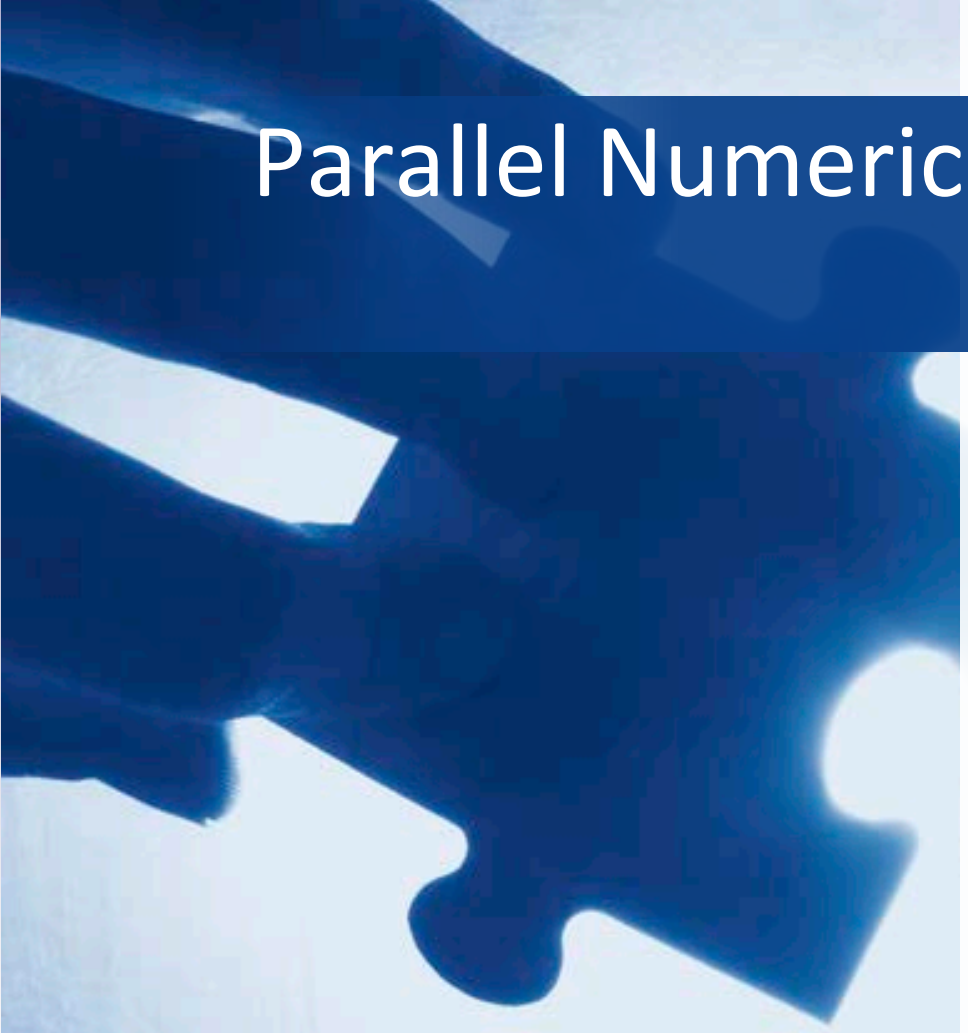
Parallel Numerical Linear Algebra in a Commercial Library

Craig Lucas

18th July 2012



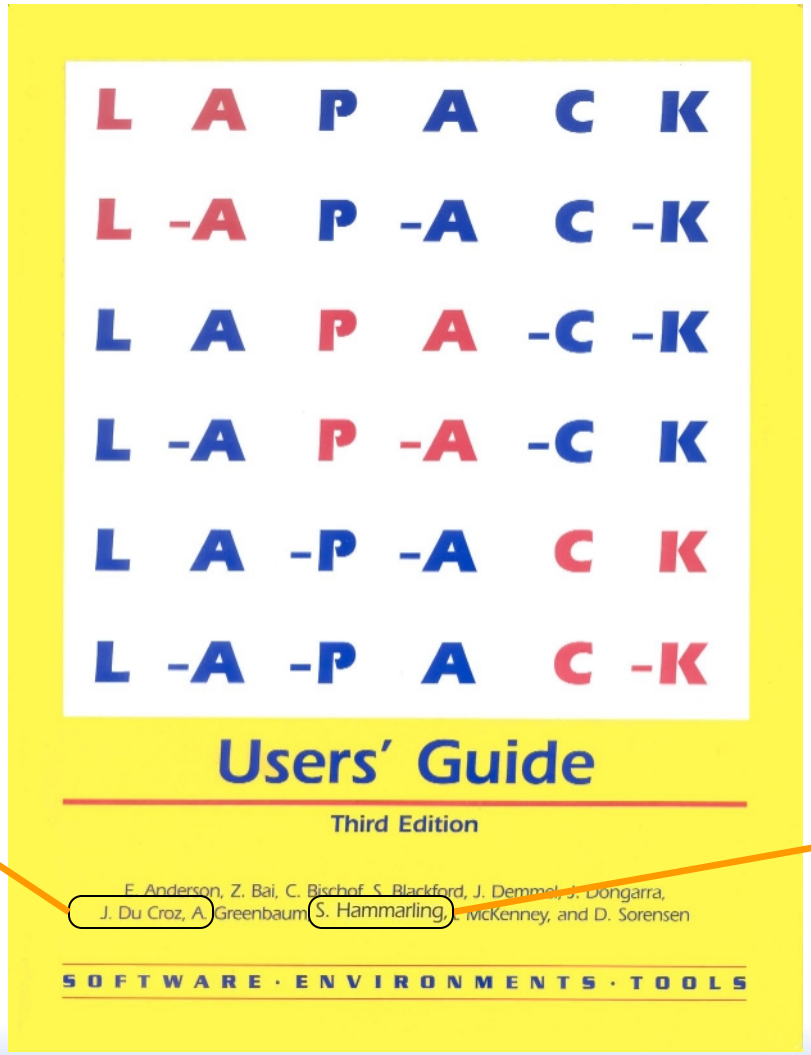
nag®

Experts in numerical algorithms and HPC services

Introduction

- For this talk we concentrate on dense parallel numerical linear algebra (PNLA) on “desktop” machines.
- That is those with shared memory and many cores, including accelerators.
- Why “commercial”? Success is measured in sales or retention, not discovering something interesting!
- In the most part our dense PNLA is “LAPACK”. And by that we mean the functionality of LAPACK.

