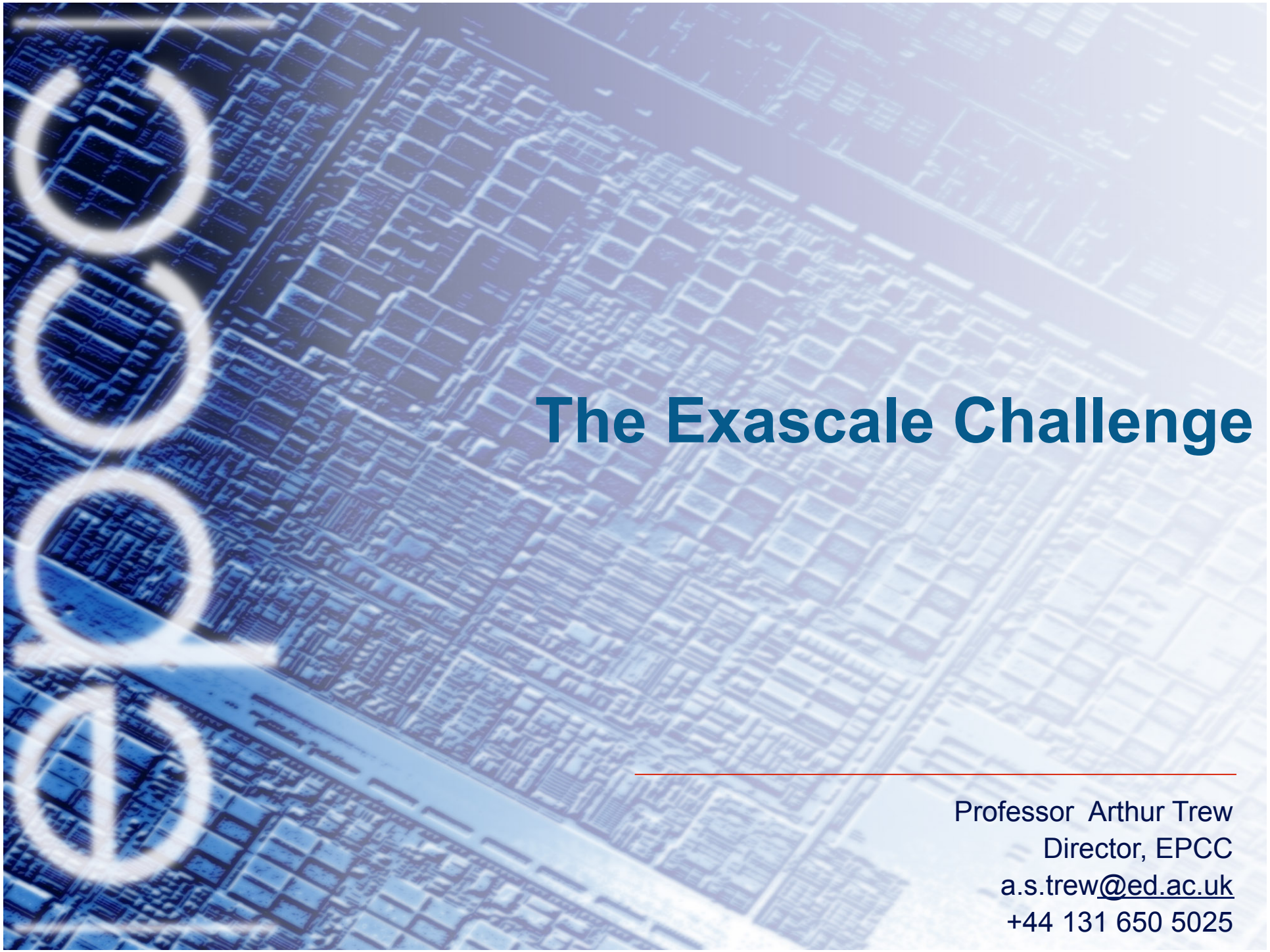


The Exascale Challenge

Professor Arthur Trew
Director, EPCC
a.s.trew@ed.ac.uk
+44 131 650 5025

parallel computing today

- the programming model is one of a set distinct memories distributed over homogeneous microprocessors
  - each microprocessor runs a Unix-like OS
- data transfers between the processors are managed explicitly by the application
- almost all programs are written in sequential Fortran or C
- they use MPI (Message Passing Interface) for data transfers between nodes/microprocessors
- some applications which exploit parallel threads on each microprocessor use a hybrid model
  - shared memory on the microprocessor, distributed memory outwith
  - this holds promise for many applications, but is still rare

parallel computing today

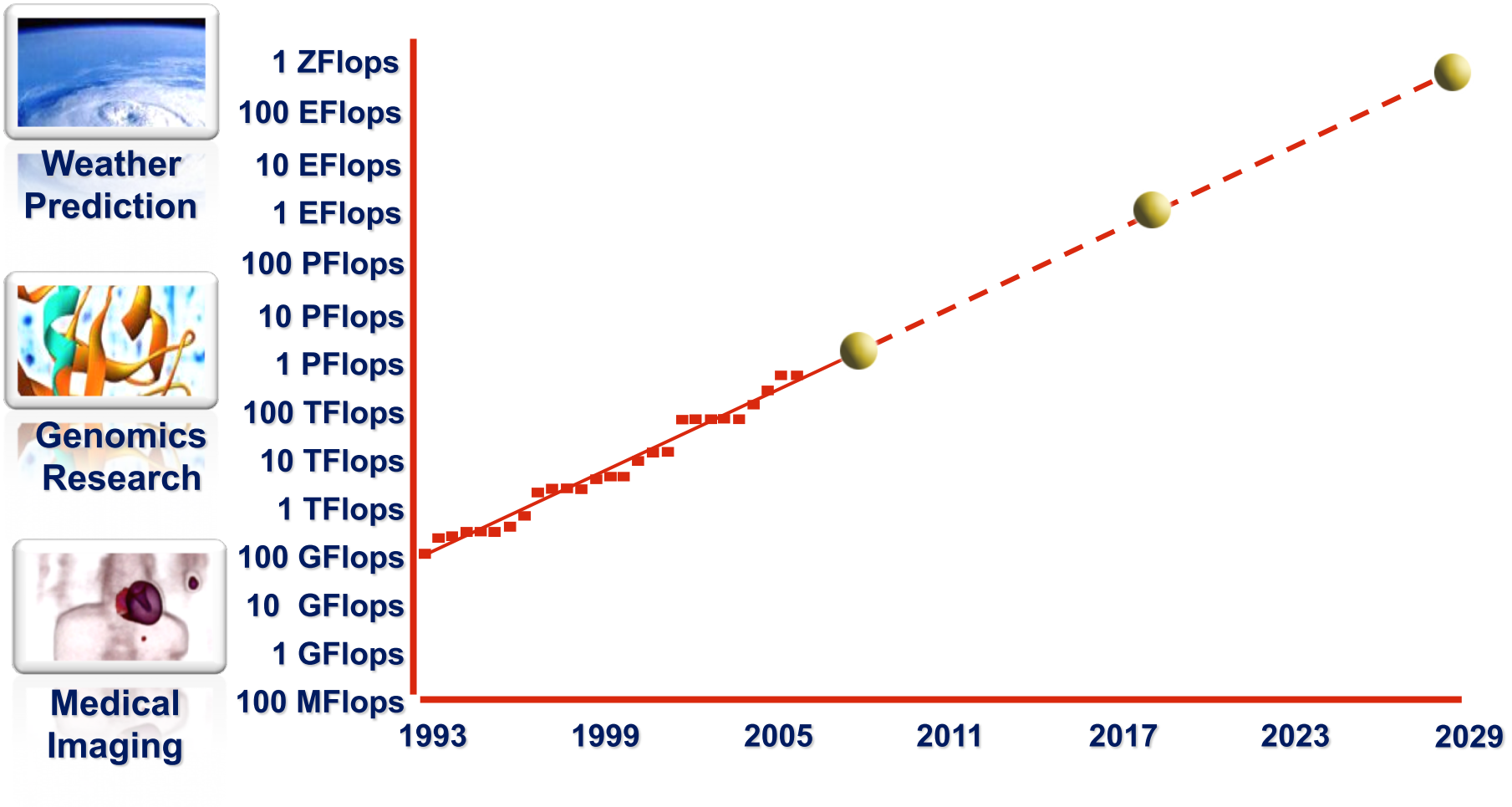
- (like the OS) few mathematical algorithms have been designed with parallelism in mind
- … the parallelism is then “just a matter of implementation”
- this approach generates much duplication of effort as components are custom-build for each application
- … but the years of development and debugging inhibits change and users are reluctant to risk a reduction in scientific output while rewriting takes place
- we may be close to a “tipping point”
  - without fundamental algorithmic changes progress in many areas will be limited

justifying the exaflops

- today, the majority of codes won't scale to a teraflops (10^{12} flops), so why bother with the exaflops (10^{18} flops)?
- there is an applications demand
- achieving it will require us to have radically new hardware and software designs
  - *"clear and widely recognised inadequacy of the current HPC software infrastructure in all component areas for supporting … escalation"*[1]
- hence there are challenges for
  - engineers for new designs for hardware and networks
  - computer scientists for compliers, software engineering, autonomic computing …
  - numerical analysts for new highly-scalable algorithms

