

Parallel basis matrix triangularisation for hyper-sparse LP problems

Julian Hall

School of Mathematics

University of Edinburgh

September 14th 2007



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
- Dominant costs of serial matrix inversion

Overview

- Nature of the challenge of matrix inversion for the revised simplex method
- Dominant costs of serial matrix inversion
- Parallel matrix inversion strategies

Overview

- Nature of the challenge of matrix inversion for the revised simplex method
- Dominant costs of serial matrix inversion
- Parallel matrix inversion strategies
- Parallel triangularisation scheme

Overview

- Nature of the challenge of matrix inversion for the revised simplex method
- Dominant costs of serial matrix inversion
- Parallel matrix inversion strategies
- Parallel triangularisation scheme
- Results

Solving LP problems

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

Solving LP problems

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

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

Solving LP problems

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

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

Simplex method and hyper-sparsity

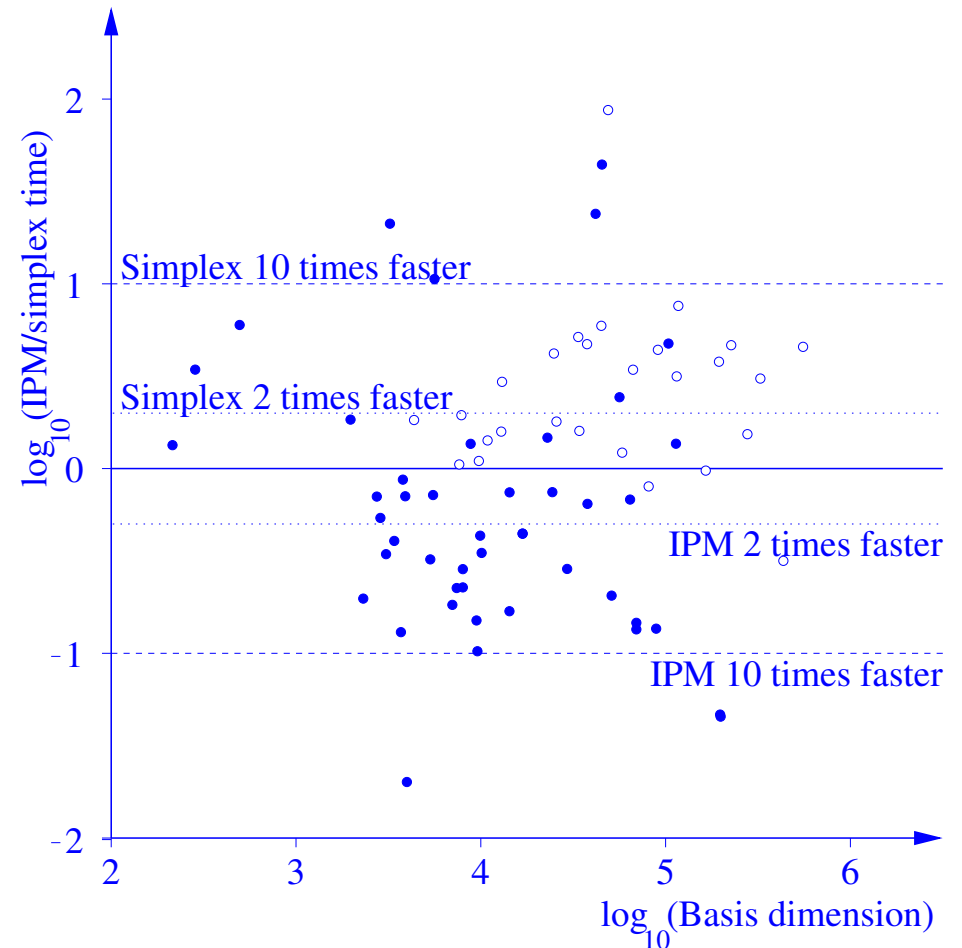
- Major computational cost each iteration is forming $B^{-1}\mathbf{r}_F$, $\mathbf{r}_B^T B^{-1}$ and $\mathbf{r}_\pi^T N$
- Vectors $\mathbf{r}_F$, $\mathbf{r}_B$ are always sparse
- Vector $\mathbf{r}_\pi$ may be sparse

Simplex method and hyper-sparsity

- Major computational cost each iteration is forming $B^{-1}\mathbf{r}_F$, $\mathbf{r}_B^T B^{-1}$ and $\mathbf{r}_\pi^T N$
- Vectors $\mathbf{r}_F$, $\mathbf{r}_B$ are always sparse
- Vector $\mathbf{r}_\pi$ may be sparse
- If the results of these operations are (usually) sparse then LP is said to be **hyper-sparse**

