An Introduction to the NAG SMP Library

Edward Smyth

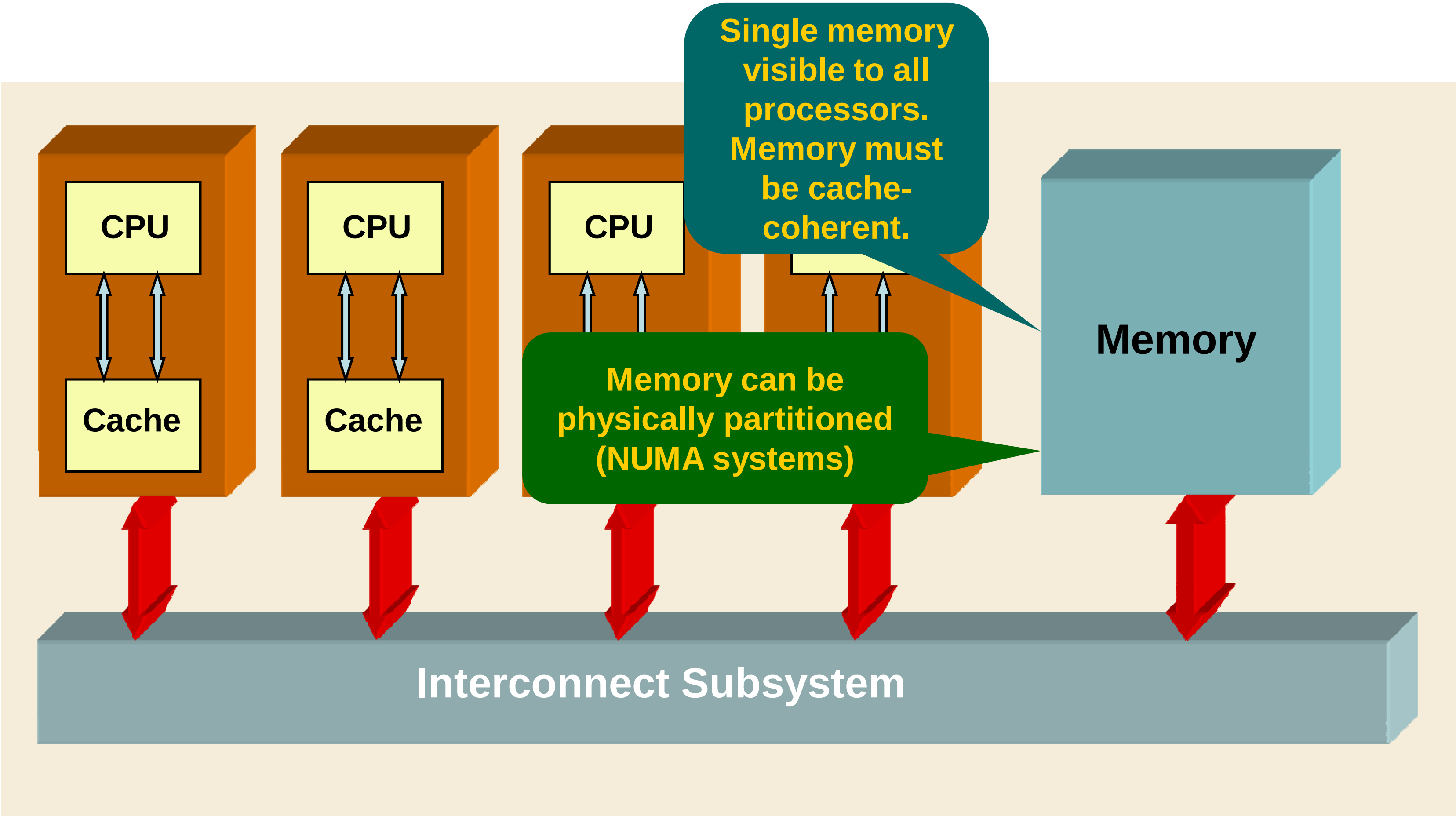
1/10/2009

Experts in numerical algorithms and HPC services

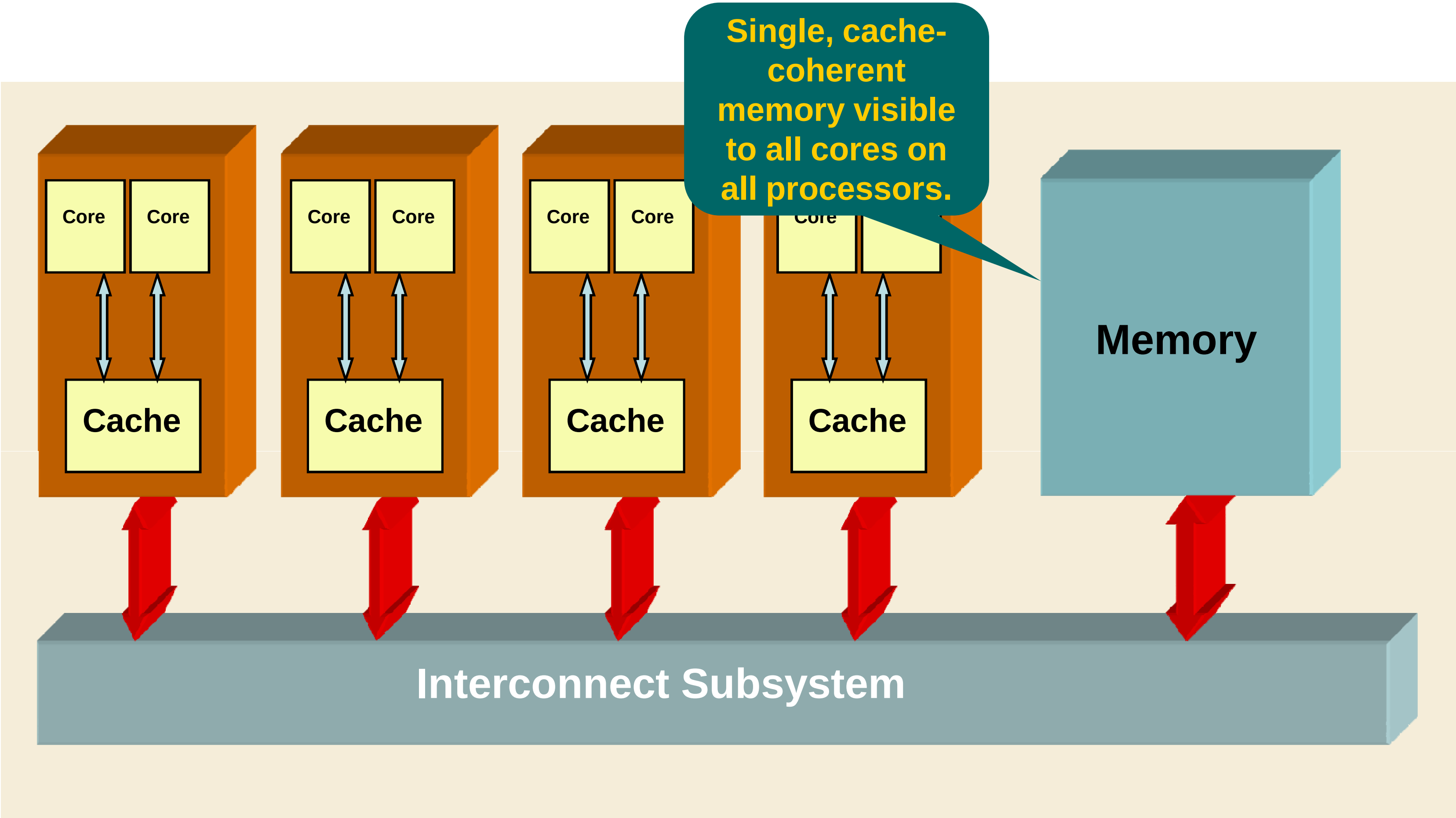
Overview

- Overview of SMP parallelism
- Description of SMP Library
- What algorithms have been parallelised?
- Future directions
- Performance issues

SMP = Symmetric MultiProcessing

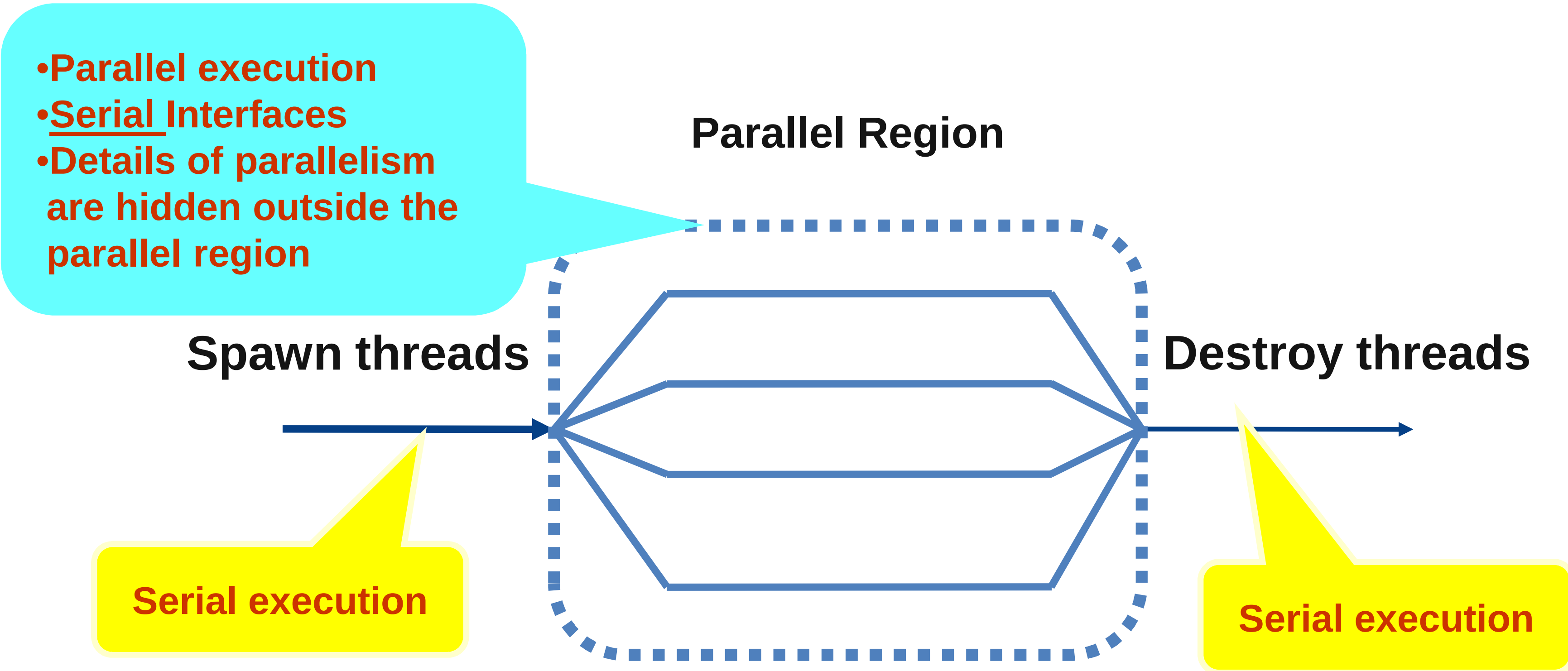


Often with multi-core processors…



SMP Parallelism

Multi-threaded Parallelism (Parallelism-on-demand)



