RESEARCH COUNCILS UK

HECToR

CSC / NAG Autumn School on

Core Algorithms in High-Performance Scientific Computing

NAG III

Edward Smyth

An Introduction to the NAG SMP Library

An Introduction to SMP Parallelism

Edward Smyth

29/9/2011

nag Experts in numerical algorithms and HPC services

Overview

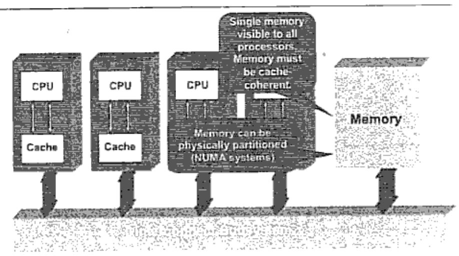
- Overview of SMP parallelism
- Very brief intro to OpenMP
- NAG Library for SMP & Multicore
- SMP performance considerations
- Two examples of performance problems
  - And solutions!

nag 2

SMP PARALLELISM & OPENMP

nag 3

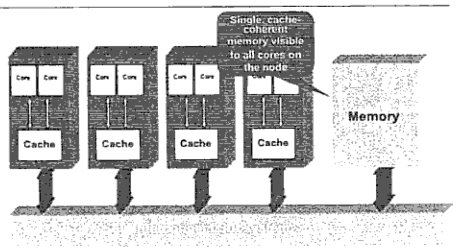
SMP = Symmetric MultiProcessing



Simple conceptual view of an SMP system (or "node")

nag 4

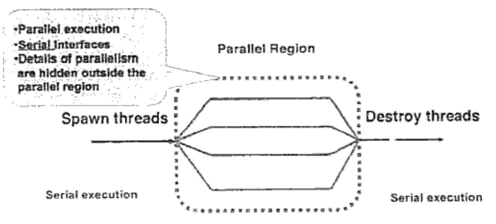
Often with multi-core processors...



nag 5

SMP Parallelism

Multi-threaded Parallelism (Parallelism-on-demand)



Parallelism carried out in distinct parallel regions

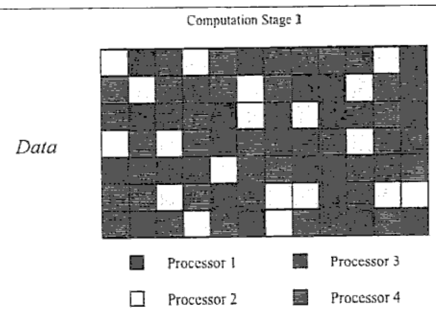
nag 6

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
- Supports incremental parallelism

nag 7

SMP Parallelism: Dynamic View of Data



nag 8

SMP Parallelism: Some Weaker Points

- Not very suitable for heterogeneous parallelism
  - Programming model generally assumes cores are identical
- Not always easy to generate efficient code
  - Distributed memory models (e.g. MPI) enforce a separation of data in memory that helps performance
- Small scale parallelism
  - ~64 cores max currently in mass market systems
  - ~2048 cores max with specialist hardware (e.g. SGI Altix)