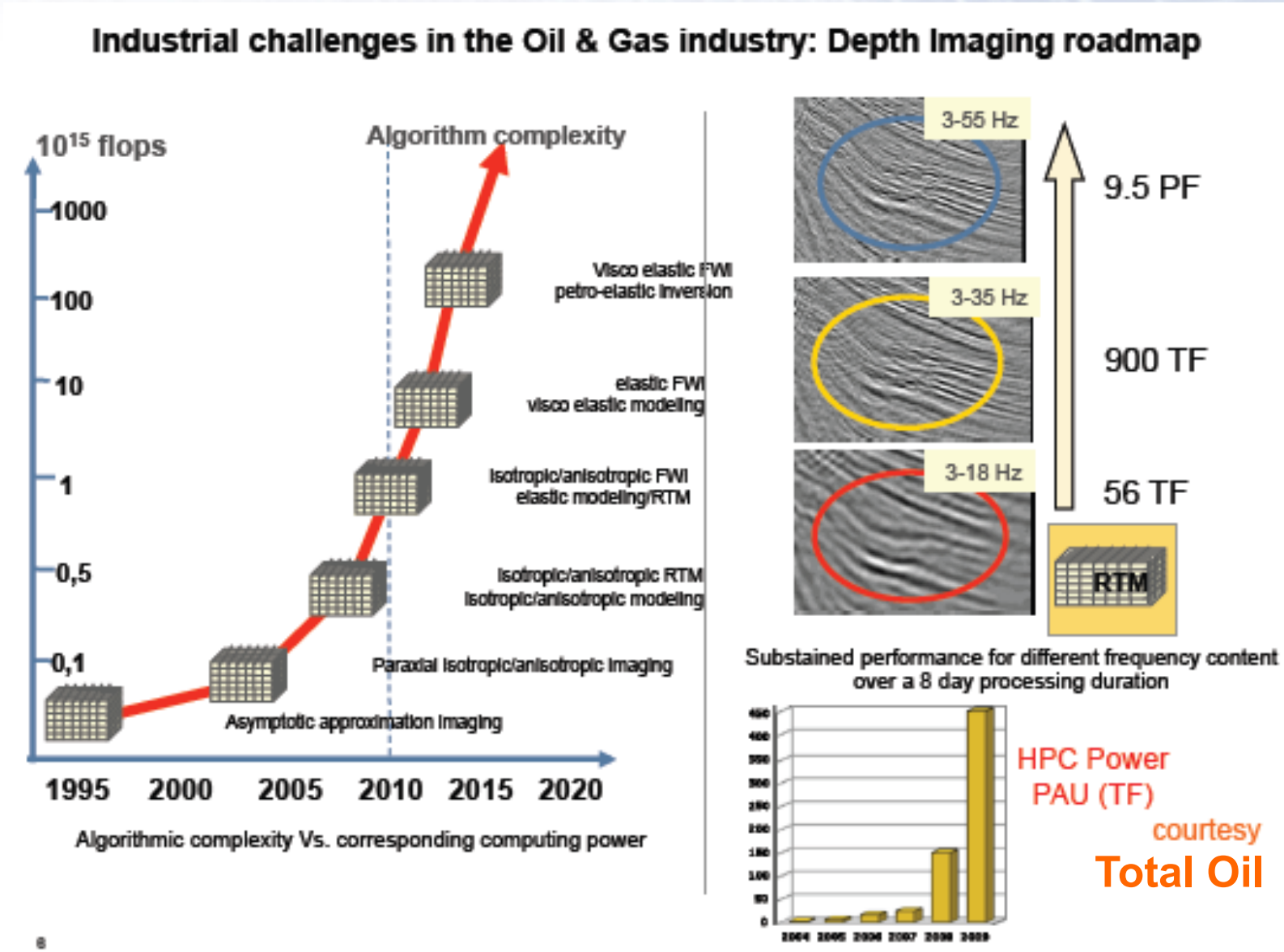
1: International Exascale Software Project Roadmap http://www.exascale.org/iesp/IESP:Documents

the need for speed

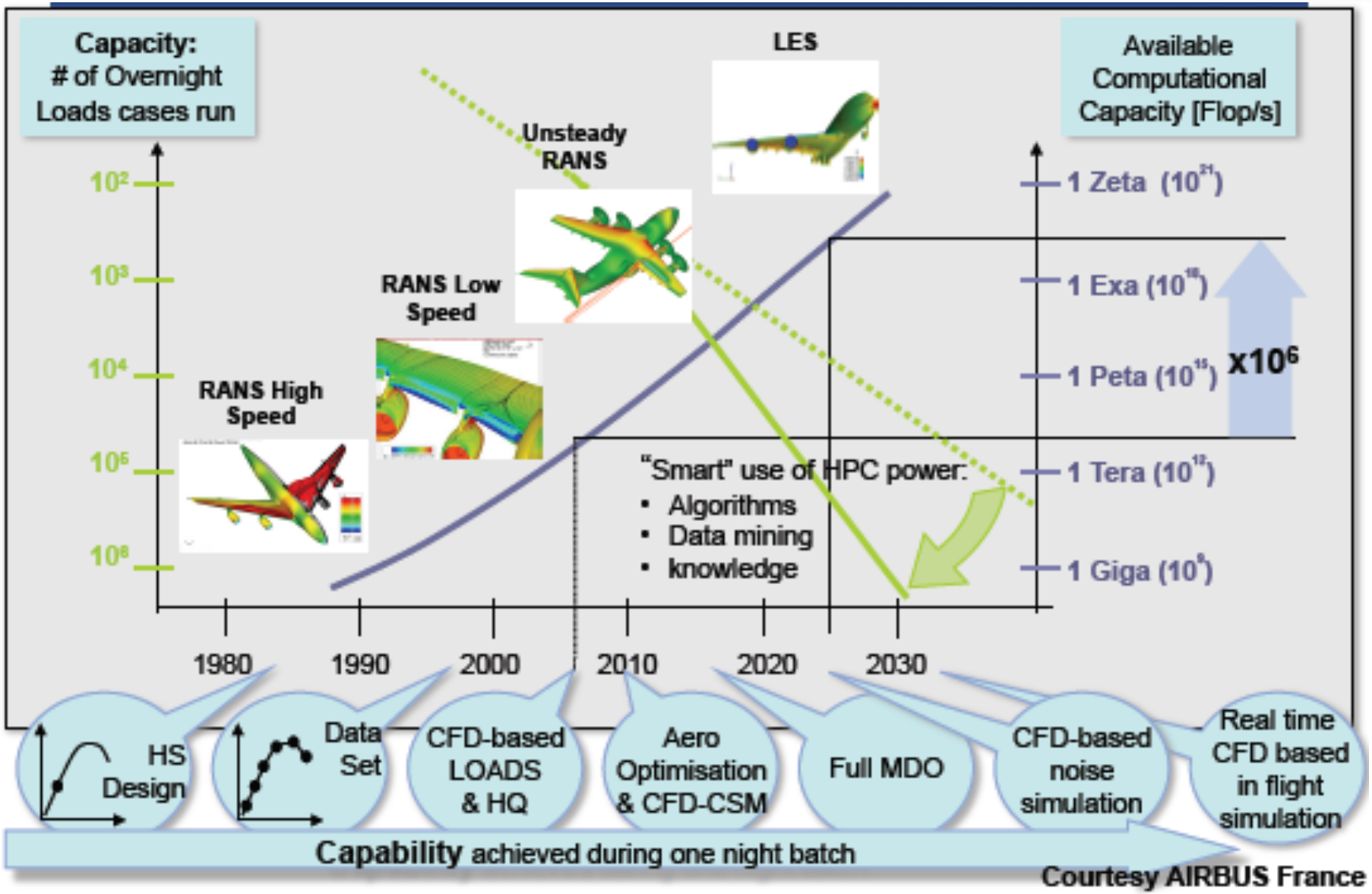


thanks to Intel Corporation

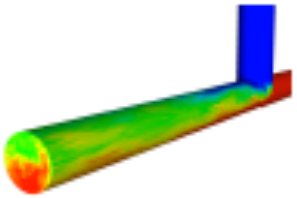
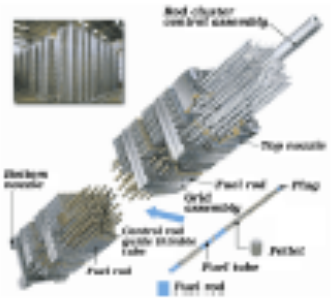
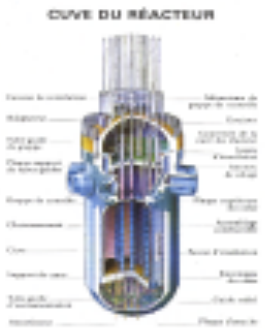
… in oil exploration