Simplex method and hyper-sparsity

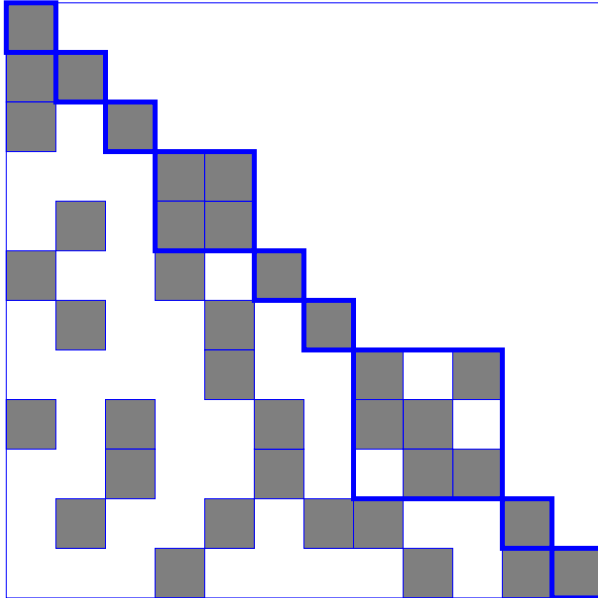
- Major computational cost each iteration is forming $B^{-1}\mathbf{r}_F$, $\mathbf{r}_B^T B^{-1}$ and $\mathbf{r}_\pi^T N$
- Vectors $\mathbf{r}_F$, $\mathbf{r}_B$ are always sparse
- Vector $\mathbf{r}_\pi$ may be sparse
- If the results of these operations are (usually) sparse then LP is said to be **hyper-sparse**
- When exploiting hyper-sparsity
 - For hyper-sparse LP problems (○)
Simplex method typically better than interior point methods
 - For other LPs (●)
Interior point methods frequently better than simplex method



Hall and McKinnon: COAP **32** (2005) 259–283

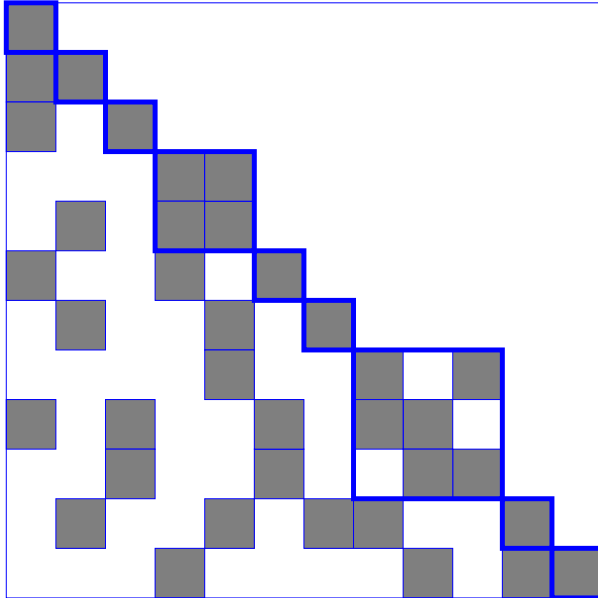


Basis matrix inversion: The nature of the challenge



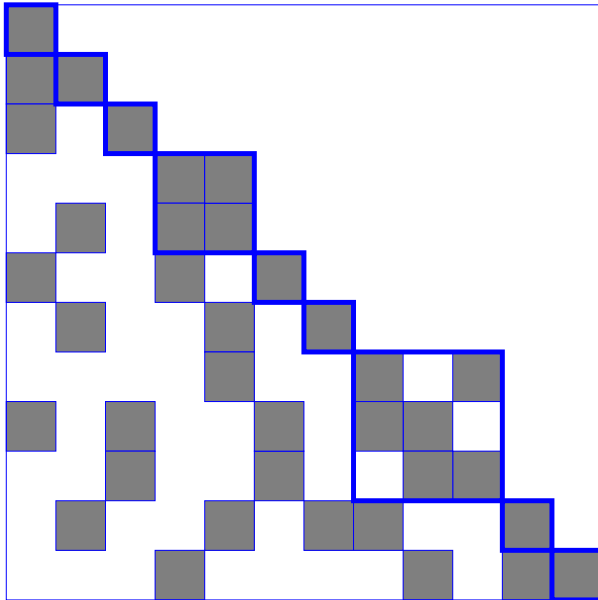
- Consider optimal block triangular form (Tarjan) of B
 - All diagonal blocks are 1×1 (trivial) or irreducible
 - For LU factors of B , elimination is restricted to diagonal blocks

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

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:

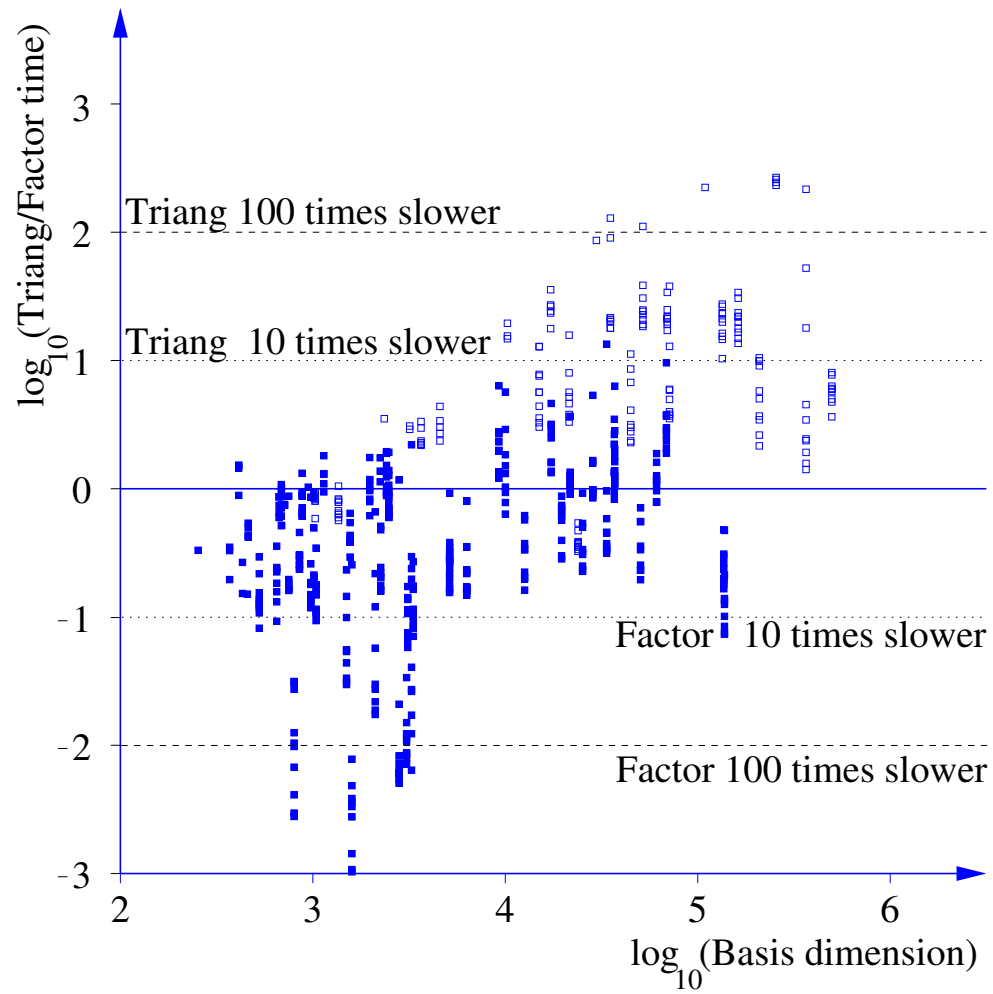
- Active row/column **singleton** entries are identified as pivots until all active rows/columns have at least *two* active nonzeros
- Corresponds to Markowitz pivot selection so long as count is zero
- Yields approximate Tarjan form at *much* lower cost

Factorisation:

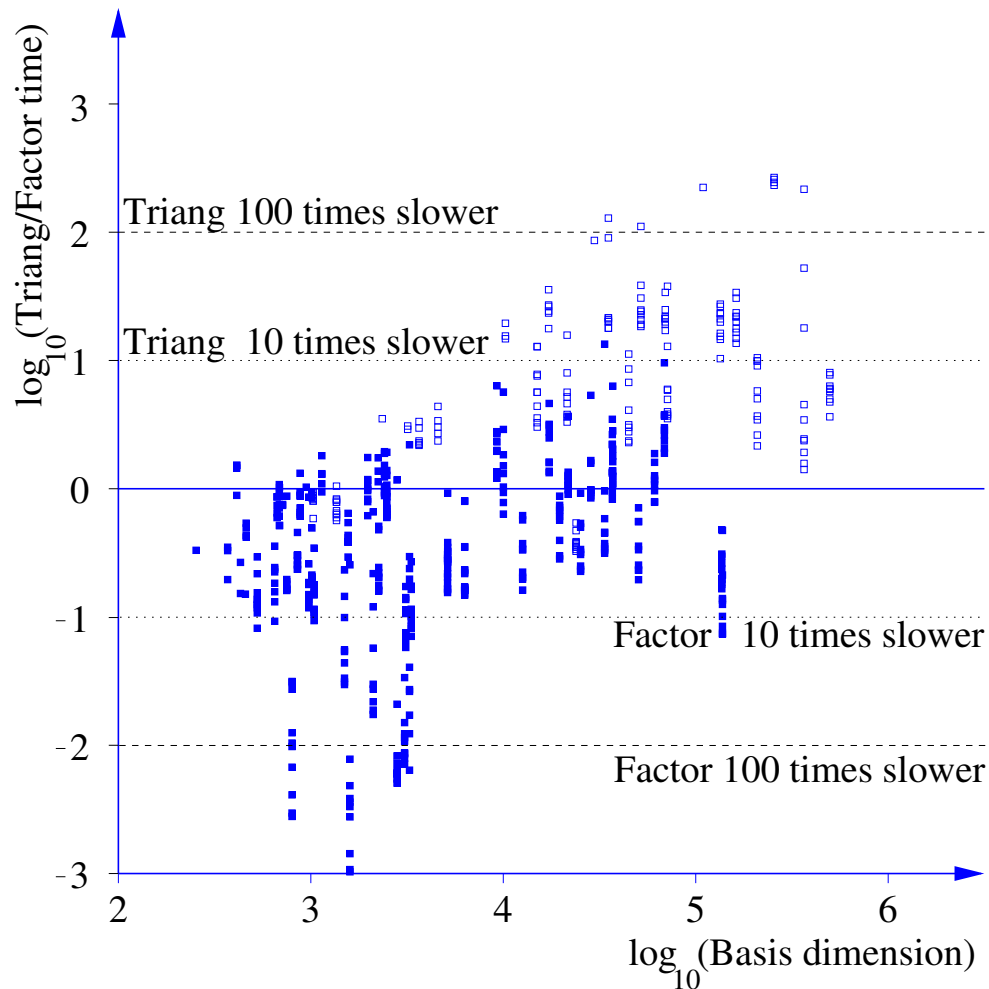
- Gaussian elimination applied the residual **bump**