Collaborations: NAG & LAPACK



J. Du Croz,

S. Hammarling,

Other Collaborations

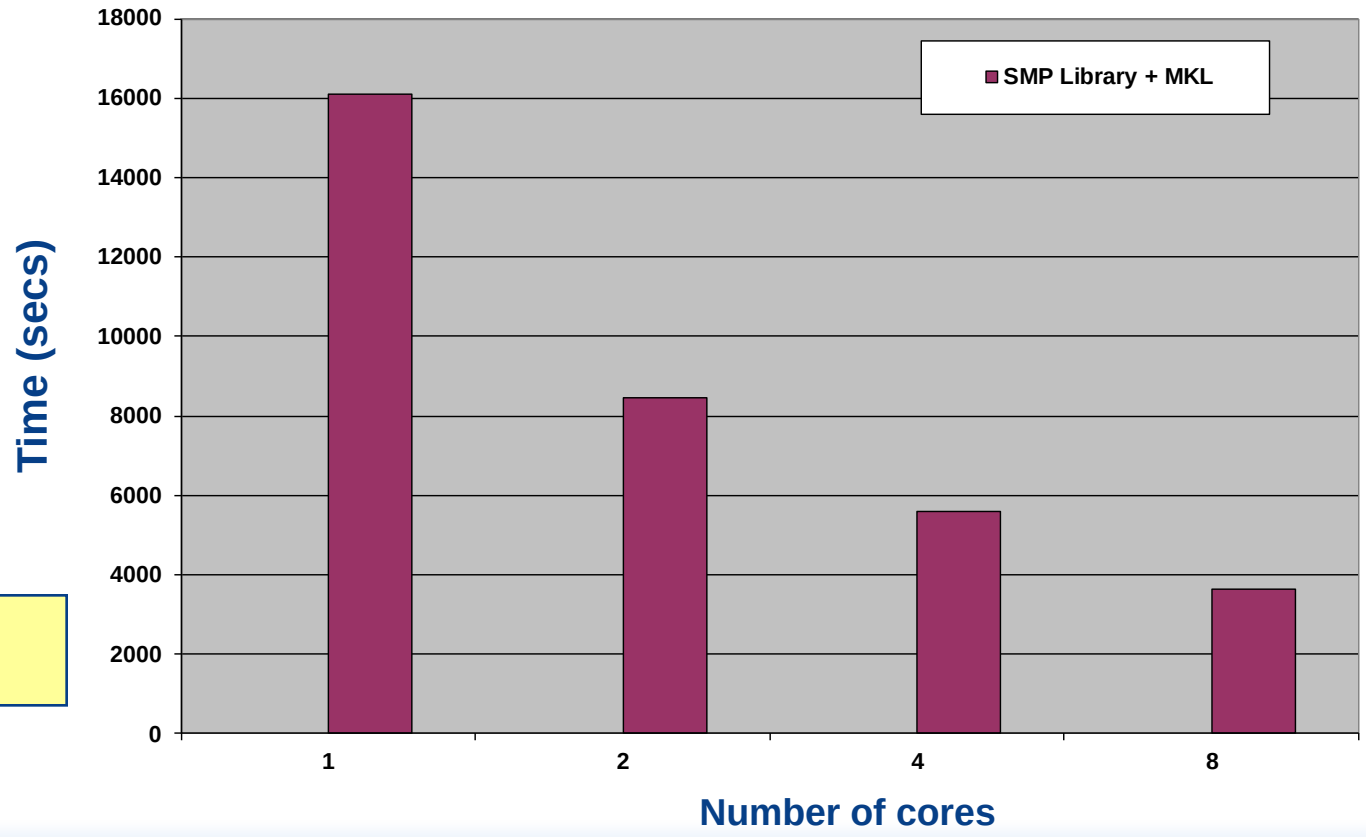
- Much work with UoManchester, more recently this has included:
  - PhD concerning Nearest Correlation Matrices
  - Two Knowledge Transfer Partnerships (KTPs)
- Latest one uses PLASMA. Some work through Open Petascale Library project with Fujitsu and Innovative Computing Lab (ICL, Jack, Tennessee)
  - Fortran Interfaces and test programs
- Other universities and many projects over the years.
- Vendors, including AMD and Intel, particularly math libraries.

NAG Parallel Libraries

- Traditionally NAG has a product for “SMP and Multicore”.
- Same interface to serial routines, out-of-the-box parallelism for “novice” users.
- Parallelism is achieved though use of threaded vendor libraries and OpenMP.
- The latter possibly being quite “easy” or possibly being a completely new algorithmic approach. (As with modern OpenMP, PLASMA et al, more later.)
- NLA used so much in the library it gives “indirect” parallelism to other routines. For example …

Nearest-correlation matrix

- Inherently serial algorithm, with BLAS, LAPACK and some OpenMP at each iteration.



