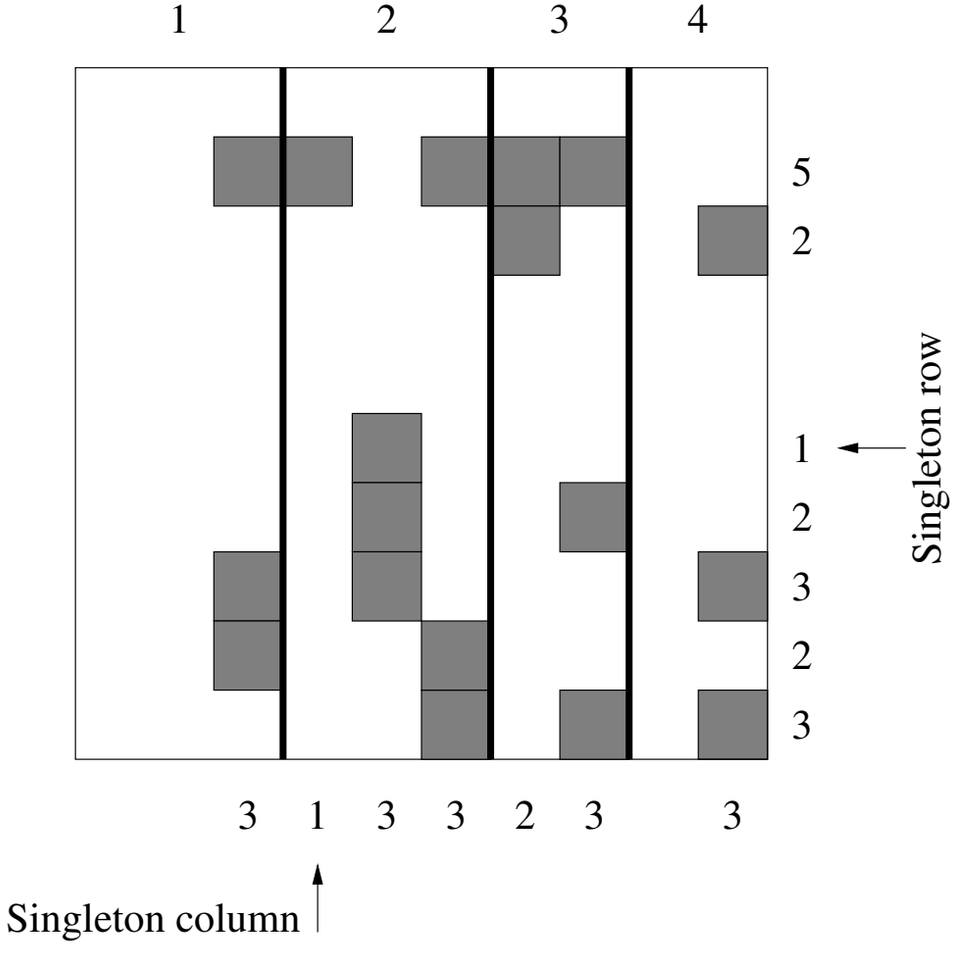
Parallel triangularisation: Iteration 1