Relative time for triangularisation and factorization



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

Col set C_p Row set R_p [illegible]

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

Col set C_p Row set R_p [illegible]

Parallel triangularisation: Overview

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

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

Parallel triangularisation: Overview

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

- **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 $p = 1, \dots, N$:

- **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 $p = 1, \dots, N$:

- **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 on each processor using pivot indices from all other processors

Parallel triangularisation: Overview

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

- **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 on each processor using pivot indices from all other processors
- Until there are no row or column singletons

Parallel triangularisation: Overview

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

- **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 on each processor using pivot indices from all other processors
- Until there are no row or column singletons

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

Perform minor iterations in parallel and update in parallel



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

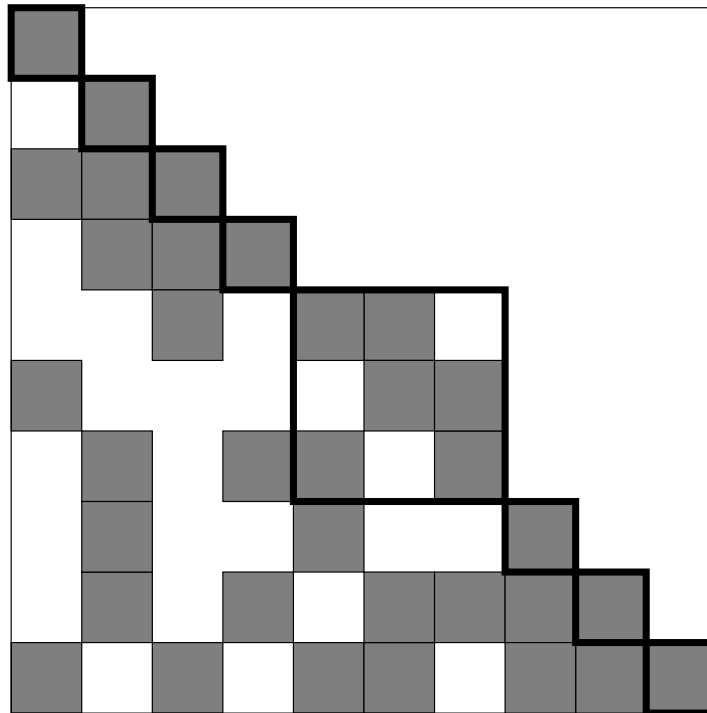
- Row and column counts are
 - initially global;
 - updated according to pivots determined locally;
 - become upper bounds on global counts
- When is it wise to stop performing minor iterations?
 - Too soon: communication overheads dominate
 - Too late: load imbalance
- Aim to find a particular proportion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

100

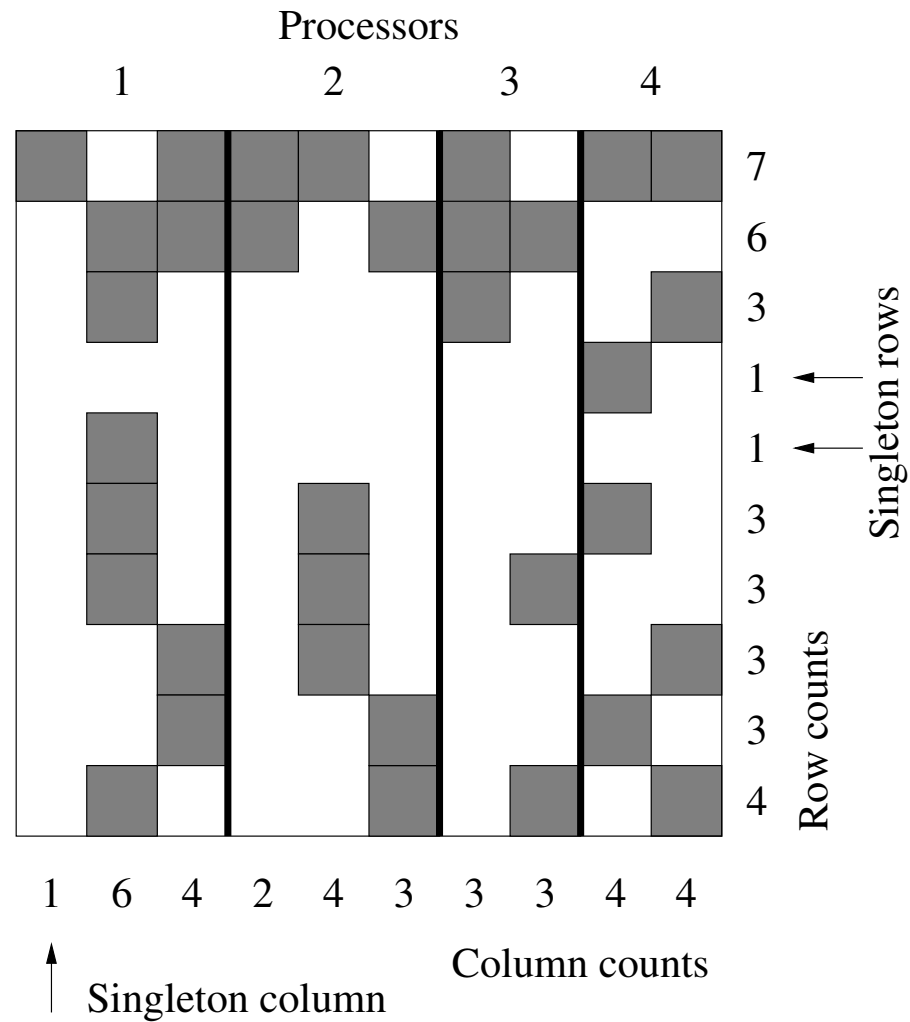
Target % of triangular pivots per major iteration



