MANCHESTER 1824

The University of Manchester

Symmetric Pivoting in ScaLAPACK

Craig Lucas
University of Manchester

Cray User Group
8 May 2006, Lugano

MANCHESTER 1824

The University of Manchester

Introduction

Introduction

- We wanted to parallelize a serial algorithm for the pivoted Cholesky factorization for ScaLAPACK [1] (Scalable Linear Algebra PACKage) , based on existing serial code [4].
- The matrix to be factored is symmetric and requires symmetric (or complete) pivoting. At each stage of the algorithm two rows and two columns are swapped
- We will look at how well this performs on Cray systems, in particular an XD1 and XT3
- The code uses existing ScaLAPACK and PBLAS [2] (Parallel Basic Linear Algebra Subprograms) routines.

Form of a Cholesky Factorization

- If *A* is symmetric positive definite then

$$A = LL^T$$

- If *A* is positive *semi*definite, of rank *r*, then

$$P^T AP = LL^T$$

where P is a permutation matrix and *L* is unique, with positive diagonal elements, in the form

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{12} & 0 \end{bmatrix}, \quad L_{11} \text{ is } r \times r$$

ScaLAPACK

- Nothing uses symmetric pivoting in ScaLAPACK at present.
- ScaLAPACK uses *block cyclic* data distribution according to the BLACS [3] (Basic Linear Algebra Communication Subroutines) process grid:

	0	1	2	3
0	0	1	2	3
1	4	5	6	7

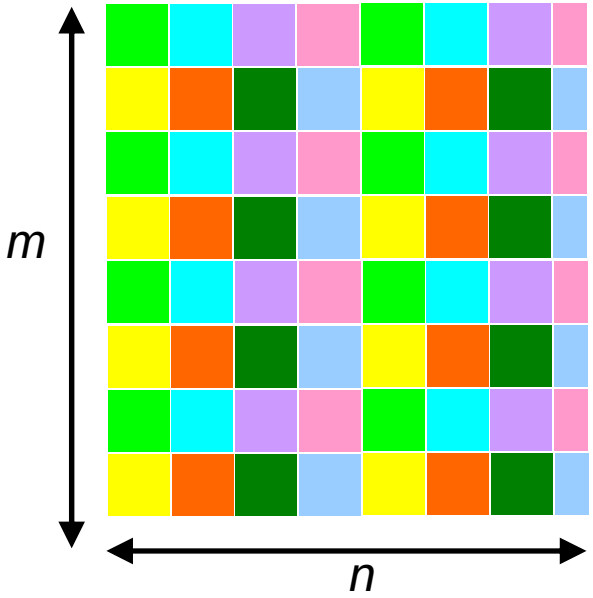
2 by 4 process grid

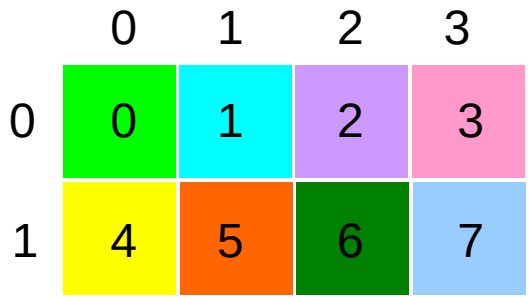
Process of rank 4 is at coordinate (1,0)

- Process grid is enclosed in a *context*, like an MPI communicator

Block Cyclic Distribution

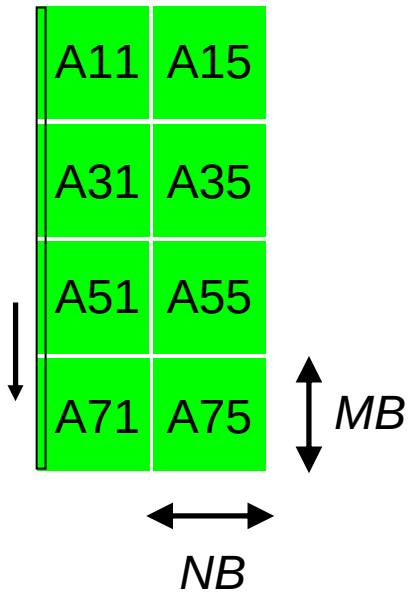
- Data is distributed in a *Block Cyclic* manner, according to the BLACS process grid and blocks of *MB* by *NB*





Process at (0,0) stores (4 X MB) X (2 X NB) of the global array.

Storage is Fortran column major order.