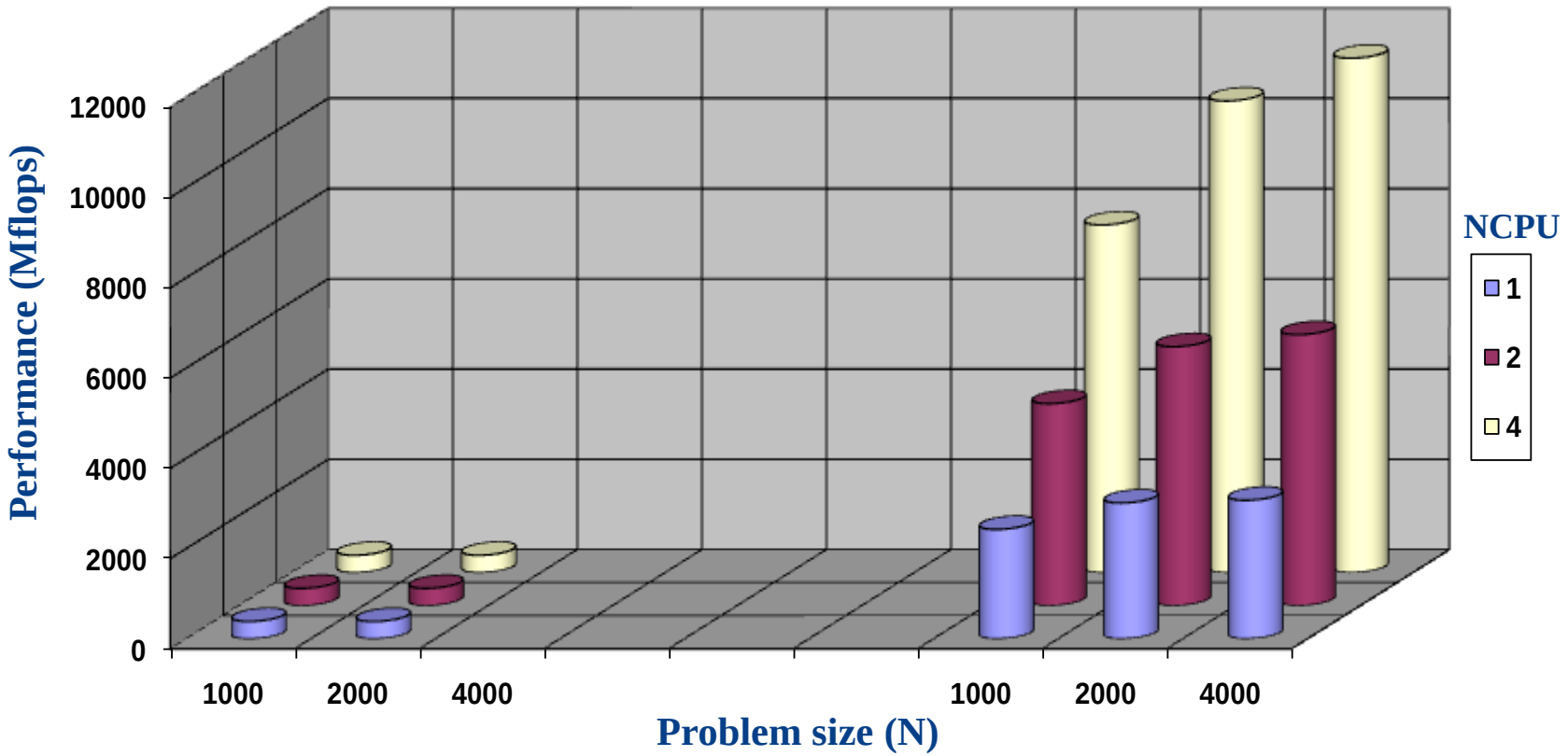
Intel Xeon E5405 2.0 GHz, N=10,000

Parallel NLA in NAG Library

- We have done work, over and above using threaded BLAS, to parallelise
  - LU factorisation
  - Cholesky factorisation
  - QR factorisation
  - Forming and applying Q
  - Reduction to tri-/bi-diagonal form
  - Computation of eigenvectors
  - Error and condition number estimation (if you have a large amount of RHS these actually dominate the runtime of the expert drivers, not the factorisation and solver.)

S.V.D. (DBDSQR)

Improved serial performance too



NAG Library Functionality

- Root Finding
- Summation of Series (e.g. FFT)
- Quadrature
- Ordinary Differential Equations
- Partial Differential Equations
- Numerical Differentiation
- Integral Equations
- Mesh Generation
- Interpolation
- Curve and Surface Fitting
- Optimisation
- Approximations of Special Functions
- Wavelet Transforms
- Dense Linear Algebra
- Sparse Linear Algebra
- Correlation and Regression Analysis
- Multivariate Analysis of Variance
- Random Number Generators
- Univariate Estimation
- Nonparametric Statistics
- Smoothing in Statistics
- Contingency Table Analysis
- Survival Analysis
- Time Series Analysis
- Operations Research

Matrix Functions

- NLA is not just LAPACK!
- First KTP with UoM is generating a suite of routines including
  - Matrix exp and log
  - General matrix function with derivatives supplied or not
  - Trigonometric functions sin, cos, sinh, cosh
  - Action of exp on vector
  - Matrix pth roots and powers
- Driven by commercial need from the finance sector.
- Edvin parallelising these now…

Parallel Matrix Functions

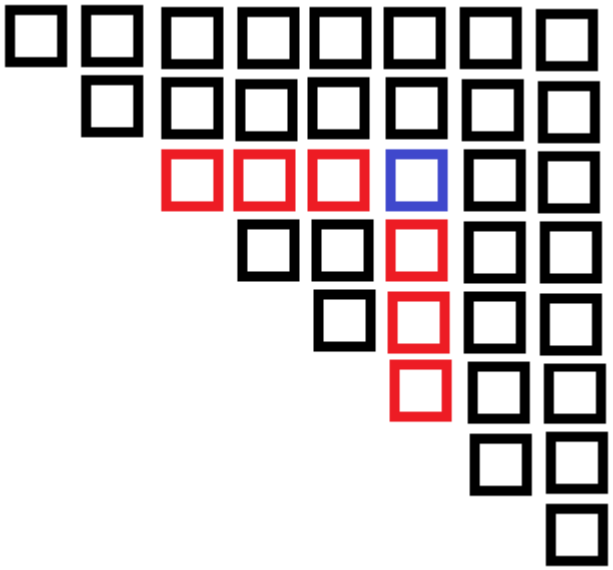
- Recall that NAG uses OpenMP for parallelism. Up until very recently this has been “old” style parallelism.
- Sharing out independent iterations in loops mainly.
- However, OpenMP 3.0 introduced TASKs. Independent units of work with some associated data.
- Allows much freedom in parallelising algorithms. Considered for matrix square root…

