Revolutionary Approaches

William Gropp
Thomas Sterling

Where can revolutionary approaches succeed?

- Very difficult: displace workable solutions
  - PDE Simulation, Many-body computations are well-served by current programming models
    - Doesn't mean things couldn't be better, but evolution of these models likely to be sufficient
    - Option: consider possible roadblocks (e.g., resilience)

Where can revolutionary approaches succeed?

- Merely difficult: enable new application area
  - Applications that have given up on extreme scale computing
  - Applications with poorly scaling current applications *and* where scaling can be related to the current programming models (not just the implementations of current models)

Where can revolutionary approaches succeed?

- Complement other (non-revolutionary) approaches
  - E.g., replace/augment parallel coordination (MPI) with new approach, but retain node compiler, other runtime support

Where can revolutionary approaches succeed

- Overcome roadblocks to Exascale
  - E.g., Power, resilience, scalability
  - These need to be quantified (to hear some tell it, there are no problems

Chicken and Egg

- How do we identify and develop new application areas?
  - ♦ Explore revolutionary architectural and programming approaches
    - E.g., graph applications with latency-hiding hardware
    - How do we match ideas with application opportunities?
  - ♦ Engage application developers
    - How do we develop interest in the absence of hardware or software?
  - ♦ Engage algorithm developers
    - How do we provide a workable and believable performance model?

Specific Reports

1. How to quantify the *need* for new approaches (requirements from exascale hardware)?
2. How to identify unserved application areas?
3. How to identify practical components to augment current programming/runtime/os capabilities?
4. How to motivate some (any) applications developers to consider new approaches?
5. What are the potential technologies?
6. What do we do about the existing niche approaches?

Need for New Approaches

- What ***quantitative*** evidence do we have that some part of the current software stack will not work at Exascale?
- What needs to be done to get more data?

Finding New Application Areas

- What application areas are good candidates for Exascale but are not currently considered or likely to use Exascale?
- How can we identify others?

Augment Current Approaches

- Where are the opportunities for augmenting current approaches? (Consider programming models, runtime, OS, I/O, fault tolerance, …)
- What is the expected benefit (be quantitative)?

Engaging Applications

- How can we engage representatives of applications?
  - E.g., Paper designs, prototypes, simulators? See *www.csm.ornl.gov/~anish12/hips08cmrdy.pdf*

Potential Technologies

- What examples do we have of technologies that might have revolutionary impact?
  - This can't be an exhaustive list; rather an "existence proof"
  - Some technologies may be well-known but not widely applied. In such cases, address why that might change

Where is the Middle?

- What needs could be satisfied by technologies/approaches that may not be considered revolutionary but are not widely deployed? Is there danger that development will bifurcate into already in widespread use and revolutionary, with no room for current niche techniques?


1867

Specific Reports

1. How to quantify the *need* for new approaches (requirements from exascale hardware)?
2. How to identify unserved application areas?
3. How to identify practical components to augment current programming/runtime/os capabilities?
4. How to motivate some (any) applications developers to consider new approaches?
5. What are the potential technologies?
6. What do we do about the existing niche approaches?

Need for New Approaches

- What ***quantitative*** evidence do we have that some part of the current software stack will not work at Exascale?
- What needs to be done to get more data?

How to quantify the *need* for new approaches (requirements from exascale hardware)

- Consider architecture component candidates that meet constraints of power = 20MW; node architecture, communications network
- Identify candidate application characteristics determined against software stack
- Quantify reliability, communications, energy efficiency
  - reliability factor is the ratio of the average time incurred per failure for fault tolerance to the mean time between failures.
  - communications factor is the ratio of average time per communication to the mean time between communications.
  - energy-efficiency speedup is the ratio of the energy-efficiency obtained by executing the programs in parallel to that in serial
- Develop models to provide data for bounds and trade-offs against metrics


1867

Finding New Application Areas

- What application areas are good candidates for Exascale but are not currently considered or likely to use Exascale?
- How can we identify others?

How to identify unserved application areas?