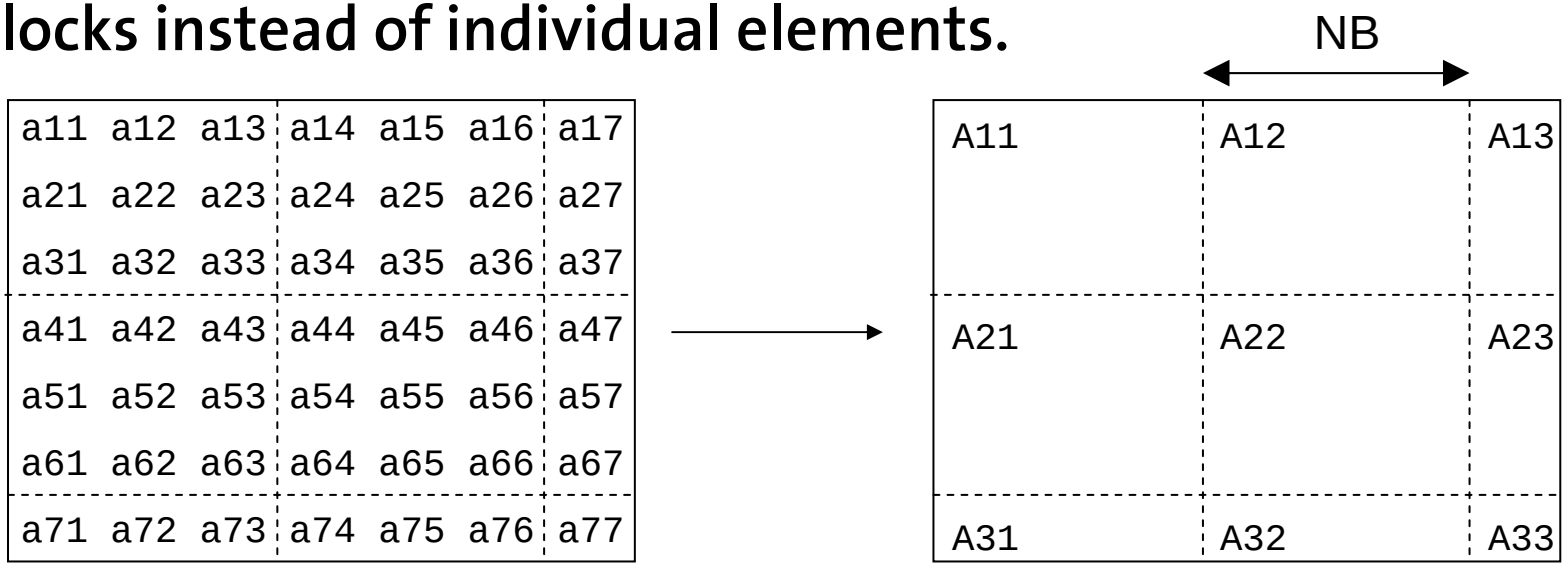
MANCHESTER 1824

The University of Manchester

Algorithm

Blocked Algorithm

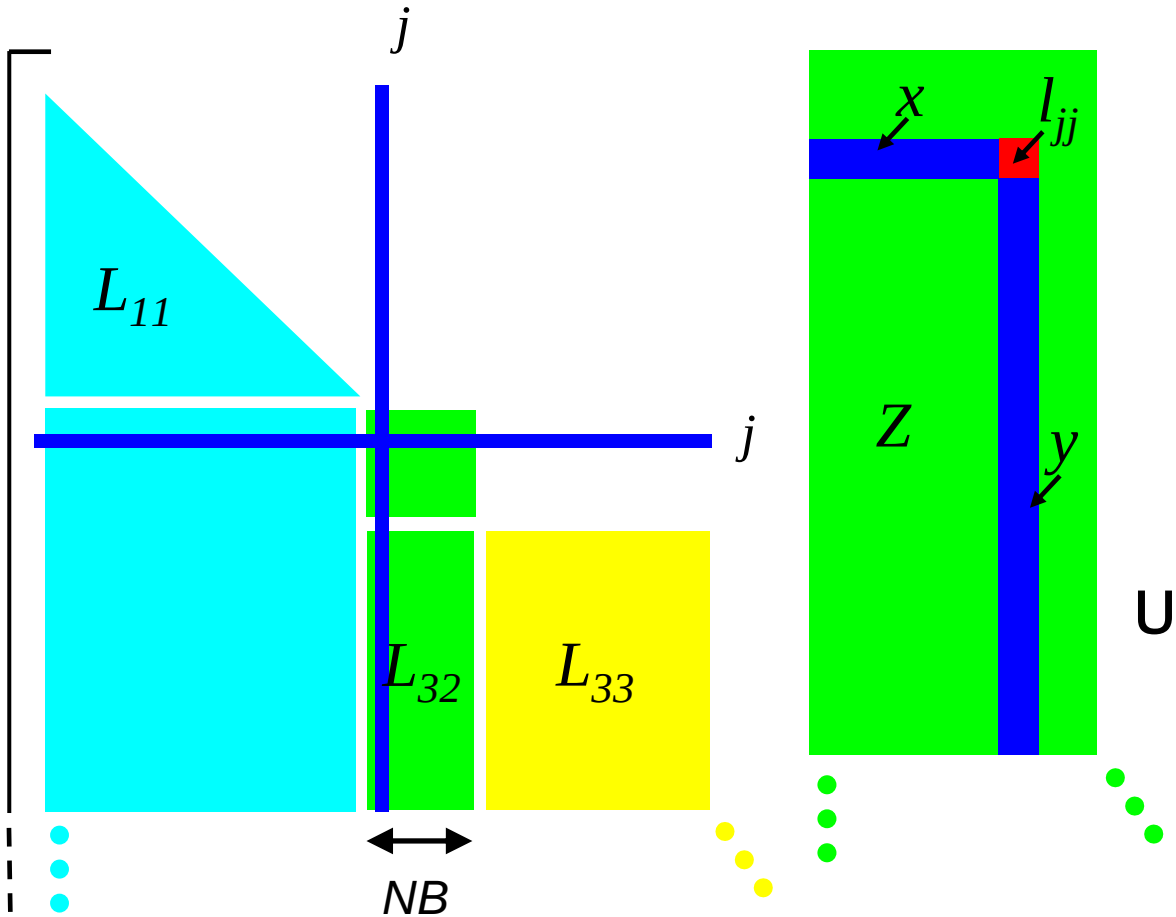
- Our algorithm is *blocked*, that is we perform operations on blocks instead of individual elements.