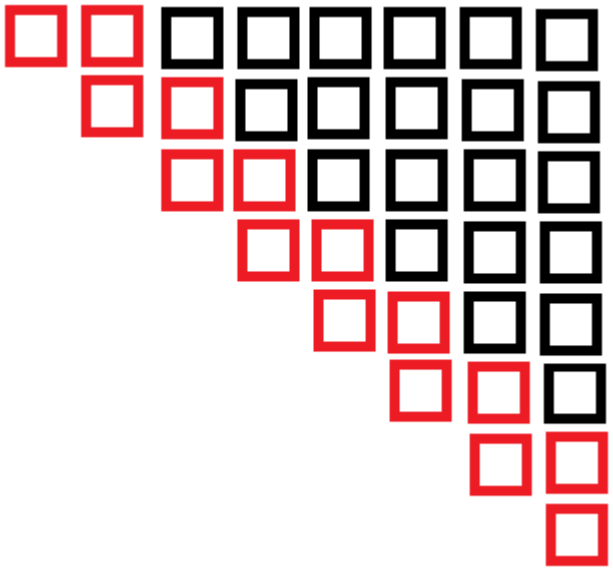
Matrix Functions

- Square root algorithm
  1. Compute a Schur decomposition: A = QTQ, with T upper triangular.
  2. Find U such that $U^2 = T$, U also upper triangular
  3. $\sqrt{A} = QUQ$
- Step 2 can be computed iteratively element wise (so called “point” algorithm), in a blocked fashion (exploiting cache memory) or recursively.
- Choice of algorithm always crucial.
- Analysis of dependencies …