nag 9

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

nag 10

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

nag 11

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

nag 12

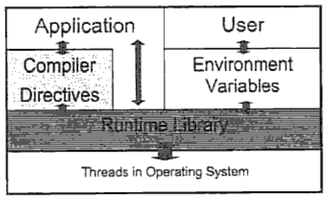
OpenMP: Introduction

OpenMP

- 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

nag 13

OpenMP: Architecture



nag 14

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

nag 15

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

nag 16

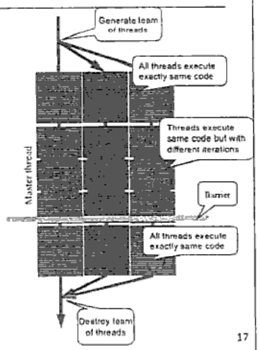
OpenMP: Parallel DO loop

All threads execute same code

Threads execute different iterations

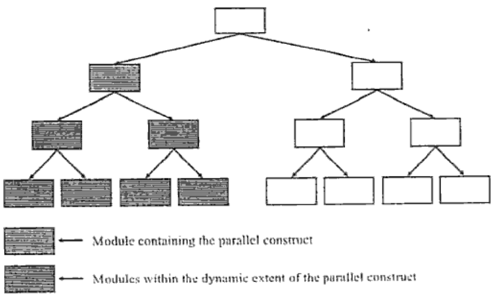
All threads execute same code

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



nag 17

OpenMP: Directives Binding



Module containing the parallel construct

Modules within the dynamic extent of the parallel construct

nag 18

Example: Dot Product

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

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

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

nag 19

NAG LIBRARY FOR SMP & MULTICORE

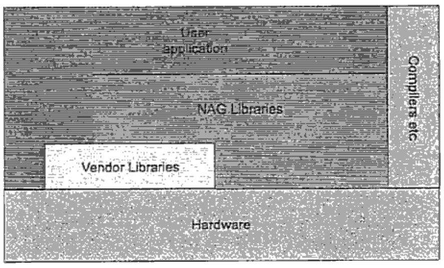
nag 20

NAG Library for SMP & Multicore

- Formerly known as the NAG SMP Library
- Based on standard NAG Fortran Library
  - Designed to better exploit SMP architecture
  - First implementations of Mark 23 due soon
- 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

nag 21

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



nag 22

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:
  - Traditional vector (Cray X2, NEC SX8, etc)
  - 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:
  - GPUs
  - FPGAs
  - Later versions of OpenMP may help us support these architectures

nag 23

Interoperability

- We want to bring benefits of SMP to multiple environments
- Currently:
  - Fortran 77 and 90 (with interface blocks available)
  - From C programs via NAG C Header files
  - Matlab (on Windows now, currently testing on Linux)
- Investigating technical issues for other possibilities:
  - Better C support via NAG C Library interfaces ("C SMP Library")
  - Excel, .NET etc.

nag 24

Compatibility issues

- NAG Library for SMP & Multicore uses OpenMP to implement parallelism
- Can work in co-operation with user's own OpenMP parallelism, or with MPI (with care!)
- Generally not compatible with other threading models, e.g. POSIX or Windows threads
  - This compatibility issue may prevent some users of NAG serial libraries from migrating to SMP parallelised library

nag 25

Parallelism in user's own code

- What is most important: Throughput on multiple tasks or turnaround on a single task?
  - Some users can parallelise their own code that makes many calls to NAG routines, e.g. different starting points for E04 local optimiser
  - Parallelism at the script level, ensemble runs, ...
  - Parallelism at higher level often more efficient
    - Tasks may be independent, so no synchronisation
    - Overheads of starting tasks amortised over long run times
    - If the number of tasks >> number of cores, efficient load balancing should be automatic

nag 26

Reasons to parallelise individual tasks

- Reduce runtime for a single task
  - e.g for Iterative processes
- Limited memory per core may restrict the number of independent tasks that can run side by side
  - Then look to make each task run faster, and to avoid wasted (idle) cores
  - Hardware trends may mean less RAM/core on average
    - e.g. SMP compute nodes on HECToR: 2-core, 3GB/core -> 4-core, 2GB/core -> 24-core, ~1.3 GB/core
- Greater number of slower cores -> need higher levels of parallelism to get same performance!?

nag 27

What to parallelise?

