Software Barriers for HPC

Moderator

Pete Beckman

Presenters

Al Gara

Jean-Yves Berthou

Mitsuhisa Sato

Peggy Williams

Vivek Sarkar

Ann Trefethen

Software Barriers for HPC

Moderator

Pete Beckman

Presenters

Al Gara

Jean-Yves Berthou

Mitsuhisa Sato

Peggy Williams

Vivek Sarkar

Ann Trefethen

Evolutionary Software Areas for Exascale:

• Extend current program models through single node threading of messaging. Eliminate “per task” scaling terms in messaging layer to allow for higher “flat scaling”.

• Allow for mixed programming models to coexist. We need a bridge to new programming models that is not an all or nothing proposition.

• Enhance job flow to enable many concurrent capability scale jobs. (similar to the emerging approach at LLNL) This is likely to be a common early usage model for Exascale.

• Open source can be a very good thing for vendors and end users but we need to find a way share the responsibility and …… risk.

• Educate young people in parallel programming.

Revolutionary Software Areas for Exascale:

- Storage class memory is coming: Technology will offer 1000x less latency but there are many other dimensions to this. Need to think through the possible directions to use this technology.

- Systems are transitioning to being power optimized. Application developers are still focused on performance optimization regardless of power. In a world where there is a power budget, software should play a role in optimizing performance through optimization of Perf/ Watt. (with total power being a hard facility constraint)

- Reliability: This is not an issue of what will we do when systems can not be made reliable. This issue is making the best trade-offs between hardware, system software and fault tolerant applications.

Actions we can take

- Value of storage class memory: Need to have the HPC community united in articulating the value proposition associated with storage class memory. The critical break points in terms of bandwidth, density, cost and latency need to be understood to help guide the technology development.

- Power : This is somewhat a mindset change. Applications will eventually need to think of their computing resource as a total energy budget and they need to optimize within this. Fortunately much of performance tuning also drives toward energy efficiency… but not always. Tools and reports that detail the energy usage need to be accessible to users.

- Reliability: The realistic adoption, cost and risk of fault tolerant algorithms must be assessed and these should be traded off against hardware cost and risk. The systems can not move in a direction that "might" be acceptable from a reliability perspective. This makes a software solution very difficult.

Software Barriers for HPC

Moderator

Pete Beckman

Presenters

Al Gara

Jean-Yves Berthou

Mitsuhisa Sato

Peggy Williams

Vivek Sarkar

Ann Trefethen

Three important software areas where *evolution* is required to improve existing open source software for extreme scale

1.1 compilers/performance analysis tools for achieving mono-processor high performance, specially with accelerators (Larrabe, GPU, Cell, …)
Goal : more than 30% of the peak performance

1.2 Efficient, “easy to use”, portable and fault tolerant implementation of Libraries, Languages/compilers for mixed parallelism : MPI/OpenMP/”cuda/Open CL like” languages
Goal: one million cores (heterogeneous, hierarchical and massively parallel)

1.3a Algorithm/solvers and data structures adapted to heterogeneous/hybrid, multilevel and hierarchical massively parallel machines.
Example: Dealing with non-structured irregular meshes for CFD computation on GPU
Goal:

⇒ No global communication involving the complete system(avoiding MPI_ALL-REDUCE, MPI_BARRIER,… on 1 million threads)
⇒ exhibiting different kind of parallelism (MPP, SIMD, …)
⇒ enabling fault tolerance techniques implementation
⇒ enabling efficient IO (data restructuring?)

1.3b Open Source scientific libraries sharing a single generic interface, targeting one million cores (heterogeneous & hierarchical)
Target: PETSc, SuperLU, ScaLAPACK, HyPre, MUMPS, PaStiX, …

Three important software areas where *revolution* is required to achieve scaling

2.1 Parallel visualization and remote/collaborative post-treatment tools

2.2 Parallel meshing, automatic hexahedral meshing, mesh healing, CAD healing for meshing and dynamic mesh refinement, hierarchical meshes (AMR like)
=> Dealing with x10^{10} cells mesh before 2015 (x10^{12} in 2020?)