- This ensures effective use of memory hierarchy, via BLAS. Serial code up to 8 times faster [4].
- Blocked algorithms fit into the data distribution of ScaLAPACK.

Algorithm

- For *L=A*, current block column:



NB steps:

$$l_{jj} = l_{jj} - xx^T$$

$$l_{jj} = \sqrt{l_{jj}}$$

$$y = y - Zx^T$$

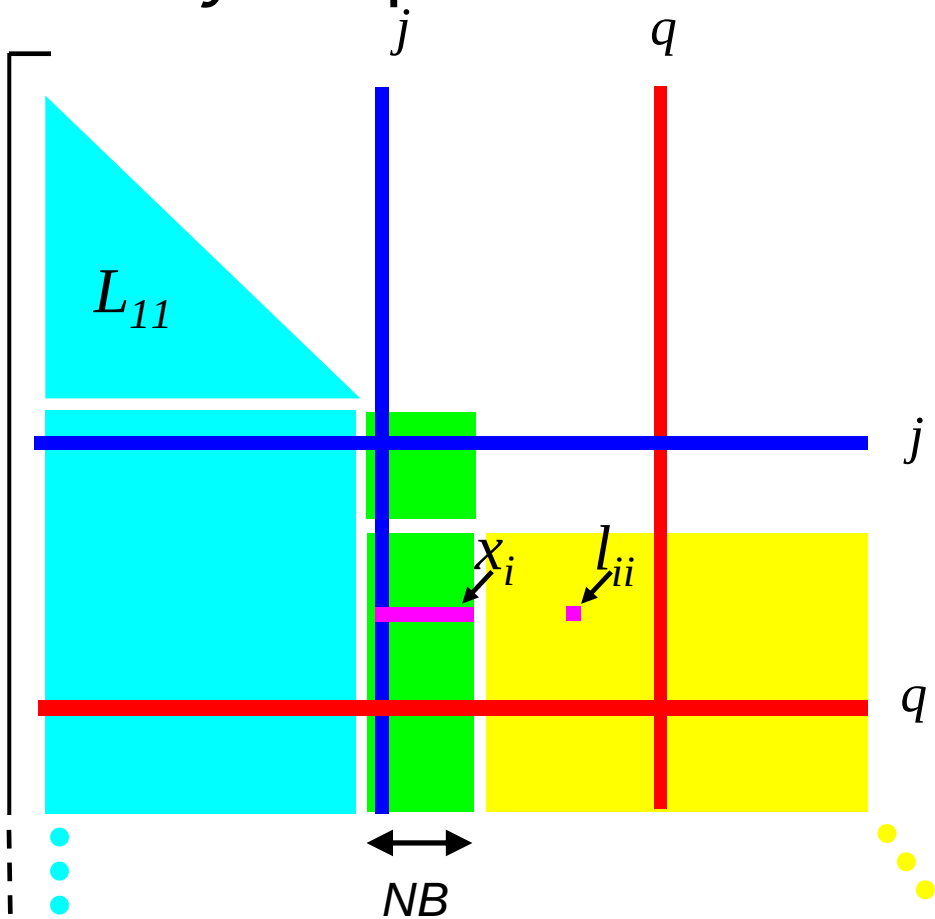
$$y = y / l_{jj}$$

Update

$$L_{33} = L_{33} - L_{32}L_{32}^T$$

Algorithm - Pivoting

- At the *j*th step:

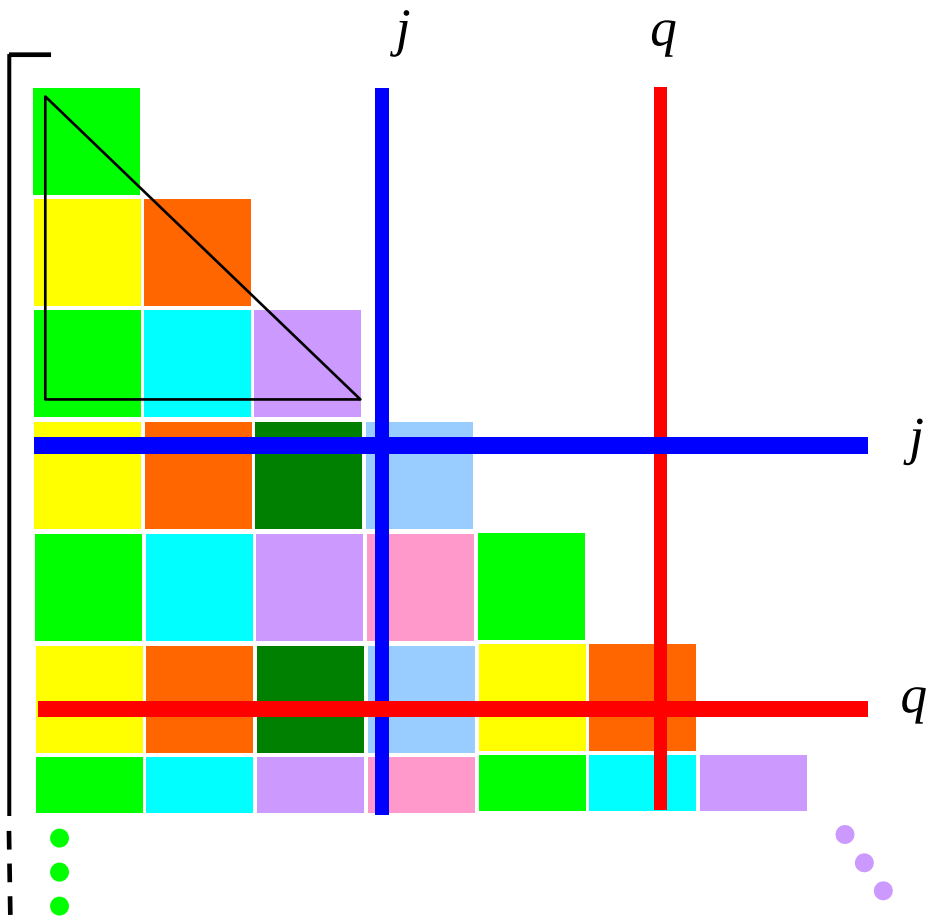


- L_{11} has been factored
- Current column:
  - Find the largest possible diagonal element, the *pivot*, in the *q*th position, say

$$\max_{i=j:n} l_{ii} - x_i x_i^T$$

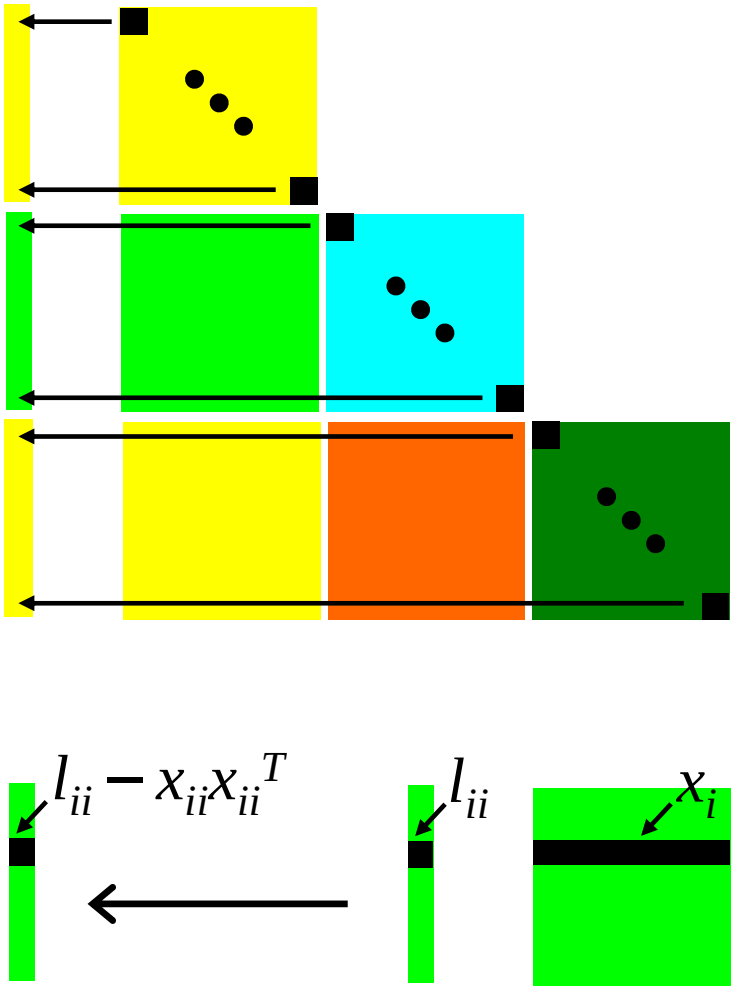
  - Swap *j*th row and column with *q*th row and column
  - Exit if pivot is "zero"
  - Compute *j*th column
- Update trailing matrix

Algorithm and Block Cyclic Data



- We need to compute the pivot, but the x_i are distributed down the process column and diagonal elements are distributed over all rows and columns.
- Swapping of rows and columns now must be done down each process row and along each process column.
- Communication can involve all processes.

Parallel Algorithm Details



- At the start of each block column we send the diagonal elements of trailing matrix(L_{33}) to a vector distributed over the current process column.
- The send is done block by block as there is no global routine for diagonal elements. We cheat by giving the leading diemnsion of local array +1 to pick out diagonal elements.
- Getting all the diagonal elements each time would be very costly.
- The processes in the current column can do its contribution of $l_{ii} - x_{ii}x_{ii}^T$ without any communication.

Parallel Algorithm Details

- The maximum value is computed with a combine operation down the process column, existing PDAMAX routine in PBLAS.
- We need to now broadcast the pivot position, q, along the process rows. We can now call global swap operations.
- We also need to swap our local copies of diagonal elements etc.
- We are now able to compute the current column, and continue until the end of the current block column.

MANCHESTER 1824

The University of Manchester

Experiments

Test Machines

- Cray XD1
  - Aston University
  - Cray XD1 2.4 GHz Opteron processors with 2GB memory per processor and two processors per node.

- Cray XT3
  - CSCS
  - Cray XT3 2.6 GHz Opteron processors with 2GB memory per processor.