Parallelism carried out in distinct parallel regions

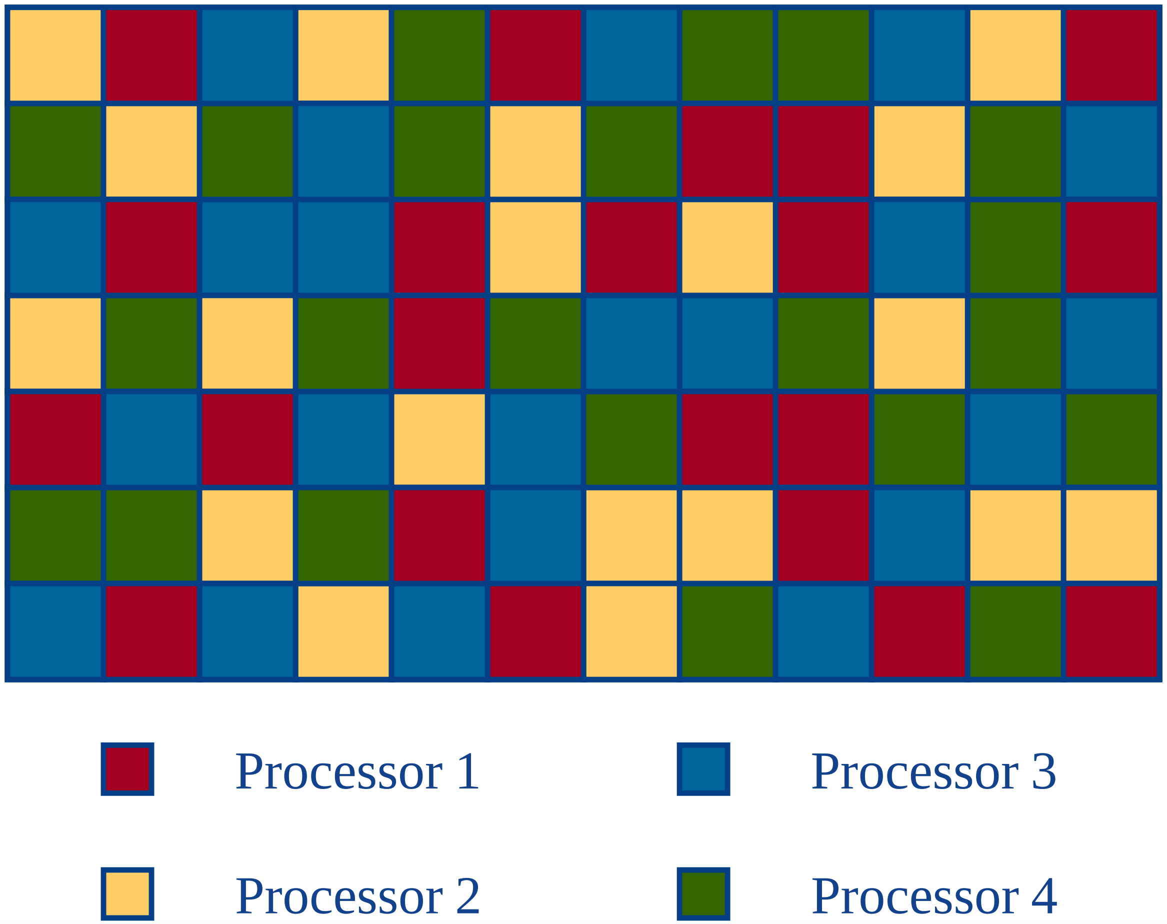
SMP Parallelism: Strong Points

- Dynamic Load balancing
  - Amounts of work to be done on each processor can be adjusted during execution
  - Closely linked to a dynamic view of data
- Dynamic Data view
  - Data can be ‘redistributed’ on-the-fly
  - Redistribution through different patterns of data access
- Portability
- Modularity
- Good programming model

SMP Parallelism: Dynamic View of Data

Computation Stage 3

Data



SMP Parallelism: Some Weaker Points

- Not very suitable for heterogeneous parallelism
- Not very suitable for complex applications
- Not easy to generate efficient code
- Non-deterministic results
  - Applies to least significant parts of solution
  - Parallel random number generators
  - May be disconcerting to some users
  - Effects on ill-conditioned problems may be dramatic

SMP Parallelism: Explicit Multi-Threading

- Call to system routines
  - Code differs significantly from original serial version
- No universal standard – different vendors may use different mechanisms
  - POSIX threads
  - Windows threads
- Difficult to write
- Difficult to maintain

SMP Parallelism: Compiler Directives

- Instruction to compiler to instrument the code with appropriate threading
- Portable to variety of SMP systems
- Ignored by compilers on serial systems
- Code executed not code written (one layer of software in between)
- Easier to write and maintain
- Identifiable by a sentinel, a special sequence of characters in a comment statement

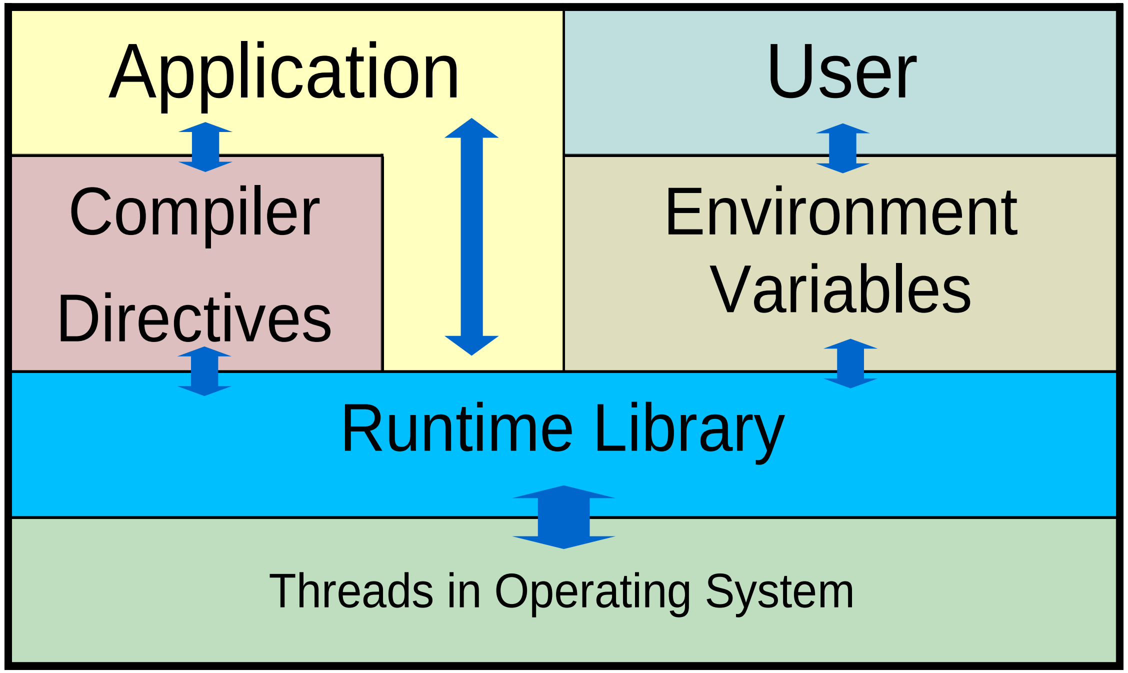
SMP Parallelism: Directives v Multi-Threading

	Compiler Directives	Explicit Threads
Portable to/from serial	**Yes**	**No**
Portable to other SMPs	**Yes**	**No**
Easy to code	**Yes**	**No**
Easy to maintain	**Possibly**	**No**

OpenMP: Introduction

- Portable, Shared Memory MultiProcessing API
  - Fortran 77 & Fortran 90
  - C & C++
  - Multi-vendor Support, for Both UNIX/Linux and Windows
- Standardizes Fine Grained (Loop) Parallelism
- Also Supports Coarse Grained Algorithms

OpenMP: Architecture



SMP Mechanisms in OpenMP

- Fork-join construct: PARALLEL
- Data attributes definition: SHARED, PRIVATE
- Global Operations: REDUCTION
- Work Sharing Constructs
  - Distribution of Loops: DO
  - Distribution of blocks of code: SECTION, TASKS

SMP Mechanisms in OpenMP

- Synchronisation
  - Processors wait until everyone calls: BARRIER
  - Sub-groups of processors synch: via locks
  - One processor at a time: CRITICAL
  - Only one processor executes: SINGLE, MASTER
- Runtime library calls to interrogate system
  - How many threads are there?
  - Which one am I?
- User control at runtime via environment variables
  - Number of threads: OMP_NUM_THREADS

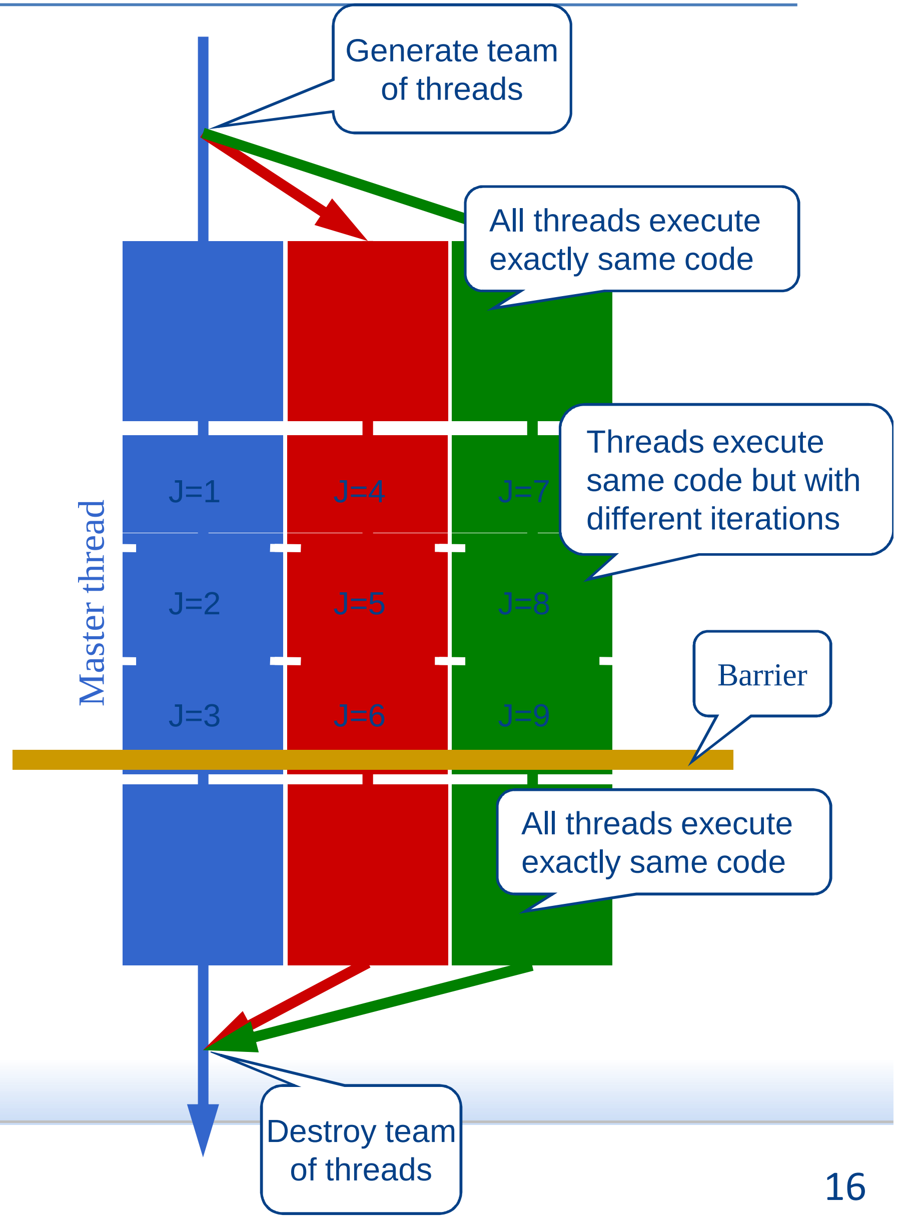
OpenMP: Parallel DO loop

All threads execute same code

```
C$OMP PARALLEL
C$OMP&   SHARED (M,N)
C$OMP&   PRIVATE (I,J)
      . . . . .
      DO 10 I = 1, M
        . . . . .
C$OMP DO
        DO 11 J = 1, N
          . . . . .
          . . . . .
          . . . . .
   11   CONTINUE
        . . . . .
   10 CONTINUE
      . . . . .
C$OMP END PARALLEL
```