2.3 Unified multiphysic/multiscale Simulation Framework and associated services, adapted to massively computing
⇒ mutualizing within a single platform pre and post-processing, calculation distribution and supervision, code coupling tools etc.
⇒ standardize integration of multiple solvers ("standard" for interoperability of scientific software components)
⇒ standardize data exchange (common data model for mesh and fields) and associated services (mesh projection, data interpolation, ..)

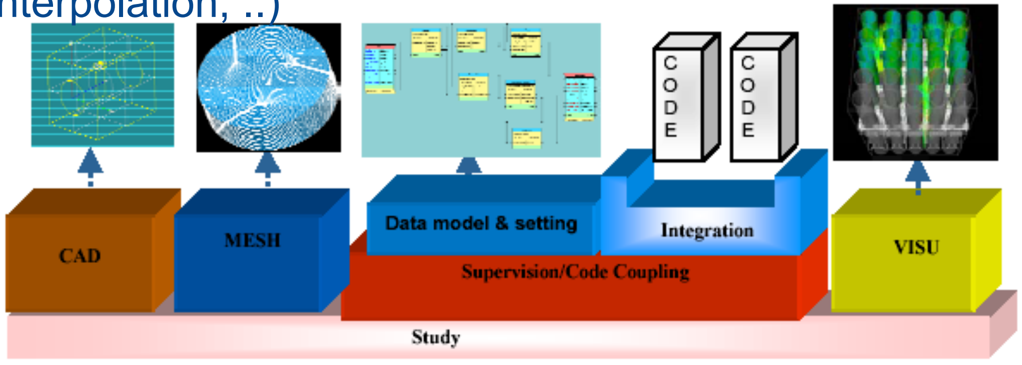


Figure 10. The Salome platform, www.platform-salome.org

How do we get it done to develop community-supported open source software to address these 6 areas, what is needed?

Some issues related to Open Source

Developing Open Source software, some conditions that may help to succeed and to keep going:

- one active leader and the recognition by key players
- A roadmap and a validated business model (at least for the leader)
- An ecosystem of partners for the software development, diffusion and associated services (installation, deployment, maintenance, specific developments)

Using Open Source software, some issues to be aware of: how they are supported, deployed, visibility of the roadmap, associated risks (as an example, moving from Qt3 to Qt4 cost 400 days of development to the SALOME project).

Suggestion:

- Identification of existing HPC Open Source software (cf.P. Beckman list)
- Promotion of an international HPC source forge for Open Source software diffusion?

How do we get it done to develop community-supported open source software to address these 6 areas, what is needed?

Need for International Task Forces on:

1. Parallel visualization tool. The community should focus on a small number of tools. VISIT and Paraview seems good candidates
2. Remote and collaborative post-treatment tools
3. Meshing tools. Need for an international joint effort between academic, commercial companies and end users
4. Common data model and associated libraries.Providing an international standard model for mesh and fields exchange and services(localization, projection, interpolation, arithmetic operations, …)
5. Supervising and code coupling tool. Unifying the software developments often driven by end users communities (climate, energy, …)
6. Uncertainties Quantification. Uncertainty analysis framework, Uncertainties referential (methodologies and tools, Open Turns)
7. Algorithm/solvers and data structures, solver interface
8. Fault tolerance. Need a joint effort involving OS, compilers, middleware/libraries, numerical solvers/algorithm researchers and engineer communities

Research policy:

- Identifying existing projects and research actions, roadmaps, cost
  - ⇒ Need for a consolidated international roadmap for HPC software
- Identifying grand challenge applications as driving forces
  - ⇒ Need for an International End User Forum structured around large communities (Climate, Health, Energy, Transport, Defense, …)

Identifying funding schemes: US/EU(or national)/Japan co-funding, single country funding with third parties participation?

