The Exascale Solution?
the exascale challenges – a summary

I. power consumption
II. memory and storage
III. application scalability
IV. resiliency
IX. validation

• all of these require computer science inputs
  – some need new hardware designs
  – some require extended CS models
  – some fundamentally new CS theories and methods

• all also need collaboration with:
  – engineers for hardware designs
  – computational scientists to understand implementation constraints
  – mathematicians on algorithms

• the intellectual challenges are truly exciting
**International Exascale Software Project**

- IESP is a CS-orientated research project investigating how to build an exascale computer\(^1\)
- although international, it is dominated by the US
  - and plays most strongly to an economy that has both the applications need for such computers and the ability to build them

**IESI is a European response**

- G8 countries make their first ever research call – exascale computing
  - it’s just a pity that their budget does not match by their aspirations!

\(^1\) [http://www.exascale.org/mediawiki/images/a/a1/iesp-roadmap-draft-0.93-complete.pdf](http://www.exascale.org/mediawiki/images/a/a1/iesp-roadmap-draft-0.93-complete.pdf)
IESP recommendations

- IESP strongly advocates a co-design model
  - the software and hardware are developed in parallel

- backed by aspiration pull and technology push
  - global challenges make the case … but the codes are too immature
  - technology push is not enough
    - politically the cost is too high, too few companies will benefit
    - technically, there are many potential hardware routes … and many likely dead-ends

- co-design vehicles
  - applications which are scientifically sound with the potential to scale
  - provide development paths
  - … while global challenge codes develop in parallel
answering the programmers’ prayer

• in IESP’s model the hardware will support a systems stack: 
  \textit{X-stack}

• X-stack will:
  
  – support concurrent programming models, applications and tools
  
  – provide software and tools will manage power directly
  
  – provide resilient software
  
  – address changes to heterogeneous nodes
  
  – solve parallel I/O bottleneck

• unfortunately, it doesn’t say how these will be done!

• … IESP advocates CS research in:
outstanding research priorities

• systems software
  – operating systems: fault-tolerance, collective OS services, power management, hierarchy management …
  – runtime systems: heterogeneity, load balancing, fault-tolerance, dynamical resource management …
  – I/O systems: integration of emerging storage devices, embed I/O into programming models …
  – systems management: resource control & scheduling, security, integration and test …
  – external environments: linking to remote resources …

• development environments
  – programming models: support for heterogeneous nodes, HPC interoperability, fault-tolerant MPI …
  – frameworks: data layouts, fault resilience, inter-component coupling …
  – compilers: MPI-aware compilers, compiler support for hybrid programming, power-aware compilers …
  – numerical libraries: asynchronous algorithms, architectural transparency, power-aware …
  – debugging tools: categorical assimilation, support for node heterogeneity, scalability …

• applications
  – algorithms: intra/inter-node scaling, fault resilience, heterogeneity, strong scaling …
  – data analysis and visualisation: integration with simulation, workflows, data extraction …
  – data management: scalable data-mining, new database technologies, search & query tools …

• crosscutting activities
  – resilience: techniques for saving/restoring state, MPI replacement, fault-oblivious software …
  – power management: node-level OS management, power-aware libraries etc …
  – performance optimization: heterogeneity, hybrid programming, enhanced concurrency …
  – programmability: new programming models, new runtime models, new compiler support …
• today most HPC facilities are homogeneous
  – perhaps with specialised processors for peripheral functions, eg I/O
• even if the nodes are compound, the components are separate with separate programming models
  – eg. microprocessors with attached FPGA
• microprocessors will increasingly be built from disparate components: “normal” core, GPGPU, SIMD Array
  – with a mix which may vary within a machine
• … somehow, that mix will have to be controlled to give optimal performance
• beyond these general statements about heterogeneity IESP did not suggest a hardware architecture
• … and hence what the software constraints would be
exascale architectures – a strawman

• DARPA also performed an exascale study focussed on the software issues\(^1\)

• to understand these they characterised the likely hardware, extrapolating from current roadmaps:
  – heavyweight strawman: based on commodity microprocessors + separate memory and routing chips (eg Cray XT4)
  – lightweight strawman: customised, low power microprocessor with integrated memory and routing (eg IBM BlueGene)

the evolutionary strawmen

• to see if they could make a *practical* exascale machine they then made a number of assumptions:
  – applications will demand same DRAM/flops ratio as today
  – processor die size will stay constant
  – Moore’s Law will drive core counts per processor, but clock rates will stabilize
  – flops/cycle will rise from 2 (today) to 8 (2015)
  – max power/rack will double every 3 years
  – number of racks may increase by 50/year

• are these reasonable?