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

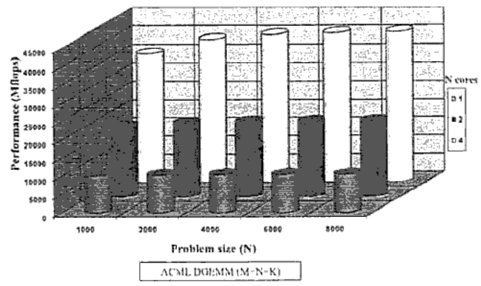
nag 28

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

nag 29

DGEMM performance



nag 30

Dense Linear Algebra: LAPACK

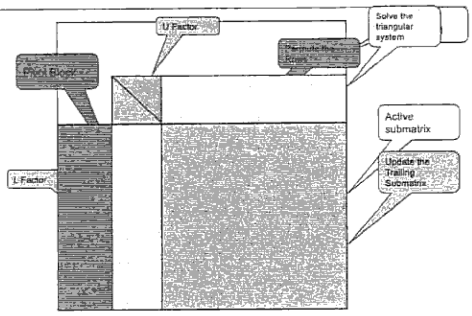
- LAPACK: Linear Algebra PACKage
  - matrix factorisations and solvers, e.g. LU, Cholesky, QR
  - eigensolvers
  - SVD and least-squares
- http://www.netlib.org/lapack
- Builds on top of BLAS
  - Gets performance from optimised BLAS
  - Strives to use BLAS3 as much as possible
- Successor to LINPACK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

nag 31

NAG & LAPACK

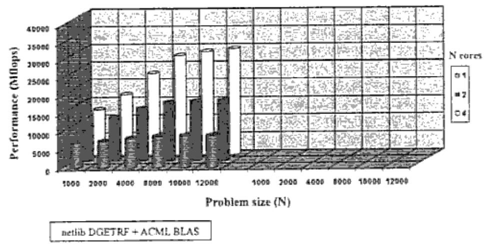
nag 32

LU Factorisation: LAPACK Style



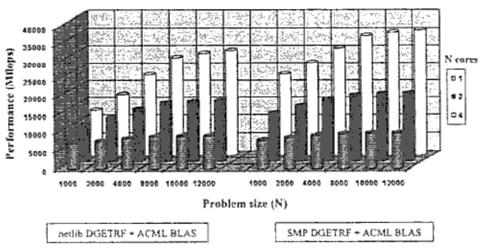
nag 33

LU factorisation (DGETRF)



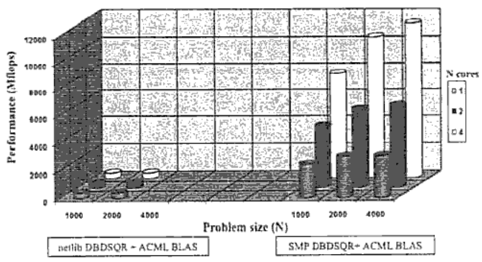
nag 34

LU factorisation (DGETRF)



nag 35

S.V.D. (DBDSQR)



nag 36

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 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

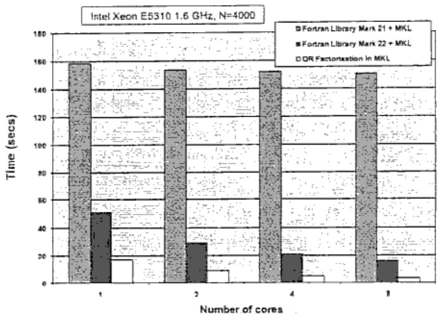
nag 37

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 Library for SMP & Multicore varies from platform to platform

nag 38

C05NCF: Non-linear equation solver



nag 39

Exploiting SMP parallelism (3)

- NAG-specific routines parallelised with OpenMP
  - Focus of future NAG SMP library development work
  - 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

nag 40

Areas with parallelism (pre-Mark 22)

- 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
- 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

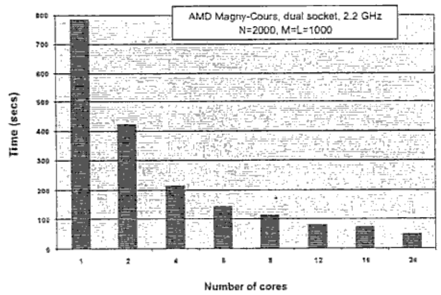
nag 41

Areas with parallelism (Mark 22 and 23)

- 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

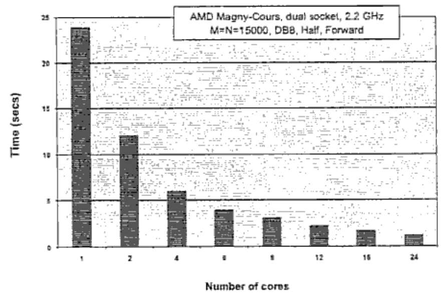
nag Note:Not all routines in these areas parallelised 42