Dependencies

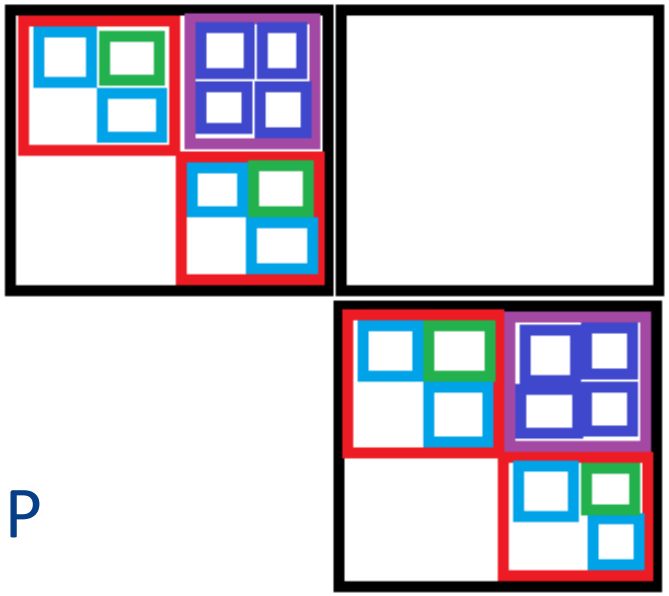
- To compute a block (or element) …





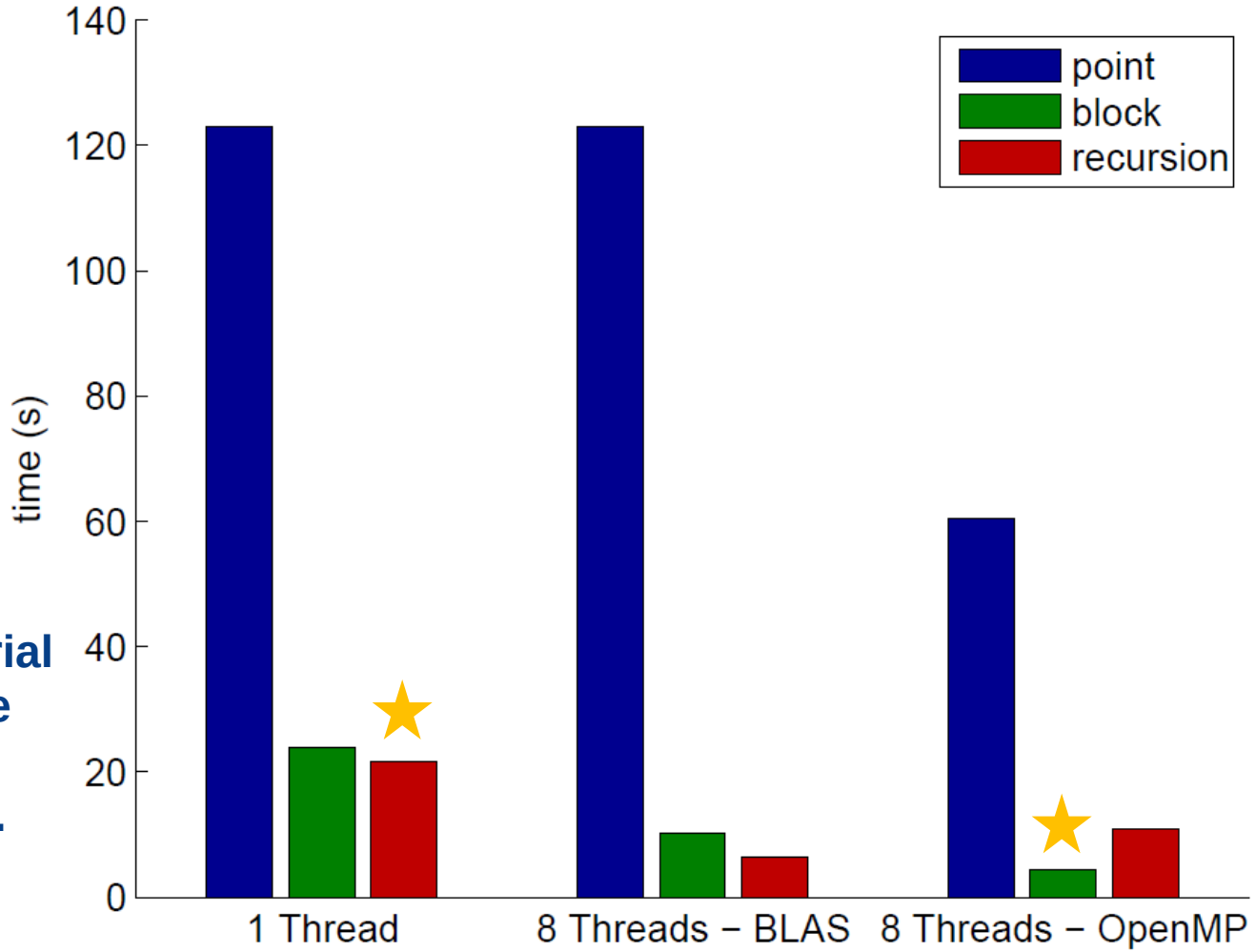
Dependencies

- Recursive algorithm
- Use OpenMP tasks
- Tasks allow recursion, not possible with older OpenMP



Performance

- n = 4000



Crucially, best serial algorithm NOT the same as best parallel algorithm.

Manycore Research Project

- We have internal manycore research project. Architectures include multicore, GPU and Intel Xeon Phi (MIC).
- As well as looking at novel algorithms and modern parallel programming also trying to leverage academic work (as always).
- This gives very rapid development of new functionality, although with some software engineering.

GPUs

- Needs no introduction, but
- Consist of 100's of "lightweight" cores, running 1000's of threads.
- The latter helping to hide the latency of data movement to the card.

- NAG has a GPU "Library" aimed at the finance sector. Some RNGs. PDEs requiring some work with NLA, coming soon. LU & Cholesky from MAGMA.

MAGMA

- Developed at ICL to target heterogeneous architectures.
- Beta version of GPU based NAG library built on MAGMA version 1.1.
- Targets MAGMA routines that have identical interfaces to LAPACK, including LU, Cholesky, QR, Eigenvalues and SVD.
- We also wrap level 3 CUBLAS routines.
- Easy (commercial) option for now.
- Each call to BLAS or MAGMA includes data transfer, we do it for the former.

MAGMA

- Need “big” (*n* in the 100’s to 1000’s) problem to make this worthwhile.
- We test for smaller problems and send those to MKL on the host instead. User tuneable via API.
- This is also done inside MAGMA but we opted for more control.
- We could be more sophisticated here and look at matrix shapes too, also
- Need to look at routines requiring data to be on GPU.
- Tested on Intel i7 860 with NVVIDIA C2050, speedup of between 2x and 7x over MKL LAPACK.

Intel Many Integrated Cores (MIC)

- 50+ cores with 4 hyper threads.
- Cores have wide vector units (512 bits) and with FMA can compute 16 operation per cycle.

- Clock speed is about 1GHz.
- Runs legacy code with offloads to MIC, targets current OpenMP parallelism.
- Or, runs entire (C/Fortran) code on the MIC.

- Product next year, **and NAG library that supports it!**

Intel MIC

- So programming style similar to OpenMP with data transfer overhead of GPU.
- So how to use NLA? MKL is provided.
- MKL can be called from the host (off loading computation) or form the MIC (using data already there.) But, crucially, can't keep it there between calls.
- MKL makes runtime decision based on problem size as to whether to offload or not in the first case.

Intel MIC

- Three choices for NAG in providing a library for MIC
  - Use MKL, with off loads, and our existing OpenMP parallel regions with the addition of offloads. Our traditional approach for a multicore library. But perhaps grouping calls into one offload.
  - Much more interaction with user. Provide for the user to have data already offloaded. NAG routines will pick this data up and leave it there when the routines exits. **A big change for users**.
  - Assume the user's application is running on the MIC so the whole library could be built for the MIC (no offload, no computation on host.)

PLASMA KTP

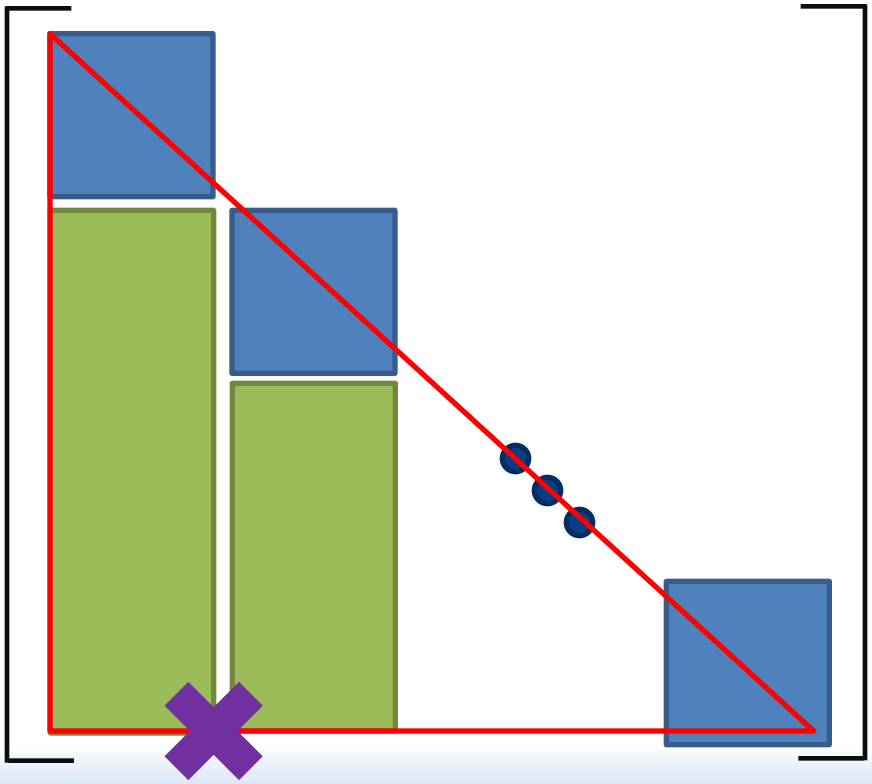
- Knowledge Transfer Partnership does what is say on the tin.
- The Numerical Algorithms Group
  - Me and our manycore research team
  - Joseph Dobson “associate”
- The University of Manchester
  - Jack Dongarra, David Silvester, Nick Higham