Software Barriers for HPC

Moderator

Pete Beckman

Presenters

Al Gara

Jean-Yves Berthou

Mitsuhisa Sato

Peggy Williams

Vivek Sarkar

Ann Trefethen

3 important software areas where *evolution* is required

- To improve existing open source software for extreme scale
- I would propose “standard” development effort for making a steps for next evolution to exa-scale

- 3 areas
  - Programming language/interface for distributed memory
    - We should make “standard” for state-of-the-art programming languages
    - PGAS and remote memory interface
    - Global views such as Chapel and HPF
  - Fault tolerant model and APIs
    - Problems are in reality more than 10,000 cores (100TF)
    - Application people want some “standard” solutions in reality
    - e.g. MPI 3 effort is going on …
  - File I/O model for large scale systems
    - Data becomes more and more important.
    - We need “standard” model for I/O and file systems in hundreds thousands nodes
    - e.g. MPI IO, Grid distributed file system …

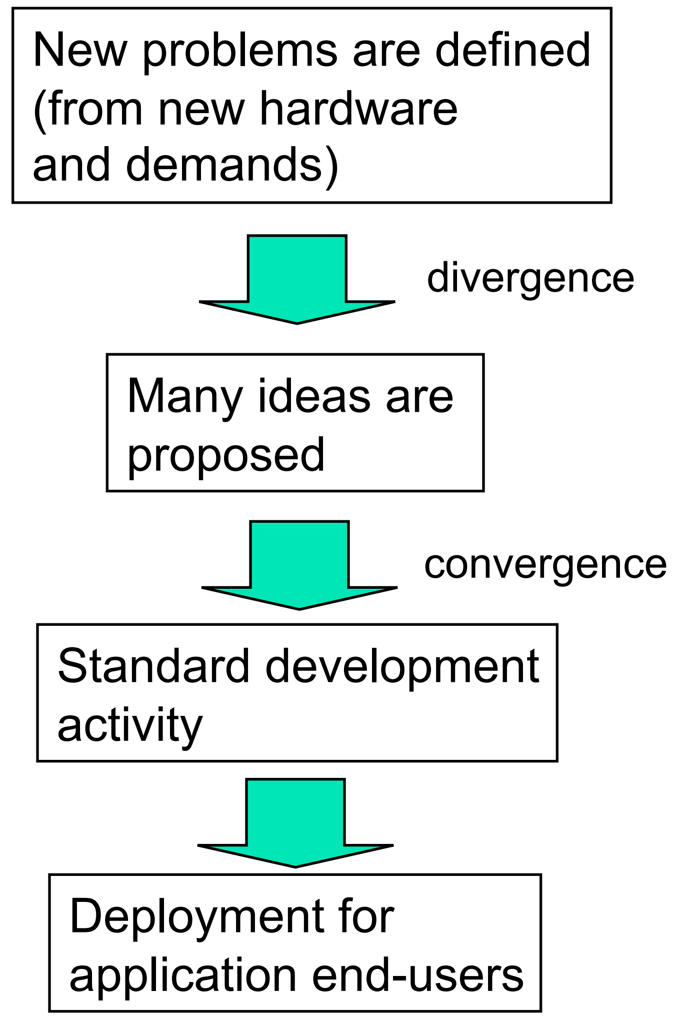
3 important software areas where *revolution* is required

- To achieve scaling … (for exa-scale)
- I would propose software supports for platforms from "weak-scaling" to "strong scaling"
  - Exascale machine != embarrassingly parallel machine!
  - Complexity from arithmetic unit, cores, SMP nodes, network to systems.
- 3 area
  - Unified programming model for a high performance node such as multicore, many cores, accelerator (GPGPU, FPGA, …)
    - Data localities, scheduling, …
  - Programming model for compos-able and scalable software
    - Module programming in parallel software
    - High level programming lang. such as functional prog., dataflow prog., tele-scope lang.
    - For multi-physics simulations, …
  - Fault tolerant / dependability in exa-scale systems
    - Model, Cost, Programming, Algorithms, …
    - FT will still be important and hard problems.

