Design and Implementation of a Parallel Threshold Markowitz Algorithm

Tim Davis, **Iain Duff** and **Stojce Nakov**

Texas A&M University, **CERFACS**, STFC Rutherford Appleton Laboratory

Sparse Days,
Toulouse, France

$11^{th} - 12^{th} July 2019$

Introduction

Goal: solve highly unsymmetric sparse systems.

Introduction

Goal: solve highly unsymmetric sparse systems.

- These do occur in, for example ...

Introduction

Goal: solve highly unsymmetric sparse systems.

- These do occur in, for example ...
  - chemical engineering
  - linear programming
  - economic modelling
  - power systems

Introduction

Goal: solve highly unsymmetric sparse systems.

- These do occur in, for example ...
  - chemical engineering
  - linear programming
  - economic modelling
  - power systems

Both code and matrices can be very evil

Highly unsymmetric systems

Highly unsymmetric systems

$$si(A) = \frac{number_{i \neq j}\{a_{ij} * a_{ji} \neq 0\}}{nz\{A\}}$$

Highly unsymmetric systems

$$si(A) = \frac{number_{i \neq j}\{a_{ij} * a_{ji} \neq 0\}}{nz\{A\}}$$

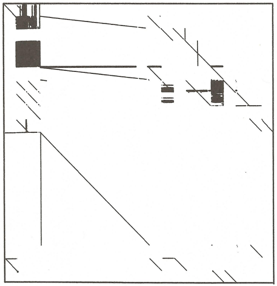
A matrix A is highly unsymmetric if

$$si(A) < 0.9$$

Highly unsymmetric systems

$$si(A) = \frac{number_{i \neq j}\{a_{ij} * a_{ji} \neq 0\}}{nz\{A\}}$$

A matrix A is highly unsymmetric if $si(A) < 0.9$

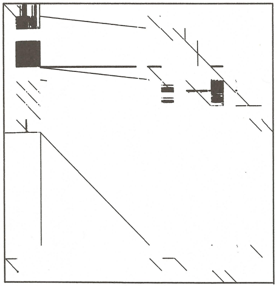


Matrix from econometric model of SE Asia

Highly unsymmetric systems

$$si(A) = \frac{number_{i \neq j}\{a_{ij} * a_{ji} \neq 0\}}{nz\{A\}}$$

A matrix A is highly unsymmetric if
$si(A) < 0.9$



Matrix from econometric model of SE Asia

State of the art solvers: MA48, UMFPACK, SuperLU, MUMPS.

Threshold Markowitz pivoting

Threshold Markowitz pivoting

- Threshold value
  Consider only entries a_{ij} that satisfy

$$|a_{ij}| \geq u * \max_k |a_{kj}|, k = 1, n$$

  where u is a threshold parameter $0 < u \leq 1.0$.

Threshold Markowitz pivoting

- Threshold value
  Consider only entries a_{ij} that satisfy

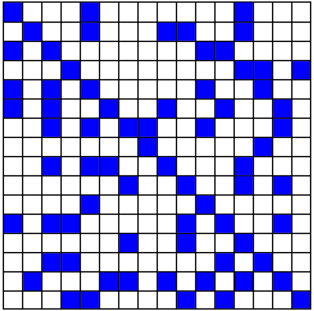
$$|a_{ij}| \geq u * \max_k |a_{kj}|, k = 1, n$$

  where u is a threshold parameter $0 < u \leq 1.0$.
- Markowitz threshold
  If there are r_i entries in row i and c_j entries in column j, the Markowitz count for the entry a_{ij} is given by $M(a_{ij}) = (r_i - 1)(c_j - 1)$. In each column, consider only the entries that that satisfy the threshold test $M(a_{ij}) \leq \alpha * \min_{markowitz}$ where $\min_{markowitz}$ is the minimum Markowitz cost for an entry in the matrix.