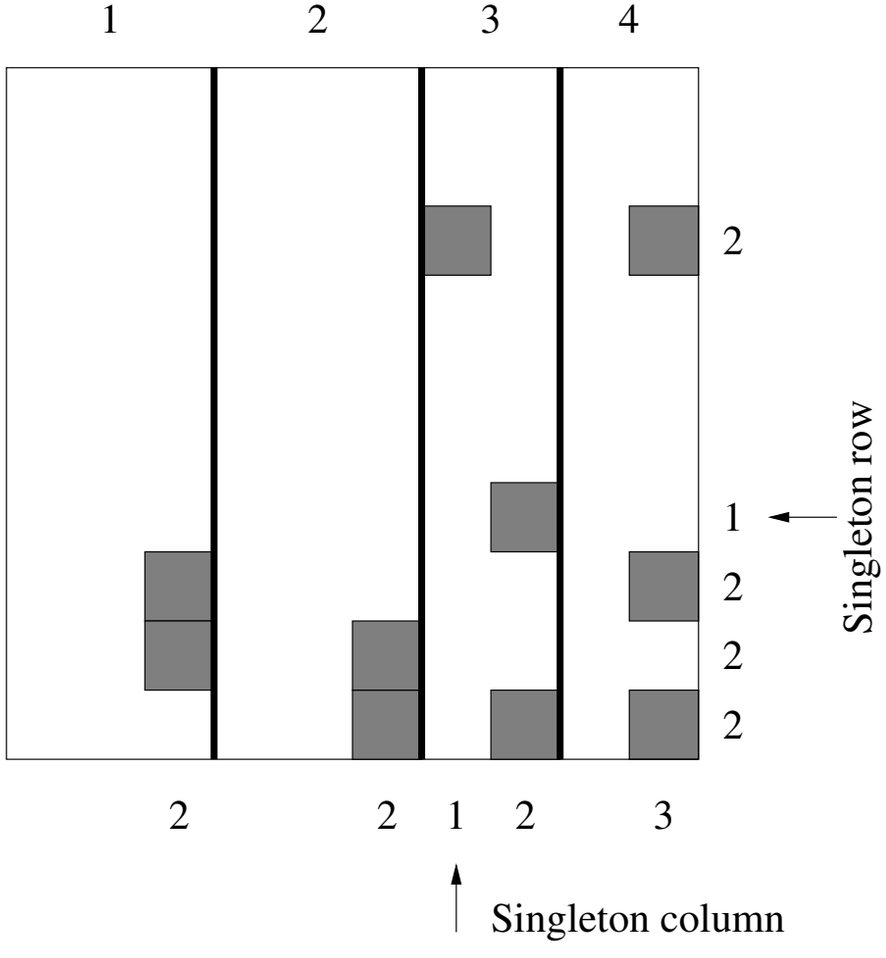
Parallel triangularisation: Iteration 1 result