… in aircraft design



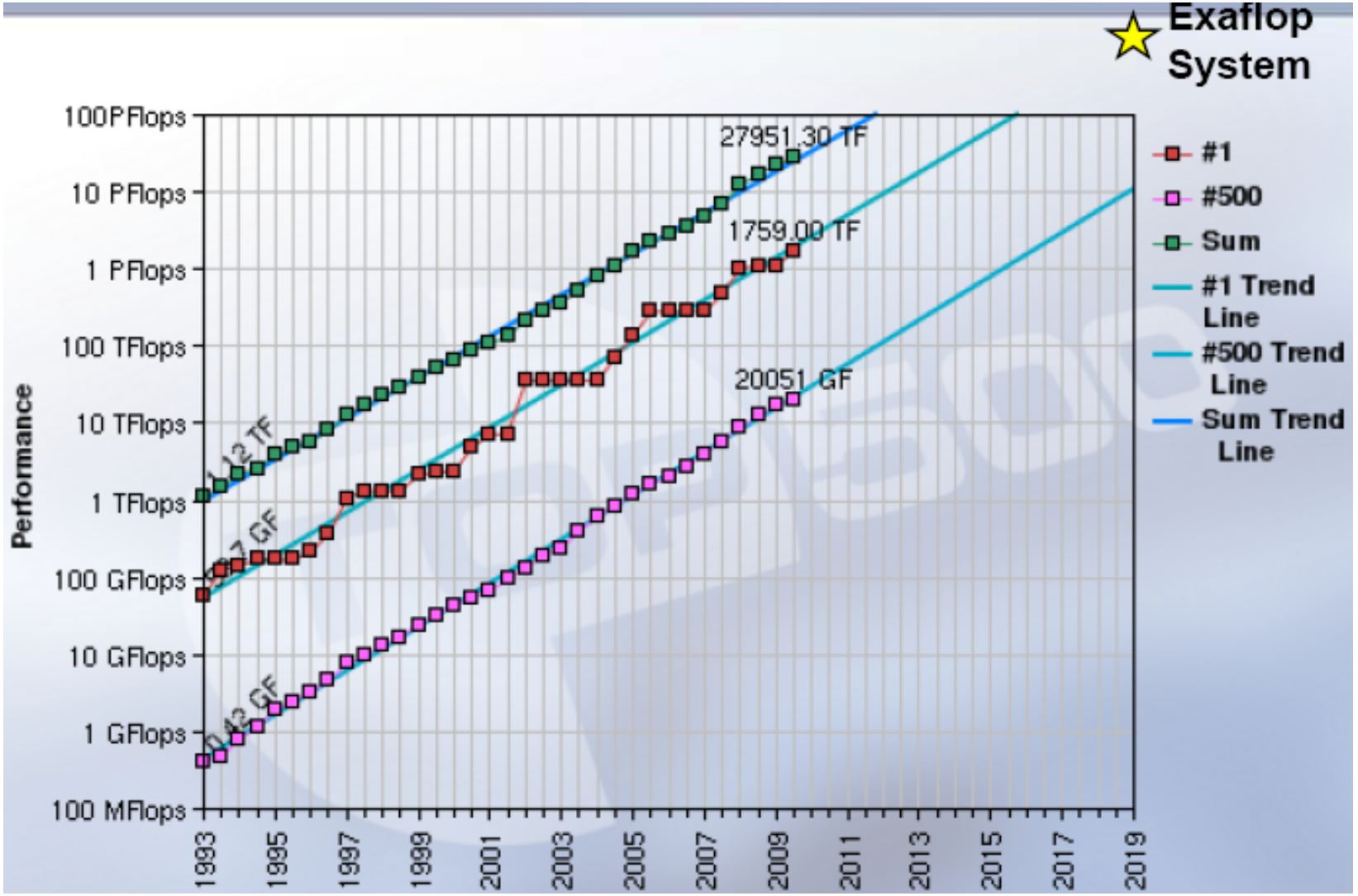
… nuclear reactor design

2003	2006	2007	2010	2015
Consecutive thermal fatigue event Computations enable to better understand the wall thermal loading in an injection. Knowing the root causes of the event ⇒ define a new design to avoid this problem.	 Computation with an L.E.S. approach for turbulent modelling Refined mesh near the wall.	 Part of a fuel assembly 3 grid assemblies	9 fuel assemblies No experimental approach up to now Will enable the study of side effects implied by the flow around neighbour fuel assemblies. Better understanding of vibration phenomena and wear-out of the rods.	The whole vessel reactor 

Computations with smaller and smaller scales in larger and larger geometries
⇒ a better understanding of physical phenomena ⇒ a more effective help for decision making
⇒ A better optimisation of the production (margin benefits)

10^8 cells 3.10^{13} operations	10^7 cells 6.10^{14} operations	10^8 cells 10^{16} operations	10^9 cells 3.10^{17} operations	10^{10} cells 5.10^{18} operations
Fujistu VPP 5000 1 of 4 vector processors 2 month length computation	Cluster, IBM PowerS 400 processors 9 days	IBM Blue Gene/L 20 Tflops during 1 month	600 Tflops during 1 month	10 Pflops during 1 month
# 1 Gb of storage 2 Gb of memory	# 15 Gb of storage 25 Gb of memory	# 200 Gb of storage 250 Gb of memory	# 1 Tb of storage 2,5 Tb of memory	# 10 Tb of storage 25 Tb of memory
Power of the computer	Pre-processing not parallelized	Pre-processing not parallelized Mesh generation IESP/Application Subgroup	… Ibid. … … Ibid. … Scalability / Solver	… Ibid. … … Ibid. … … Ibid. … Visualisation

shooting for an exaflops



thanks to top500.org

what are the challenges?

- DARPA conducted a study on Exascale hardware in 2007[1]
- Objective: understand the course of mainstream technology and determine the primary challenges to reaching 1EFlops by 2015, or soon thereafter
- they concluded the four key challenges were:
  I. power consumption
  II. memory and storage
  III. application scalability
  IV. resiliency
- … to which I would add:
  V. validation

1: http://www.darpa.mil/ipto/personnel/docs/ExaScale_Study_Initial.pdf

I: the power problem

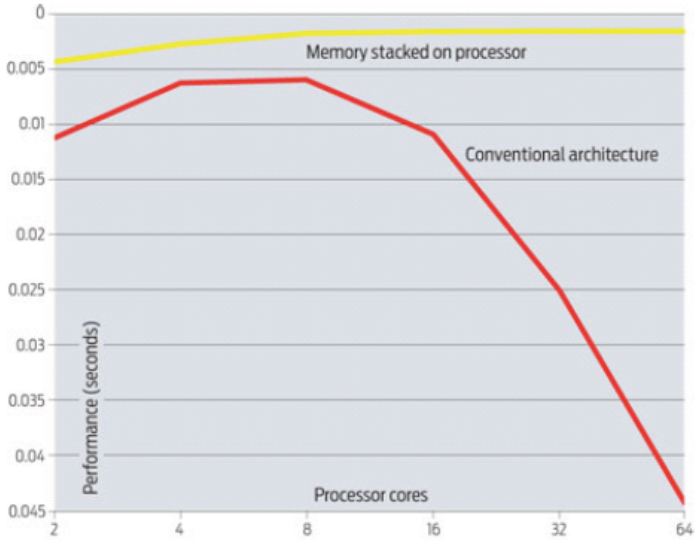
- the most power-efficient microprocessors available today deliver ~450 Mflops/W on Linpack
  - ie ~2.2 MW per Pflops … or 2.2GW per Eflops
  - excluding cooling which adds 20-100% to the power draw

Longannet: 2.4 GW

- … clearly, we have to do better!
  - DARPA goal: 50 Gflops/W in 8 years
  - 100x improvement

I: how do we reduce power consumption?