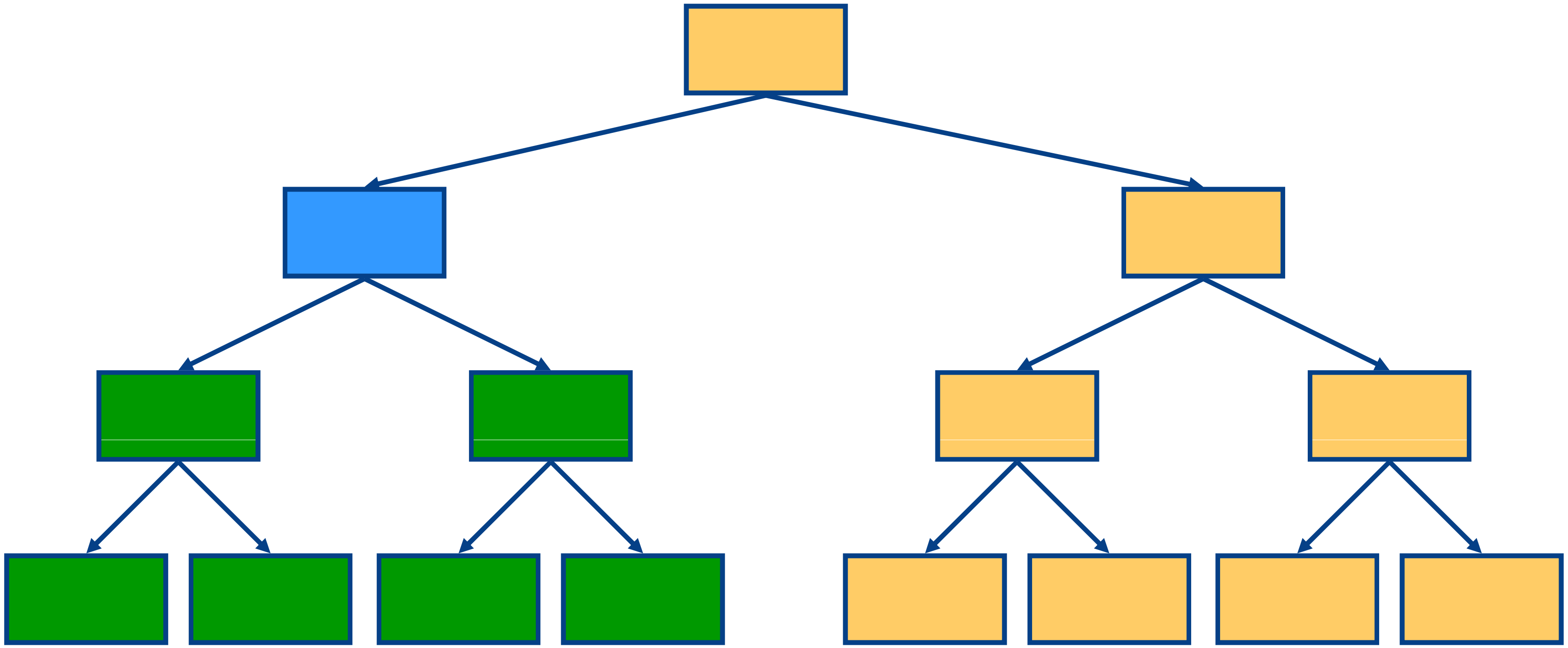
Threads execute different iterations

All threads execute same code



nag®

OpenMP: Directives Binding



Module containing the parallel construct

Modules within the dynamic extent of the parallel construct

Example: Dot Product

```
C$OMP PARALLEL
C$OMP&   SHARED (N,DOT,X,Y)
C$OMP&   PRIVATE (I,DOTL)
      DOTL = 0.0D0
C$OMP DO
      DO 1 I=1,N
         DOTL = DOTL + X(I)*Y(I)
    1 CONTINUE
C$OMP END DO NOWAIT
C$OMP CRITICAL
      DOT = DOT + DOTL
C$OMP END CRITICAL
C$OMP END PARALLEL
```

```
C$OMP PARALLEL DO
C$OMP&   SHARED (N,X,Y)
C$OMP&   PRIVATE (I)
C$OMP&   REDUCTION (+:DOT)
      DO 1 I=1,N
         DOT = DOT + X(I)*Y(I)
    1 CONTINUE
C$OMP END PARALLEL DO
```

```
C$OMP PARALLEL DO
C$OMP&   SHARED (N,DOT,X,Y)
C$OMP&   PRIVATE (I)
    DO 1 I=1,N
C$OMP ATOMIC
         DOT = DOT + X(I)*Y(I)
  1 CONTINUE
C$OMP END PARALLEL DO
```

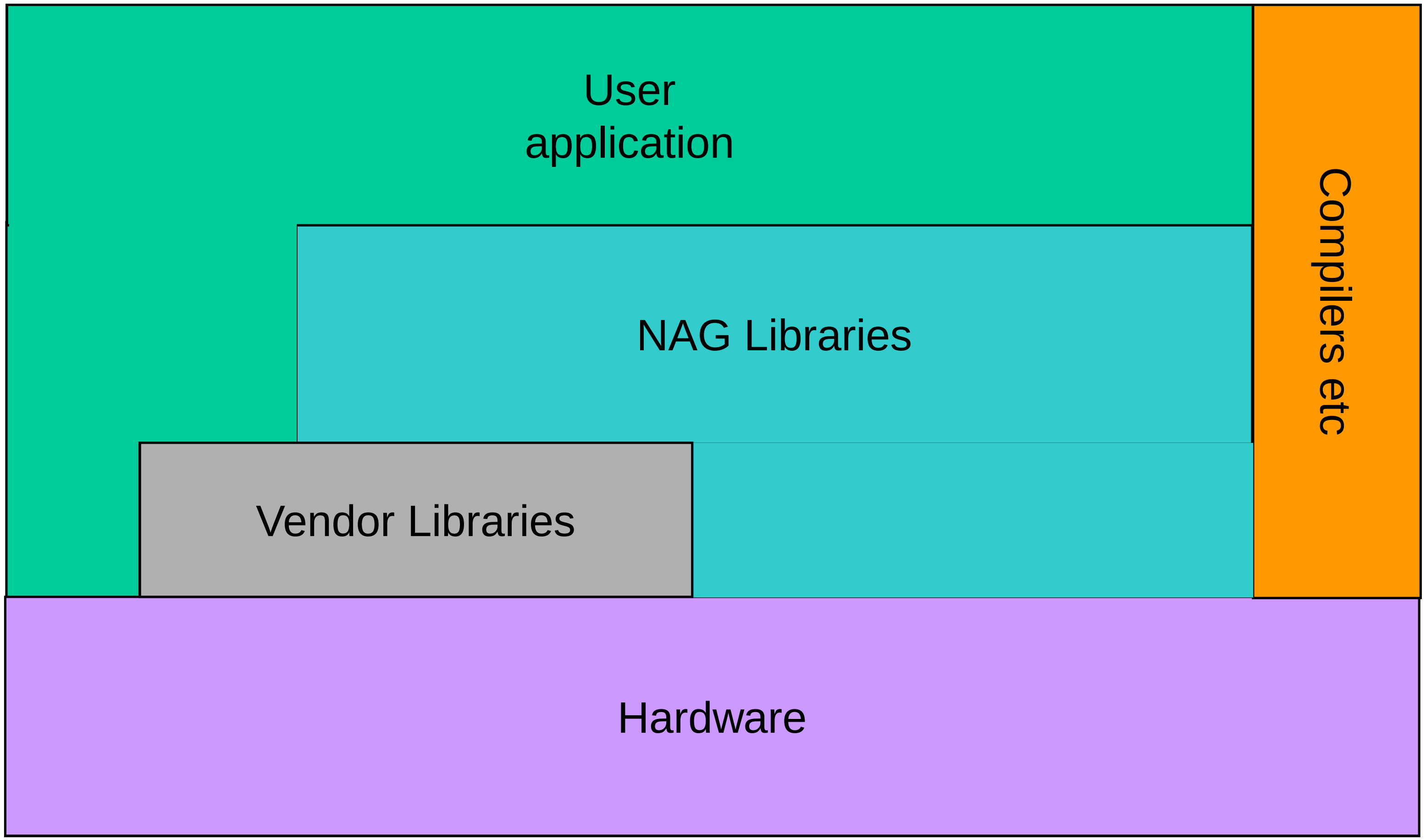
NAG SMP Library

- Based on standard NAG Fortran Library
  - designed to better exploit SMP architecture
  - Current version Mark 21, soon we will release Mark 22
- Identical interfaces to standard Fortran Library
  - just re-link the application
  - easy access to parallelism for non-specialists
    - user is shielded from details of parallelism
    - assists rapid migration from serial code
  - can be used along with user's own parallelism
    - for expert users
- Interoperable
  - call NAG SMP Library routines from other languages

NAG Library Functionality

- Root Finding
- Summation of Series
- Quadrature
- Ordinary Differential Equations
- Partial Differential Equations
- Numerical Differentiation
- Integral Equations
- Mesh Generation
- Interpolation
- Curve and Surface Fitting
- Optimisation
- Approximations of Special Functions
- 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

NAG & vendor libraries (e.g. ACML, MKL)



Target systems

- Multi-socket and/or multi-core SMP systems:
  - AMD, Intel, IBM, SPARC processors
  - Linux, Unix, Windows operating systems
  - Standalone systems or within nodes of larger clusters or MPPs
- Other possibilities:
  - Cray or NEC vector
  - Virtual Shared Memory over clusters in theory, but efficiency may be poor on many algorithms due to extreme NUMA nature of such configurations
- Notable exceptions:
  - IBM Cell?
  - Sun Niagara I
  - GPUs, FPGAs, etc

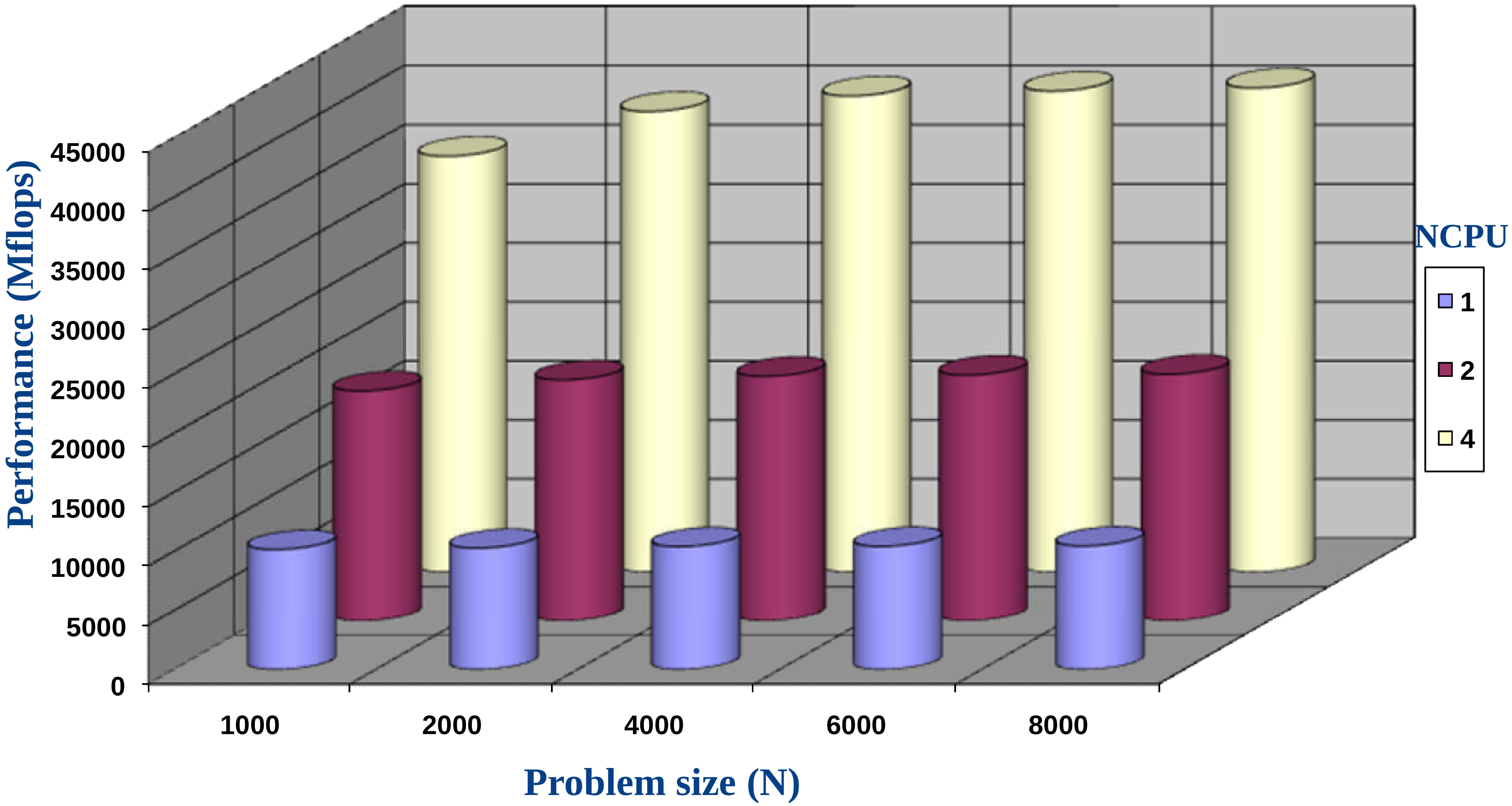
What to parallelise?

