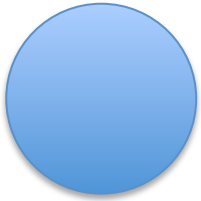


Blue Dot Discussion

Computational Challenges and Needs for Academic and Industrial Application Communities

Oxford IESP Meeting

12-14 April 2010

Participants

- Giovanni Aloisio
- Jean-Claude Andre, co-chair
- Venkat Balaji
- Jean-Yves Berthou, co-chair
- Peter Coveney
- Robert Harrison
- Stefan Heinzel
- Ryutaro Himeno
- Herbert Huber
- Richard Kenway
- David Keyes, scribe
- Norihiro Nakajima
- Jean-Philippe Nominé
- Mike Papka
- Joel Stiles
- Fred Streitz
- Makoto Taiji
- Toshikazu Takada
- Hiroshi Takemiya
- Bill Tang
- John Taylor

Technology Charge

- Assessing the short-term, medium-term, and long term software and algorithm needs of applications for peta/exascale systems
- Refining the roadmap for software and algorithms on extreme-scale systems
- Setting a prioritized list of software components of exascale computing as outlined in the roadmap

Response to Charge

- As applications custodians and exascale customers, we respond by considering how particular applications (nicknamed “co-design vehicles,” or CDVs, after the principal new programming paradigm in the exascale regime) would migrate to the exascale
- We assume that most of the apps for which exascale systems will be built already exist today in high-performance implementations
  - We remain open to brand new apps and brand new formulations of existing apps, natively suited to the exascale
- We assume that all apps will need to be rewritten, substantially
  - From scratch in terms of data structures
  - Substantially in terms of algorithms
  - Possibly even in terms of mathematical formulations

Mode of breakout operation

- The group included many veterans of previous IESP workshops, including the Tsukuba workshop
- We made sure we captured the results of the Tsukuba workshop by reviewing the applications needs discussed there
- We need to dialog further with enabling technologists in algorithms, software, and architecture before closing the loop
  - This report is merely a step along the path of prioritization

Caveats

- In prioritizing software components, we considered many schemes
  - by what important applications identify as needs
  - by position along the critical path
  - by what an exascale initiative needs to do for itself (as opposed to what the market will do for us, anyway) and then, within such an exascale initiative, by
    - what *application designers* must build
    - what *enabling technologists* in mathematics and computer scientists should build
- The first is the simplest and is the only one attempted here

A prevalent theme

- There is a disconnect between an initiative to pursue the exascale and the reality that some applications currently do not successfully exploit the petascale
  - They encounter difficulty obtaining support and recruiting interdisciplinary teams to get there
  - "Have to walk before you can run." – Bill Tang
- It is difficult to write a detailed exascale roadmap from the current sub-petascale status for some applications
- On the other hand, there is *no question* that exascale power is desired for progress in all applications represented around the table
  - Easy to quantify for some applications, e.g., QCD, cosmology, seismic inversion, that are scaling extremely well and processing bottlenecked today
  - Harder to quantify but equally important for less uniform applications with complex geometry, adaptivity, multiple phases with different physics
  - We must not assume that the former are full proxies for the latter

Prototype exascale hardware: a heterogeneous, distributed memory GigaHz KiloCore MegaNode system

Systems	2009	2018	Difference Today & 2018
System peak	**2 Pflop/s**	**1 Eflop/s**	**O(1000)**
Power	**6 MW**	**~20 MW**	
System memory	0.3 PB	32 - 64 PB [**.03 Bytes/Flop**]	O(100)
Node performance	125 GF	1,2 or 15TF	O(10) – O(100)
Node memory BW	25 GB/s	2 - 4TB/s [**.002 Bytes/Flop**]	O(100)
Node concurrency	12	O(1k) or 10k	O(100) – O(1000)
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)	O(100)
System size (nodes)	18,700	O(100,000) or O(1M)	O(10) – O(100)
Total concurrency	225,000	O(billion) [O(10) to O(100) for latency hiding]	O(10,000)
Storage	15 PB	500-1000 PB (>10x system memory is min)	O(10) – O(100)
IO	0.2 TB	60 TB/s (how long to drain the machine)	O(100)
MTTI	days	O(1 day)	- O(10)

c/o P. Beckman

Some common moans

- The most commonly envisioned path to exascale is thousand-fold manycore at 1GHz each, within a trightly coupled network of about one million such nodes
  - However, memory bandwidth is already limiting today's low core count nodes to less than 10% of peak on most apps, whose kernels offer little cache reuse (e.g., stencil ops or sparse matvecs)
  - Processors are cheap and (relative to memory) small in chip area and relatively low in power, so there is no harm in having them in excess most of the time, but the opportunities for exploiting the main new source for performance are undemonstrated for most applications
- While there is opportunity for combining today's individually high capability simulations into complex simulations, there is no silver bullet for merging the data structures of the separate applications
  - the data copying inherent in the code coupling will likely prevent exploitation of the apparent concurrency opportunities