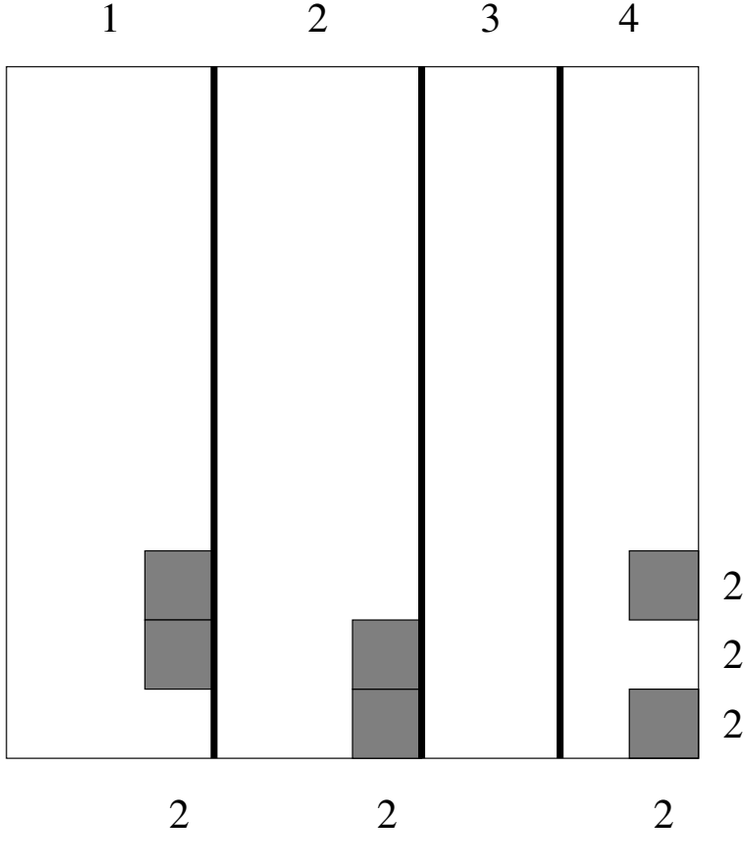
Parallel triangularisation: Iteration 2



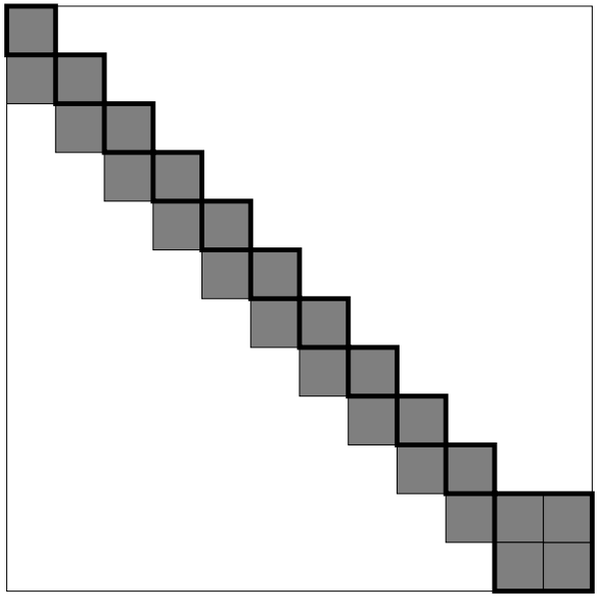
Parallel triangularisation: Iteration 3



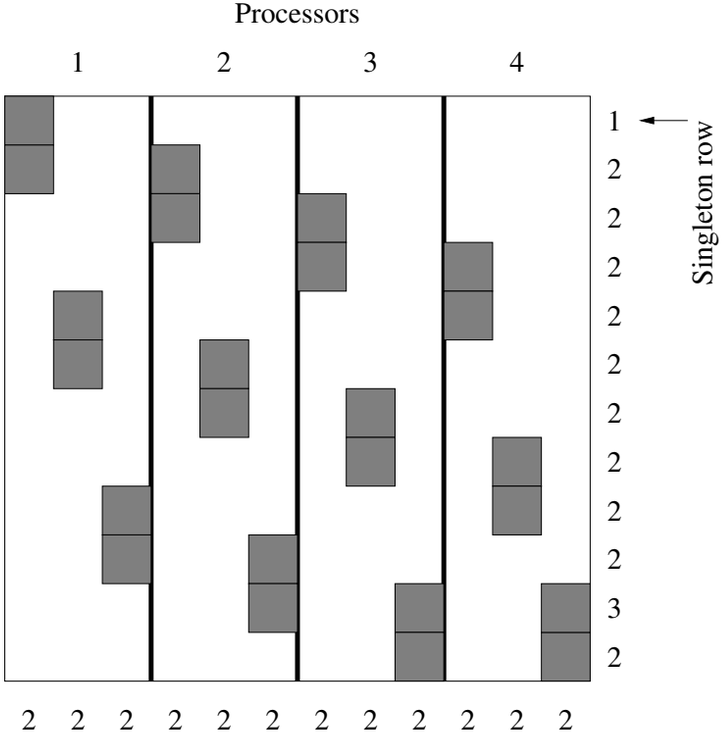
Parallel triangularisation: Iteration 4



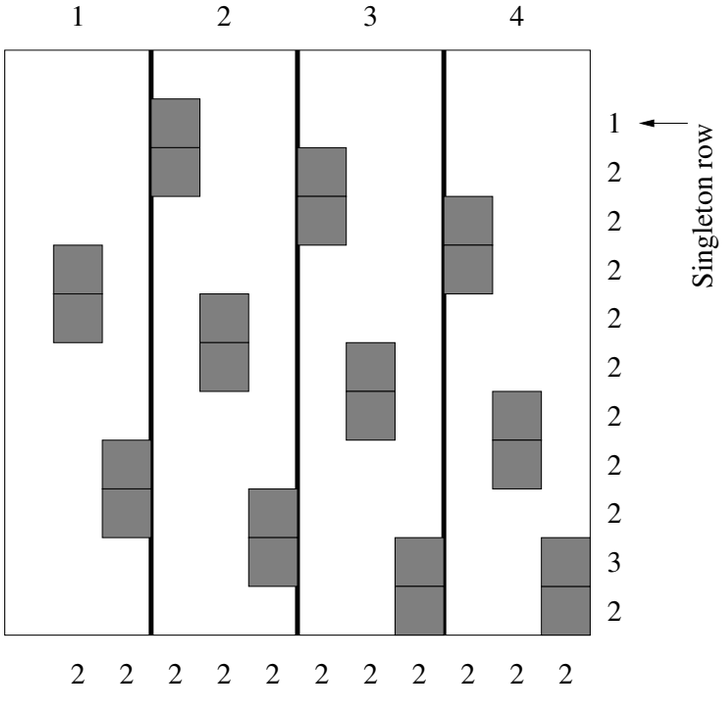
Parallel triangularisation: Worst case behaviour



Worst case behaviour: Iteration 1



Worst case behaviour: Iteration 2



- Only one pivot is identified on one processor until pivot is broadcast
- Reduces to serial case with considerable overhead

Measures of performance

Assess viability of parallel scheme using serial simulator

Measures of performance

Assess viability of parallel scheme using serial simulator

- **Load balance**: Radically different numbers of pivots on processors

Measures of performance

Assess viability of parallel scheme using serial simulator

- **Load balance**: Radically different numbers of pivots on processors
  Processor idleness

Measures of performance

Assess viability of parallel scheme using serial simulator

- **Load balance**: Radically different numbers of pivots on processors
  Processor idleness
- **Communication overhead**: Excess numbers of major iterations

Measures of performance

Assess viability of parallel scheme using serial simulator

- **Load balance**: Radically different numbers of pivots on processors
  Processor idleness
- **Communication overhead**: Excess numbers of major iterations
  Communication costs dominate

Measures of performance

Assess viability of parallel scheme using serial simulator

- **Load balance**: Radically different numbers of pivots on processors
  Processor idleness
