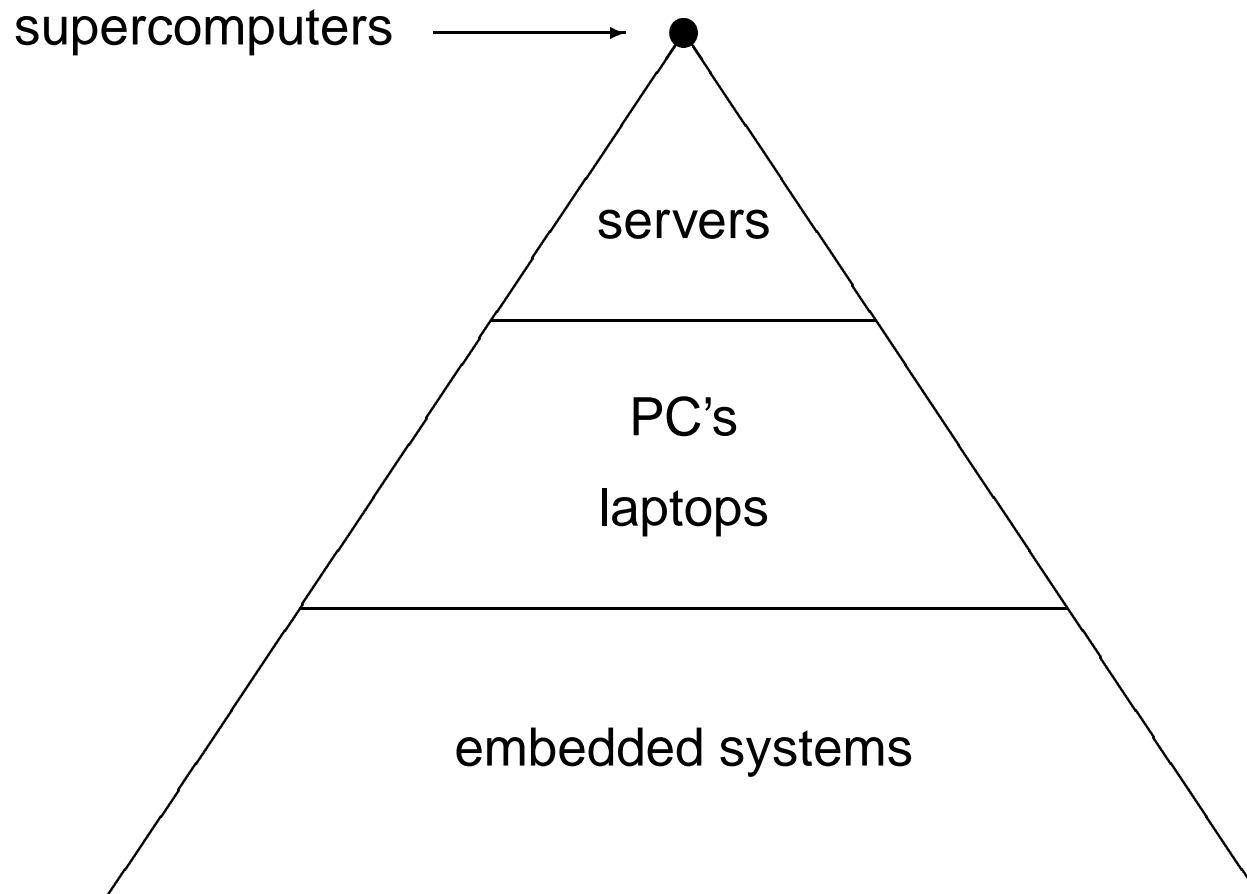


Numerically Intensive Computing in Science and Finance

Mike Giles

`giles@comlab.ox.ac.uk`

The Hardware Pyramid



Hardware

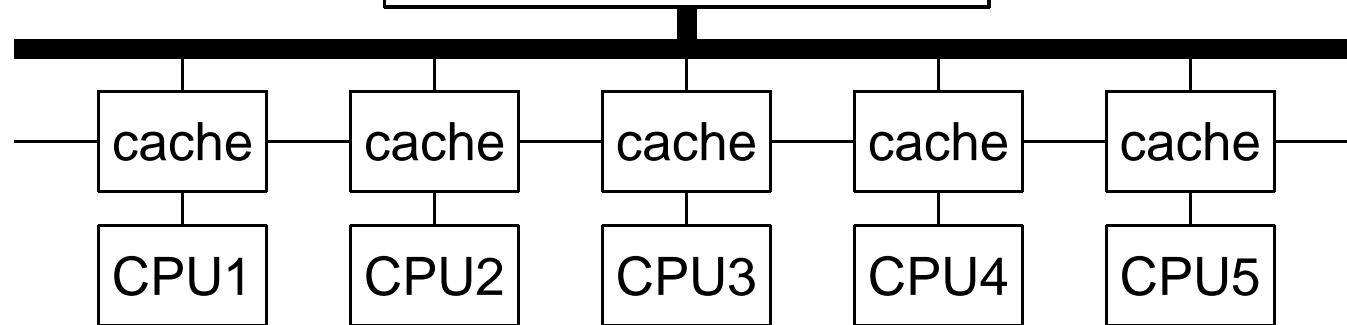
- almost all scientific computing now done on systems built of commodity components
- 4 : 2 : 1 ratio in performance for servers : PCs : embedded systems
- advances in computing driven by needs, not technology: ask “why?” not “how?”
- Moore’s Law to continue for next 5 – 10 years, at least (speech recognition, image processing, expert systems)
- rest will keep pace, with memory struggling (ever more complex caching) and networking improving more rapidly (video streaming)

Hardware for high-end computing

- 1) shared-memory multiprocessor
 - the modern mainframe – high-end products from Sun and IBM used widely, e.g. database applications
 - single very large memory, accessed by multiple processors
 - hardware challenge is high bandwidth memory access – costly
 - often has high-reliability features such as hot-swap disks, redundant power supplies – adds to cost