Applications' messages (next slides)

- Programming models and algorithms
- I/O
- Fault tolerance
- Reproducibility

Programming model is a key

- Prior to possessing exascale hardware, apps groups can prepare themselves by exploring new programming models
  - on manycore and heterogeneous nodes
- Attention to locality and reuse is valuable at all scales
  - will produce performance paybacks today *and* in the future
- New algorithms and data structures can be explored under the assumption that flop/s are cheap and moving data is expensive
- Bandwidth pressure can also be reduced by considering mixed-precision algorithms, using lower precision wherever possible

I/O is a key

- For many apps, I/O is a potential bottleneck
  - For input, for output (including visualization), or for checkpointing, or any combination

Fault tolerance is a key

- Applications people reluctantly recognize that fault tolerance is a shared responsibility
  - It is too wasteful of I/O and processing cycles to handle faults purely automatically
- Different types of faults may be handled different ways, depending upon consequences evaluated by scientific impact
- Strategic, minimal workingset checkpoints can be orchestrated by application developers and users

Reproducibility is a key

- Applications people realize that bit-level reproducibility is unnecessarily expensive most of the time
- Though scientific outcomes must be run- and machine-independent, we have no illusions about bit-level reproducibility for individual pairs of executions with the same inputs
  - Since operands may be accessed in different orders, even floating point addition is not commutative in parallel and on inhomogeneous hardware platforms; this has been true for a long time
  - A new feature in the context of co-design, with an emphasis on low power (low voltage switching), is that lack of reproducibility may emerge for many *other* (hardware-based) reasons
  - If applications developers are tolerant of irreproducibility for their own reasons, e.g., for validation and verification through ensembles, then this has implications for considering less expensive, less reliable hardware

Priorities for exa-scaled enabling technologies

Development-related

- **Programming models**
- Compilers
- Debuggers
- Profilers
- Source-to-source transformations*
- Configuration systems

Production-related

- **I/O systems**
- **Fault monitoring, application reporting, and recovery**
- Dynamic resource management
- Data management
- Visualization systems
- Frameworks
- Data miners
- Dynamic performance optimization
- Workflow controllers

Model-related

- **Uncertainty quantification**
- **Dynamic load balancing**
- Meshers
- Solvers / integrators
- Adaptivity systems
- Graphs and combinatorial algorithms
- Geometric modelers
- Discretizers
- Partitioners
- Random number generators
- Compression

* to provide some hardware independence for codes

Roadmap Components (Beckman)

Big data is a key horizon

- An exascale machine (once loaded up) is a 32 PB fast store, with lots of processors to graze over it
- There may be lots of interesting things to do with it that we don't do yet, e.g., in data mining in climate modeling, astrophysics, etc.
- Such applications can begin to be explored today in "miniature" on petascale computers with 300 TB

Issues for scaling up CDVs

- Weak scale applications up to distributed memory limits
  - Proportional to number of nodes
- Strong scale applications beyond this
  - Proportional to cores per node/memory unit
- Scale the workflow, itself
  - Proportional to the number of instances (ensembles)
  - Integrated end-to-end simulation
- Co-design process is staged, with any of these types of scaling valuable by themselves
- Big question: does the software for co-design factor? Or is all the inefficiency at the data copies at interfaces between the components after a while?

Limitations to be explored by CDVs

- Strong scaling algorithms
  - sufficient coarse grain parallelism but
  - expect load imbalance to be the main problem due to irregular task/data size
  - bulk synchronous algorithms on one million nodes do not tolerate load imbalance worse than one part per million for a synchronous task
- Single node performance
  - Compiler-generated code for hybrid/multicore
  - linear algebra kernels
    - typically come with autotuning
  - nonstandard linear algebra kernels
    - we need the autotuning tools, not just their output