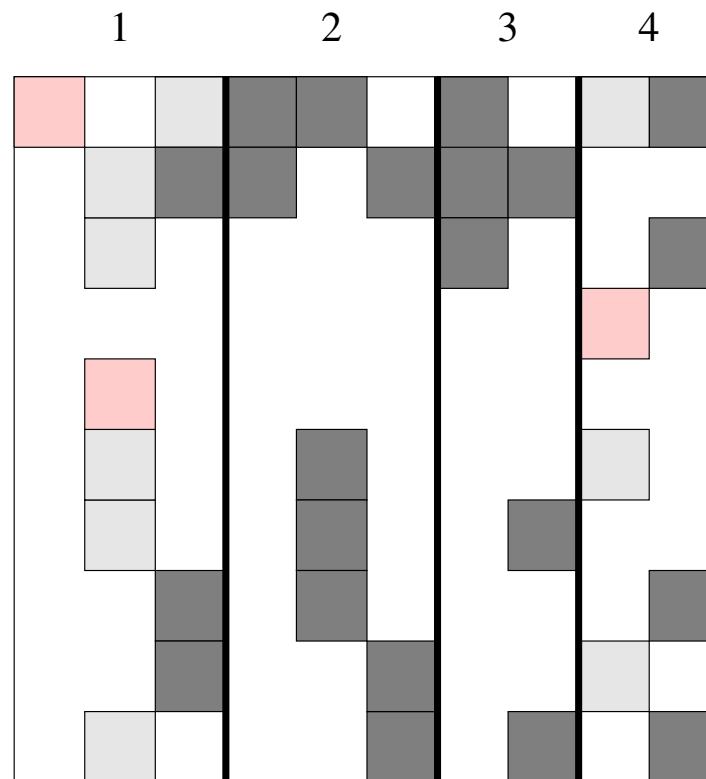
Parallel triangularisation: Example



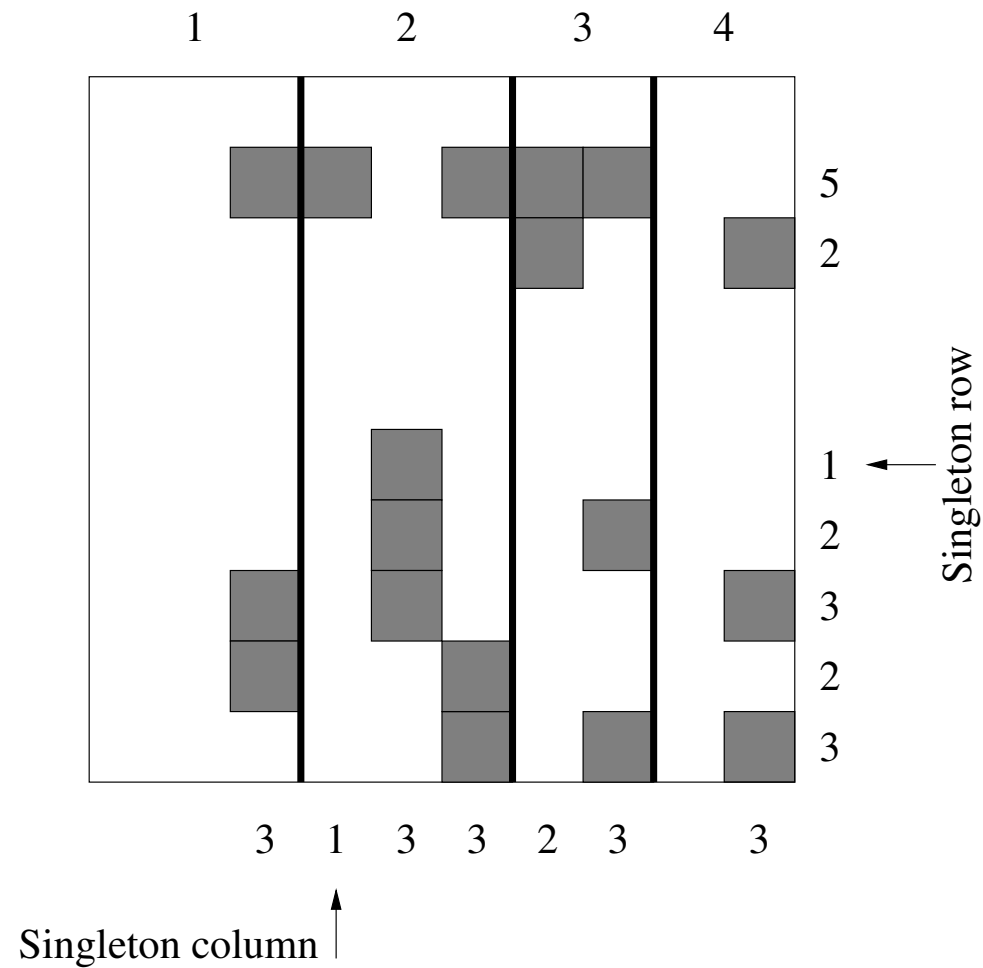