- **Communication overhead**: Excess numbers of major iterations
  Communication costs dominate
- Relate performance to **ideal number of major iterations**

$$\frac{100}{\text{Target \% of triangular pivots per major iteration}}$$

Parallel triangularisation: Good performance

- For model `nsct2`
- 23003 rows
- 16329 logicals
- Bump dimension is 183

Parallel triangularisation: Good performance

- For model `nsct2`
- 23003 rows
- 16329 logicals
- Bump dimension is 183
- 4 processors
- 10% of pivots per major iteration
- Ideal number of major iterations is 10

Parallel triangularisation: Good performance

- For model `nsct2`
- 23003 rows
- 16329 logicals
- Bump dimension is 183
- 4 processors
- 10% of pivots per major iteration
- Ideal number of major iterations is 10

	Pivots found on processor			
It	1	2	3	4
1	163	163	163	163
2	163	163	163	163
3	163	163	163	163
4	163	163	163	163
5	163	163	163	163
6	163	163	163	163
7	163	163	163	163
8	163	163	163	163
9	163	163	163	163
10	162	148	156	155
11	0	1	1	0
12	0	0	0	0

Serial simulation of parallel triangularisation: "Poor" performance

- For model `pds-06`
- 9881 rows
- 952 logicals
- Bump dimension is 55
- 4 processors
- 10% of pivots per major iteration

Serial simulation of parallel triangularisation: "Poor" performance

	Pivots found on processor				
It	1	2	3	4	Pivots
1	222	222	222	222	10%
⋮	⋮	⋮	⋮	⋮	
8	222	222	222	222	80%

- For model `pds-06`
- 9881 rows
- 952 logicals
- Bump dimension is 55
- 4 processors
- 10% of pivots per major iteration

Serial simulation of parallel triangularisation: "Poor" performance

- For model `pds-06`
- 9881 rows
- 952 logicals
- Bump dimension is 55
- 4 processors
- 10% of pivots per major iteration