G13EAF: Kalman filter (1 iteration)



nag 43

C09EAF: 2-D discrete wavelet transform



nag 44

Future algorithms

- Not all algorithms can be parallelised
  - Thus it may be better to replace existing routines rather than try to parallelise them
- Potential for parallelism is now a key criteria for selecting future algorithms, but note that:
  - Accuracy and stability are still most important criteria
  - Not all algorithms are computationally demanding enough to need to be parallelised

nag 45

Example: Numerical Optimisation

- Current algorithms for local optimisation (chapter E04) were written for optimal serial performance, and are extremely difficult to parallelise
- Global optimisation offers more possibilities:
  - MCS algorithm (Mark 22) also tries to optimise serial performance.
    - Scope for parallelism exists, but initial attempts have been disappointing
  - Particle Swarm Optimisation algorithm (Mark 23)
    - Stochastic method
    - Poor performance on one thread but should scales extremely well, thus PSO will be a complement, not replacement, for existing routines

nag 46

SMP PERFORMANCE CONSIDERATIONS

nag 47

Performance considerations

- In ideal world, performance would scale linearly with the number of cores used
  - Sadly this is rarely achieved
- Performance and scalability depends upon
  - Nature of algorithm
  - Problem size(s) and other parameters
  - Hardware design
  - OS, compiler and load on system
- These factors are often interconnected!

nag 48

Performance considerations

- Choice of algorithm, e.g.
  - Global optimisation: MCS vs PSO
  - Quasi-RNG: Sobol, Sobol (A659) and Niederreiter parallelised, Faure is serial
- Nature of algorithm
  - Some algorithms can sub-divide work and then each thread proceeds independently
  - Others require threads to synchronise frequently
- Not all parts of an algorithm may have been parallelised
  - Limits scalability of overall routine due to Amdahl's law

nag 49

Performance considerations

- Problem size(s)
  - Spawning threads incurs an overhead that may be significant for small problem sizes
  - In some cases we try to determine an appropriate threshold and only parallelise problems bigger than this
    - Difficult to get optimal value on all systems in a single family
  - Performance may vary depending upon whether the problem size fits in cache or not
    - We block algorithms for cache where possible, to try to minimise this variation

nag 50

Performance considerations

- Hardware design
  - Number of cores, cache sizes, memory bandwidth all affect performance
  - Significant differences between compatible processors in the same family, e.g.
    - AMD K8 vs GH (Barcelona, Shanghai etc)
    - Intel Nehalem (Core i5, i7) vs Penryn and Woodcrest (Core 2)
- Some processors have option of more virtual cores than actual hardware cores (also known as SMT)
  - Intel Hyperthreading on latest Nehalem (i5, i7 etc)
  - IBM POWER6

nag 51

Performance advice

- In general, don't oversubscribe the cores
  - Sharing is bad!
  - (but may be good for some codes on systems with SMT)
- Invest time to benchmark commonly used problem sizes/parameters
  - e.g. calculation performed during every timestep of long simulation
- Do this on the system you plan to use for real calculations if possible
  - Most large clusters/MPPs use commodity processors, which does make this easier than in the past

nag 52

Performance advice

- When benchmarking:
  - Beware of other things running on the system
    - GUI, web browser etc on your own PC
    - Other users or OS daemons (e.g. search, virus scanning)
  - Power saving OS modes
    - Can significantly affect performance from run to run
  - Dynamic overclocking, such as Intel Turbo Boost/AMD Turbo Core, e.g. Intel i7-920XM:
    - Normal speed 2.0 GHz
    - Turbo Boost, 3 or 4 cores active: 2.26 GHz
    - Turbo Boost, 2 cores active: 3.06 GHz
    - Turbo Boost, 1 core active: 3.20 GHz