High-level algorithm description

while $size(A) > 1$ **do**

end while

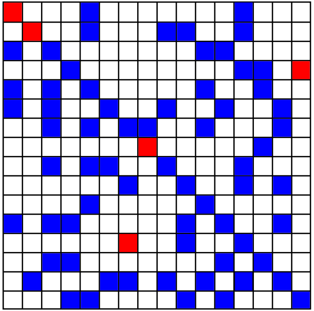


High-level algorithm description

while $size(A) > 1$ **do**

  Find a set of independent pivots

end while

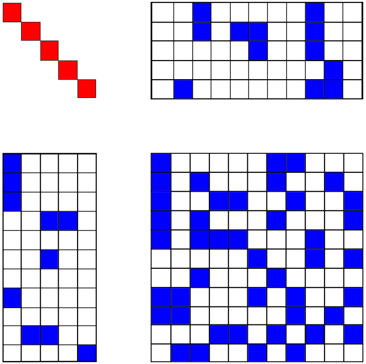


High-level algorithm description

while $size(A) > 1$ **do**

Find a set of **independent** *pivots*

end while



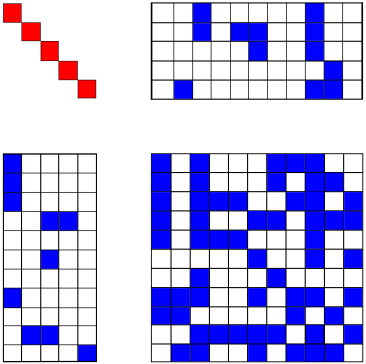
High-level algorithm description

while $\mathit{size}(A) > 1$ **do**

  Find a set of independent pivots

  Update the trailing matrix

end while



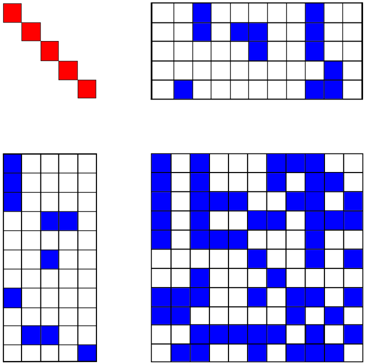
High-level algorithm description

while $\quad density(A) < eps \quad$ **do**

 Find a set of independent pivots

 Update the trailing matrix

end while



High-level algorithm description

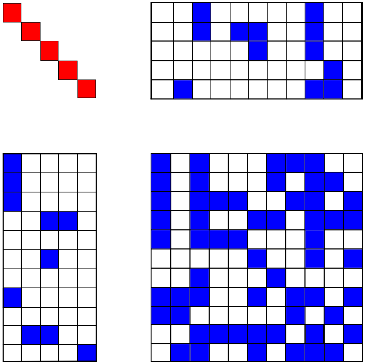
while $\color{red}{density(A) < eps}$ **do**

Find a set of independent pivots

Update the trailing matrix

end while

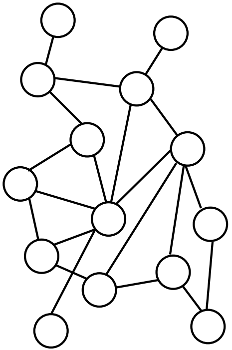
Switch to dense factorization.



Luby's Algorithm for a **M**aximal **I**ndependent **S**et (**MIS**)

Input G = (V,E) an undirected graph
Output $I \subseteq V$, an MIS

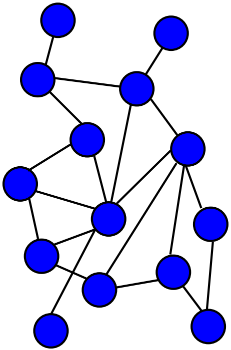
$I \leftarrow \varnothing$
$G' = (V', E') \leftarrow G = (V, E)$
while $V' \neq \varnothing$ **do**
  assign random score to each node in V'
  $I' \leftarrow$ *nodes having highest score among their neighbours*
  $I \leftarrow I \cup I'$
  $Y \leftarrow I' \cup N(I')$
  Set $G' = (V', E')$ *to the induced subgraph on* $V' - Y$
end while



Luby's Algorithm for a **M**aximal **I**ndependent **S**et (**MIS**)