Hardware for high-end computing

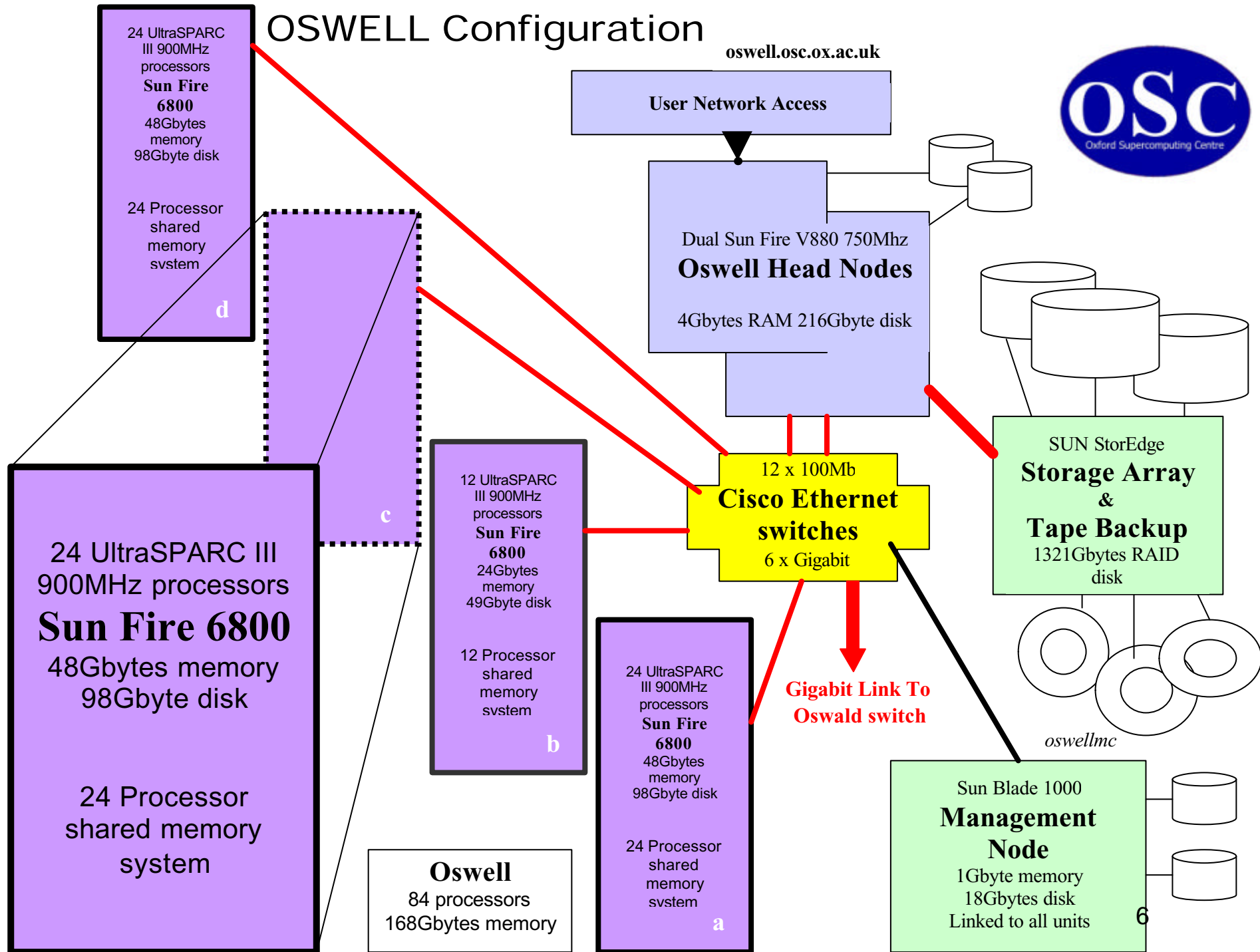
Main Memory



Really good performance requires an understanding of memory caches to avoid conflicts.

OSWELL Configuration

oswell.osc.ox.ac.uk

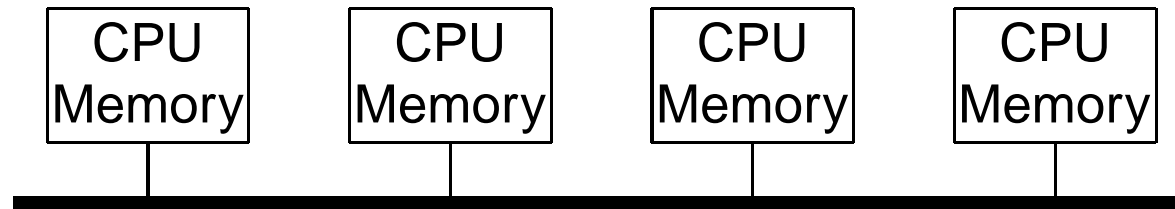


Hardware for high-end computing

2) tightly-coupled distributed-memory system

- multiple nodes (with 1 – 4 processors) each with own memory
- high-bandwidth low-latency network connection (Myrinet, Gigabit Ethernet)
- in academia, used to be collections of PCs on a shelf, but now there are tailored packages from the leading vendors
- a key issue is system management; you lose all of the price/performance benefits if you have to employ lots of system managers