- Fundamental building blocks
  - Linear algebra and FFTs
  - Focus for first few releases
- Broaden out to different areas
  - Especially in new Mark 22
- Make potential for parallelism a key design criteria for future algorithms

Dense Linear Algebra: BLAS

- BLAS: Basic Linear Algebra Subprograms
  - BLAS1: vector-vector operations, e.g. dscal, ddot, daxpy
  - BLAS2: matrix-vector operations, e.g. dgemv, dtrsv
  - BLAS3: matrix-matrix operations, e.g. dgemm, dtrsm
- http://www.netlib.org/blas
- Optimised for cache-based architectures
- NAG SMP Library uses vendor library for fast BLAS
  - e.g. ACML, MKL, ESSL, Sunperf, Fujitsu SSL2, etc

DGEMM performance

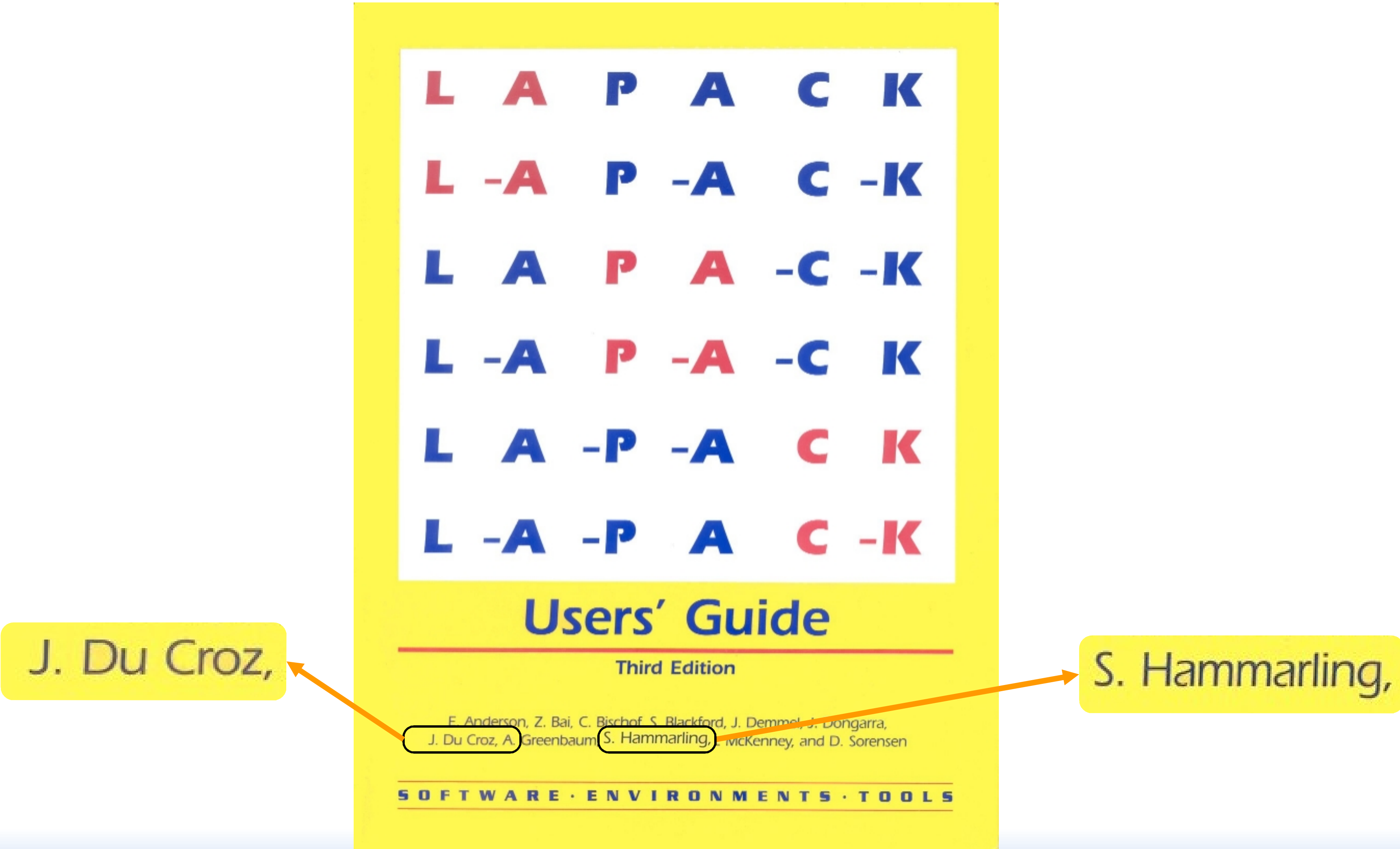


ACML DGEMM (M=N=K)

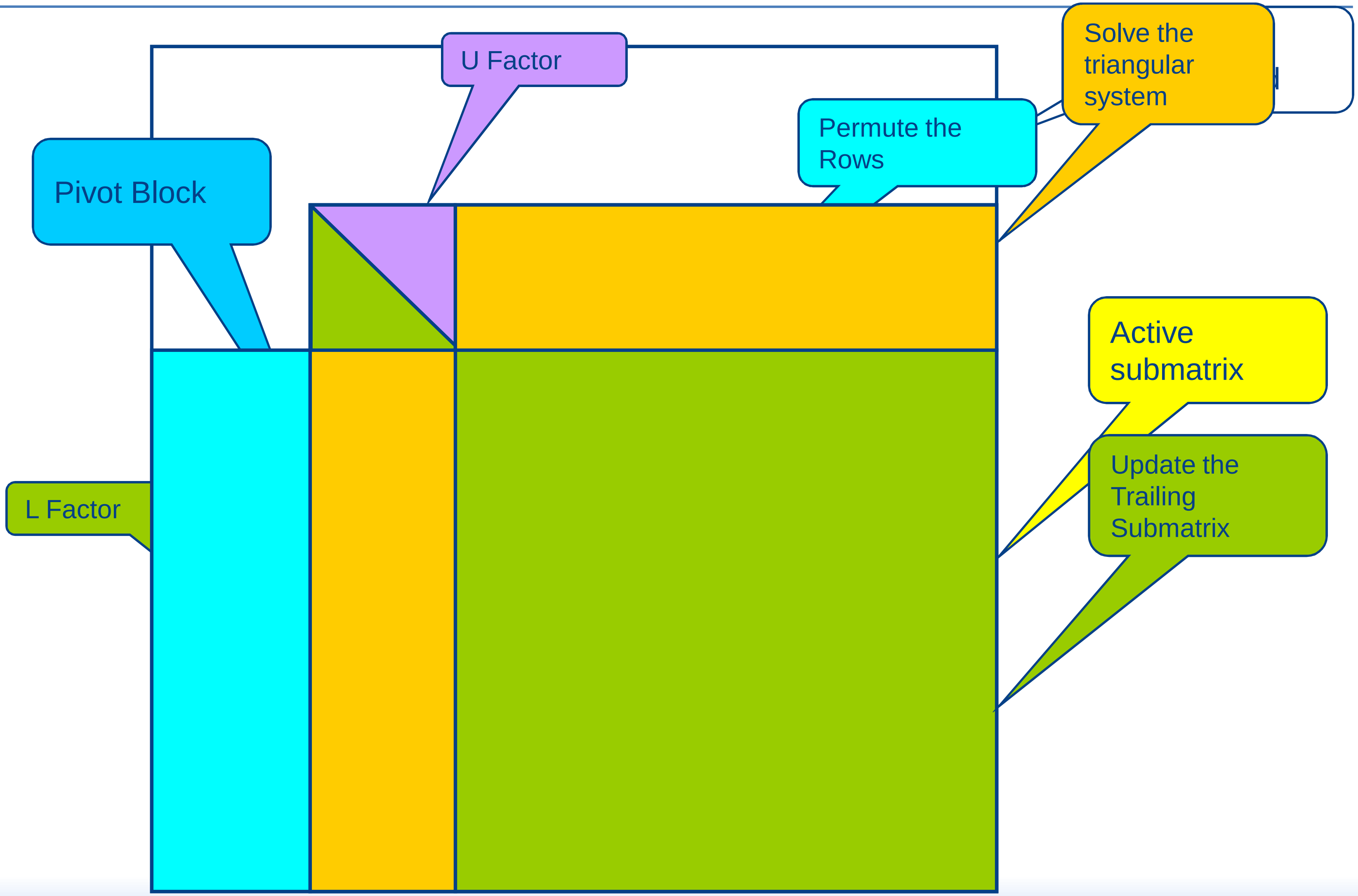
Dense Linear Algebra: LAPACK

- LAPACK: Linear Algebra PACKage
  - matrix factorisations and solvers, e.g. LU, Cholesky, QR
  - eigensolvers
  - SVD and least-squares
- http://www.netlib.org/lapack
- Builds on top of BLAS
  - Gets performance from optimised BLAS
  - Strives to use BLAS3 as much as possible
- Successor to LINPACK and EISPACK
  - also successor to earlier NAG dense linear algebra

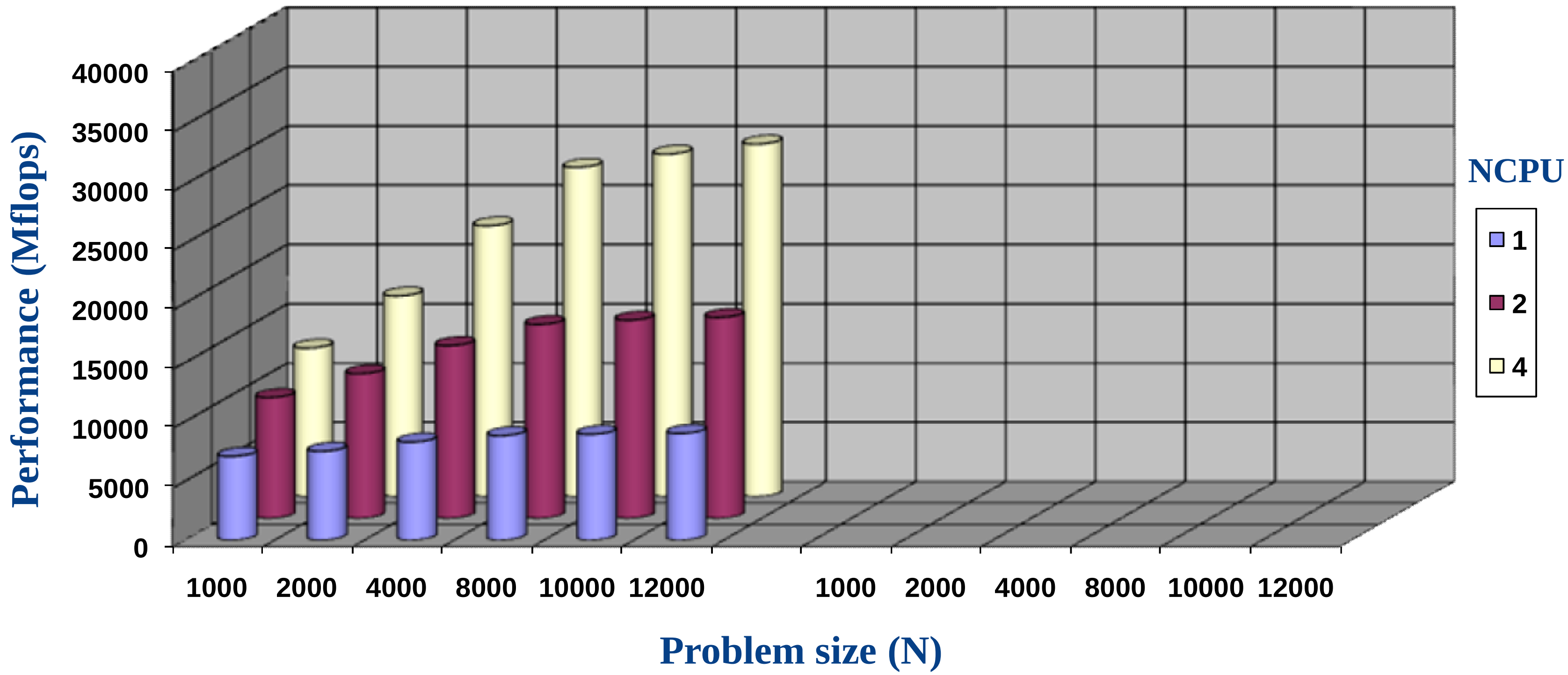
NAG & LAPACK


J. Du Croz,
L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K
Users' Guide
Third Edition
E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, Dongarra,
J. Du Croz, A. Greenbaum, S. Hammarling, McKenney, and D. Sorensen
SOFTWARE · ENVIRONMENTS · TOOLS
S. Hammarling,