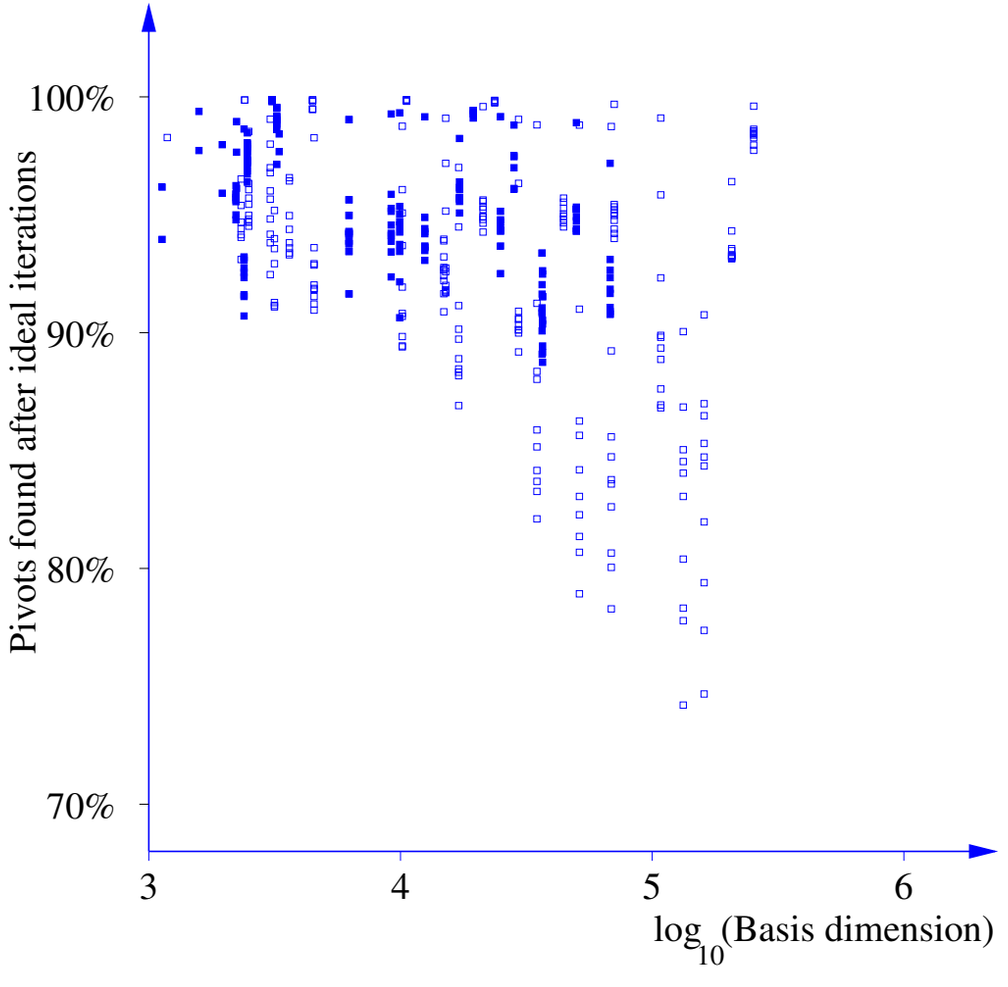
	Pivots found on processor				
It	1	2	3	4	Pivots
1	222	222	222	222	10%
⋮	⋮	⋮	⋮	⋮	
8	222	222	222	222	80%
9	162	186	169	195	88%
10	105	78	87	84	92%
11	41	52	47	38	94%
15	11	10	10	4	97%
⋮	⋮	⋮	⋮	⋮	

Serial simulation of parallel triangularisation: "Poor" performance

- For model pds-06
- 9881 rows
- 952 logicals
- Bump dimension is 55
- 4 processors
- 10% of pivots per major iteration