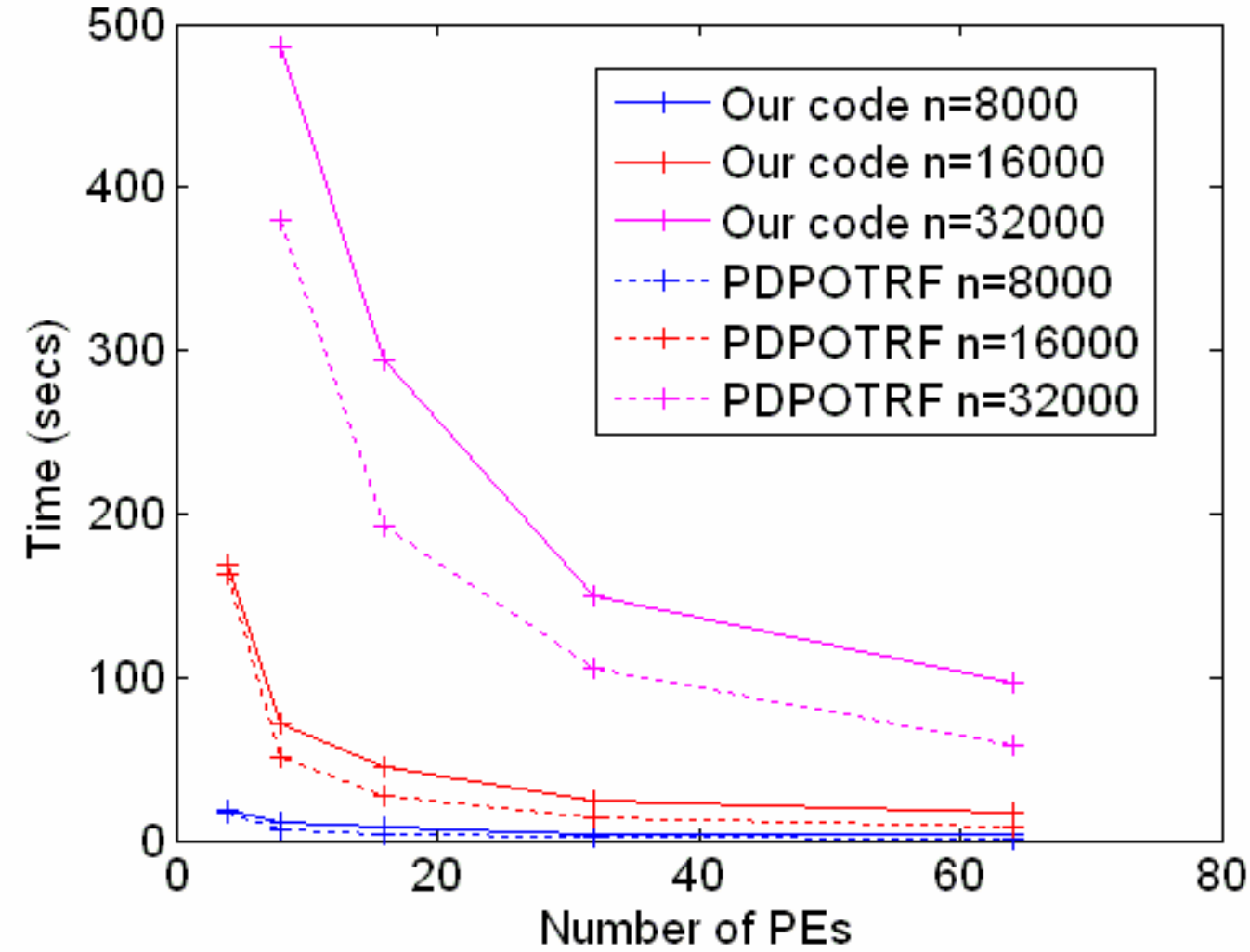
Timings

- We compare our code with the ScaLAPACK routine PDPOTRF, in ACML, which computes the factorization for positive definite matrix. This code is optimized.
- This is done so we can see the pivoting overhead in our algorithm. Which is the point of this talk!
- We do not look at the numerical behaviour and rank detection.
- Tuning parameters looked at empirically.

Timings

- Here we time the code for different problem sizes
  - n = 8000, 16000, 32000
- Different block sizes
  - NB = 16, 32, 64, 128, 256
- Different process grids
  - 2x2, 2x4, 4x2, 2x8, 4x4, 8x2, 4x8, 8x4, 4x16, 8x8, 16x4, 8x8, 16x4
  - 64 PEs XD1 only
- In each case we give the best time to compute the factorization.