LU Factorisation: LAPACK Style

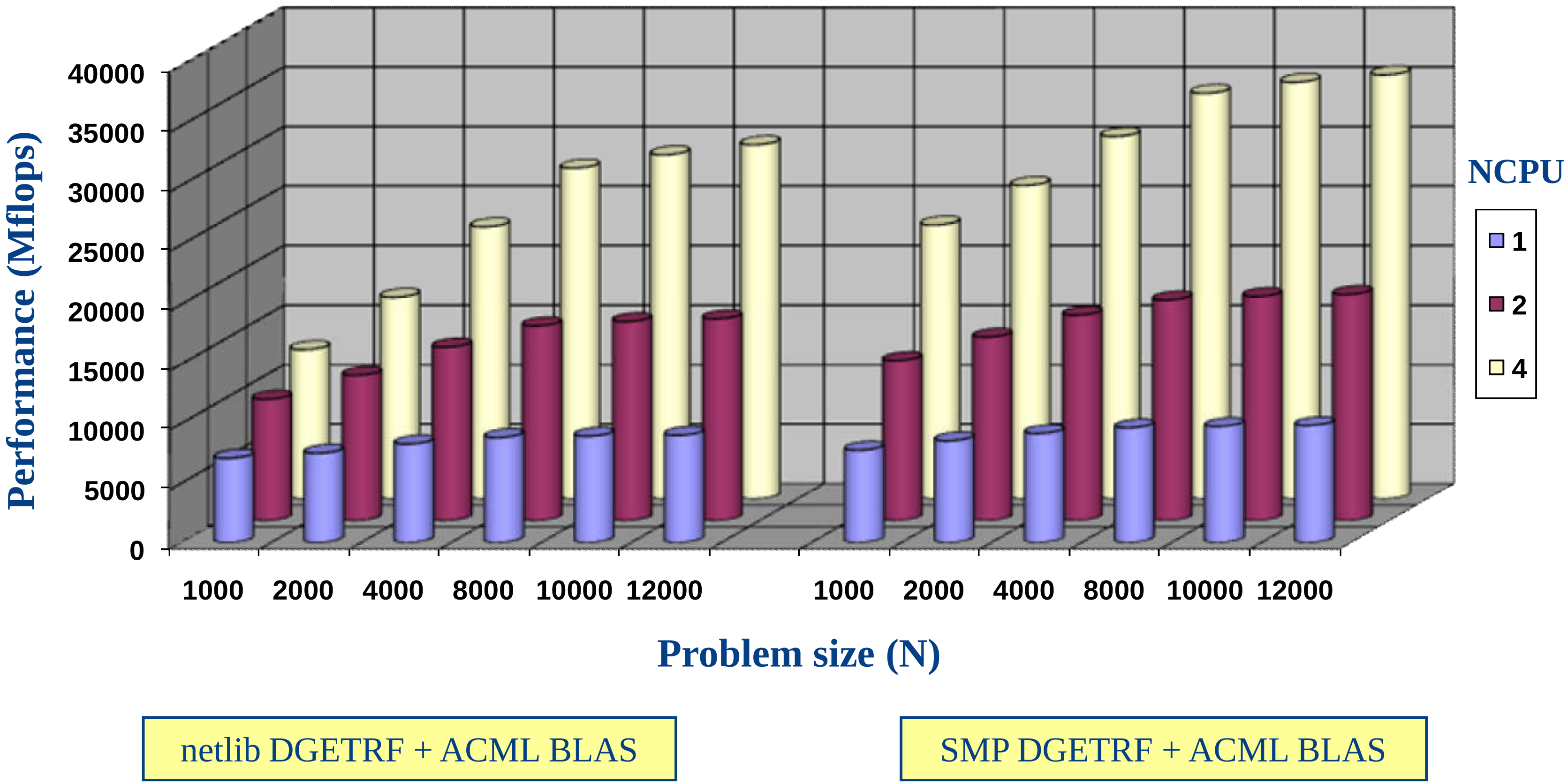


LU factorisation (DGETRF)

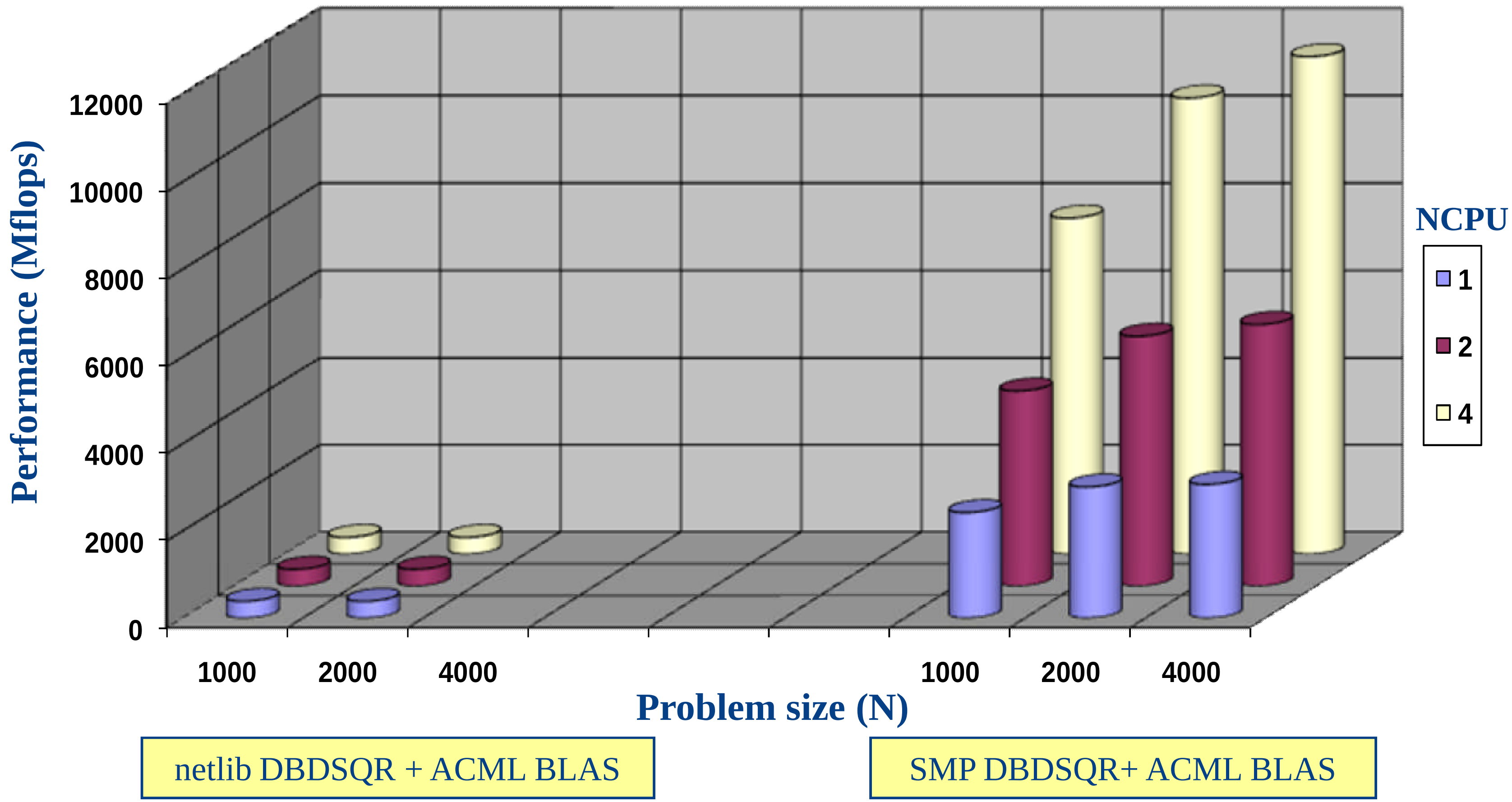


netlib DGETRF + ACML BLAS

LU factorisation (DGETRF)



S.V.D. (DBDSQR)



Performance (Mflops)
12000
10000
8000
6000
4000
2000
0
1000
2000
4000
1000
2000
4000
Problem size (N)
NCPU
1
2
4
netlib DBDSQR + ACML BLAS
SMP DBDSQR+ ACML BLAS

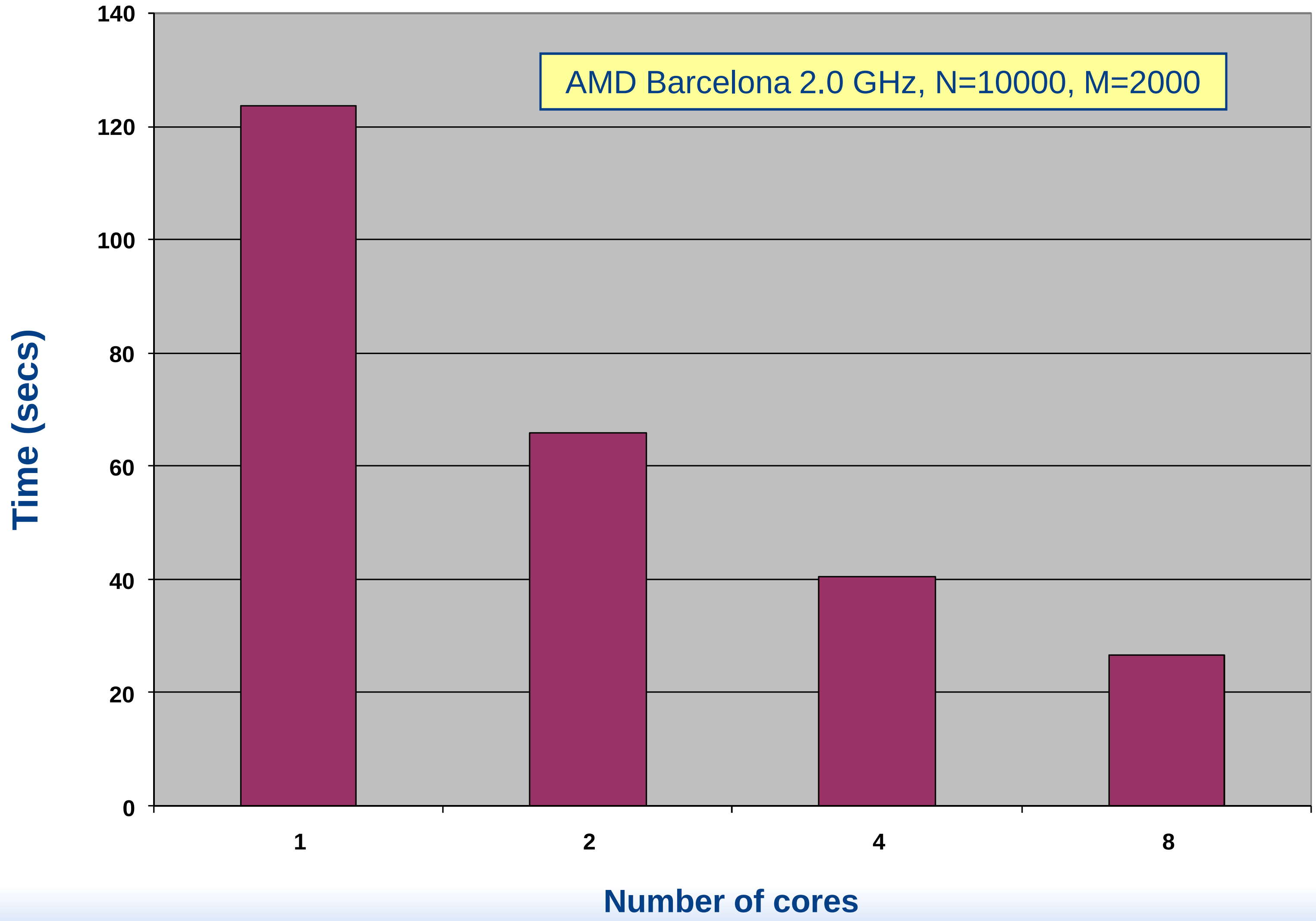
Exploiting SMP parallelism (1)