Input G $=$ (V,E) an undirected graph
Output $I \subseteq V$, an MIS

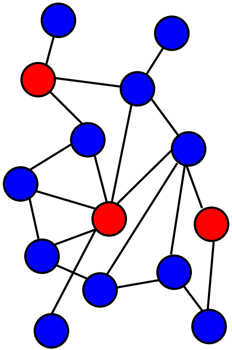
$I \leftarrow \varnothing$
$G' = (V', E') \leftarrow G = (V, E)$
while $V' \neq \varnothing$ **do**
  assign random score to each node in V'
  $I' \leftarrow$ *nodes having highest score among their neighbours*
  $I \leftarrow I \cup I'$
  $Y \leftarrow I' \cup N(I')$
  Set $G' = (V', E')$ *to the induced subgraph on* $V' - Y$
end while



Luby's Algorithm for a **M**aximal **I**ndependent **S**et (**MIS**)

Input G = (V,E) an undirected graph
Output $I \subseteq V$, an MIS

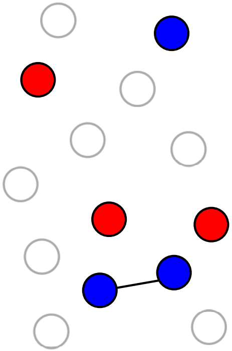
$I \leftarrow \varnothing$
$G' = (V', E') \leftarrow G = (V, E)$
while $V' \neq \varnothing$ **do**
  assign random score to each node in V'
  $I' \leftarrow$ *nodes having highest score among their neighbours*
  $I \leftarrow I \cup I'$
  $Y \leftarrow I' \cup N(I')$
  Set $G' = (V', E')$ *to the induced subgraph on* $V' - Y$
end while



Luby's Algorithm for a **M**aximal **I**ndependent **S**et (**MIS**)

Input G = (V,E) an undirected graph
Output $I \subseteq V$, an MIS

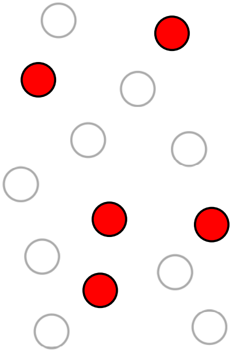
$I \leftarrow \varnothing$
$G' = (V', E') \leftarrow G = (V, E)$
while $V' \neq \varnothing$ **do**
  assign random score to each node in V'
  $I' \leftarrow$ *nodes having highest score among their neighbours*
  $I \leftarrow I \cup I'$
  $Y \leftarrow I' \cup N(I')$
  Set $G' = (V', E')$ *to the induced subgraph on* $V' - Y$
end while



Luby's Algorithm for a **M**aximal **I**ndependent **S**et (**MIS**)

Input G = (V,E) an undirected graph
Output $I \subseteq V$, an MIS

$I \leftarrow \varnothing$
$G' = (V', E') \leftarrow G = (V, E)$
while $V' \neq \varnothing$ **do**
  assign random score to each node in V'
  $I' \leftarrow$ *nodes having highest score among their neighbours*
  $I \leftarrow I \cup I'$
  $Y \leftarrow I' \cup N(I')$
  Set $G' = (V', E')$ *to the induced subgraph on* $V' - Y$
end while

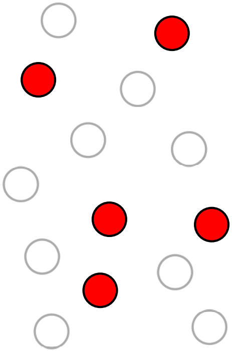


We then continue the process to obtain an MIS with 5 nodes.