• they are not conservative
  – for example, all of HECToR is 60 racks
• assume one of two possible system power models:
  – simplistic (aka highly optimistic model)
    – max power per die grows in line with ITRS projections
    – power/memory chip remains constant
    – power for routers remains constant –
      – even though we know that we need to increase bandwidth
    – true, if energy/bit moved/accessed decreases as fast as flops increases
  – fully scaled (aka pessimistic/realistic model)
    – as above, except memory and router power scales with flops
      – true, if energy/bit moved/accessed remains constant
heavyweight strawman

• 4x target … and we still have to add on cooling, peripherals etc
lightweight strawman

lightweight power costs

- ... no better than the heavyweight option overall, though more flops/kW
• whichever strawman model you choose, you still have to manage $\sim 10^9$ threads
the aggressive strawman

• they then tried the “clean sheet of paper” approach
  – aggressive strawman: microprocessors designed to maximise performance for minimal power

• system architecture:
  – 32 nm silicon technology, but with aggressive voltage scaling
<table>
<thead>
<tr>
<th>Characteristic</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Flops – peak (PF)</td>
<td>997</td>
</tr>
<tr>
<td>- microprocessors</td>
<td>223,872</td>
</tr>
<tr>
<td>- cores/microprocessor</td>
<td>742</td>
</tr>
<tr>
<td>Cache (TB)</td>
<td>37.2</td>
</tr>
<tr>
<td>DRAM (PB)</td>
<td>3.58</td>
</tr>
<tr>
<td>Total power (MW)</td>
<td>67.7</td>
</tr>
<tr>
<td>Memory Bandwidth (B/s per flops)</td>
<td>0.0025</td>
</tr>
<tr>
<td>Network bandwidth (B/s per flops)</td>
<td>0.0008</td>
</tr>
</tbody>
</table>
critical concerns

- the memory and network bandwidth/flops ~1% of current Pflops machines
  - this is already a limiting factor for most applications
    - ideally 3 word/flops; practically ~few Byte/flops; achieved ~0.3 Byte/flops
    - vital to be able to increase number of operations on each datum

- adaptively-balanced node
  - this *may* overcome this imbalance on an application-by-application basis
  - designed to enable power to be used *either* processing or moving data to memory or network
    - too little power to drive both the ALU and memory at 100% simultaneously
six ways not to do it

1. ignore Little’s Law
   - required concurrency = latency*bandwidth, or waste memory bandwidth

2. use processors engineered for serial applications
   - out-of-order execution, speculation, hardware-controlled caches … waste energy for dubious benefit

3. rely on weak scaling

4. synchronize all data communications
   - one-sided communications are more efficient

5. add global synchronization
   - not only does it introduce delays, but it is not fault-tolerant

6. use algorithm design to minimize flops
   - data movements dominate energy usage
• weak scaling (problem size α machine concurrency) has been the mainstay of parallelism for 30 years
• strong scaling (scaling with a fixed problem size) has been hard to find
• for some applications there is no more weak scaling because the system being studied is already large enough
  – eg classical MD for many chemistry applications only requires 100 -1000 molecules
• but a larger set is constrained by algorithmic complexity
algorithmic complexity

- consider a problem which scales as $O(n^2)$, running on $P$ processors

$$F_j = \sum_{i=1 \neq j}^{N} \frac{Gm_i m_j}{r_{ij}^2}$$

- if $n \to 2n$, the memory requirement doubles, but the computational load quadruples
- if we map this new load to $4xP$ processors then each processor has the same workload as before
  - but there are only 50% as many particles in each processor’s memory
  - so, the data exchange ratio rises
• only if we have an algorithm which scales linearly (or slower) can we maintain the compute/communications ratio
• … and these algorithms don’t exist
final thoughts

- the aggressive strawman is unlikely to be based on a technology which is of interest to anyone outside HPC
  - so, who will build it?
  - is this a return to the path of custom-built machines?

- more generally, if Moore’s Law requires ever more parallelism where is the demand from “normal users”
  - drivers for renewal may be increased memory, improved graphical performance …
  - but unless we solve the parallel working problem soon, the industry faces potentially fatal challenges
  - can you design the “killer app”?