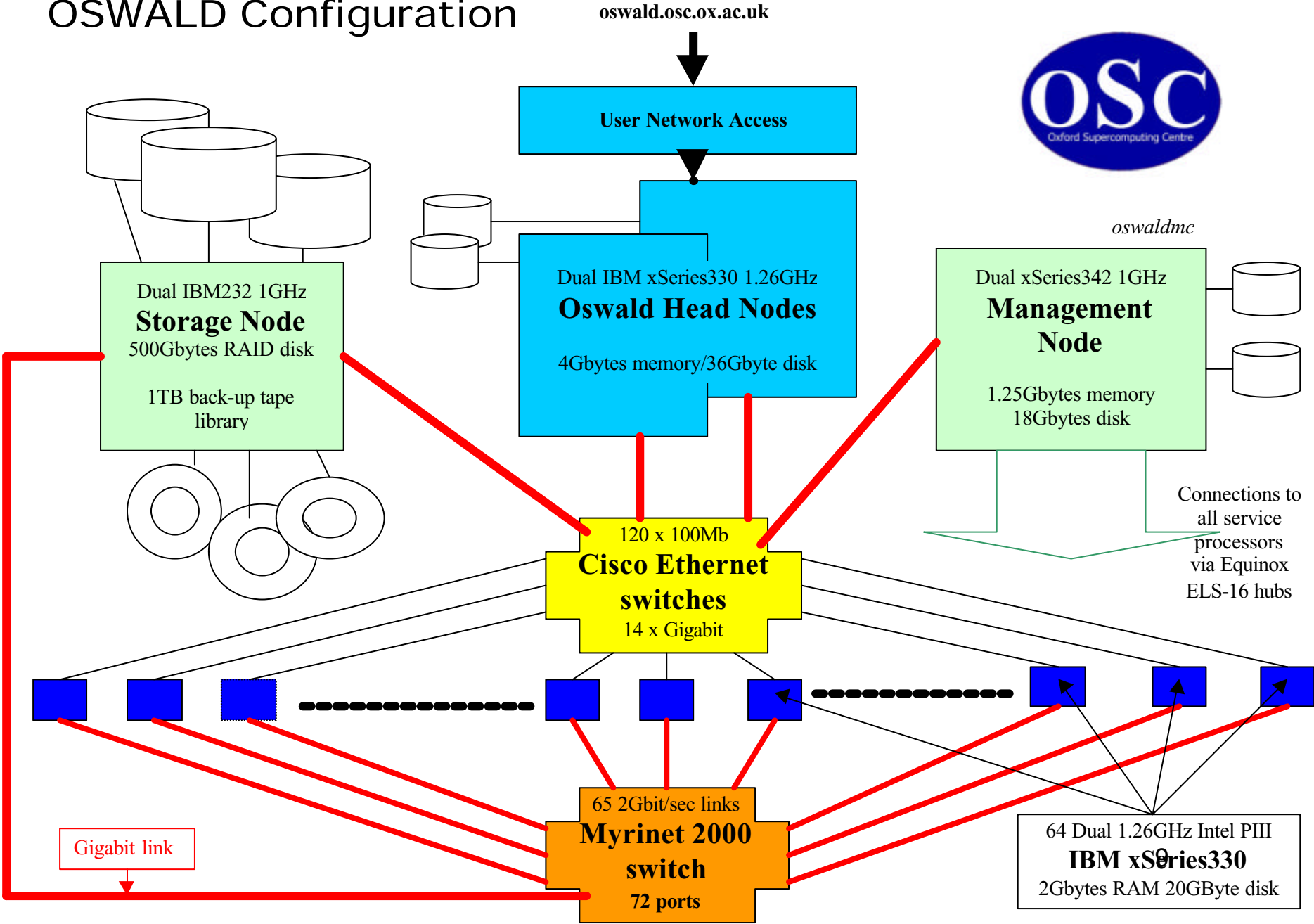
Hardware for high-end computing



Good performance depends on

- the application
- the network hardware
- the communication software

OSWALD Configuration



Hardware for high-end computing

Another example: OCCF cluster

- 24 Sun Ultra-80 nodes each with 4 UltraSPARC processors and 2GB memory
- connected by Myrinet for parallel computing, and Fast Ethernet for file i/o and external network access
- separate mini-network has Reuters datafeed with MIS software link to an Informix database

Hardware for high-end computing

- 3) loosely-coupled PC/workstation “farms”
- similar to 2) but with relatively low-speed interconnect (switched Fast Ethernet)
 - ideally suited for “trivially-parallel” applications like Monte-Carlo
 - system management and resource management are again the key issues

Hardware for high-end computing

Dedicated farms:

- racks of up to 1024 PCs (in special dense packaging) at bio-informatics companies

Collection of “idle” resources:

- computer teaching labs in the university, unused most of the time!
- traders’ workstations/PCs which are idle overnight and at weekends

Hardware for high-end computing

Comparison:

- 8 : 2 : 1 cost/performance ratio for three categories
- shared-memory and distributed-memory systems built of high-end processors because of cost of interconnect
- PC/workstation farms built of low-end processors for best price/performance ratio

Hardware for high-end computing

Trends:

- business/finance is replacing science as main user of “supercomputers”
- big shared-memory systems (64+ procs) may start to decline because of new distributed-memory database software
- PC farms moving to mobile processors for low power, maximum packing density

Hardware for high-end computing

Predictions for financial computing in 5-10 years:

- multiple 24-processor SMP systems for database applications, finite difference calculations
- PC farms with 256 processors per rack for Monte Carlo calculations

Hardware vs. Software

Hardware:

- phenomenal technological advances, driven by user needs
- new products every year, new architectures every 10 years

Software:

- disappointingly slow progress, limited by “people” issues
- new languages and standards every 10 years or so

Software “People” Issues

- need global standards, agreed by committee which takes time – Fortran 90 was motivated by vector computing, which was on the way out by the time the standard was agreed!
- need (re)training of staff – in the worst case have to wait for existing staff to retire!
- staff can be reluctant to learn new skills which might not be transferable to a new employer – another reason for standards
- companies have been happier investing in hardware than software – changing these days?