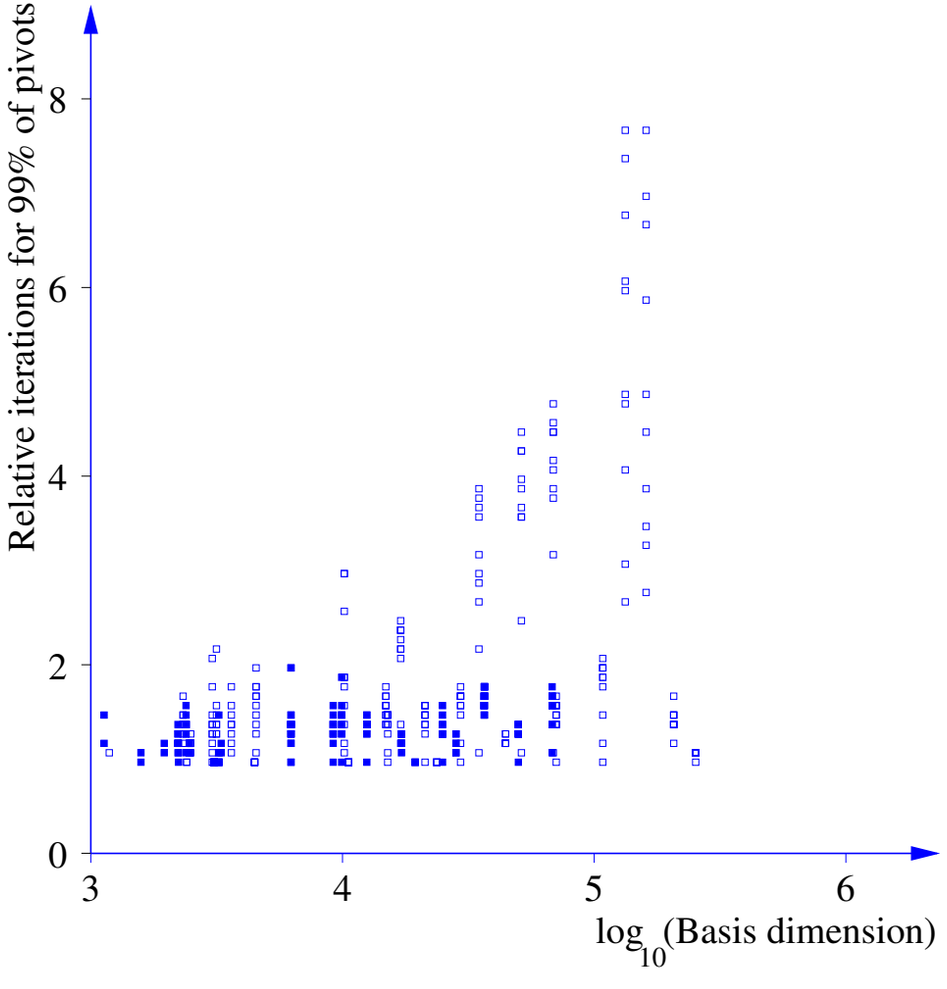
	Pivots found on processor				
It	1	2	3	4	Pivots
1	222	222	222	222	10%
⋮	⋮	⋮	⋮	⋮	
8	222	222	222	222	80%
9	162	186	169	195	88%
10	105	78	87	84	92%
11	41	52	47	38	94%
15	11	10	10	4	97%
⋮	⋮	⋮	⋮	⋮	
20	3	3	5	4	98%
⋮	⋮	⋮	⋮	⋮	
40	2	1	0	0	99%
⋮	⋮	⋮	⋮	⋮	
57	0	0	0	0	100%