Luby's Algorithm for a **M**aximal **I**ndependent **S**et (**MIS**)

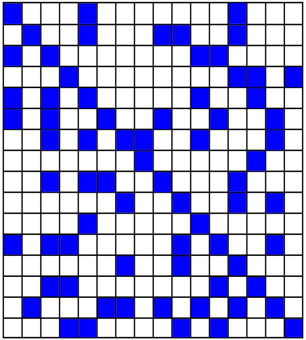
Input G = (V,E) an undirected graph
Output $I \subseteq V$, an MIS

$I \leftarrow \varnothing$
$G' = (V', E') \leftarrow G = (V, E)$
while $V' \neq \varnothing$ **do**
  assign random score to each node in V'
  $I' \leftarrow$ *nodes having highest score among their neighbours*
  $I \leftarrow I \cup I'$
  $Y \leftarrow I' \cup N(I')$
  Set $G' = (V', E')$ *to the induced subgraph on* $V' - Y$
end while



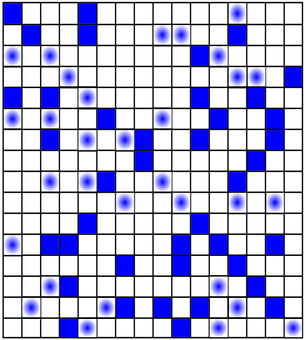
We have adopted the Luby's algorithm for directed graphs.

Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library



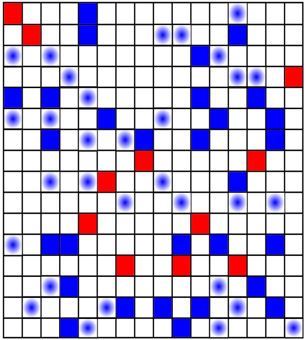
Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

Discard all numerically ineligible entries and calculate the minimum Markowitz cost.



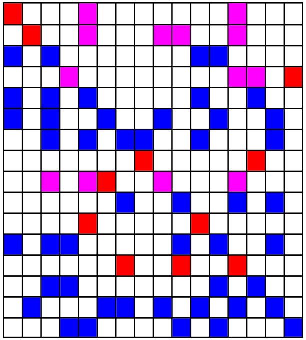
Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

Choose potential pivots that satisfy the Markowitz threshold test, one for each column and assign a score for each one. The pivot score is associated with the column.



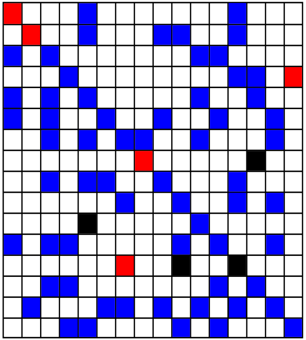
Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

For each potential pivot scan its row, comparing its score with the score of the columns with entries in the row.



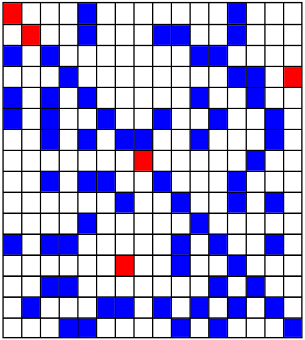
Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

Discard the column with a potential pivot with lower score.



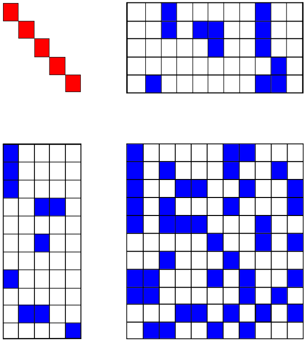
Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

The set of independent pivots.



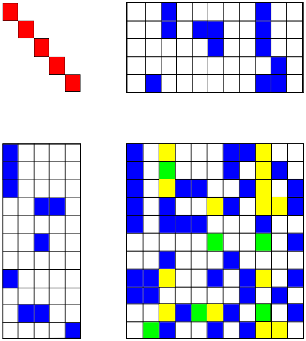
Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

This results in a reordered matrix of the form:



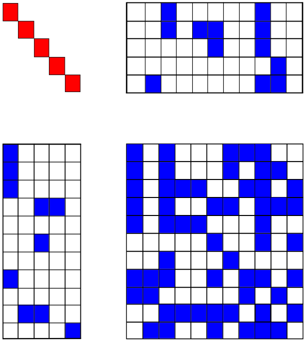
Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

The trailing matrix is then updated by a sparse matrix-matrix multiply.



Parallel Solver for Highly Unsymmetric Matrices (ParSHUM) library

The pivot search is then repeated on the reduced matrix.