Languages

Fortran and C:

- for those who want the highest performance
- closest to the level of the operating system
(written in C)

C++, Java, C#:

- object-oriented computing for better software design and re-use of code (in principle)

Visual Basic, Matlab, Python/Jython:

- “niche” languages, each with a strong following
- emphasis on ease of use

Parallel Computing

OpenMP is the standard for multithreaded computing on shared-memory systems

- available for Fortran, C and C++
- supported by all major vendors ensuring code portability
- very easy to implement, just a few lines of comments/compiler directives
- much higher level than using Posix threads

Parallel Computing

OpenMP:

- a single “process” with multiple “threads” working on the same data
- all thread generation and management is handled automatically
- programmer simply tells the compiler which bits (e.g. loops) can be carried out in parallel by multiple threads
- programmer also has to say which variables are private (each thread has its own working copy) and which are shared (all threads see the same value)

Parallel Computing

MPI is the standard for distributed-memory computing

- again available for Fortran, C and C++
- again supported by all major vendors ensuring code portability
- much harder to develop MPI code, and to maintain it

Parallel Computing

MPI:

- multiple “processes” each with its own data
- in finite difference applications, each process will “own” just one piece of the whole grid
- when data is needed from a neighbouring “piece” (partition) messages have to be sent from one process to another
- rather low-level, tedious – for programmer productivity need higher-level libraries to hide the nasty details

Distributed Resource Management

Using loosely-coupled PC/workstation farms for Monte Carlo calculations needs distributed resource management:

- which machines are available?
- how heavily are they being used?
- do they have the necessary software/licenses?
- what rights do I have to use them?

Distributed Resource Management

Products such as Grid Engine (Sun) and LSF (Platform Computing) deal with this through users submitting tasks to a unified queue which dispatches jobs based on:

- matching job requirements to machine properties
- taking account of current interactive/batch usage of machine
- taking account of different priorities of different user groups
- doing charging if necessary

Maybe sounds simple – but very important

Grid Services

Something new which is evolving very rapidly:

- builds on web services for eBusiness
- provides a modular approach to heterogeneous computing across the internet, or within a corporate intranet (the successor to CORBA?)
- uses new standards such as XML/SOAP (communication) and UDDI (resource discovery)
- DTI and research councils are funding £200M academic research in eScience using grid services

Grid Services

- Microsoft is building its services on C# & .NET
- everyone else is building on Java, J2EE
- adherence to standards by both sides ensures interoperability
- first impact in finance may be through simple coupling of Microsoft-based user interfaces (e.g. Excel spreadsheet) to back-end computation on Unix systems

An Application

Modelling the value of a gas storage facility:

- gas price modelled as mean reverting stochastic process with a known volatility and time-varying mean value
- PDE used to model evolution of value $M(V, S, t)$ as a function of filled volume fraction, gas spot price and time
- Bellman optimality principle used to decide when to buy/hold/sell gas.

An Application

$$\begin{aligned} \frac{\partial M}{\partial t} - \kappa S \log\left(\frac{S}{\mu(t)}\right) \frac{\partial M}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 M}{\partial S^2} \\ + \max\left(0, Q_{out} \left(-\frac{\partial M}{\partial V} + S - C_{out}\right)\right) \\ + \max\left(0, Q_{in} \left(\frac{\partial M}{\partial V} - S - C_{in}\right)\right) = rM \end{aligned}$$

on the domain $S_{min} < S < S_{max}$, $0 < V < 1$.

Boundary conditions:

- No selling on $V = 0$; no buying on $V = 1$
- $\frac{\partial^2 M}{\partial S^2} = 0$ on $S = S_{min}, S_{max}$.

An Application

Development effort for an explicit finite difference approximation:

- Matlab implementation – 1 week
- Porting to Fortran – 1 day
- OpenMP implementation – 1/2 day
- MPI implementation – 1 day (+1 week for high-level library)

An Application

Execution speed relative to Fortran code:

- Matlab – 0.02
- OpenMP – 3.9 using 4 processors, on grid of 200×400 .
- MPI – 3.7 using 4 procs, 9.3 using 16 procs, on grid of 200×400 .

The MPI performance is limited by bandwidth. For 2D calculations, OpenMP is best. MPI becomes more appropriate for 3D and 4D applications. For higher dimensions, Monte-Carlo methods are a better approach.

An Application

Next step is Monte-Carlo simulation:

- using buy/sell thresholds from PDE model
- validate and assess variability

Not yet implemented, but previous experience with a simple Monte-Carlo application:

- execution time is dominated by random number generation; Matlab is half the speed of NAG's Fortran routines
- easy to use Matlab as a scripting language to launch multiple Fortran codes within Sun's Grid Engine environment

Conclusions

- computer hardware continues to get faster
— no end in sight
- PC farms most cost-effective choice for Monte-Carlo calculations
- OpenMP on shared-memory systems best choice for most PDE applications
- MPI on distributed-memory systems viable for high-dimensional problems
- web/grid services becoming important for distributed computing and interoperability between Windows and Unix