Timings on the XD1

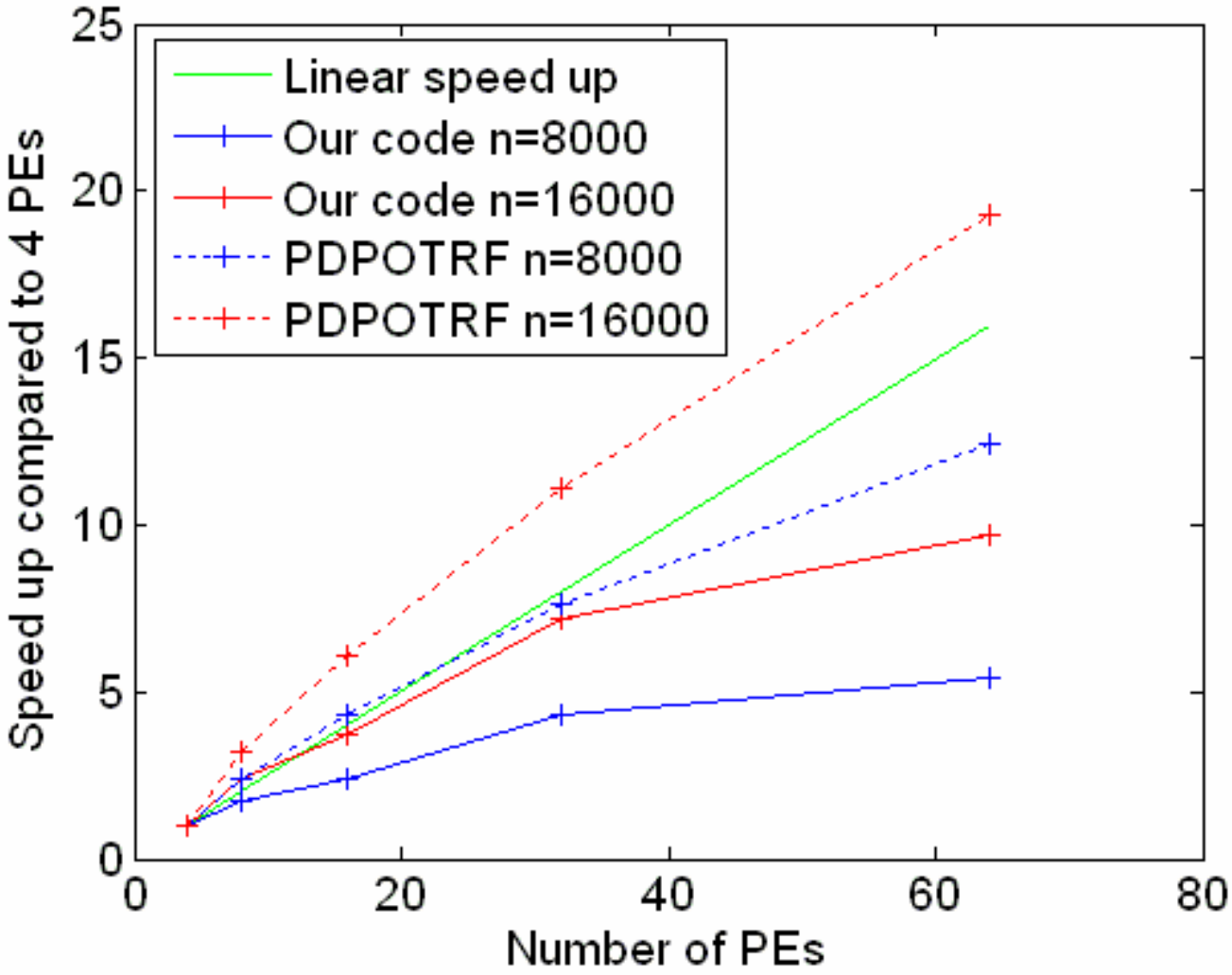


Pivoting Overhead

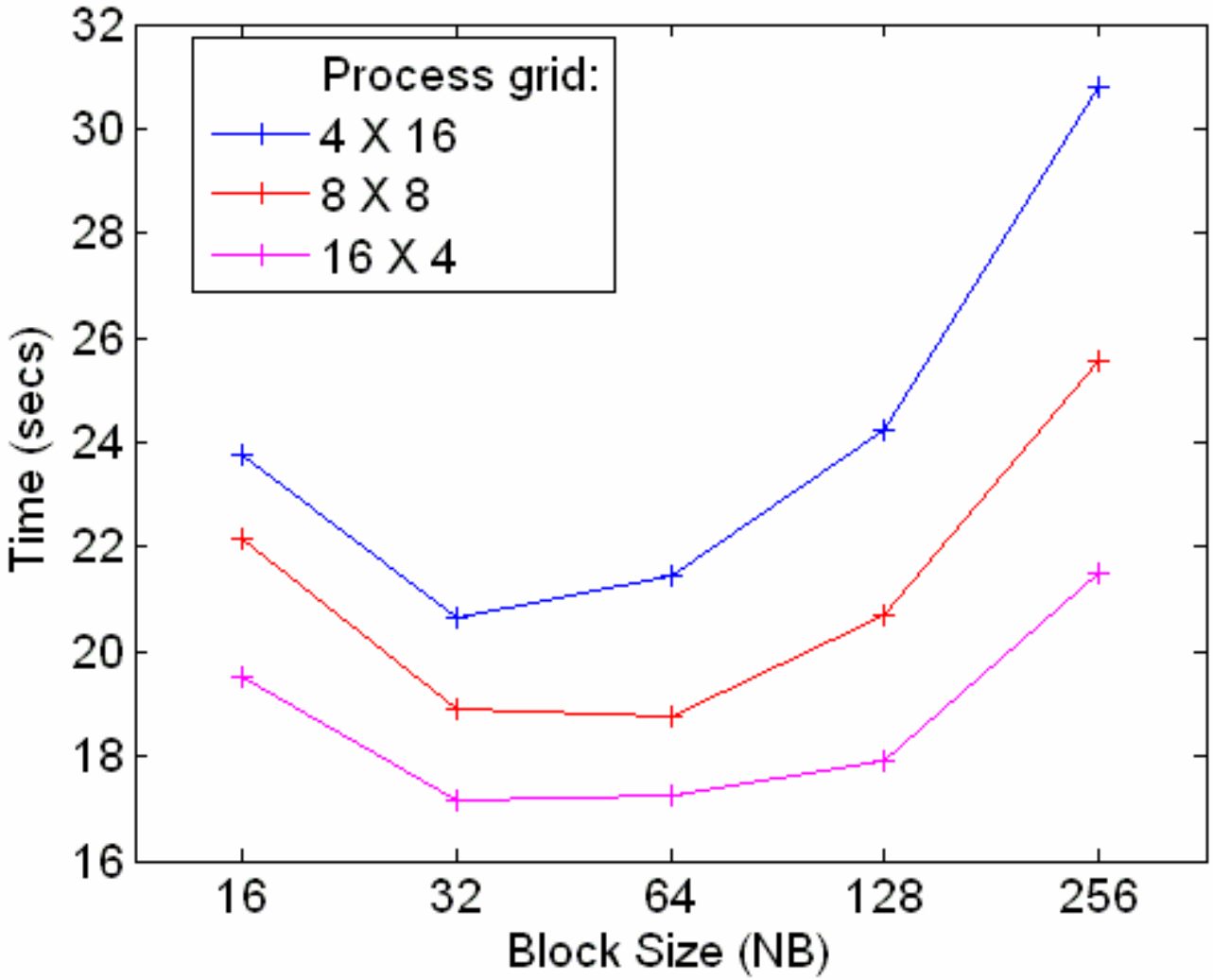
- Difference between pivoted and non-pivoted codes as a percentage non-pivoted compute time.