Parallel triangularisation: Iteration 1



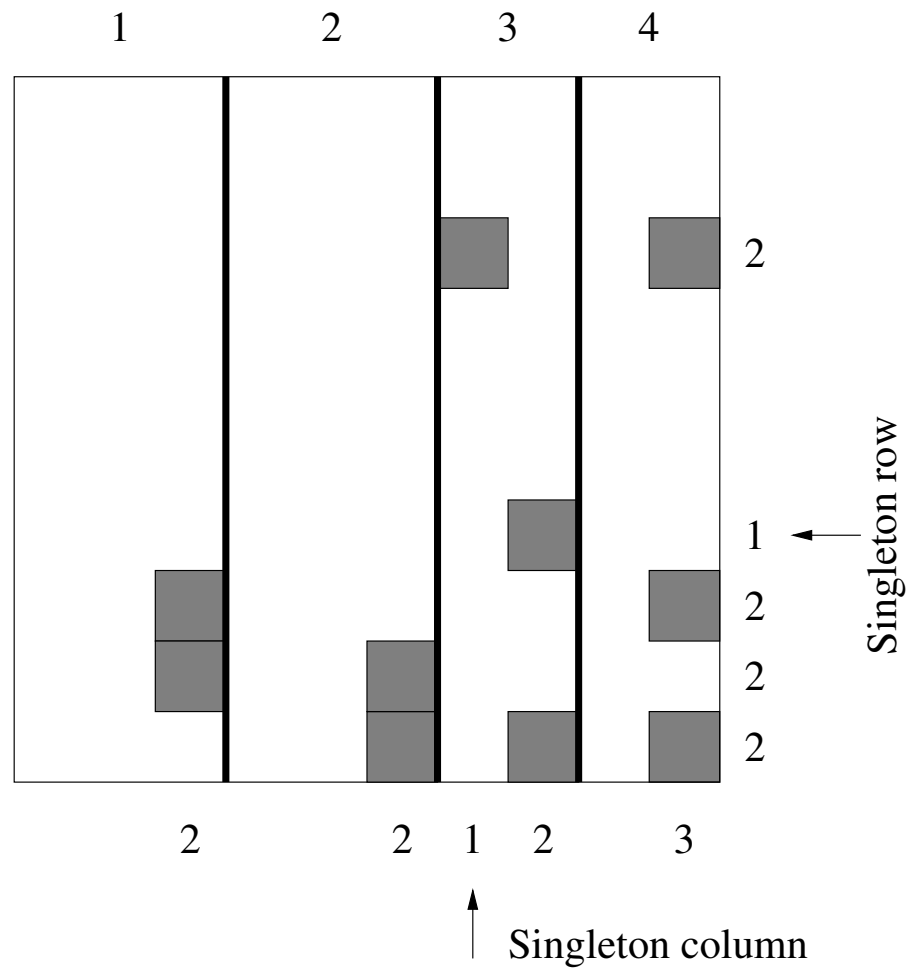
Parallel triangularisation: Iteration 1 result