How do we get it done?

- We should promote standard development effort of APIs between several levels and components of existing software (for "evolution")
  - For end-users, education, …
  - For development of higher-level software technologies
  - Improvement of technologies by defining clear APIs
  - e.g. MPI, OpenMP, …
- We should encourage the exchange ideas (for "revolution")
  - Diversity is important

A software research/development process model


New problems are defined (from new hardware and demands)
divergence
Many ideas are proposed
convergence
Standard development activity
Deployment for application end-users

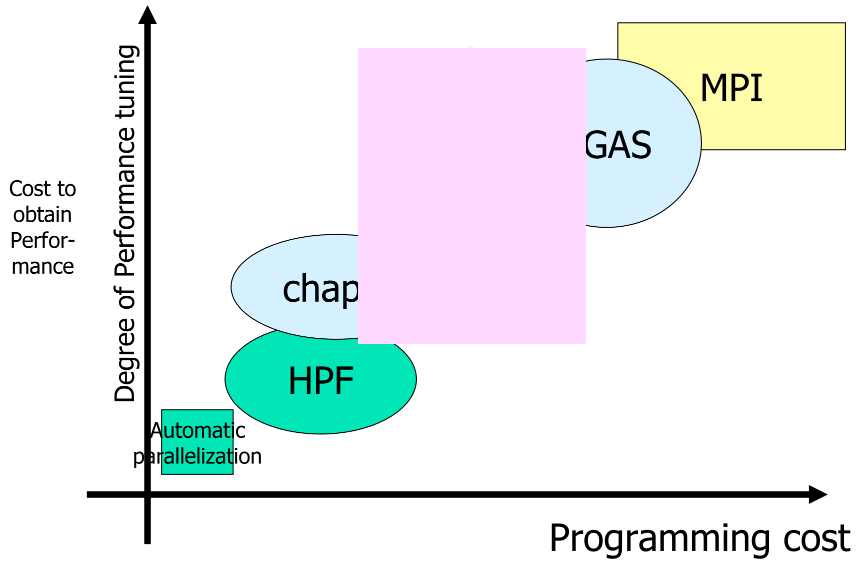
Proposal for "Parallel Programming Languages" area

- Many parallel programming languages have been proposed, but …
  - Many people still use MPI …
- OpenMP is now "standard" for programming multi-cores
- What about distributed memory programming?

- NOTE: Restrict us parallel extension of existing languages (C/F95) for end-users.
  - NOT HPCS languages and Java-based.

- How about PGAS (UPC and CAF)?
  - Local view parallel programming
  - Already standard?

- Global view parallel programming
  - We should learn from HPF history
  - Locality, efficient communication …
  - Any way to Combine to local view programming?



Cost to obtain Perfor-mance
Degree of Performance tuning
Programming cost
MPI
GAS
chap
HPF
Automatic parallelization

XcalableMP

http://www.xcalablemp.org

Software Barriers for HPC

Moderator

Pete Beckman

Presenters

Al Gara

Jean-Yves Berthou

Mitsuhisa Sato

Peggy Williams

Vivek Sarkar

Ann Trefethen

Where is evolution required?

- MPI
  - It’s portable
  - It’s ubiquitous
  - It isn’t going away
- Thread Packages
  - Stable, portable, general user-level
  - Enable easier implementation of OpenMP
  - Allow oversubscription of HW threads
- Operating System
  - Extremely lightweight with global functionality (memory management, communication, etc.)
  - Heavily multithreaded locally for latency tolerance

Where is revolution required?