- Core-math routines (LAPACK, FFTs)
  - We aim to give best combination of vendor library and NAG routines
  - Choice varies from platform to platform
    - NAG SMP Library version may be faster on some platforms
    - If not, we recommend you just use the relevant vendor library
    - In particular, NAG works with AMD on ACML, hence all NAG SMP LAPACK routines are available in ACML
  - NAG FFT routines provide a portable interface to different underlying vendor FFT routines
    - No BLAS-equivalent standard for FFT interfaces

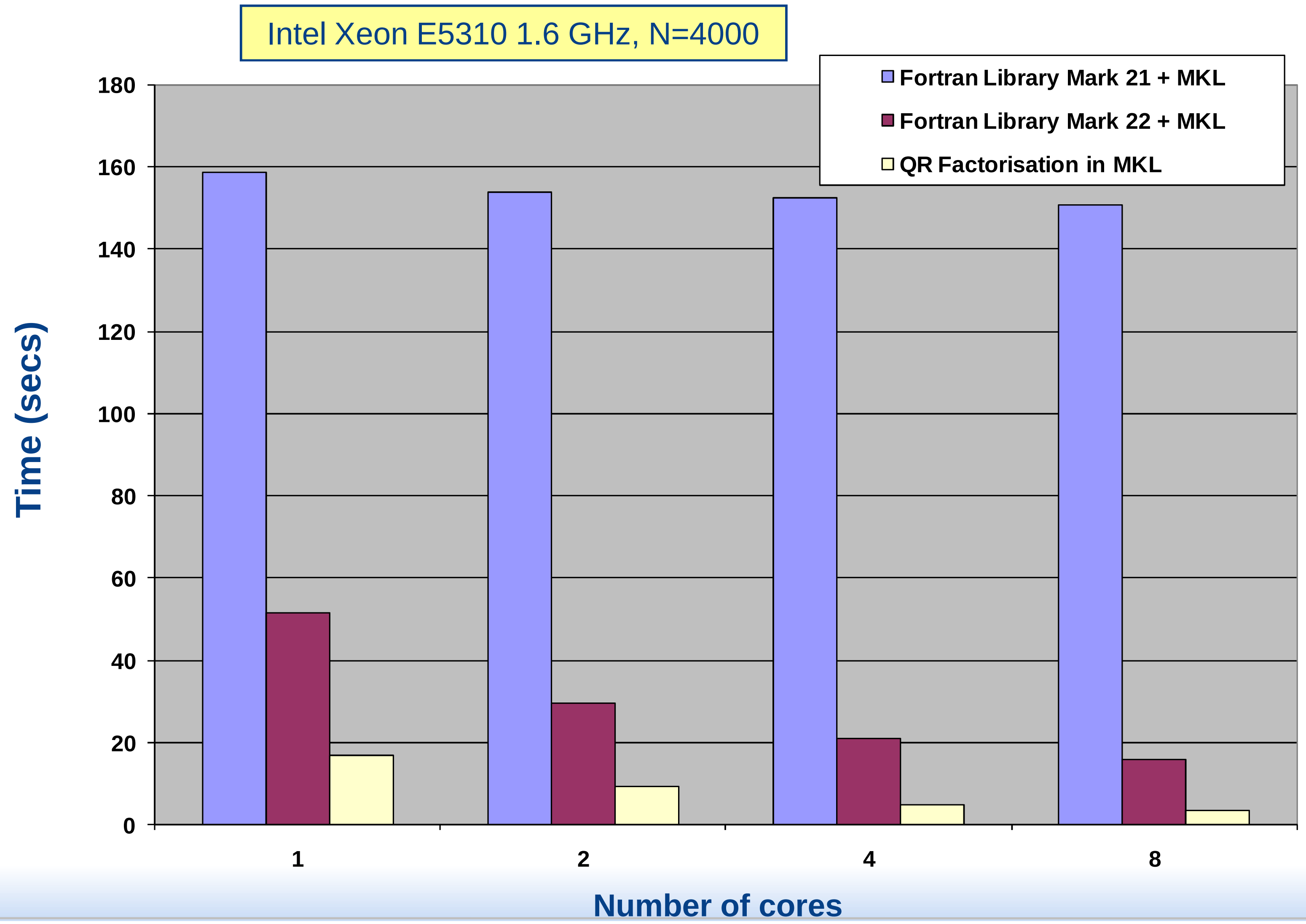
Exploiting SMP parallelism (2)

- NAG routines which use core-math routines
  - Exploit parallelism in underlying BLAS, LAPACK and FFT routines where possible
  - Development programme includes renovation of existing routines as well as adding new functionality
- Following on from (1), best choice of NAG Fortran Library vs NAG SMP Library varies from platform to platform

G03AAF: Principal Component Analysis



C05NCF: Non-linear equation solver



nag

Exploiting SMP parallelism (3)

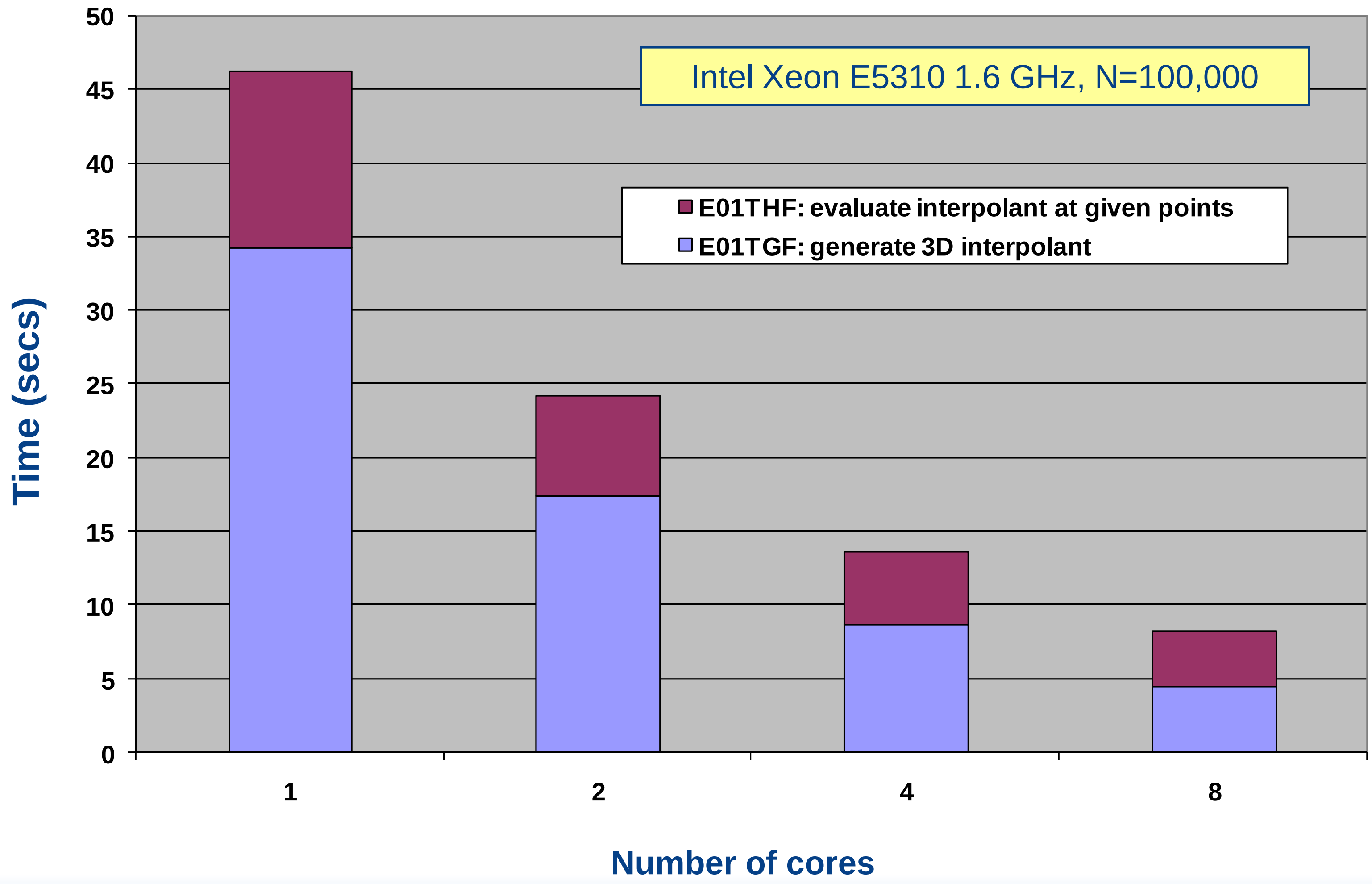
- NAG-specific routines parallelised with OpenMP
  - Focus of future NAG SMP Library development
  - Seeking to broaden scope of parallelism to different parts of the library
    - to a wide variety of algorithmic areas
    - to routines that do not use BLAS, LAPACK or FFTs

Exploiting SMP parallelism (3)

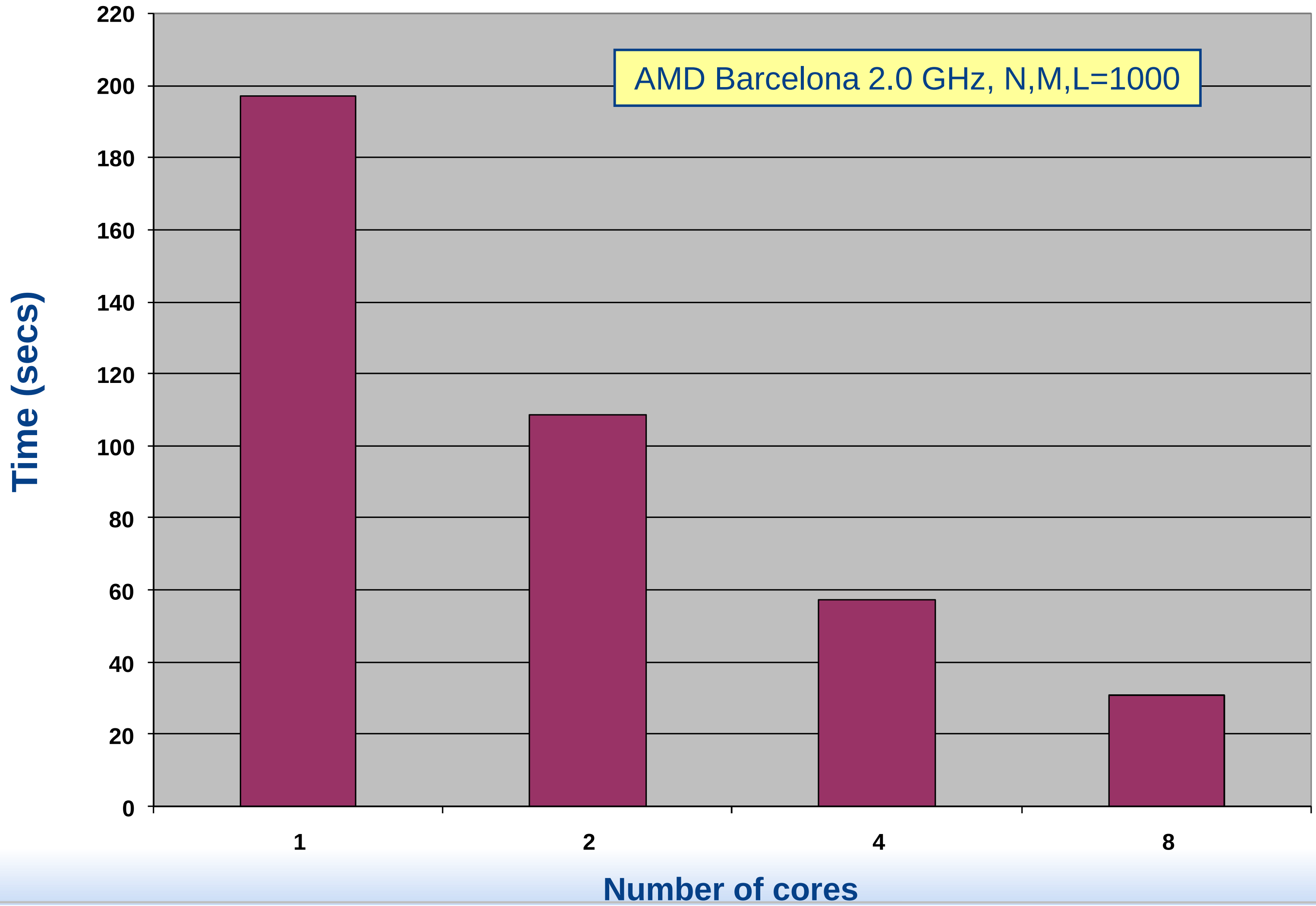
- Routines parallelised in Mark 22 in the areas of:
  - Sparse direct and iterative solvers
  - Sparse eigenproblems
  - Random Number Generators
  - Interpolation
  - Curve and Surface Fitting
  - Correlation and Regression Analysis
  - Multivariate statistics
  - Time Series Analysis
  - Financial Option Pricing