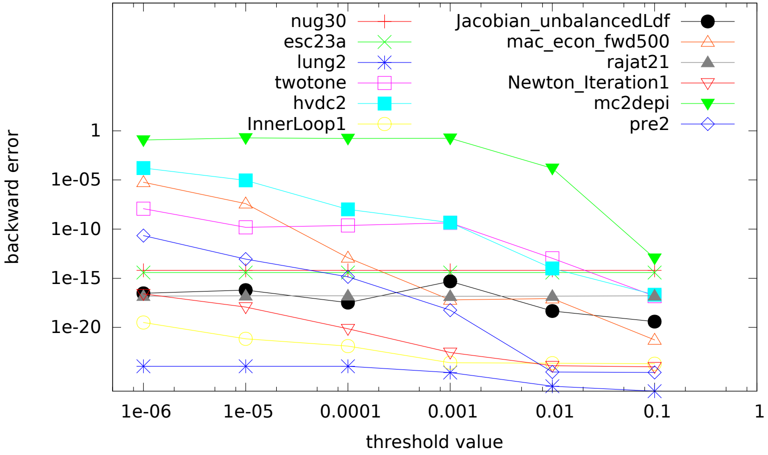
Experimental set-up

HPC2N platform at
Umeå University

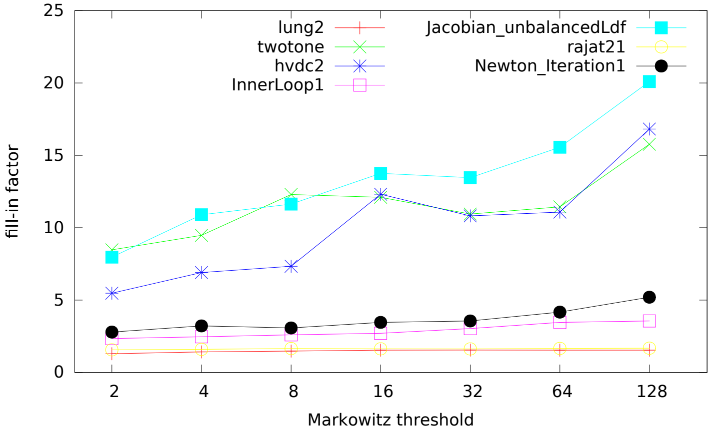
Each node:
28 Intel Xeon E5-2690v4
35 MB of shared L3 cache
128 GB RAM memory

Matrix	Order $\times 10^3$	Entries $\times 10^6$	*si*
nug30	52.4	0.24	0.00
esc32a	63.6	0.31	0.00
lung2	109	0.49	0.57
twotone	120	1.22	0.26
hvdc2	190	1.35	0.99
InnerLoop1	197	0.75	0.44
Jacobian_unbalancedLdf	203	2.41	0.80
mac_econ_fwd500	206	1.27	0.07
rajat21	411	1.89	0.76
Newton_Iteration1	427	2.38	0.14
esc64a	504	2.40	0.00
mc2depi	525	2.10	0.00
pre2	659	5.96	0.36

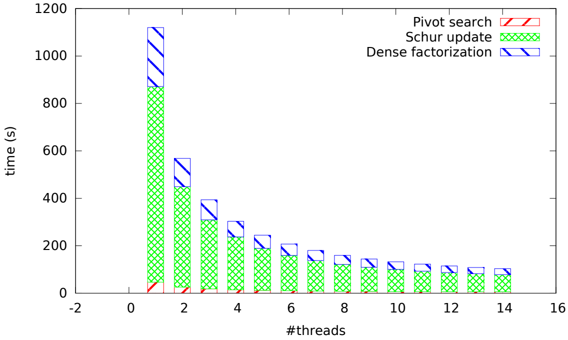
Effect of the threshold value on the backward error