- Software to enable reliable systems built with unreliable parts
  - Infrastructure to enable application resiliency
  - Programming Models
  - System Software
  - APIs
- Finding and Expressing parallelism
  - User perspective (how to code it)
  - Compiler perspective (how to render what the user has expressed)
  - Make extremely fine-grain, massive, μthreading practical and effective
  - Exploit heterogeneous concurrency (computation, communication, I/O)
- Programming Tools
  - Intelligently collect data
  - Provide space efficient format for data storage
  - Collapse, reduce, filter data

How do we get it done?

- Define the overall architecture
  - Can we converge on a common architecture?
  - Establish well-defined interfaces between SW layers
  - Dedicated architects throughout the effort
- Establish a community for key projects
  - Dedicated maintainers
  - Research + Industrial partnerships with funding for both
  - User community participation
- Avoid “Design by Committee”
  - HPF, Ada are examples to avoid
  - Respected leaders make the tough calls

How do we get it done?

- Focus on the full SW life-cycle, not just the initial development
  - Test and integration
  - Maintenance
  - Management of the rate of change
- Provide a common exascale test and integration platform
  - All components tested at scale on a reference platform
  - Strong focus on:
    - Mainline testing
    - Error-path testing
    - Edge-condition/interface testing
- Resolve Differentiation Needs vs. Commonality Needs
  - Hardware has been commoditizing over time
  - Can common SW provide opportunities for vendor differentiation?

Software Barriers for HPC

Moderator

Pete Beckman

Presenters

Al Gara

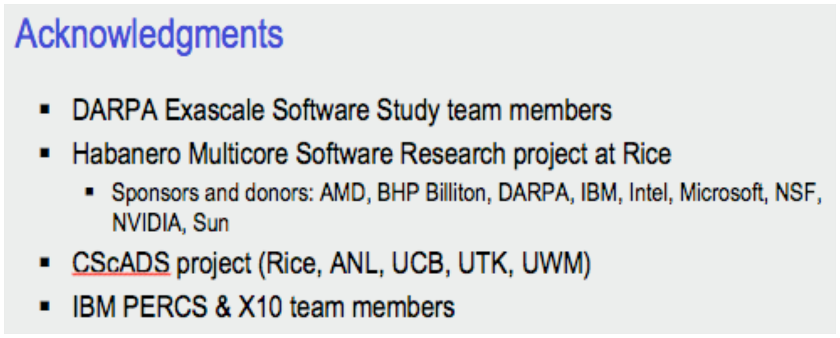
Jean-Yves Berthou

Mitsuhisa Sato

Peggy Williams

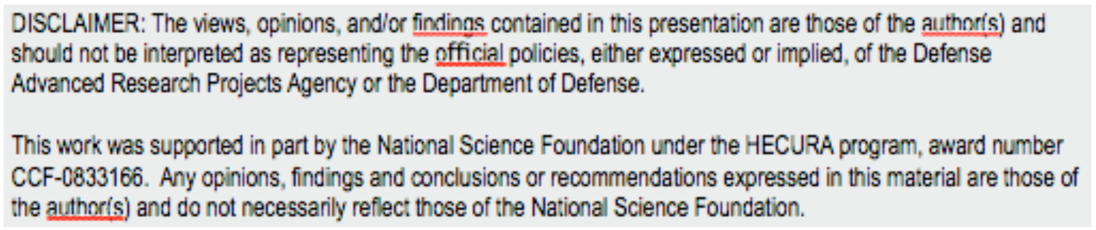
Vivek Sarkar

Ann Trefethen



Acknowledgments

- DARPA Exascale Software Study team members
- Habanero Multicore Software Research project at Rice
  - Sponsors and donors: AMD, BHP Billiton, DARPA, IBM, Intel, Microsoft, NSF, NVIDIA, Sun
- CScADS project (Rice, ANL, UCB, UTK, UWM)
- IBM PERCS & X10 team members



This work was supported in part by the National Science Foundation under the HECURA program, award number CCF-0833166. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

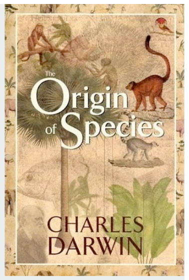
Context for ExaScale Software Study (in progress)