- the simplest way is to reduce the clock rate
  - the power consumption of a microprocessor depends on many factors
  - … empirically, the power consumption α v^3
  - a 20% drop in clock rate gives an 50% reduction in power[1]
- however, lowering the clock reduces the speed
  - and, hence, increases the number of cores required
  - bad news for HPC
  - especially if you want to use data![2]
- recently, we upgraded HECToR: dual-core 2.3 GHz -> quad-core 2.0 GHz
  - one application **reduced** its performance by 1.7x



1: http://spectrum.ieee.org/computing/hardware/why-cpu-frequency-stalled
2: http://spectrum.ieee.org/computing/hardware/multicore-is-bad-news-for-supercomputers

I: microprocessor architectures

- conventional microprocessor architectures are optimised for single thread performance, rather than energy efficiency
  - fast clock rate with latency (performance)-optimised memory
  - heavy use of speculative execution => large structures supporting various types of predictions
  - relatively little energy spent on actual ALU operations
- could be much more energy efficient with multiple slow, simple processors exploiting vector/SIMD

I: microprocessors not the only problem

- which takes more?
  - performing a 64-bit FMA

```
                  893,500.288914668
x                 43.90230564772498
= 39,226,772.78026233027699
+                  2.02789331400154
= 39,226,724.80815564
```

- or, moving the three operands 20mm across the die?
- moving the data uses 3x the energy

- loading the data from off-chip takes >10x more yet
  - flops are cheap, communications are expensive
  - exploiting data locality is critical

II: memory and power

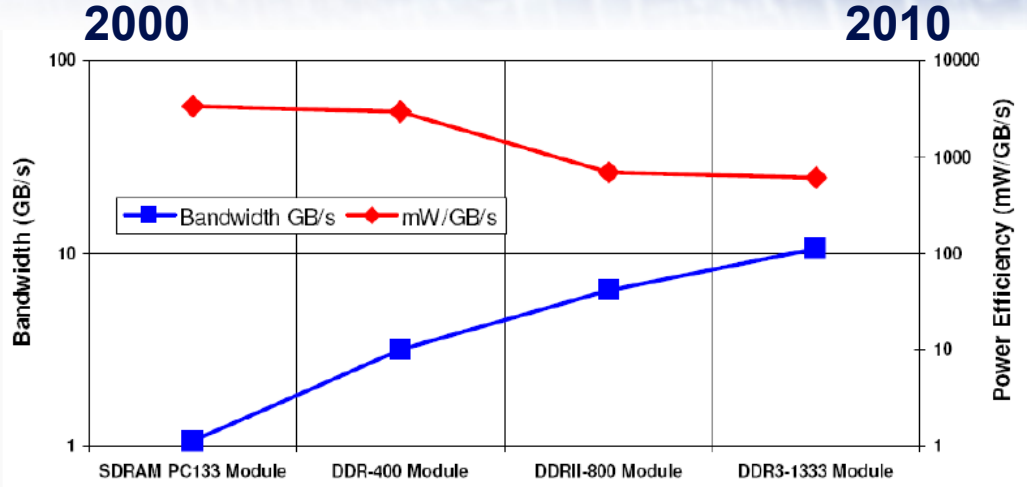
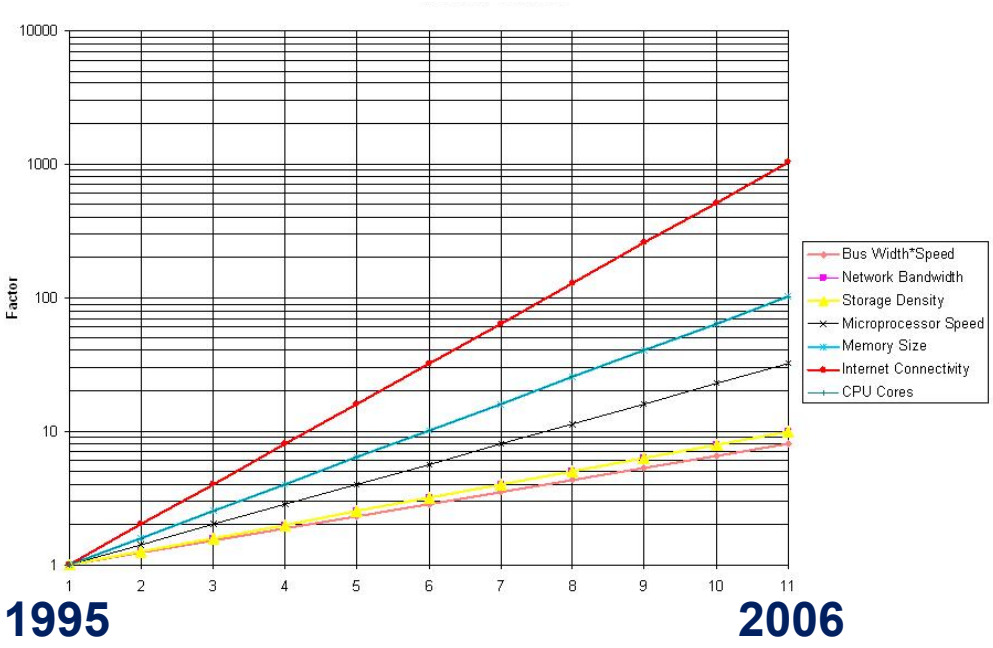


Figure 6.22: Commodity DRAM module power efficiency as a function of bandwidth.

- memory bandwidth has increased *~10x* over the past decade
- the energy cost/bit transferred has declined by *2.5x*
- … the energy cost of driving the memory at full bandwidth has risen *4x*
- memory DIMMs can't provide bandwidth at acceptable energy costs