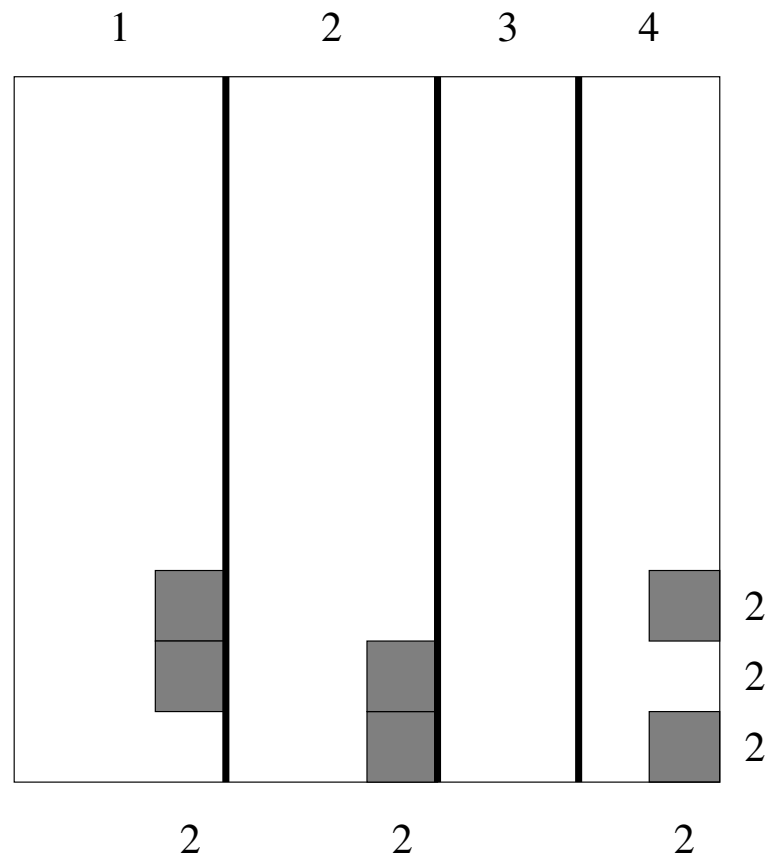
Parallel triangularisation: Iteration 2



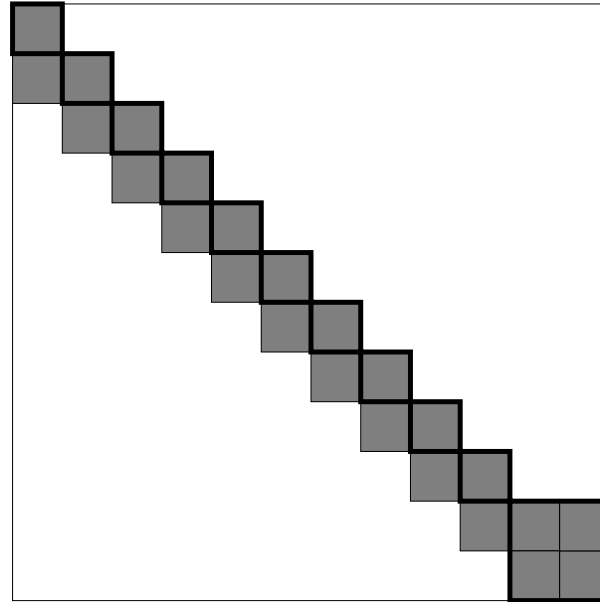
Parallel triangularisation: Iteration 3



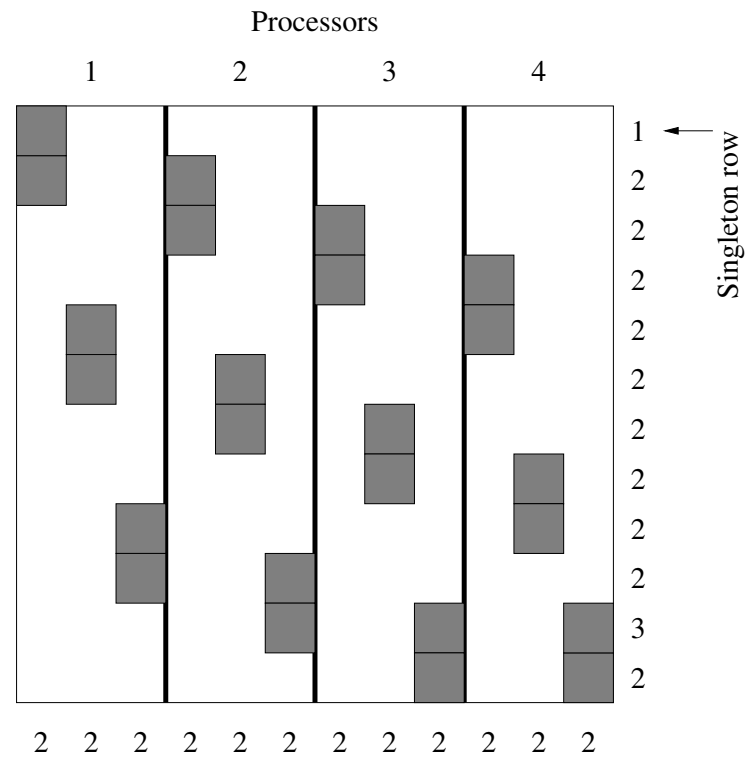
Parallel triangularisation: Iteration 4



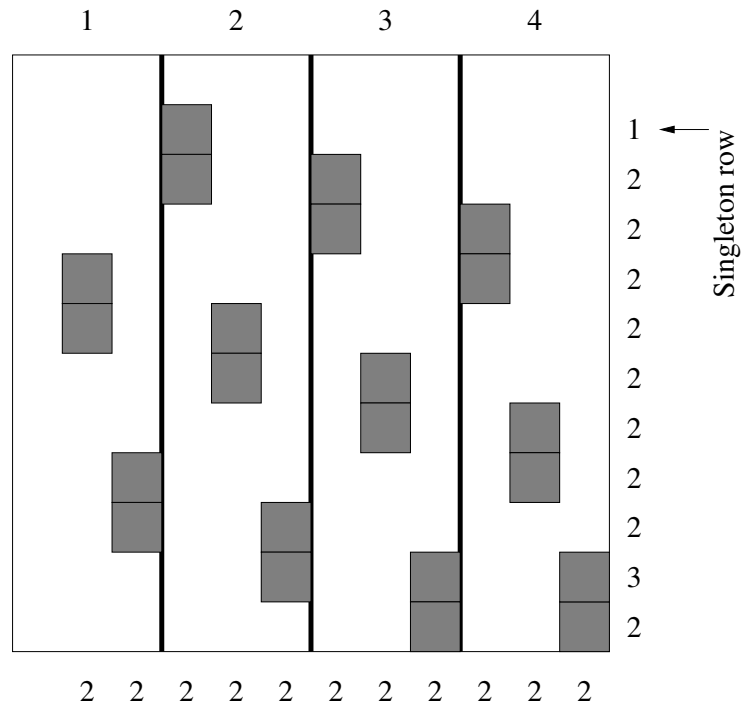
Parallel triangularisation: Worst case behaviour



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

SunFire E15K implementation: good performance

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

SunFire E15K implementation: 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

SunFire E15K implementation: 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

It	Pivots found on processor			
	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



SunFire E15K implementation: “poor” performance

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

SunFire E15K implementation: “poor” performance

It	Pivots found on processor				Pivots
	1	2	3	4	
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

SunFire E15K implementation: “poor” performance

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

It	Pivots found on processor				Pivots
	1	2	3	4	
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%
⋮	⋮	⋮	⋮	⋮	

SunFire E15K implementation: “poor” performance

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

It	Pivots found on processor				Pivots
	1	2	3	4	
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%



SunFire E15K implementation: speedup

- For model `pds-100`
- 156243 rows
- 7485 logicals
- Bump dimension is 1655
- 10% of pivots per major iteration

SunFire E15K implementation: speedup

- For model pds-100
- 156243 rows
- 7485 logicals
- Bump dimension is 1655
- 10% of pivots per major iteration

	Processors				
