

# Issues in the Design of Scalable Out-of-core Dense Symmetric Indefinite Factorization Algorithms

Peter Strazdins

Department of Computer Science,  
Australian National University,

<http://cs.anu.edu.au/~Peter.Strazdins/seminars>

2 June 2003

# 1 Talk Outline

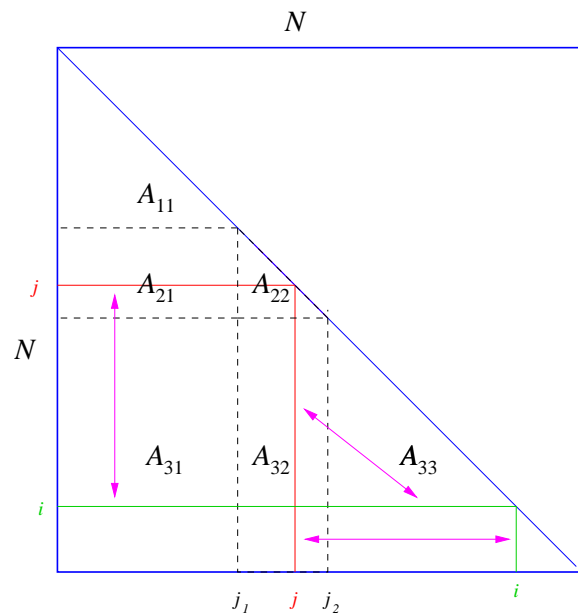
- introduction to (large) general dense symmetric systems factorization
- diagonal pivoting algorithms:
  - challenges for the parallel out-of-core case
  - the (exhaustive) elimination block search strategy
- left-looking parallel out-of-core algorithms:
  - blocking and data layout issues
  - a slab-based algorithm
  - memory scalability issues and a block-based algorithm
- conclusions and future work

## 2 Introduction to Large Symmetric Indefinite Systems Solution

- $N \times N$  **general** (ie. indefinite:  $x^T A x \not\geq 0, \|x\| > 0$ ) symmetric systems of linear equations, eg.  $Ax = b$ , arise in:
  - incompressible flow computations, linear and non-linear optimization, electromagnetic scattering & field analysis & data mining
- a direct solution for  $x$  is the most general and accurate method
  - require  $O(N^3)$  FLOPS, dominated by the fact'n of  $A$ , ie.  $A \rightarrow PLDL^T P^T$
- only parallel out-of-core algorithms for LU,  $LL^T$  and QR developed so far
  - left-looking versions of blocked algorithms are preferred:
    - the number of writes to disk is only  $O(N^2)$ ; easier to checkpoint
  - two approaches:
    - slab-based: whole block of columns being eliminated are kept in core
    - block-based: only part of the column block in core
      - ✓ permits a wider column block  $\Rightarrow$  better memory scalability
      - ✗ can't be (efficiently) applied for LU with strict partial (row) pivoting
    - seemingly harder still for  $LDL^T$ , which requires symmetric pivoting

### 3 Diagonal Pivoting Algorithms

- store  $A$  in lower triangular half; overwrite with  $L$  and  $D$
- partial left-looking blocked algorithm: ( $A_{*1}$  factored,  $A_{*2}$  being factored,  $A_{*3}$  untouched)



1. apply pivots from  $A_{11}$  to  $A_2 = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix}$
2.  $A_{22} -= A_{21}W_{21}^T$ ,  $W_{21} = D_{11}A_{21}$
3.  $A_{32} -= A_{31}W_{21}^T$
4. eliminate block column  $A_2$   
(applying pivots from within  $A_2$ )
5. apply pivots (row interchanges) from  $A_2$  to  $A_1 = \begin{pmatrix} A_{21} \\ A_{31} \end{pmatrix}$

- diagonal pivoting methods use symmetric (row & column) interchanges based on  $1 \times 1$  or  $2 \times 2$  'pivots'
  - nb. interchange  $i \leftrightarrow j$ :  $A_{i,j}$  is not moved;  $A_{j,j} \leftrightarrow A_{i,i}$
- recently developed stable methods include the bounded Bunch-Kaufman and exhaustive block search methods

## 4 Challenges for the Parallel Out-of-core Case

- here,  $A$  is distributed over a  $P \times Q$  processor grid with an  $r \times r$  block-cyclic matrix distribution:
  - i.e. (storage) block  $(i, j)$  will be on processor  $(i \bmod P, j \bmod Q)$
  - assume this applies to both disk and memory
  - assume storage is column-oriented for both
- in the left-looking algorithm, consider candidate pivot  $i$  lying outside  $A_2$ 
  - $a_i$  must be aligned with  $a_j$  (read a large number of remote disk blocks)
  - *all* updates from  $A_1$  and  $A_2$  (so far) must be applied to it (large number of disk accesses)
  - if suitable,  $a_i$  (in  $A_{33}$ ) must be over-written by the *original value* of  $a_j$
  - pivot  $i$  may not even be suitable!
- if pivot  $i$  is inside  $A_2$ , only overhead is in message exchanges ...