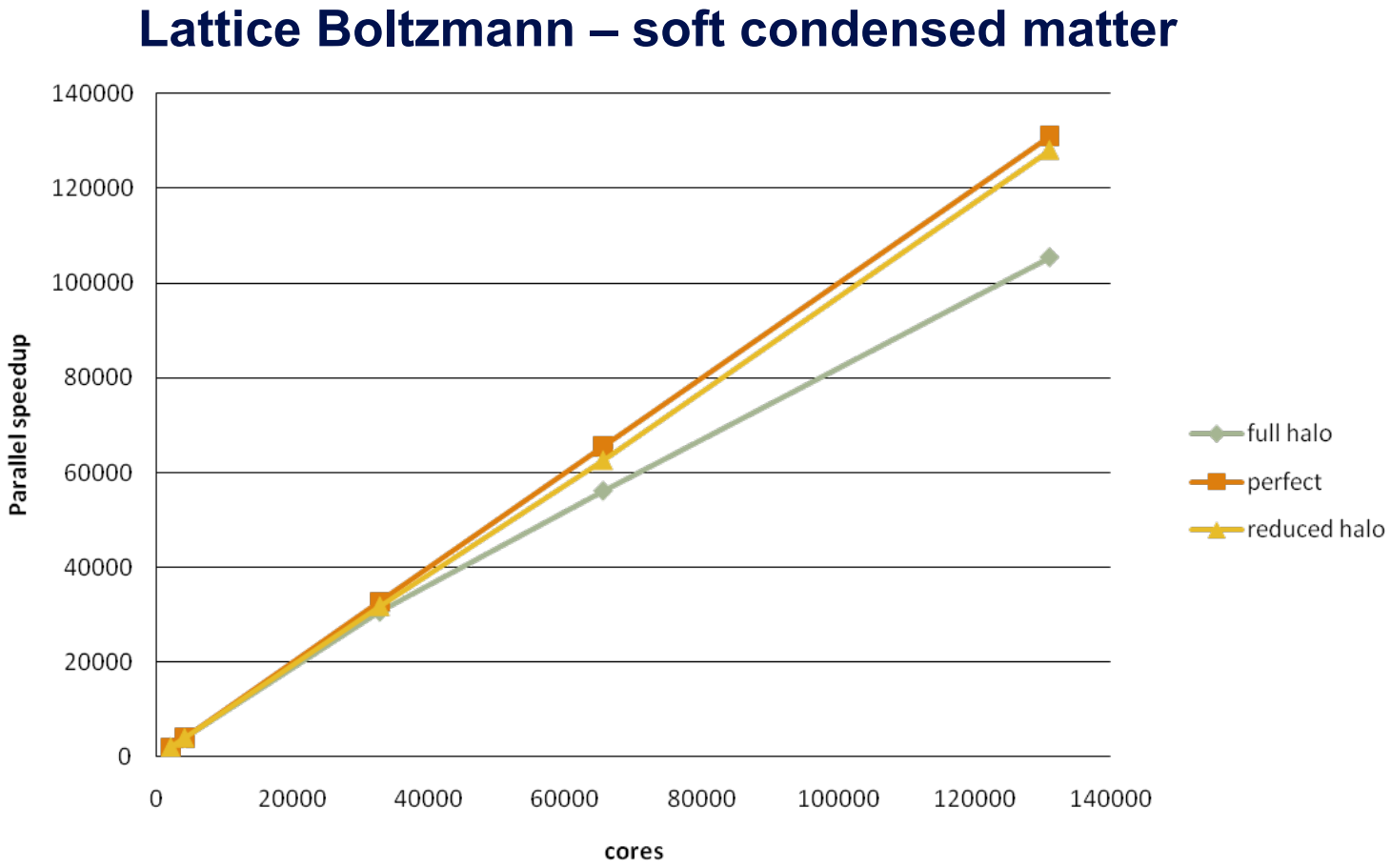
II: memory performance

- over the past 30 years DRAM density has increased ~*75x* faster than bandwidth
- … memory bandwidth is the limiting factor in future designs
- novel memory technologies needed :
  - phase-change memory, holographic memory, graphene …

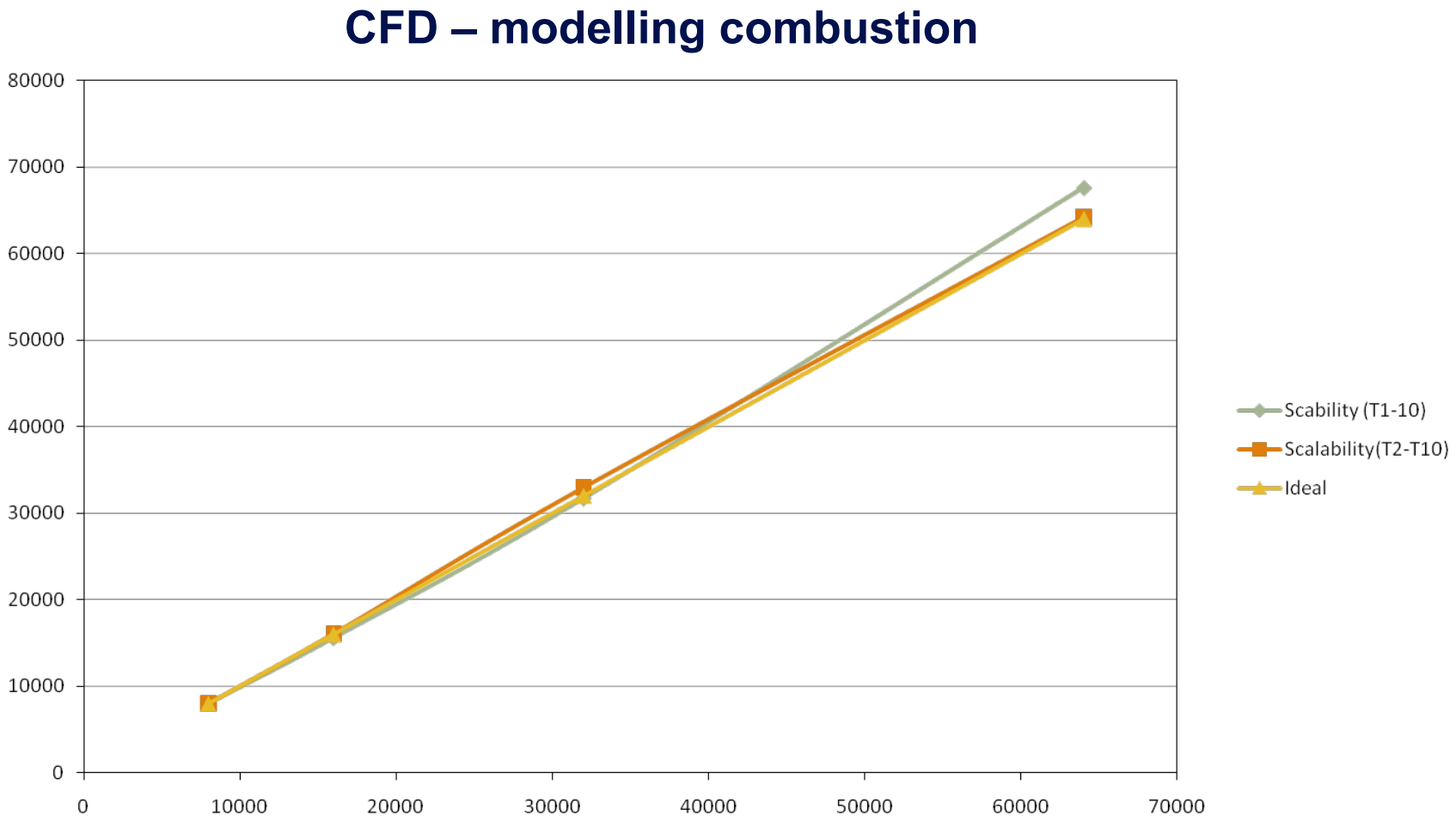


III: applications scalability

- those codes with low communications overheads and which can exploit weak scaling do well:

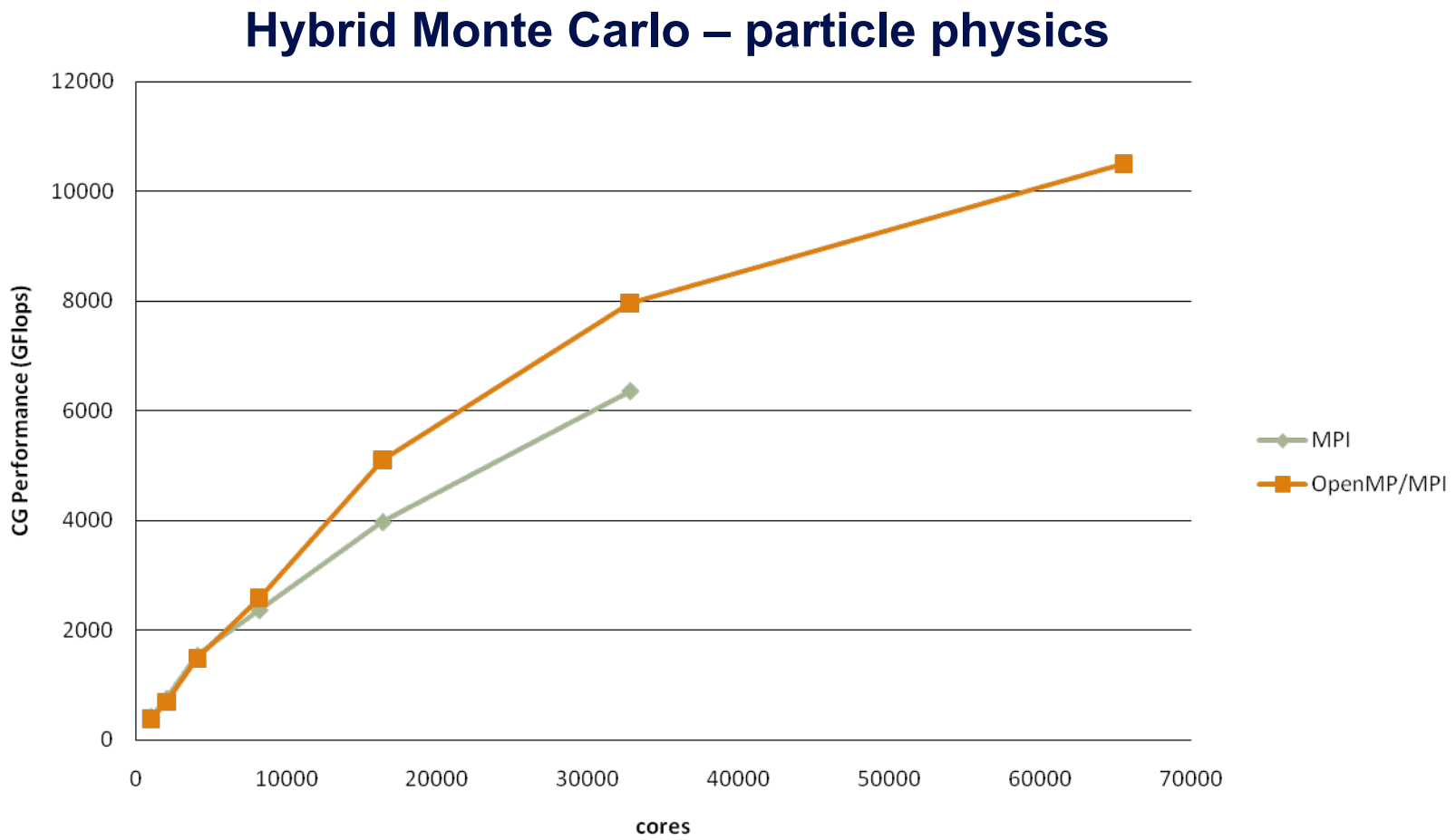


III: applications scalability …



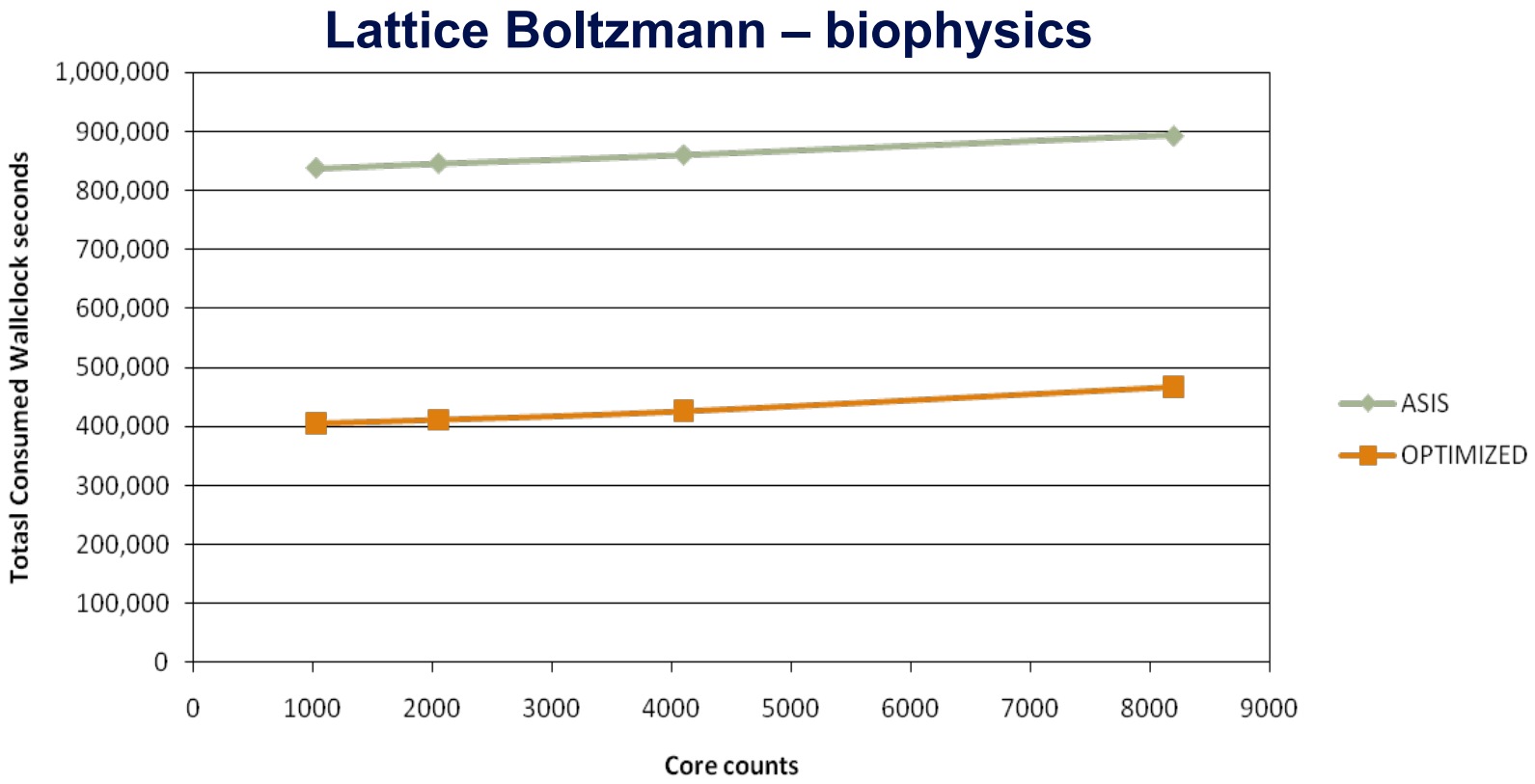
III: applications scalability …

- … some do pretty well



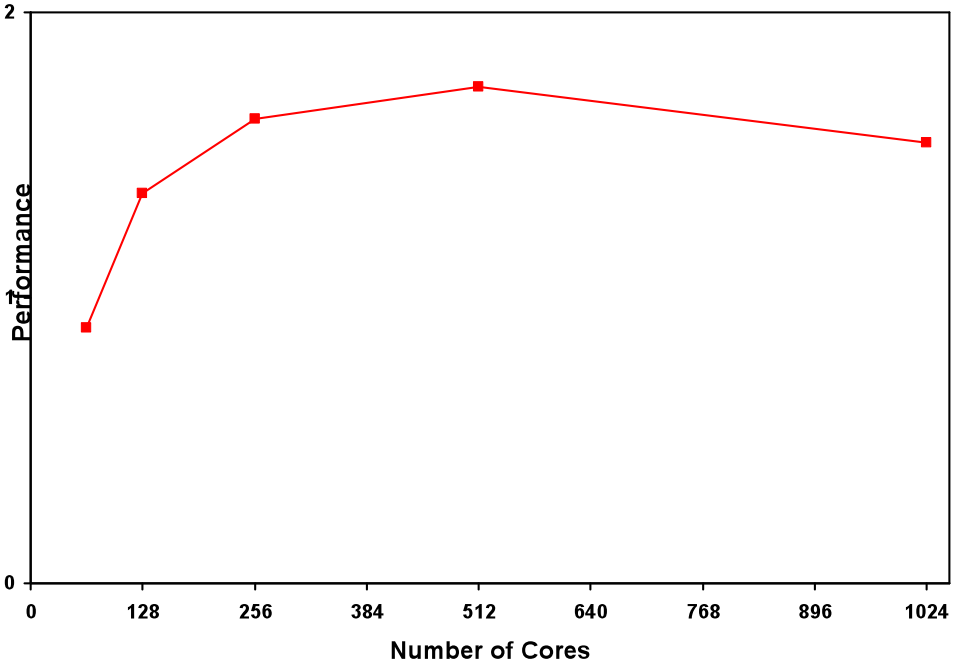
III: applications scalability … again

- … but most are disappointing
  - this behaviour is caused by the overheads of global communications
  - applications only when communications are highly infrequent, or local