nag 53

Performance advice

- Profiling tools may help but:
  - Not standard across all systems
  - Beware of chasing the wrong target! The goal is not to maximise scalability, GFLOPS, cache re-use etc, but rather we should want to **minimise runtime.**
- The maximum number of cores may not give optimal performance
  - Experiment with different values
  - OpenMP allows you to change the number of threads to use (within a max limit) for different parts of your program, e.g. before calls to different subroutines

nag 54

Performance advice

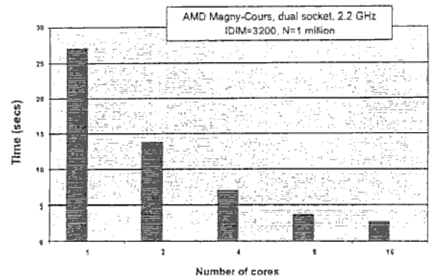
- For NAG library routines
  - Consult documentation to see which routines have been parallelised
  - Also which routines may get some benefit because they internally call one or more of the parallelised routines
  - Library introduction document gives some extra advice on using some of the parallelised routines
  - Consult NAG for advice if required
  - Feedback on which routine(s) you use and typical problem parameters etc is very useful for us for planning

nag 55

TWO EXAMPLES OF PERFORMANCE PROBLEMS (AND SOLUTIONS!)

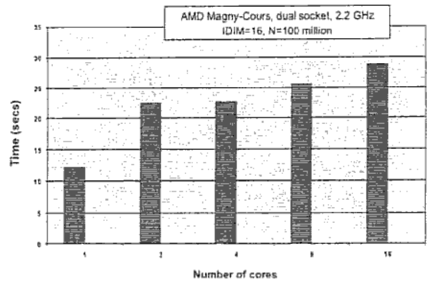
nag 56

G05YMF: Sobol quasi-RNG (original code)



nag 57

G05YMF: Sobol quasi-RNG (original code)



nag 58

Sobol quasi-RNG: False sharing problem

```
!$OMP PARALLEL &
!$OMP SHARED (IDIM,N,IREF,OUT,S) &
!$OMP PRIVATE (I,J,L)
  ...
!$OMP DO
   DO I = 1, IDIM
    DO J = 1, N
      L = Function(J)
      IREF(I) = IEOR(IREF(I),IREF(I+L))
      OUT(J,I) = REAL(IREF(I))*S
     END DO
   END DO
!$OMP END DO
  ...
!$OMP END PARALLEL
```

nag 59

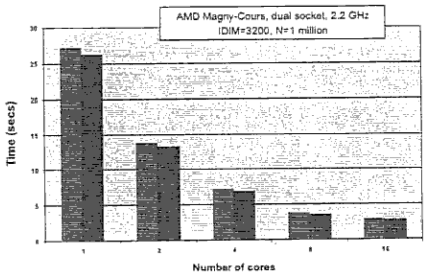
Sobol quasi-RNG: False sharing solution

```
!$OMP PARALLEL &
!$OMP SHARED (IDIM,N,IREF,OUT,S) &
!$OMP PRIVATE (I,J,L)
  ...
!$OMP DO
   DO I = 1, IDIM
    DO J = 1, N
      L = Function(J)
      IREF(I) = IEOR(IREF(I),IREF(I+L))
      OUT(J,I) = REAL(IREF(I))*S
     END DO
   END DO
!$OMP END DO
  ...
!$OMP END PARALLEL
```

```
!$OMP PARALLEL &
!$OMP SHARED (IDIM,N,IREF,OUT,S) &
!$OMP PRIVATE (I,J,L,ITMP)
  ...
!$OMP DO
   DO I = 1, IDIM
    ITMP = IREF(I)
    DO J = 1, N
      L = Function(J)
      ITMP = IEOR(ITMP,IREF(I+L))
      OUT(J,I) = REAL(ITMP)*S
    END DO
    IREF(I) = ITMP
   END DO
!$OMP END DO
  ...
!$OMP END PARALLEL
```