n \ PE	4	8	16	32	64
8000	11.2	54.8	96.8	95.6	155.1
16000	3.9	40.6	70.5	60.35	106.4
32000		27.8	52.8	42.1	63.8

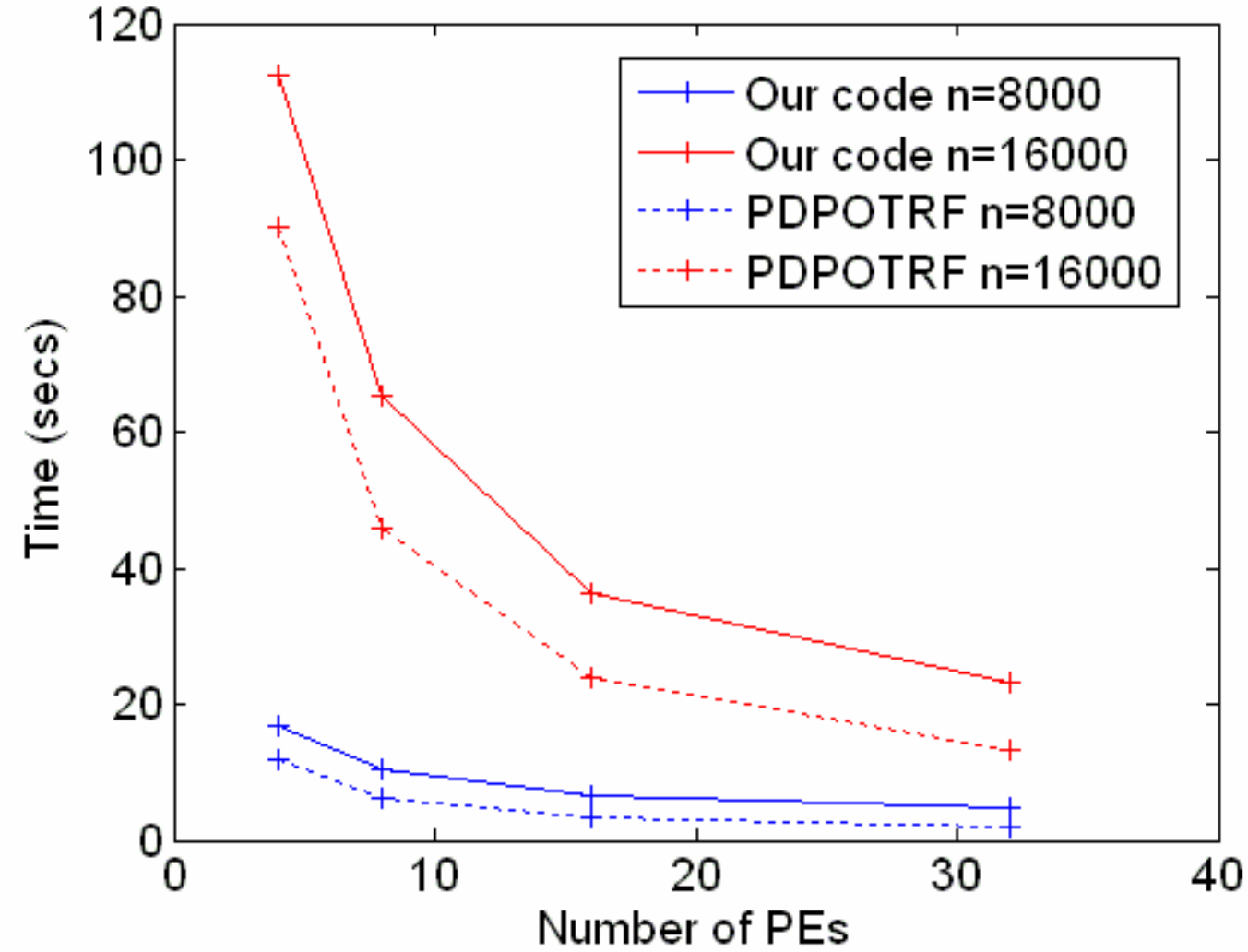
Scaling on the XD1



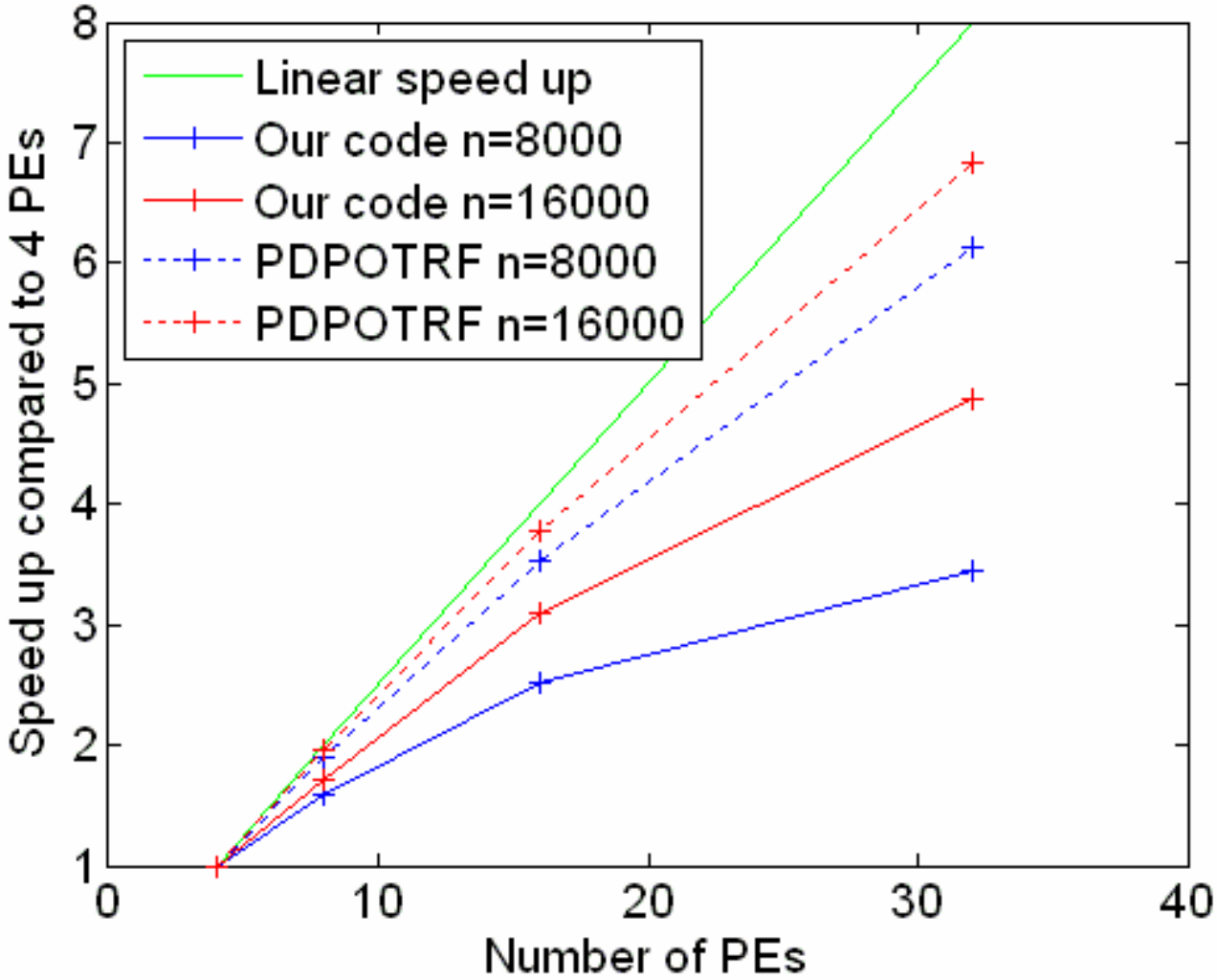
Block Sizes and Process Grids, n=16000



Timings on the XT3



Scaling on the XT3



Conclusions

- Wasn't sure we could get anything practical at all!
- Scaling is OK, for the problem sizes we ran, up to 32 processors.
- Pivoting overhead decreases with problem size...
- ... but increases with the number of processes. Pivoted code can take 2.5 times longer.

MANCHESTER 1824

The University of Manchester

And finally...

Further Work

- Work in process, code probably needs “optimization”

- Larger problems on more processors.
- Better communication pattern?
- One sided communication?
- Packed Storage?

And thanks to .....

- Aston for the XD1, particularly Andrey Kaliazin
- CSCS for the XT3, particularly Marie-Christine Sawley and Neil Stringfellow
- Kevin Roy at Manchester for running codes

References

- [1] L. S. Blackford et al. *ScaLAPACK user's guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
- [2] J. Choi et al. *A proposal for a set of parallel basic linear algebra subprograms.* LAPACK Working Note 100, May 1995.
- [3] J. Dongarra et al. *A user's guide to the BLACS v1.1*. LAPACK Working Note~94, May 1997.
- [4] Craig Lucas. *LAPack-style codes for level 2 and 3 pivoted Cholesky factorizations.* LAPACK Working Note 161, February 2004.
- LAPACK Working Notes: www.netlib.org/lapack/lawns/downloads

MANCHESTER 1824

The University of Manchester

Manchester Computing

Combining the strengths of UMIST and
The Victoria University of Manchester