## 5 The Exhaustive Block Search Strategy

- search for (stable) pivots in the current elimination block ( $A_2$ )
  - consider *any*  $1 \times 1$  or  $2 \times 2$  pivots in remaining columns  $j : j_2$
  - well-known stability tests can be used
- originally developed for sparse matrices (Duff & Reid, 1983)
  - has a large payoff in preserving the sparsity
- useful in the parallel in-core case if search succeeds within the current storage block
  - ✓ little message overhead in searches / interchanges
  - ✗ overhead of extra searches (finding column maximums) can outweigh
    - ⇒ limit search to  $\omega_s = 16$  columns found to be optimal
      - for highly indefinite matrices: only  $0.15N$  searches outside  $A_2$  required
      - for weakly indefinite matrices, this reduces to  $< 0.05N$
    - the bounded Bunch-Kaufman algorithm used for searches outside  $A_2$
- a successful block search can minimize the overheads in the out-of-core case

## 6 Parallel OOC Algorithms: Blocking and Data Layout Issues

- must consider all levels of the (parallel) memory hierarchy, exploiting locality wherever possible:
  - $\omega_a$ : the top-level algorithm's blocking factor
    - should be as large as possible, to maximize re-use at disk level
    - for slab-based algorithm,  $\frac{N\omega_a}{PQ} \leq M$  ( $M$  is memory capacity)
  - $\omega_c$ : optimal blocking factor for matrix multiply
    - need  $\omega_a \geq \omega_c$  to maximize re-use at cache level
  - $r$ : the storage block size; can be chosen to minimize message overheads
    - $r = \omega_a$  best for this but may cause unacceptable load imbalance (disk and CPU)
  - $\omega_s$ : the number of columns to be searched,  $\omega_s \leq \omega_a$ 
    - must be sufficiently large to achieve a very high success rate
  - $\omega_d$ , where  $\omega_d^2 =$  disk block size
    - for block-based algorithm,  $\omega_a^2 \gg \omega_d^2$  to amortize disk latencies
- organization of  $A$  upon disk: could be [ $(\omega_s$  or  $r$  - sized) block] row/column oriented, depending on algorithm

## 7 A Slab-based Algorithm Exploiting Pivoting Locality

- a modification of the left-looking algorithm:
  - the  $0 \leq u_1 < \omega_a$  un-eliminated columns left over from previous stage become the 1st  $u_1$  columns of  $A_2$  for this stage
  - steps 1–3 only operate on last  $\omega_a - u_1$  columns
  - insert step 3a: block interchange of the 1st & last  $u'_1 = \min(u_1, \omega_a - u_1)$  columns of  $A_2$
  - step 4: uses the exhaustive block search (+ a non-local search to ensure  $\geq 1$  columns eliminated)
- provided no non-local searches were needed, step 1 is empty, and all interchanges are kept within slabs
  - as  $\omega_a > r$  for other aspects of performance, this will require some communication
- column-oriented disk stage will be optimal
- performance will rely on highly successful block searches, eg.  $u_i \leq 0.2\omega_a$ 
  - for  $u_1 \approx 0$ , the number of words read from disk (dominated by steps 2–3) is:

$$\sum_{i=0}^{N/\omega_a} i\omega_a(N - i\omega_a) = \frac{N^3}{6\omega_a} + O(N^2)$$

## 8 A Block-Based Algorithm

- only for very large problems (or small processor grids) will slab-based algorithms result in a too small  $\omega_a$
- can convert the slab-based algorithm to a block-based one in which only  $\omega_a$  ( $\omega_c \leq \omega_s \leq \omega_a$ ) columns are kept in core:
  - apply a left-looking factorization internally to **step 4** (factorize  $A_2$ ), using a blocking factor of  $\omega_s$ 
    - the columns of  $A_2$  must now be read repeatedly as they were for  $A_1$  for the slab-based algorithm
    - the number of extra reads is given by:
 
$$\sum_{i=0}^{N/\omega_a} \sum_{j=0}^{\omega_a/\omega_s} i\omega_a \cdot j\omega_s = \frac{N^2\omega_a}{4\omega_s} + O(N)$$
  - for the top-level algorithm, **steps 2–3** proceed using  $\omega_a \times \omega_a$  sized blocks
  - as these form the dominant accesses, a row-block disk storage is optimal

## 9 Conclusions

- solving general symmetric systems has an accuracy–performance trade-off
  - for the parallel out-of-core case, potentially very large disk/message overheads from non-local symmetric interchanges
- however, efficient slab- and even (a limited) block-based algorithms exist, based on applying exhaustive pivots searches in (overlapping) elimination blocks
  - seems likely that *most* searches and interchanges can be kept within these blocks
  - performance is highly dependent on overlap being small
  - block-based algorithm has better memory scalability but slab-based algorithm will normally be adequate in practice
- future (current!) work includes:
  - investigating whether the overlap can be kept small (eg.  $\frac{u_1}{\omega_a} \leq 0.2$ ) for any indefinite matrices
  - developing and evaluating these highly complex algorithms!