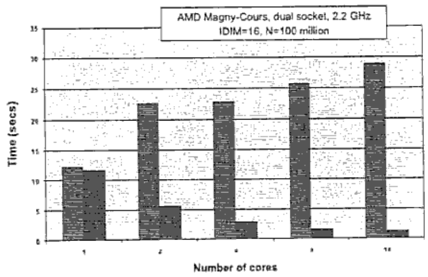
nag 60

G05YMF: Sobol quasi-RNG (orig vs opt)



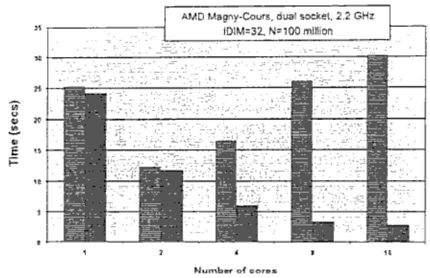
nag 61

G05YMF: Sobol quasi-RNG (orig vs opt)



nag 62

G05YMF: Sobol quasi-RNG (orig vs opt)



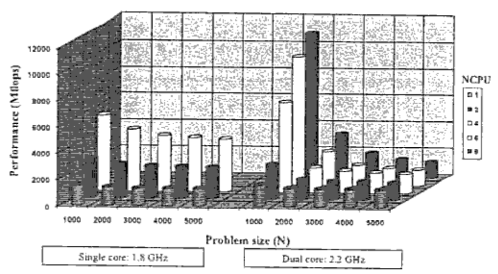
nag 63

Example of a system related issue

- Example from 2005, still relevant today
- 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

nag 64

Opteron: Reduction to Tridiag (DSYTRD)



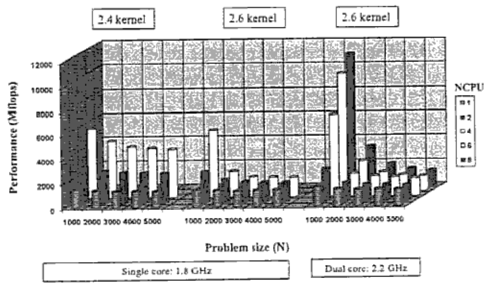
nag 65

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

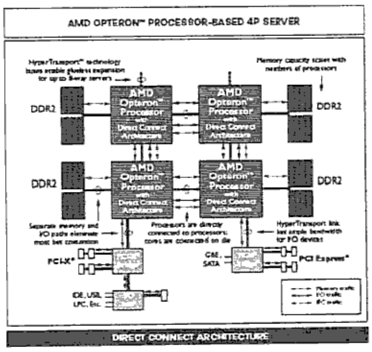
nag 66

Opteron: Reduction to Tridiag (DSYTRD)



nag

67



nag

68

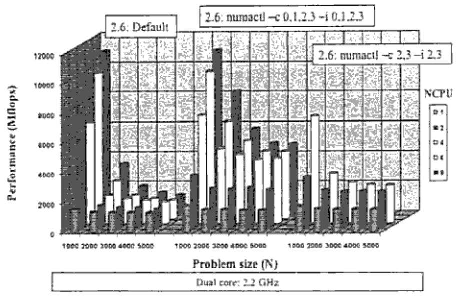
numactl

- 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
- Less benefit from using multiple cores per socket

nag

69

Opteron: Reduction to Tridiag (DSYTRD)



nag

70

NUMA effects

- NUMA architecture now common
  - Intel QuickPath Interconnect on "Nehalem" chips (Core i5, i7, etc)
  - IBM POWER 6, POWER 7
  - Larger scale NUMA on SGI Altix
- Behaviour may be different will different OS
  - Windows, AIX, Solaris
- Be aware of issue, check OS setting and consider how memory is being initialised

nag

71

Summary

- SMP systems now the norm
  - in large part due to multi-core chips
- NAG Library for SMP & Multicore provides an easy-to-use option for exploiting SMP hardware
  - Identical interfaces to standard NAG Fortran Library
  - Works with vendor core-math library to get best performance on dense linear algebra and FFT routines
  - Increasing number of NAG-specific routines parallelised
- Many factors influence performance of SMP code
  - Benchmark commonly used cases and experiment with different parameters to get best performance

nag

72