Recall LAPACK DPOTRF

```
            DO J = 1, N, JB

*              Update and factorize the current diagonal BLOCK

               CALL DSYRK( 'Lower', 'No transpose', JB, J-1, -ONE,
     $                     A( J, 1 ), LDA, ONE, A( J, J ), LDA )
               CALL DPOTF2( 'Lower', JB, A( J, J ), LDA, INFO )
               IF( J+JB.LE.N ) THEN

*                 Compute the current PANEL.

                  CALL DGEMM( 'No transpose', 'Transpose', N-J-JB+1, JB,
     $                        J-1, -ONE, A( J+JB, 1 ), LDA, A( J, 1 ),
     $                        LDA, ONE, A( J+JB, J ), LDA )
                  CALL DTRSM( 'Right', 'Lower', 'Transpose', 'Non-unit',
     $                        N-J-JB+1, JB, ONE, A( J, J ), LDA,
     $                        A( J+JB, J ), LDA )
               END IF
            END DO
```

Recall Parallel DPOTRF

- Excessive synchronisation
- Parallelism shoe horned in



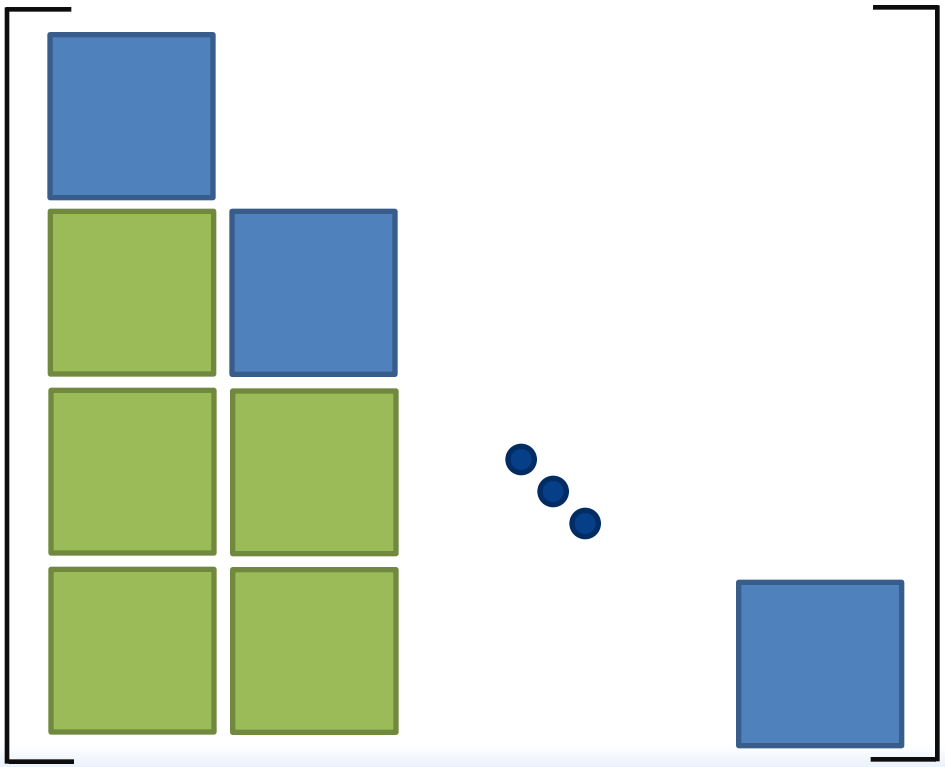
Pre PLASMA algorithm:

1. Compute diagonal blue blocks (DSYRK and DPOTF2) and on one core
2. Compute the green panel (DGEMM and DTRSM) on multi-cores with threaded BLAS
3. Repeat on next column

✖ **We can't start the next blue block until the previous panel is done....**

Need new algorithmic approach, PLASMA

- Consider the matrix now broken down into *tiles*:



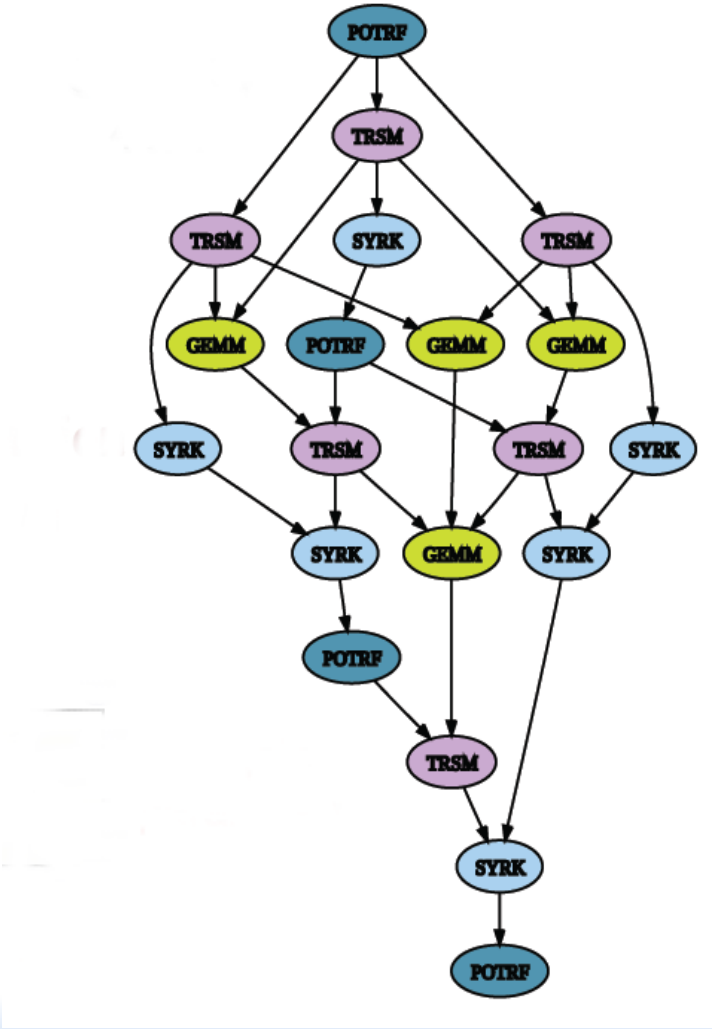
PLASMA algorithm

1. Analyse the dependencies between tiles, gives:
2. Compute first diagonal blue block on one core
3. Compute the first green tile on a single core
4. Now free to continue down panel, but also do next blue block.
5. etc

Note a tile is still blocked

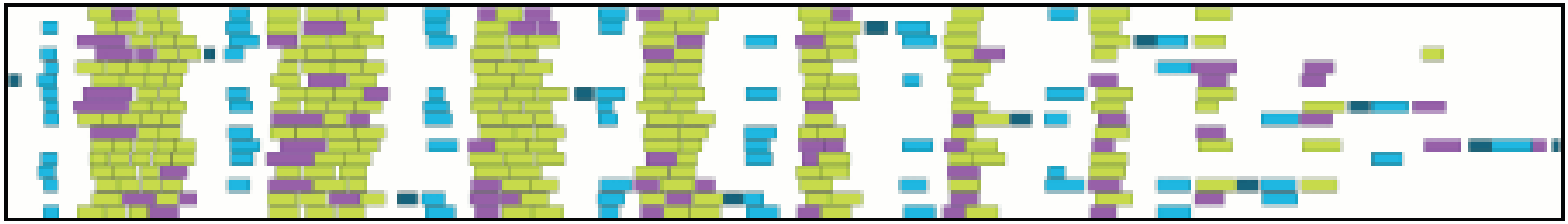
Directed Acyclic Graph (4X4 Cholesky)

- Each node is a *task*, not the computation of a complete tile.
- The largest piece of data is a tile. So several tasks can make up the computation of a tile.
- Directed edges show dependencies.

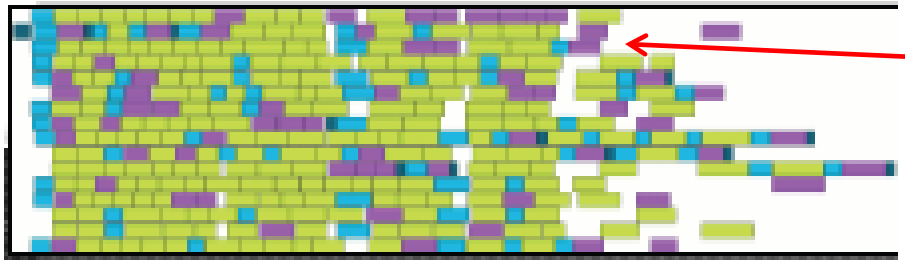


Computation Time

- LAPACK with threaded BLAS



- PLASMA



PLASMA

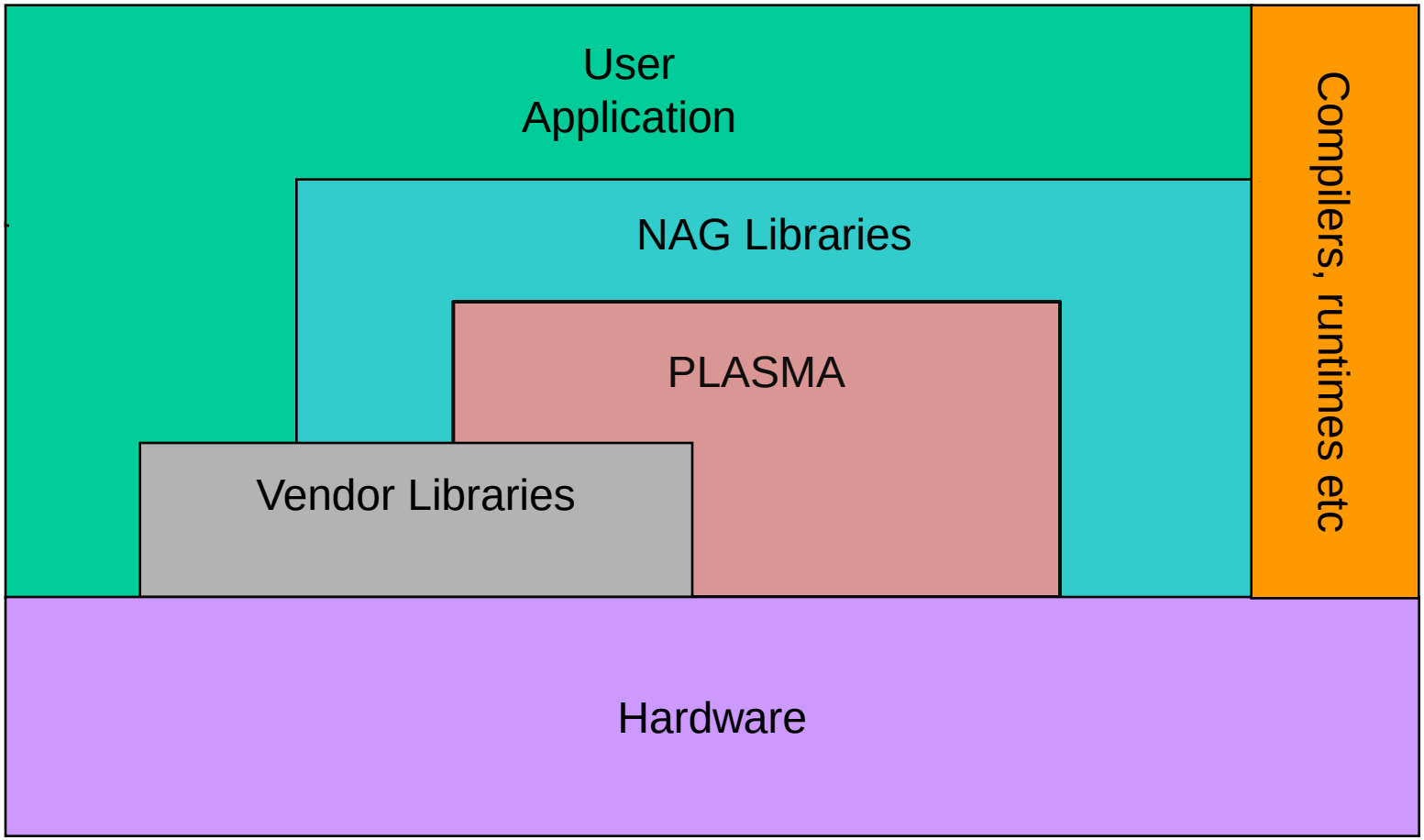
- Much bigger investment by NAG compared to MAGMA.
- We will work on integrating the different interfaces, that we avoided with MAGMA so far.
- Other software engineering challenges
  - Interoperability issues of Pthreads and OpenMP
  - Using the PLASMA scheduler, Quark
  - Mixing of serial and threaded BLAS
  - Integrating the asynchronous routines, that allow DAGs to span routines boundaries.
- We will need consider tuning of tile and block sizes.