Parallelised in previous release

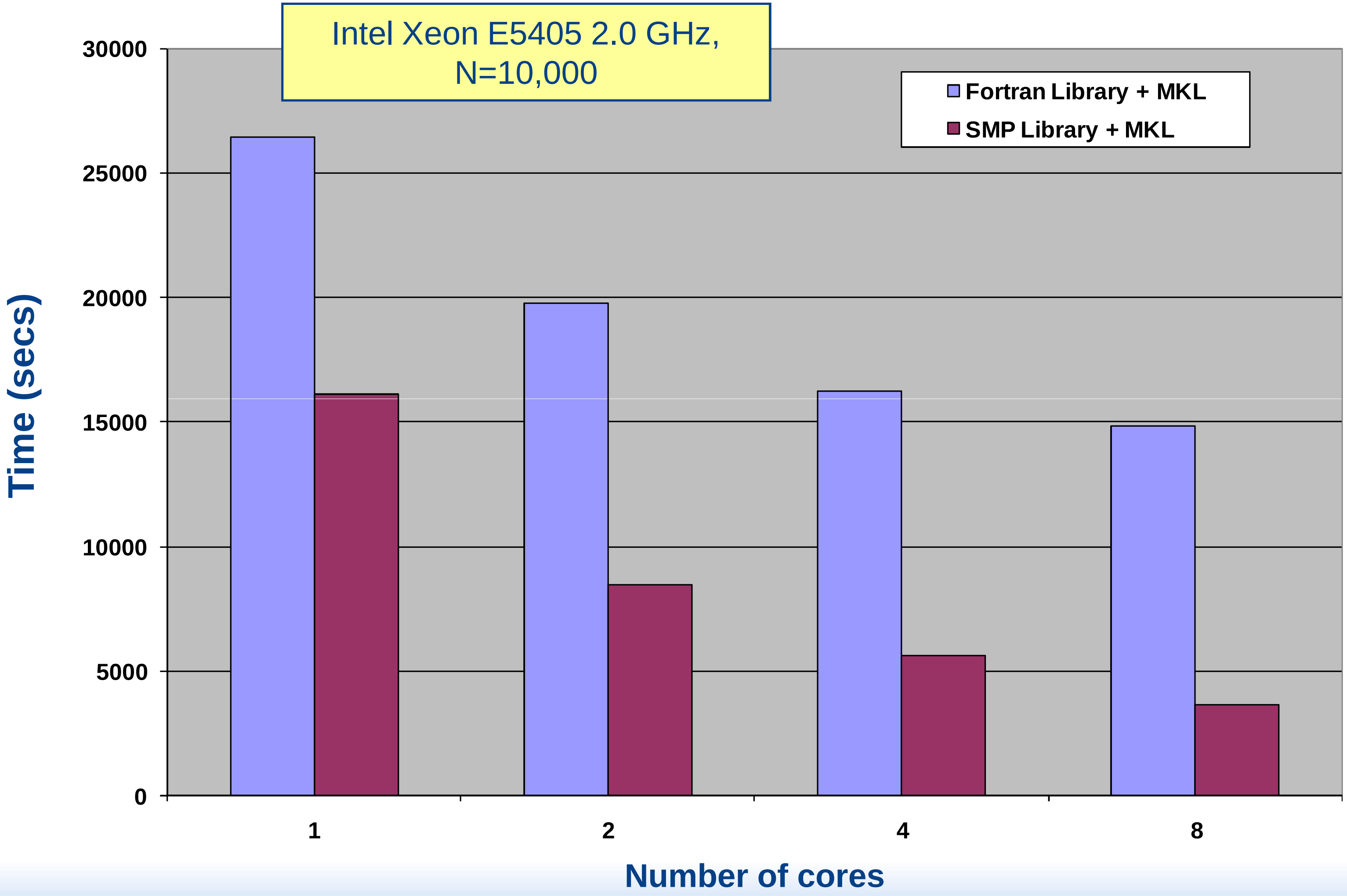
E01TGF/E01THF: Interpolation



G13EAF: Kalman filter (1 iteration)



G02AAF: Nearest-correlation matrix



Future algorithms

- Potential for parallelism is now a key criteria for selecting future algorithms
- Example: Numerical Optimisation
  - Current algorithms were written for optimal serial performance, and are not suitable for parallelism
  - Currently working on a Parallel Swarm Optimisation algorithm
    - Stochastic method
    - Poor performance on one thread but scales extremely well, thus PSO will be a complement, not replacement, for existing routines

Performance considerations

- Performance and scalability depends upon
  - Nature of algorithm
  - Problem size(s) and other parameters
  - Hardware design
  - OS, compiler and load on system
- Maximum number of threads may not be optimal
  - Important to benchmark frequently used problems on your system
  - Consult NAG for advice if required

Example 1: Sparse iterative solvers

- Problem: Iterative solver may not converge, or may converge very slowly
  - Runtime proportional to number of iterations
- Preconditioners can help reduce number of iterations required for convergence
  - at the cost of increased memory requirements in many cases
- Should we choose preconditioner parameters to minimise the number of iterations of the solver?

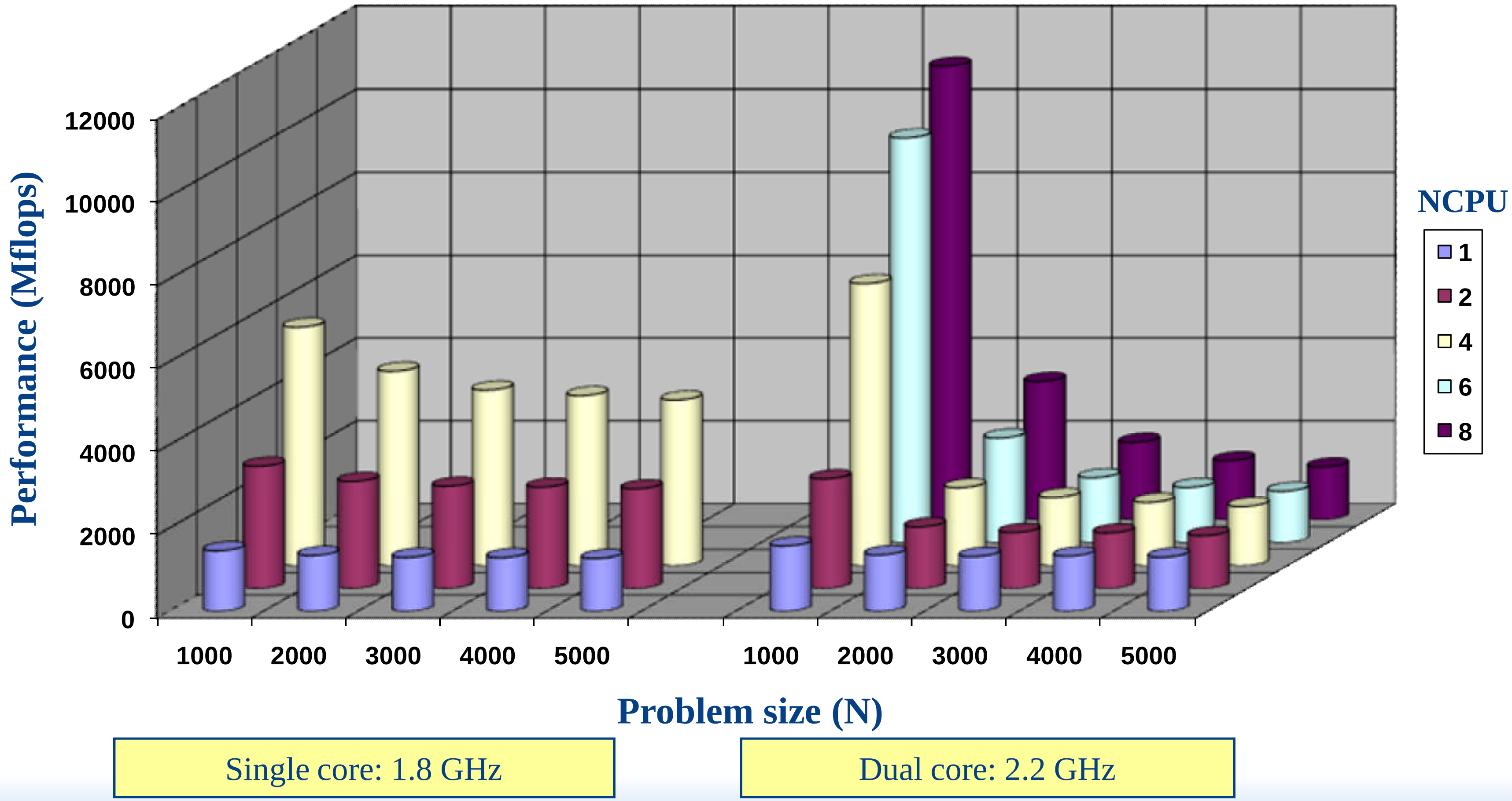
Example 1: Sparse iterative solvers

- Not necessarily!
- Need to consider cost of preconditioner
- In NAG SMP Library
  - Iterative solvers have been parallelised
  - Preconditioners are still serial
- On multiple processors:
  - turning down preconditioner, so that proportionally more time is spent in parallel solvers, may be beneficial
  - choice of parameters depends on nature of sparse matrix, system design and number of threads

Example 2: System issues

- Comparing two quad-socket Opteron systems
  - 4 x Single core, 1.8 GHz processors
  - 4 x Dual core, 2.2 GHz processors
- Linux OS
- PGI compiler
- Note: LAPACK code different from netlib source:
  - Same algorithm
  - Optimised, and parallelised with OpenMP

Opteron: Reduction to Tridiag (DSYTRD)