- Characteristics of Extreme Scale systems:
  - Massive multi-core (~ 1000 cores/chip)
  - Performance driven by parallelism, constrained by energy
  - Three system classes --- Exascale Data Center, Petascale Departmental, Terascale Embedded
- Key Software Challenges:
  - ***Concurrency***
  - ***Energy***
  - ***Resilience***
- Software stack:
  - Application frameworks & Tools
  - Programming models and languages
  - Libraries
  - Compilers
  - Runtimes for scheduling, memory management, communication, performance monitoring, power management, resilience, storage (including metadata access)
  - Operating & Storage System – persistence support

Extreme Scale software need long-term research that goes beyond industry efforts in cloud computing and manycore accelerators

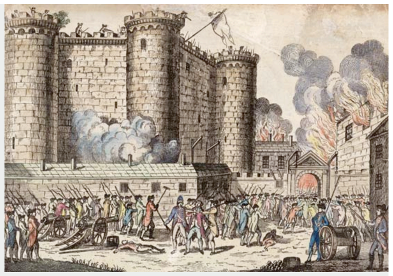
Software-hardware co-design will be critical to the success of future Exascale systems

Three Software areas where Evolution is Necessary



1. ***Performance Analysis Tools***
   - Extensions for multithreaded code
   - Extensions for calling contexts
   - Progress under way in SciDAC centers such as CScADS & PERI
2. ***Node Compilers***
   - Adjust and adapt to proliferation of new multicore processors
   - Extend auto-tuning techniques with online & offline learning
   - DARPA AACE program will provide a major boost to this area
3. ***MPI + Dynamic Parallelism***
   - MPI Communicators are founded on fixed process structures
   - Process structures will need to change dynamically to address needs of emerging HPC applications (adaptive/unstructured grids, coupled models) and architectures (manycore)

Three Software areas where Revolution is Required



1. ***Fine-grained Asynchronous Parallelism***
   - Weak scaling and bulk-synchronous parallelism will not deliver billion-way concurrency needed in Exascale systems
   - Instead require unified abstractions of asynchrony and concurrency for multi-core & cluster parallelism
     - Subsumes threads, shared memory, message-passing, active messages, …
2. ***Locality Models***
   - Data movement will be major contributor to energy consumption in Exascale systems
   - Need locality models that enable programmer, compiler, and runtime to manage data movements across multiple levels of memory hierarchy
3. ***Software-Hardware co-design for Exascale systems***
   - IESP effort should identify software interfaces that are critical bottlenecks, and drive vendors to provide hardware support for software-hardware co-design of these interfaces
   - Examples to follow

Example Opportunities for Software-Hardware Co-Design

- Dynamic parallelism with fine-grained tasks (async, spawn, …)
  - Hardware support for scheduling data structures
- Distribution and co-location of tasks and data (places, locales, …)
  - Hardware support for virtual-to-physical translation and inter-place data transfers
- Collective and point-to-point synchronization with dynamic parallelism (barriers, phasers, …)
  - Hardware support for intra-node & inter-node synchronization and communication
- Producer-consumer parallelism (single-assignment vars, futures, …)
  - Hardware support for full-empty bits
- Isolation and mutual exclusion
  - Transactions, fine-grained locks
- Data parallelism
  - Vectors, SIMD, SIMT

Candidate items for Software-Hardware Interface

- Memory hierarchy configurations
  - Cache sizes & geometries, hardware vs. software cache coherence
  - Register file sizes and data widths
- Memory access patterns
  - Address ranges that should bypass cache
  - Address ranges that require hardware coherence
  - Address ranges for which coherence will be managed by software
  - Address ranges with values that are guaranteed to be read-only (immutable) for certain application phases
- Network bandwidth partitioning for different forms of data movement and communication
  - PGAS, RDMA, Message passing, Stream processing, …
- Other network reconfigurability parameters
  - Topology, Packet size, …
- Power management
  - Frequency scaling, Voltage scaling, …
- Performance profiling
  - Lightweight profiling, Identification of events to be counted and sampled, …
- Resilience
  - Identification of threads with lower resilience requirements e.g., for which software can perform error detection and recovery