Data storage

- Rather than storing matrices as column major, PLASMA stores tiles contiguously, and provides routines for conversion between the two formats.
- We could completely hide PLASMA or do we allow user to supply data in PLASMA tiled storage?
  Another big change for users.
- Should we supply other routines that act upon tiled storage?

- How do all these libraries fit together…

PLASMA


User
Application
NAG Libraries
PLASMA
Vendor Libraries
Compilers, runtimes etc
Hardware

More OpenMP

- OpenMP is developing very quickly.
  - More support for nested parallelism.
  - Affinity - mappings threads to hardware
  - Accelerators - OpenACC
  - Error model - exiting OpenMP loops and regions
  - TASKs - dependencies defined between them

- In particular, for PNLA this last addition will give us implicit DAGs, and PLASAM-esque routines.

OpenMP TASK Dependencies

- In our Cholesky example each call to a BLAS routine could be an OpenMP task, perhaps.
- OpenMP 4.0 will allow us to define data to be *in*, *out*, *inout*.
- (Plus possibly another way to define graph edges.)
- So we could modify DPOTRF, thus …

- (Note I haven't reorganised data into tiles.)

```
*           Loop over tiles
*
            DO 20 J = 1, N, NB
*
*              Update and factorize the current diagonal block and test
*              for non-positive-definiteness.
*
               JB = MIN( NB, N-J+1 )
*
*              Loop across tiles contributing to current tile
*              Note we use NB as previous columns will be NB wide
*              and only the last column is JB
*
               DO L = 1, J-1, NB
                  !$OMP TASK in(A(J:J+NB-1,J-NB:J-1) &
                  !$OMP inout(A(J:J+NB-1,J:J+NB-1)
                  CALL DSYRK( 'Lower', 'No transpose', JB, NB, -ONE,
     $                        A( J, L ), LDA, ONE, A( J, J ), LDA )
               END DO
*
               !$OMP TASK inout(A(J:J+NB-1,J:J+NB-1)
               CALL DPOTRF( 'Lower', JB, A( J, J ), LDA, INFO )
```

Implies data is read only and defines a dependency (graph edge)

Defines a dependency for the next task accessing the array section

```
*                 Compute the current block column.
*                 Loop down tiles in panel
*
                  DO I = J + NB, N, NB
                     IB = MIN( NB, N-I+1 )
*
*                    Loop across tiles contributing to current tile
*
                     DO K = 1, J-1, NB
                        !$OMP TASK in(A(J:J+NB-1,J-NB:J-1) &
                        !$OMP in(A(I:I+NB-1,J-NB:J-1) &
                        !$OMP inout(A(I:I+NB-1,J:J+NB-1)
                        CALL DGEMM( 'No transpose', 'Transpose', IB, JB,
     $                             NB, -ONE, A( I, K ), LDA, A( J, K ),
     $                             LDA, ONE, A( I, J ), LDA )
                     END DO
                     !$OMP TASK in(A(J:J+NB-1,J:J+NB-1) &
                     !$OMP inout(A(I:I+NB-1,J:J+NB-1) &
                     CALL DTRSM( 'Right', 'Lower', 'Trans', 'Non-unit',
     $                          IB, JB, ONE, A( J, J ), LDA,
     $                          A( I, J ), LDA )
                  END DO
               END IF
   20       CONTINUE
```

User Experience

- NAG will change the way we present parallelism to the user.
- We have already seen this with the likely ways we will implement MIC and PLASMA based libraries.
- The complexity of modern architecture demands this.
- Users also want to use their own OpenMP with NAG.
- OpenMP support for nested parallelism and affinity will help with NUMA architecture, but again more complexity for and interaction from the user.

Summary

- So is it the end of our “out-of-the-box” parallelism?
- Probably not. With “expert” and “novice” interfaces.
- But the latter may ultimately be at the cost of performance.
  - Extra data movement and/or redistribution
  - Blind to NUMA effects etc.

- New architecture are exciting and challenging.
- New software and programming is embracing it.
- New challengers for the user.