Parallel triangularisation: Percentage of pivots after ideal iterations



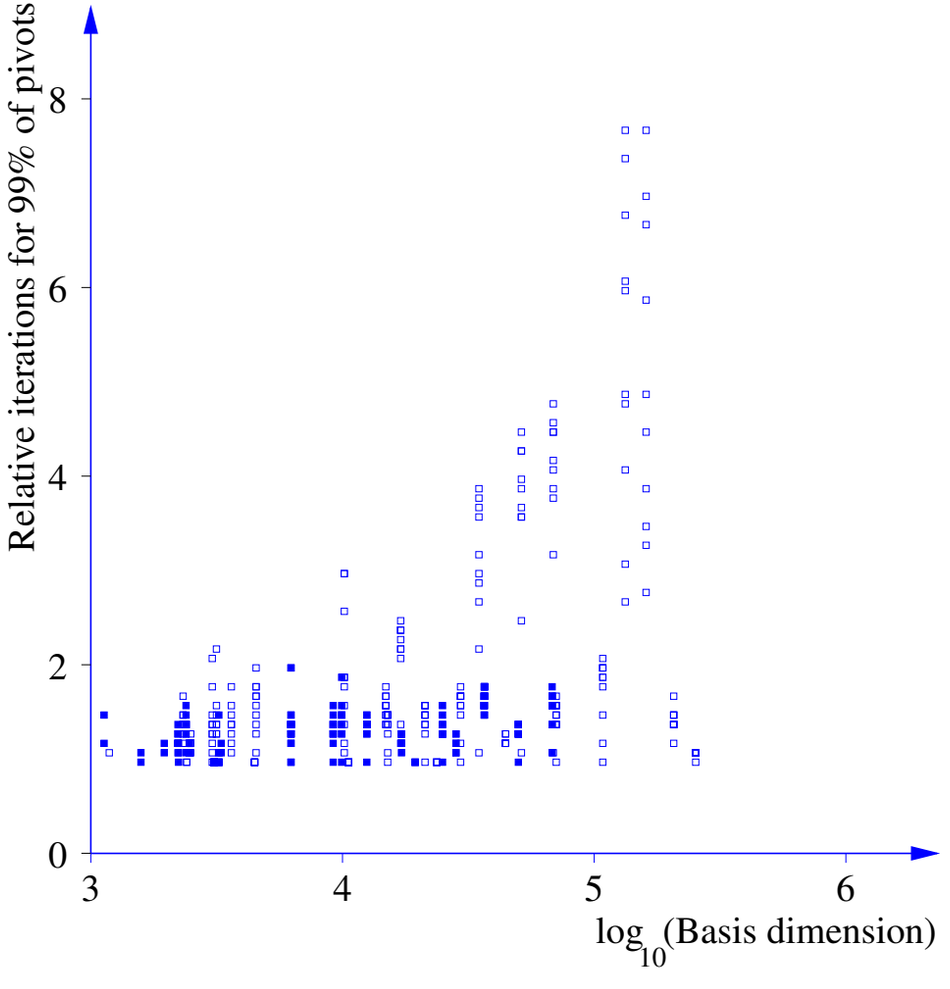
- Typically get 90%
- Generally get at least 80%
- Indicates good load balance

Parallel triangularisation: Relative number of iterations for 99% of pivots



- Typically get 99% of pivots within a small multiple of the ideal number of iterations
- Occasionally requires large multiple of the ideal number of iterations

Parallel triangularisation: Relative number of iterations for 99% of pivots



- Typically get 99% of pivots within a small multiple of the ideal number of iterations
- Occasionally requires large multiple of the ideal number of iterations
- **Recall:**
  - Additional iterations may be very much faster than "ideal" iterations
  - Communication overhead will dominate if too few pivots are found in an iteration

Conclusions

- Matrix triangularisation identified as dominant serial cost for hyper-sparse LPs

Conclusions

- Matrix triangularisation identified as dominant serial cost for hyper-sparse LPs
- Significant scope for parallelisation with scheme presented
  - Communication cost of $O(m)$ for computation cost $O(\tau)$

Conclusions

- Matrix triangularisation identified as dominant serial cost for hyper-sparse LPs
- Significant scope for parallelisation with scheme presented
  - Communication cost of $O(m)$ for computation cost $O(\tau)$
  - Limited cost of switching to serial in case of poor ultimate parallel performance

Conclusions

- Matrix triangularisation identified as dominant serial cost for hyper-sparse LPs
- Significant scope for parallelisation with scheme presented
  - Communication cost of $O(m)$ for computation cost $O(\tau)$
  - Limited cost of switching to serial in case of poor ultimate parallel performance
- Implementation required for proper assessment of scheme