Needs to be explored by CDVs

- Tools to generate domain-specific languages and for powerful source-to-source transformations
  - to enhance composability
    - want to enable new science and expand developer and user communities, so we must *decrease* complexity as we go to exascale
  - for writing performance-portable code (retargetable)
  - to extend effective lifetime of code over generations of hardware
  - to implement domain-specific frameworks
    - frameworks are widely perceived as successful solutions of many HPC problems
    - the future is increasingly multidisciplinary, so these must interoperate
- Expanded/different programming models
  - move more of the burden of managing scheduling of computation and placement of data to runtime
  - expand intrinsically fault tolerant programming models to be relevant to a broader class of algorithms
  - interoperability of programming models (GAS, MPI, Cilk, HPCS, etc.)
- Understanding the design space
  - tradeoffs associated with options for power consumption and resilience
  - the nature of expected faults, including common signaled faults and especially silent faults

Criteria for Co-Design Vehicles (CDVs) [Tsukuba + 2]

- Application running at at least the terascale today, with obvious need for more
- In progressing to the exascale, should be able to achieve scientific goals in its own domain
  - amenability to experimental validation
- Should possess a well-defined set of steps to progress to exascale
- Should have a community supporting the application that has the skills and experience to engage with the co-design process
- Has to be open and should ideally spawn modules for common use
- Overall portfolio should attempt span application space

Different categories of CDVs

- Societally relevant simulations (e.g., climate, patient-specific medicine)
- More likely readily scaled simulations (e.g., QCD, cosmology)
- Data processing problems (e.g., Square Kilometer Array in Australia, which generates 1 EB/s of data and needs FFTs per image while data is streaming)
- Perhaps a “surprise outsider”, not currently practical at the tera-/petascale
- Our breakout does not designate the CDVs, but seeks to identify the features by which the CDV can itself be identified

Algorithmic Priority Research Directions (1)

- Advanced mathematical methods for scientific understanding in exascale simulations, including *in situ*
  - Uncertainty quantification, intrusive and nonintrusive
  - Optimization, inverse problems, sensitivity
  - Analysis and Visualization
  - Validation and Verification

Algorithmic Priority Research Directions (2)

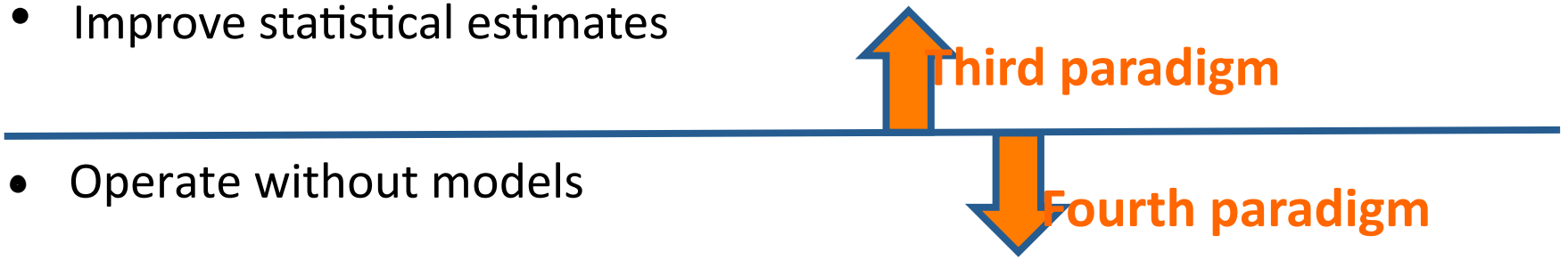
- Exascale algorithms that expose and exploit multiple levels of parallelism
  - Communication-reducing algorithms
  - Synchronism-reducing algorithms
  - Fault resilient algorithms
- Support for multiphysics, multiscale methods
  - Break the SPMD and BSP paradigms in joining multiple different codes
  - Stability of coupling

Algorithmic Priority Research Directions (3)

- Exascale algorithms for constructing and adapting discrete objects
  - Algorithms that deal with unpredictable, dynamic workloads and have few flops to hide

Why push to extreme scale?

- Better resolve model's full, natural range of length or time scales
- Accommodate physical effects with greater fidelity
- Allow the model degrees of freedom in all relevant dimensions
- Better isolate artificial boundary conditions or better approach realistic levels of dilution
- Combine multiple complex models
- Solve an inverse problem, or perform data assimilation
- Perform optimization or control
- Quantify uncertainty
- Improve statistical estimates



- Operate without models

How these needs are felt in, e.g., reservoir modeling

- Better resolve model's full, natural range of length or time scales
  - Discretize a reservoir into more horizontal cells and layers
- Accommodate physical effects with greater fidelity
  - Replace black oil assumption with fuller compositional effects
- Allow the model degrees of freedom in all relevant dimensions
  - Here, all three space dimensions, plus time
- Better isolate artificial boundary conditions or better approach realistic levels of dilution
  - Use additional cells outside of the production and injection zones to buffer the zones where data is needed from unknown geology and fluxes