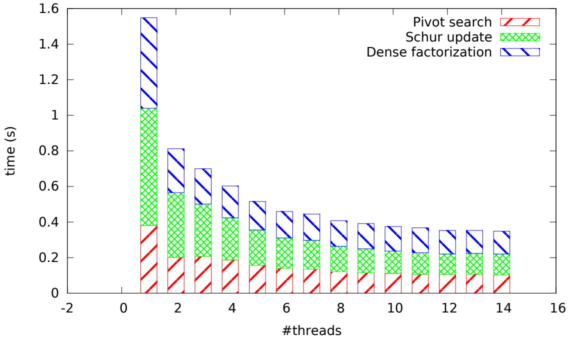


Effect of the Markowitz threshold on the fill-in factor $(nz(LU)/nz(A))$



Scalability on the mc2depi and twotone matrices





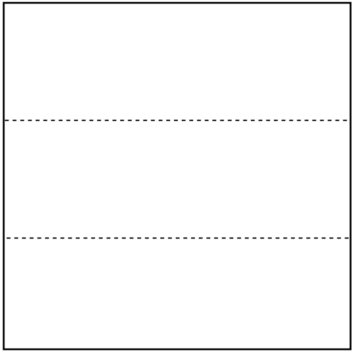
Comparison of ParSHUM, MUMPS and UMFPACK on single NUMA node (14 cores)

	ParSHUM		MUMPS		UMFPACK	
Matrix	time	fill-in	time	fill-in	time	fill-in
nug30	**0.71**	142.	6.19	730	48.15	463.
esc32a	**0.46**	70.9	10.59	758	71.0	341.
lung2	**0.04**	1.48	—	—	0.10	1.44
twotone	**0.35**	7.37	2.22	25.4	0.49	5.45
hvdc2	**0.38**	6.91	1.13	2.14	**0.38**	2.06
InnerLoop1	**0.13**	2.71	1.64	3.90	0.30	2.48
Jacobian_unbalancedLdf	0.99	10.9	1.52	3.45	**0.74**	3.88
mac_econ_fwd500	33.5	450.	10.5	56.4	**4.62**	57.5
rajat21	**0.11**	1.64	—	—	44.5	1.89
Newton_Iteration1	**0.46**	3.46	4.26	6.02	1.00	2.55
esc64a	**12.8**	155	—	2151	—	—
mc2depi	104.	302.	4.60	25	**4.36**	38.6
pre2	13.0	57.0	**9.73**	18.4	25.8	32.3

Table: The execution time and the fill-in factor for ParSHUM, MUMPS and UMFPACK.