From Powerpoint to Action

- Directed research needed for all 6 topics (and more)
  - Revolutionary areas --- let a thousand flowers bloom
    - Users will vote with their feet (and noses)
  - Evolutionary areas --- opportunities for consolidation starting with performance tools
- Application drivers
  - Application stakeholders should contribute sample applications and/or SSCA's --- requires effort, but will pay great dividends
- Platform drivers
  - Platform stakeholders should contribute to development, testing and integration for their platform --- requires effort, but will pay great dividends
- Coordination
  - Follow best practices of successful open source projects --- open development, continuous integration, continuous testing, customer focus, community involvement, meritocratic leadership, …
  - Open source participation in selected areas can be strategic to vendors too
    - For example, see IBM Systems Journal special issue on Open Source Software, Volume 44, Number 2, June 2005 for open source experiences by a range of IBM project
  - Software-hardware co-design – don't let software play second fiddle to hardware!

Software Barriers for HPC

Moderator

Pete Beckman

Presenters

Al Gara

Jean-Yves Berthou

Mitsuhisa Sato

Peggy Williams

Vivek Sarkar

Ann Trefethen

UK Roadmap activity

Leveraging work in the US and Europe together with UK specific workshops and discussions groups have lead to barriers for software development that fall into five themes

1. Cultural Issues
   - some people won't share...
2. Applications and Algorithms
   - Need to bring application and algorithm development closer
   - Need new algorithms for new architectures
3. Software Challenges
   - Engineering, portability, programming models, ...
4. Sustainability
   - Need better models for sustainability not only for UK efforts but those that we depend on!
5. Knowledge base
   - It would be good to know who is doing what and where
   - We need to train more people with this cross cutting set of skills.

http://www.oerc.ox.ac.uk/research/hpc-na♪

Evolution x 3

- Communication libraries
  - Cleverer
- Numerical and visualisation algorithms and libraries and tools {need both evolution and revolution}
- Integration of systems of models across scales and the like are increasingly important – need to evolve support for this – error propagation.
- Best practice software engineering....

Revolution x 3

- Portability
  - Architecture dependent code-generation
  - Dynamic adaptation
  - Check out on one platform check in on another
- Programmability
  - Develop systems that let us drive the machine with the hood down – better abstractions
- Dependability
  - On this scale things will fail – but it shouldn't mean they're broken
- Validation
  - Garbage generated in milliseconds is still garbage

- What can we learn from our formal methods colleagues?

Playing together

- Collaborative development of a roadmap for exascale software – several such already underway at the national level
- We need better coordination at the international programme level including mechanisms for collaboratively funded research and development
- Integration of applications, numerical and system software – silos of activity will not achieve our aims – US is better at than UK at this.
- Better models for sustainability
  - Community support?
  - Industry take-up
  - Need to ensure exascale efforts are not for the few
- Shared knowledge base required.

Playing together

- Success stories include BLAS, LAPACK, MPI, GPNL (what is that library called), PetSC
  - Good requirements capture, careful design, well engineered, well supported, used by many
- Support models:
  - Community support with funding agency investments
  - Vendor supported due to user requirements (eg MPI)
  - Industry support through direct licensing (library that Rolls Royce using), through integration into products, Matlab, NAG, ...
- Failures
  - Too many to mention – badly designed and/or engineered, no industry leverage.. Etc..
  - Created for a single audience or application area (CCPs)
  - Support model has relied on continuing investment from research councils (much grid software)
  - Tied to a particular architecture (CMSSL)

Playing together

- Ongoing activity – apace development site
  http://apace.myexperiment.org/