III: alarming applications scalability …

Density Functional Theory – Physical Chemistry



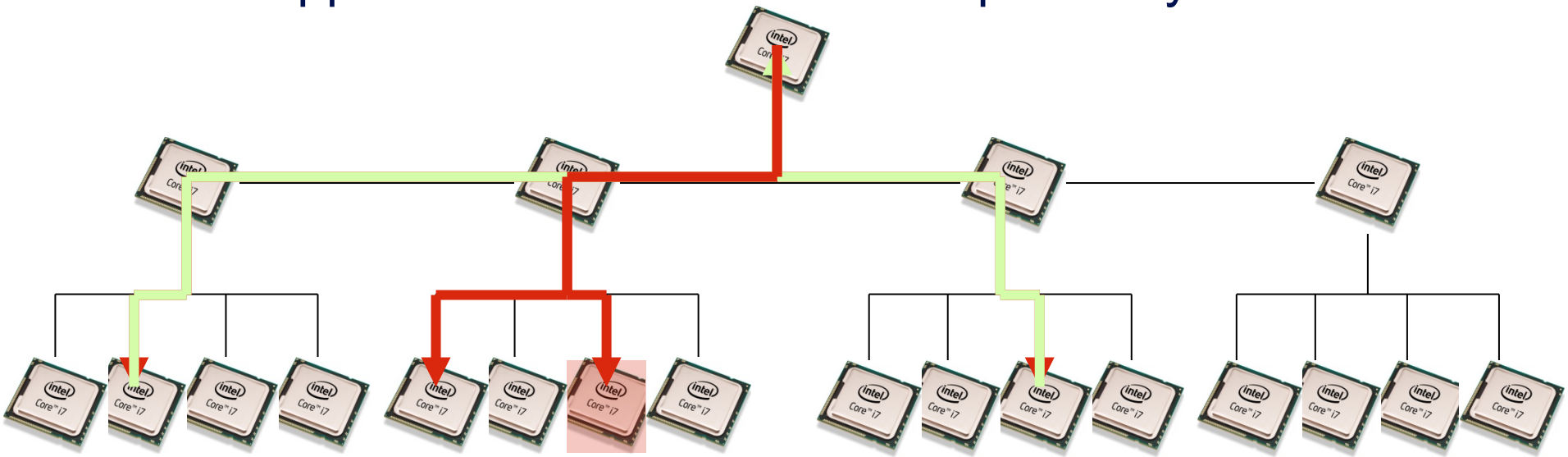
- users, especially in chemistry and engineering, are locked-in to poorly-scaling third-party codes
- summary: widespread need for good software engineering and parallel techniques

IV: resiliency

- an Eflops machine is likely to have ~10^6 processors
- if each processor had a lifetime of 10 years (unlikely)
- … then the machine will have a MTBF of ~5 minutes!
- we therefore have to be able to operate it in a way which is resilient to single-node failures
- unfortunately, most scientific applications use synchronous algorithms
- … which would halt when something blocks the data flows
- fault tolerance is not a new problem
  - von Neumann considered this is detail as early computers failed often

IV: fault-tolerant computing

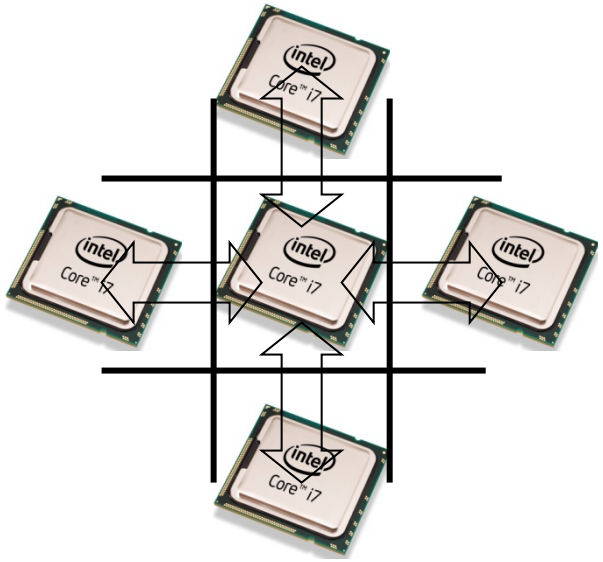
- … is common in many high-throughput applications
  - Google, Amazon's Availability Zones …
- here, the focus is to maximize overall throughput, not to minimize the execution time of every individual job
- these applications have elaborate supervisory structures



- why not transplant these approaches to HPC?

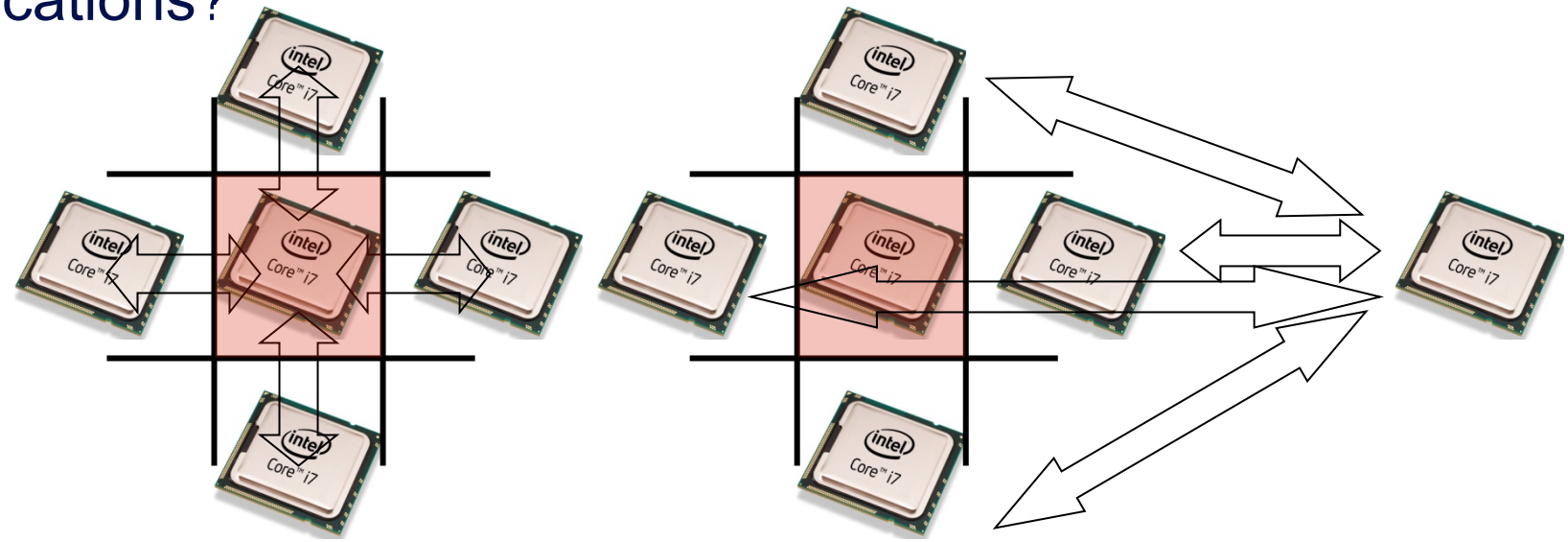
IV: fault-tolerant HPC

- this approach is directly applicable to HPC where the problem can be decomposed as a task farm
  - eg. DNA sequence analysis, LHC simulations, SETI@home …
- however, this is a (small) subset of applications
- most require tight coupling between processes
  - data must flow between worker processes and not just between a master and a pool of workers



IV: fault-tolerant HPC

- what happens when a processor fails in such synchronous applications?



- now, neighbouring processes don't have to run on neighbouring processors (though it is faster if they do)
- so, we can reserve processors to substitute for failed ones
  - fault-tolerant MPI provides a framework to achieve this

IV: fault-tolerant HPC

- … so, the problem is solved?
- No
- while it may be possible to reconnect the processors in a new configuration to exclude failed components, how do we reconstruct the state of the failed processor?
- we could checkpoint each processor's state to a neighbour and then transfer this to the spare, when required
- … however, this will be computationally expensive
  - memory/core is decreasing
  - memory and network bandwidths already limit performance
- most codes use checkpoint/restart
  - crude and unscalable to exascale

V: validation

- if the application does not mimic reality then there is no point
- there are many levels at which errors can creep in:
  - hardware unrepeatability
  - inappropriate choice of algorithm
  - wrong coding

hardware errors

- of these, hardware problems may be surprising, but:
  - 1994: Pentium divide error
    - Intel: “1 in 9 billion divides wrong”
    - at this rate an Eflops machine would make ~10^8 mistakes s^{-1}
  - 1991: Meiko i860
    - race conditions produce errors which are scientifically significant
    - run every simulation three times, if two agree, accept
  - 2003: QCDOC (Bluegene prototype)
    - need to reduce clock rate to prevent race conditions
  - 2008: Cray XT4
    - undiagnosed network problems give lack of reproducibility
    - example of “silent errors” which are all too prevalent[1]
      - these are different from “soft errors” because they can persist

1: Cappello et al 2009 Int J HPC Applns, **23**, 374

hardware(?) errors

- in these last 3 cases the errors were only uncovered by a particularly diligent user group
  - "normal" users would never have noticed
- understanding how to improve matters requires us to understand where the problems originate
  - little consensus, different studies have suggested different sources
  - but most likely that most problems originate in the system software
- but, most applications are very sensitive to a single soft error[1]
- fault oblivious, "self-stabilisation" algorithms have been investigated for many years[2]