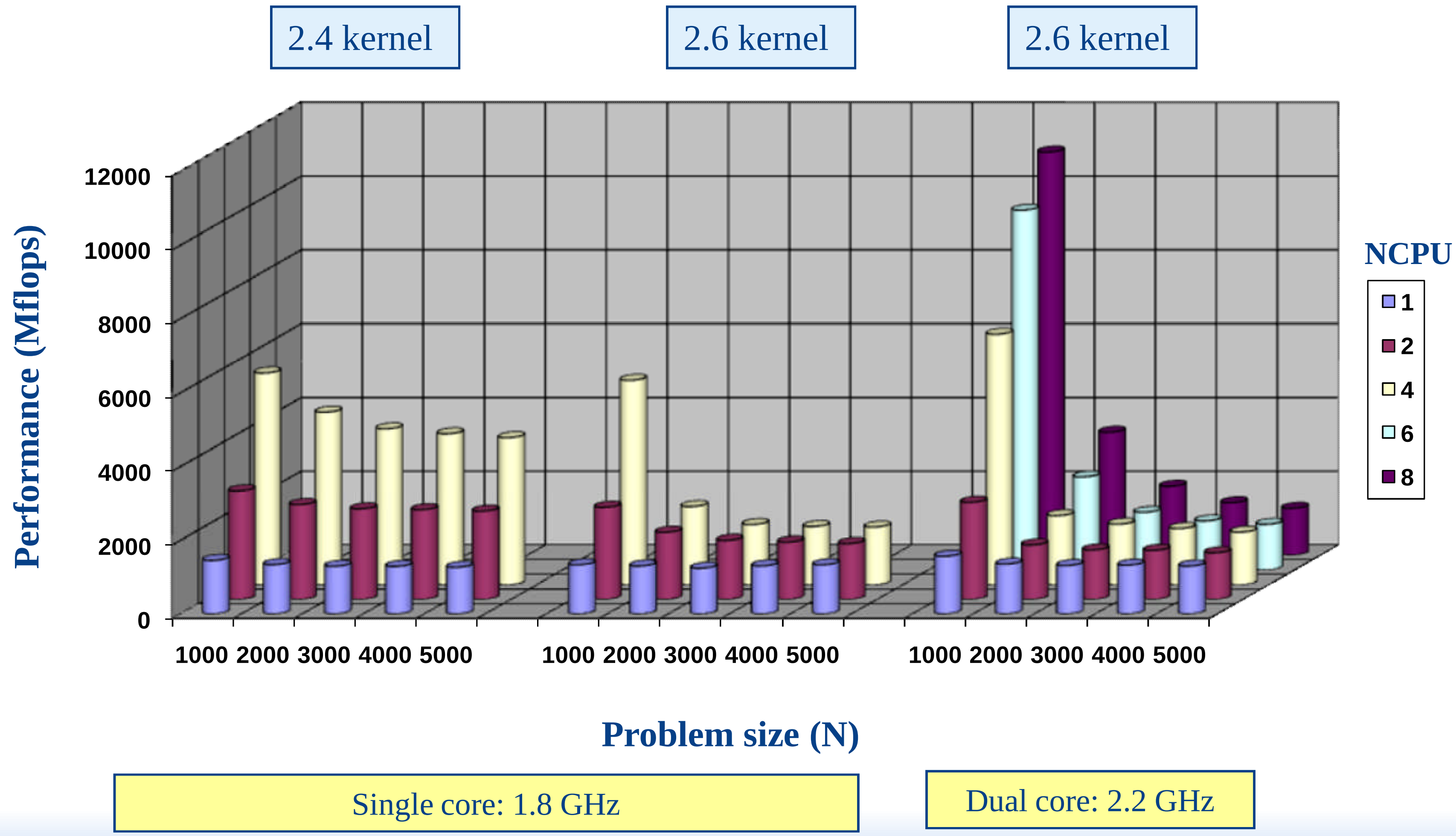


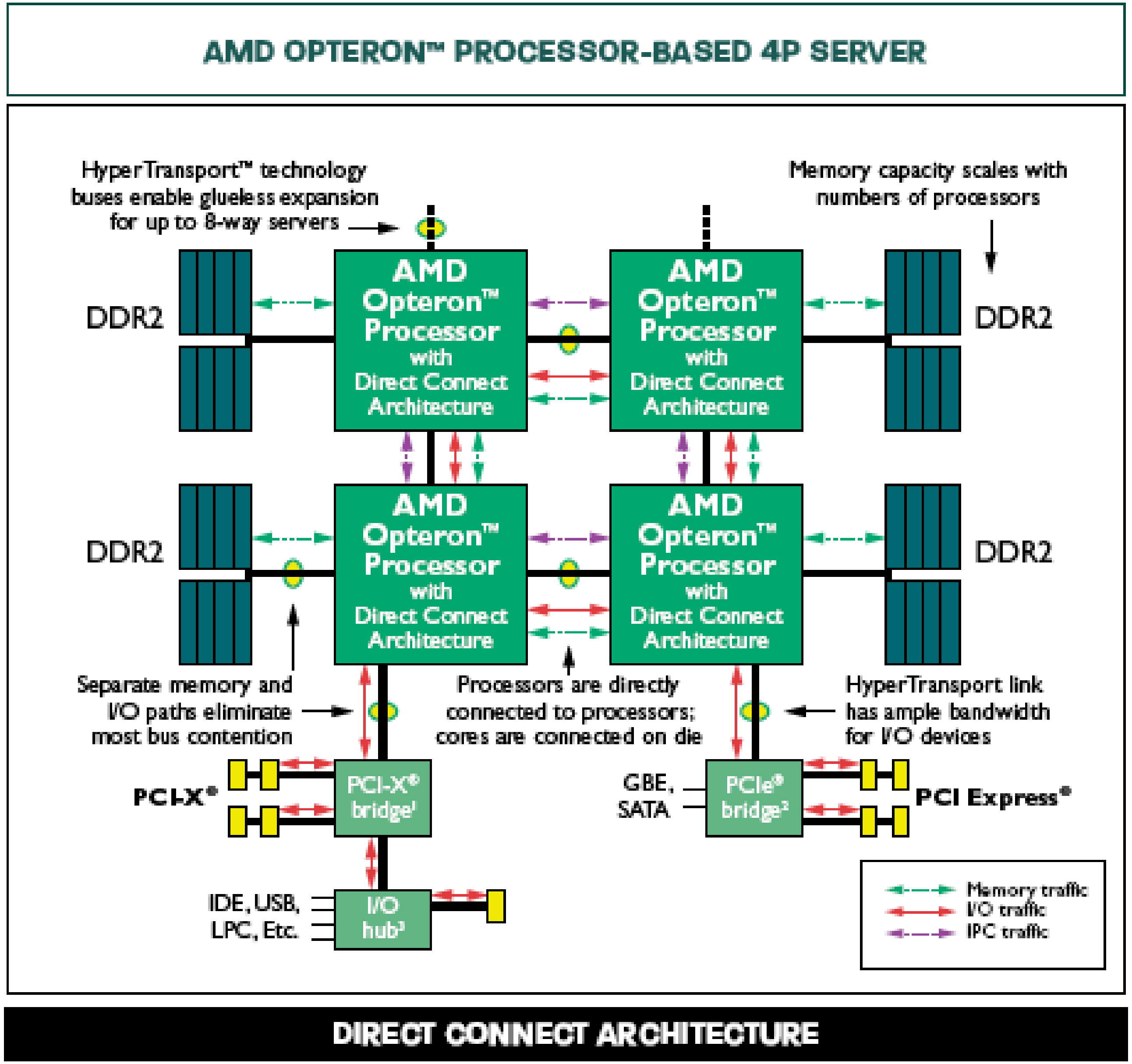
Where did the performance go???

- Q: Is multi-core to blame?
- A: No, machines had different OS (and kernel)
  - Single-core was SuSE SLES 8 (2.4.x kernel)
  - Dual-core was SuSE 9.3 (2.6.x kernel)

- Q: What if we use a 2.6.x kernel on single-core?
- A: Same effect as on dual-core with 2.6.x

Opteron: Reduction to Tridiag (DSYTRD)



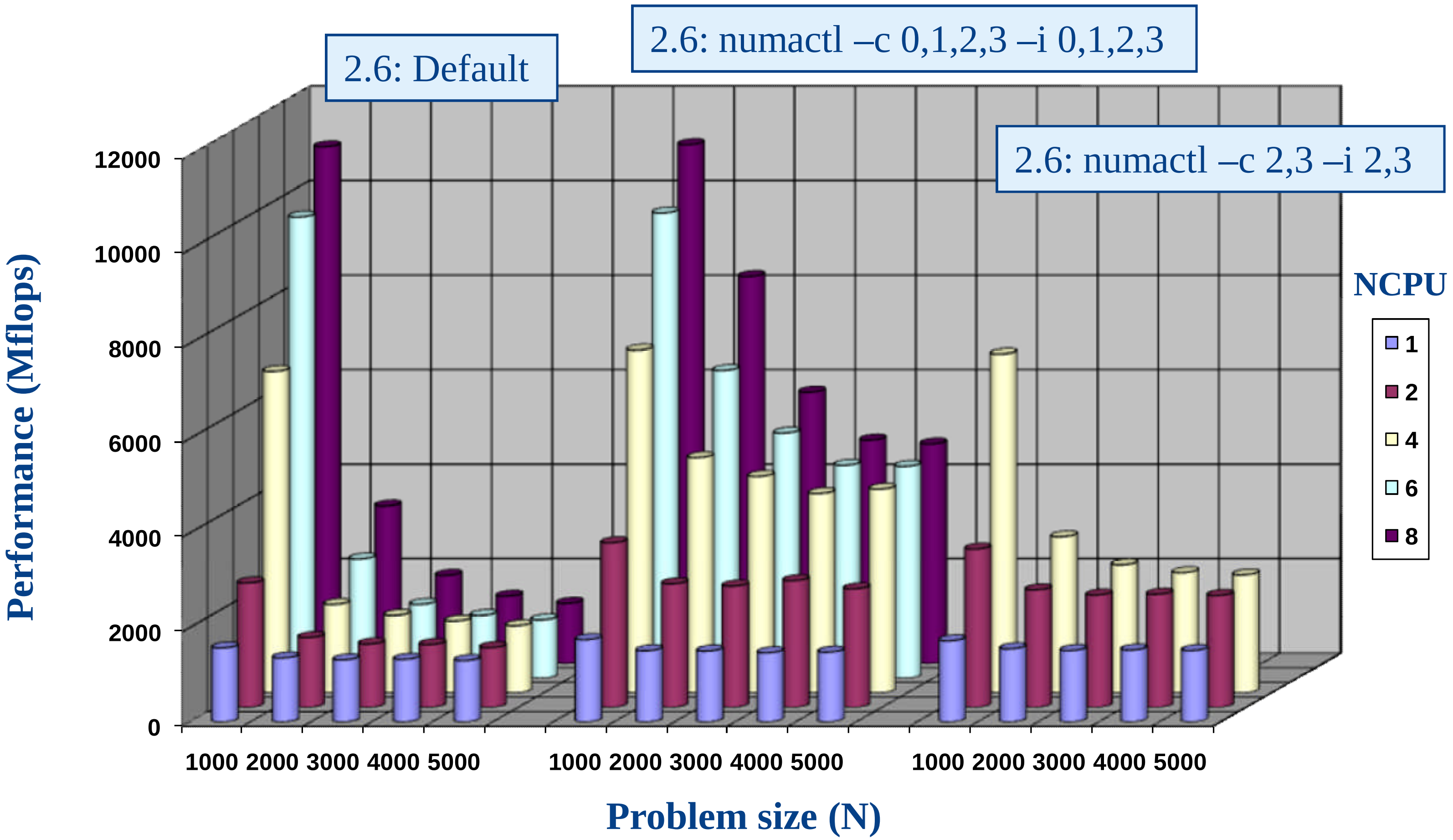

AMD OPTERON™ PROCESSOR-BASED 4P SERVER
HyperTransport™ technology buses enable glueless expansion for up to 8-way servers
Memory capacity scales with numbers of processors
DDR2
AMD Opteron™ Processor with Direct Connect Architecture
Separate memory and I/O paths eliminate most bus contention
Processors are directly connected to processors; cores are connected on die
HyperTransport link has ample bandwidth for I/O devices
PCI-X®
PCI-X® bridge¹
GBE, SATA
PCIe® bridge²
PCI Express®
IDE, USB, LPC, Etc.
I/O hub³
Memory traffic
I/O traffic
IPC traffic
DIRECT CONNECT ARCHITECTURE

nag®

numactl

- First: Check BIOS and kernel versions
- *numactl* controls NUMA policy for processes and memory, e.g.
  numactl –c 0,1,2,3 –i 0,1,2,3 program.exe
- **Interleaving of memory across nodes vital**
- DSYTRD in SMP library is memory-bandwidth hungry
- Thus better to use single core per socket, if possible

Opteron: Reduction to Tridiag (DSYTRD)



Dual core: 2.2 GHz

What about other systems?

- Questions:
  - Windows?
  - Solaris?
  - Intel systems, e.g. new QuickPath Interconnect?
  - SGI Altix?
  - IBM POWER4/5 MCM?
- Answer:
  - YMMV! (Your Mileage May Vary)
  - But be aware of this issue

Summary

- SMP systems now the norm
  - in large part due to multi-core chips
- NAG SMP Library provides an easy-to-use option for exploiting SMP hardware
  - Identical interfaces to standard NAG Fortran Library
  - Interoperable with other languages
  - Works with vendor core-math library to get best performance on dense linear algebra and FFT routines
  - Increasing number of NAG-specific routines parallelised
  - Potential for parallelism key criteria for future routines
- Mark 22 available in Q4 2009