Pivots	2	4	8	16	32
90%	2.1	3.9	6.1	7.5	5.8
99%	2.0	3.7	5.6	6.4	4.6

SunFire E15K implementation: speedup

- For model pds-100
- 156243 rows
- 7485 logicals
- Bump dimension is 1655
- 10% of pivots per major iteration

Pivots	Processors				
	2	4	8	16	32
90%	2.1	3.9	6.1	7.5	5.8
99%	2.0	3.7	5.6	6.4	4.6
100%	1.9	3.7	5.3	5.6	3.8



SunFire E15K implementation: speedup

Model	Rows	Processors				
		2	4	8	16	32
ken-13	28632	1.2	1.6	1.6	1.2	0.6
mod2	35664	1.1	1.7	1.6	1.1	0.5
stormG2-125	66185	1.2	1.8	2.4	2.4	1.7
deteq27	68672	1.1	1.8	2.3	2.3	1.6
pds-40	66844	0.8	1.4	1.7	1.5	0.8

SunFire E15K implementation: speedup

Model	Rows	Processors				
		2	4	8	16	32
ken-13	28632	1.2	1.6	1.6	1.2	0.6
mod2	35664	1.1	1.7	1.6	1.1	0.5
stormG2-125	66185	1.2	1.8	2.4	2.4	1.7
deteq27	68672	1.1	1.8	2.3	2.3	1.6
pds-40	66844	0.8	1.4	1.7	1.5	0.8
ken-18	105127	1.3	2.7	4.3	4.6	3.4
pds-80	129181	1.8	3.5	5.3	5.1	3.1
pds-100	156243	1.9	3.7	5.3	5.6	3.8
sgpf5y6	246077	1.6	3.8	7.0	8.3	8.0
watson_2	352013	1.7	3.3	5.8	7.4	6.3

- Good parallel performance is aided by cache effects
- Parallel performance limited by
 - Communication costs
 - Overheads of update loop parallelisation



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 $O(m \log N)$ 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 $O(m \log N)$ for computation cost $O(\tau)$
- Parallel implementation gives good speed-up, particularly for large problems

Conclusions

- Matrix triangularisation identified as dominant serial cost for hyper-sparse LPs
- Significant scope for parallelisation with scheme presented
Communication cost of $O(m \log N)$ for computation cost $O(\tau)$
- Parallel implementation gives good speed-up, particularly for large problems

See

- **Slides:** <http://www.maths.ed.ac.uk/hall/Talks>

Thank you