How these needs are felt in, e.g., reservoir modeling

- Solve an inverse problem, or perform data assimilation
  - Match simulation with drilling and historical production records to better estimate unknown parameters in the model and nudge the simulation towards reality, where and when reality is known
- Combine multiple complex models
  - Unify the simulations of adjacent fields to capture the effects of producing one of them on the other, or combine a model for the reservoir with a model of the pipeline network to provide feedback on reservoir production from transportation
- Perform optimization or control
  - Select a strategy for injection and pumping in the thousands of wells per reservoir

How these needs are felt in, e.g., reservoir modeling

- Quantify uncertainty
  - Given the many unresolvable uncertainties in program inputs, bound the error in the outputs in terms of errors in the inputs
- Improve statistical estimates
  - Treating uncertain inputs as random, improve output estimates

Need for extreme scale goes far beyond these, however!

- Oil companies are vast, with hundreds of reservoirs "upstream" and many refineries and transportation systems "downstream"
- Companies are under contractual constraints for supplying products and must seek to *maximize profit* while *satisfying output demands* in many different product streams
- Tens of thousands of "valves" (literal and figurative, like workforce and other controls) need to be scheduled continuously
- Mathematically, this is a massive, nonlinear and possibly nonrobust constrained optimization problem – insatiably power hungry

How should resources such as reservoirs be managed?

- They are too complex for a self-contained, self-consistent theory
- They are unsuitable for experiment, because you can only do the experiment (*e.g.*, exploit the reservoir) once
- Engineering heuristics are useful (and used!), but
  - gone are the days when employees spent decades understanding a single reservoir
  - the external forcings (*e.g.,* world economy) change daily, making history less useful
- Extreme-scale simulation is an irreplaceable tool for exploring scenarios experimentally in a virtual world
- Data mining and machine learning may be even more useful tools in the future.

Extra slides

(particular CDVs contributed by group to be added according to template, not in this collection)

How to pursue the exascale?

- Applications as co-design vehicles

or

- Libraries, kernels, factored components, etc.

Building up CDVs

- Capability (scaling up individual codes)
- Complexity (combining multiple codes)
- Scientific or engineering understanding (exploring parameters space for sensitivity, stability, optimization, inversion, etc.)

Exascale candidates

- Multiscale, multiphysics problems
- Problems that have a superlinear scaling of work with memory capacity
  - for cost- and power-feasible exascale hardware design, which is memory- and communication bandwidth-limited

Organization Charge

- Developing a governance, management, and organizational structure for IESP
- Exploring ways for funding agencies to coordinate their support of IESP-related R&D so that they complement each other
- Exploring how laboratories, universities, and vendors can work together on coordinated HPC software
- Creating a plan for working closely with hardware vendors and application teams to co-design future architectures

Execution Charge

- Developing a strategic plan for moving forward with the roadmap
- Creating a realistic timeline for constructing key organizational structures and achieving initial goals
- Exploring community development techniques and risk plans to ensure key components are delivered on time
- Exploring key components of any needed intellectual property agreements

The wrong reward

- Since a flop is by far cheaper to provision and to power up, relative to memory and memory transfer rate, good designs over-provision flops, so that they are never limiting
  - Percentage of peak is therefore a counter-productive metric, since it penalizes for over-provisioning something cheap
  - Percentage of the instantaneously limiting resource, typically memory bandwidth, is the most interesting figure of merit for a given execution of the "right problem"
  - The "right problem" is the one that provides the requisite accuracy with the requisite confidence at some combination of lowest energy and shortest execution time

Some major challenges that stay the same in peta to exa

- Poor scaling of collectives due to internode latency
- Poor performance of SpMV due to shared intranode bandwidth ("*Dear Santa …*")
- Poor scaling of coarse grids due to insufficient concurrency
- Lack of reproducibility due to floating point noncommutativity and algorithmic adaptivity (including autotuning) in efficient production mode
- Difficulty of understanding and controlling vertical memory transfers

Library software design principle: multiple layers

- **Top layer (all users)**
  - **Abstract interface featuring language of application domain, hiding details, with conservative parameter defaults**
  - ***Robustness, correctness, ease of use***
- **Middle layers (experienced users)**
  - **Rich collection of state-of-the-art methods and data structures, exposed upon demand, highly configurable**
  - ***Capability, algorithmic efficiency, extensibility, composability, comprehensibility of performance and resource use***
- **Bottom layer (developers)**
  - **Support for variety of execution environments**
  - ***Portability, implementation efficiency***

EOF