Towards distributed memory: Singly Bordered Block decomposition using Zoltan

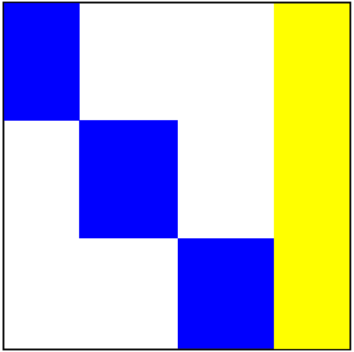
Global view



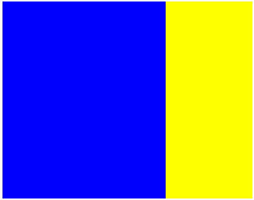
Local view

Towards distributed memory:
Singly Bordered Block decomposition using Zoltan

Global view

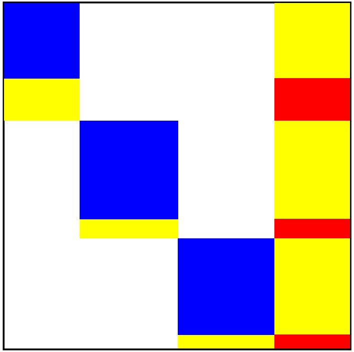


Local view

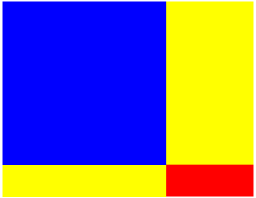


Towards distributed memory:
Singly Bordered Block decomposition using Zoltan

Global view

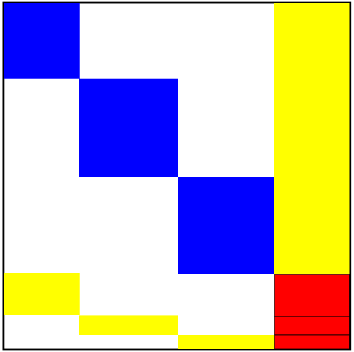


Local view

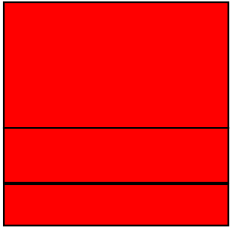


Towards distributed memory: Singly Bordered Block decomposition using Zoltan

Global view



Local view



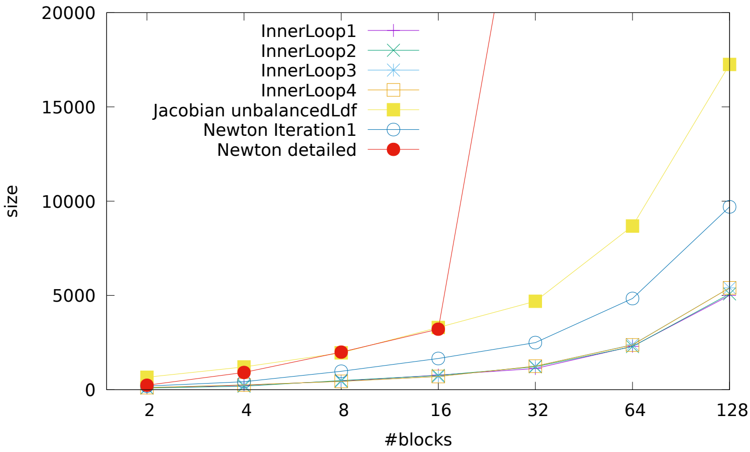
Experimental set-up

HPC2N platform at
Umeå University

Each node:
28 Intel Xeon E5-2690v4
35 MB of shared L3 cache
128 GB RAM memory

Matrix	Order $\times 10^3$	Entries $\times 10^6$	*si*
InnerLoop1	197	0.75	0.44
InnerLoop1	197	0.75	0.44
InnerLoop1	197	0.75	0.44
InnerLoop1	197	0.75	0.44
Jacobian_unbalancedLdf	203	2.41	0.80
Newton_Iteration1	427	2.38	0.14
Newton_detailed	7355	24	0.29

Bordered Block size



Scalability

	#MPI processes			
Matrix	1	2	4	8
InnerLoop1	0.15	0.08	0.06	0.05
InnerLoop2	0.15	0.08	0.06	0.05
InnerLoop3	0.15	0.09	0.06	0.05
InnerLoop4	0.15	0.09	0.06	0.05
Jacobian_unbalancedLdf	1.11	0.52	0.35	0.18
Newton_Iteration1	0.49	0.30	0.14	0.09
Newton_detailed	7.39	3.01	1.67	1.63

The execution time in seconds for ParSHUM on the test matrices partitioned in SBBD form. One numa node (14 cores) is used per process.

Comparison with MUMPS and SuperLU

	ParSHUM		MUMPS		SuperLU	
Matrix	time	fill-in	time	fill-in	time	fill-in
InnerLoop1	**0.05**	3.03	0.23	3.90	1.11	6.02
InnerLoop2	**0.05**	2.82	0.25	3.72	1.11	5.57
InnerLoop3	**0.05**	2.78	0.17	3.72	1.10	5.60
InnerLoop4	**0.05**	2.41	0.15	3.73	1.15	5.56
Jacobian_unbalancedLdf	**0.18**	6.75	0.28	3.45	1.05	3.59
Newton_Iteration1	**0.09**	3.57	0.42	6.02	2.59	5.60
Newton_detailed	**1.63**	5.32	7.88	6.09	52.4	5.39

Conclusions and future work

Conclusions:

- We have developed a generalization of Luby's algorithm for directed graphs.
- We have used this to develop a multi-threaded threshold Markowitz code (ParSHUM library).
- In general it outperforms established codes.

Future work:

- Ongoing work on SBBD to exploit distributed systems.
- Develop a GPU version of the solver.

THANK YOU FOR YOUR ATTENTION