1: Lu and Reed 2004 Proc 2004 ACM/IEEE Conf of Supercomputing
2: Dijkstra 1974 Commun, ACM **17**(11), 643

self-healing machines

- self-stabilisation requires that *all* software used in the program's execution is fault-tolerant
  - not just the application and numerical algorithm
- … so, a lot of work
- moreover:
  - such algorithms have only been investigated in basic distributed system operations
  - the duration of the stabilisation phase is unknown
  - … and, errors during the stabilisation phase restart the clock
- thus, it's not obvious how to have self-stabilising numerical algorithms
- … but many aspects of the runtime environment could make use of this approach

Algorithm vs. Implementation

$$\sum f(x)$$

```
r = 0
do i=1, n
r = r + f(i)
end do
```

1234567.4440
1234567.4444
1234567.4448

3703702.4000

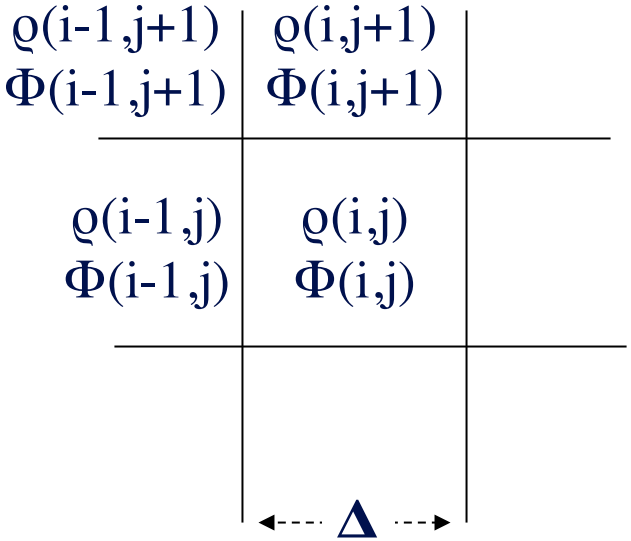
1234567.4448
1234567.4444
1234567.4440

3703702.5000

- … and the correct answer is ….. neither
  - 3703702.3332

algorithmic choice

- discretising a continuous system on to a grid necessarily introduces errors
- … the algorithm must be chosen to ensure that these do not propagate excessively:
- eg. Poisson's Equation ($\nabla^2\Phi = \rho$)
  - –we wish to solve this on some surface with some boundary conditions



$$\frac{\partial\Phi(i,j)}{\partial x} = \frac{\Phi(i+1,j)-\Phi(i,j)}{\Delta}$$ Forwards

$$\frac{\partial\Phi(i,j)}{\partial x} = \frac{\Phi(i+1,j)-\Phi(i-1,j)}{2\Delta}$$ Centred

$$\frac{\partial\Phi}{\partial x} = \frac{\partial\Phi(i,j)}{\partial x} - \frac{\Delta}{2!}\frac{\partial^2\Phi}{\partial x^2}$$ Forwards

$$\frac{\partial\Phi}{\partial x} = \frac{\partial\Phi(i,j)}{\partial x} - \frac{\Delta^2}{3!}\frac{\partial^3\Phi}{\partial x^3}$$ Centred

- the problem worsens for higher differentials: $O(\frac{1}{\Delta})$ *vs* $O(\Delta^2)$ for ∇^2

initial value problems

- … not going to go through the algebra but IVP problems require stability, eg Diffusion Equation:

$$D\nabla^2\Phi = \frac{\partial\Phi}{\partial t}$$

- this is a parabolic PDE
- if we calculate $\Phi_{t+1}(x)$ from $\Phi_t(x)$ (FTCS in 1D)

$$\Phi_{t+1}(x) = \Phi_t(x) + \frac{D\Delta t}{\Delta x^2}\left\{\Phi_t(x+\Delta x) + \Phi_t(x-\Delta x) - 2\Phi_t(x)\right\}$$

- but, this scheme is unstable unless $\Delta t < \Delta x^2/2D$
  - … and with this condition it is computationally very expensive
- we can remedy that by using an *implicit* integration scheme
  - here, $\Phi_{t+1}(x)$ is calculated using $\Phi_{t+1}(x+\Delta x)$ etc at the advanced time

$$\Phi_{t+1}(x) = \Phi_t(x) + \frac{D\Delta t}{\Delta x^2}\left\{\Phi_{t+1}(x+\Delta x) + \Phi_{t+1}(x-\Delta x) - 2\Phi_{t+1}(x)\right\}$$

initial value problems

- unfortunately, this is only accurate to $O(\Delta t)$
- … so we have to use Crank Nicholson
  - an average of implicit and explicit, accurate to $O(\Delta t^2)$
- but we have to use a different integration scheme for, say, the Wave Equation:

$$v^2 \nabla^2 \Phi = \frac{\partial^2 \Phi}{\partial t^2}$$

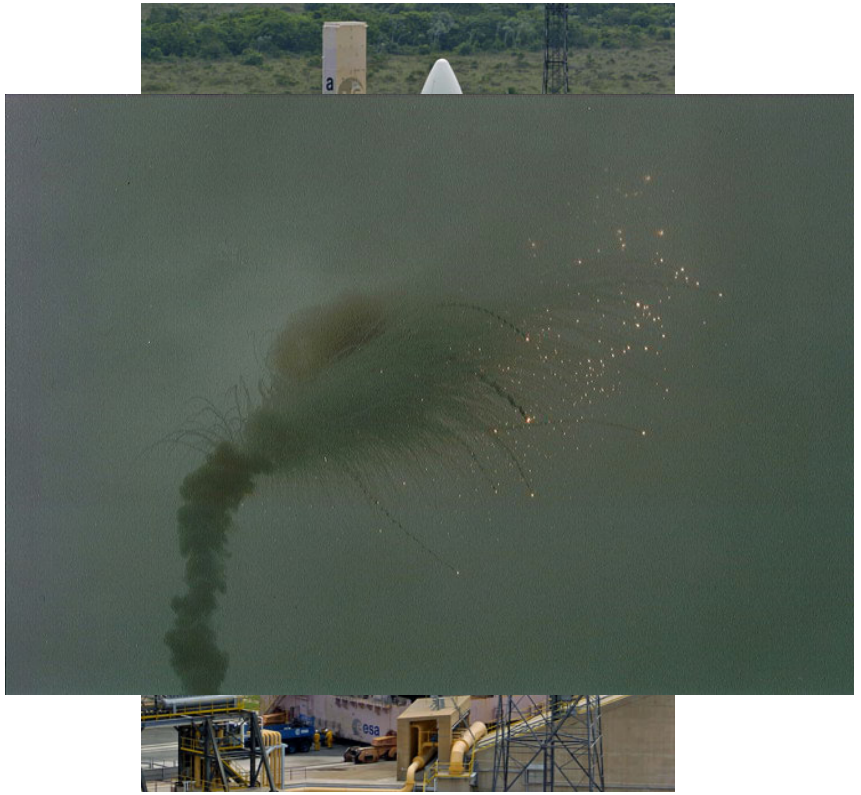
- - this is a hyperbolic PDE
- … so, choice of algorithm is not necessarily straightforward
  - especially if you are trying to simulate a phase change, as the character of the pde can change
- nor is timestep necessarily constant throughout a simulation

wrong coding

- making mistakes is only human
- finding, and correcting, them requires a process
- … unfortunately few academic software developers understand the software development process
  - this is an area steeped in mystery
- but, fortunately, academic software developers aren't likely to kill anyone through their mistakes

Bad Software

- Ariane 5 Explosion
- Code from Ariane 4 re-used
- Faster engines in Ariane 5 triggered a bug which caused buffer overflows
- Oops!!
  - No comprehensive testing of old code in the new platform
- Result – A very big bang



Bad Software … cont

- Therac-25: Medical Linac
  - two modes of operation: “Electron” (low power) and “X-ray” (high power)
- early example of concurrent programming
- only partial understanding of the need for control of inter-thread communications
- eg.
  - user entered “X” by mistake
  - quickly corrected sequence, entering “E”
  - ran sequence
    - one thread controlled the output power
    - another controlled the collimator
  - mis-prioritisation permitted the high power setting to run without the collimator plate in place
- several deaths occurred

formal verification

- … provides a method for rigourous verification of correctness as an alternative to ad hoc testing
  - formal specification methods can show critical interactions between program components
- however, for scientific applications the applications for formal testing within the component is limited
  - they use floating point numbers => requires us to know what tolerance is important
  - … hence has been the responsibility of the application and currently few such tests are made
  - at exascale the volumes of data increase and the practicality of even this is unclear
- in parallel we need to validate results because of the high probability of soft errors