- Anything that runs for a long time (days, weeks)
  - One way to cope with flat clock speeds is to run longer, such people and applications are candidates for new methods
- Look at scaling laws
  - Algorithms whose time or data movement power laws make them prohibitive at exascale
- Applications for which extracting more parallelism is known to be hard
  - But can we do anything for them?
- Big data sources from instruments (accelerators, light sources, telescopes, etc.)
  - Often have planned computing trajectories which are an opportunities to examine for exascale potential. Those groups, unlike SKA, who have not recognized exascale.
- Applications or researchers who currently rely on one or a small number of simulations
  - Scaling up through UQ enhances confidence in research and brings larger computing needs
- Applications for which MPI+X are known to be hard to implement or perform
  - E.g. phylogenetics, anomaly detection, noisy/graph problems


1867

Augment Current Approaches

- Where are the opportunities for augmenting current approaches? (Consider programming models, runtime, OS, I/O, fault tolerance, …)
- What is the expected benefit (be quantitative)?

How to identify practical components to augment current programming / runtime/ os capabilities?

- A very important point to consider is that we need a means to share information between components of the stack and applications. This allows us to reach a higher plane of existence that will allow for a **holistic** approach to performance, power, programmability, resources, and resiliency.
- Some of this work cannot be done without prototype systems and these prototypes need to be funded even if they don't immediately demonstrate the promise of added productivity / advancement of the state of the art, and … and … and …
- Some of the co-design centers in Japan, Europe and the USA and will most likely be willing to work on these new architectural ideas. We should make use of them if we can.

1: Applications

If a lot of information is available on the state of the system, the bleeding edge applications _will_ make use of it to make the machine usable. How to provide this information is going to be one of the key considerations. A standards based approach will allow different vendors to plug-and-play into this model of programming, computing, and analysis.

- Need to identify common patterns: parallel motifs, memory operations (gather-scatter, etc), I/O
- Need to identify canned solutions that are broadly used
- Need to solve issues related to small memory footprint
- Need to consider issues related to power
- Need to change the way programmers think about data / increase data locality

2: Tools for application development

- compilers that are more open and reason at a higher level to do translations to improve data locality, provide info on why loops didn't parallelize, etc. (example caravel from LANL, PGI's compiler with directive disgnostics, cray vectorizing compiler)
- **tools for hierarchical memory management. A very simple example is visualization, or even gather-scatter at different scales: on-socket, on-node, and on-machine.**
- Storage solutions at all levels need better interfaces that reduce the burden on the programmer
- debuggers that can work on a subset (e.g. totalview/stat from LLNL)
- performance tools that work with compilers to give better feedback on why performance is not optimal, also something that works better with the runtime / os to do power optimization, and resource management / allocation / reallocation to adapt to changing system conditions
- Provide information back to applications (see dumb programmers in #3 on next slide)
- New tools that we don't know about yet, like power analysis and optimization, data locality / movement optimizers, scalability analysis, runtime network bandwidth reallocator, in-situ analysis, concurrency analysis …

3: Runtime / OS

Runtime / OS should not assume that developers are dumb or unwilling to go the extra mile. You don't have to fix everything in the stack: if applications have more information on the state of the program, they can (and will) handle a lot of failures and inefficiencies given the right tools

- provide more information to application
- don't pull the rug from under the feet of the application (how most schedulers work today)
- dynamic adaptation and resource management interfaces to allow applications to compensate for lost hardware assets
- in situ, viz, etc are part of resource management and need to be considered seriously
- OS is probably going to play a limited role and is expected to be light weight
- power and resiliency problems are best solved in conjunction with applications


1867

Engaging Applications

- How can we engage representatives of applications?
  - E.g., Paper designs, prototypes, simulators? See *www.csm.ornl.gov/~anish12/hips08cmrdy.pdf*

#4 How to motivate some (any) application (and system sw) developers to consider new approaches?

How to motivate?

- Find influential and wiling parties and turn them into evangelists
- Make it easy (relatively) to try and adopt innovative approaches
- Give the risk takers measurable goals so they know what success looks like
- Make the rewards greater than the costs - though no guarantees
- Make it painful or embarrassing to avoid innovation

Carrot Approach

Stick Approach

Carrot Approaches

- Develop and publicize promising and practical components/tools
  - ♦ fund experimentation to discover which components/tools are promising and quantify their benefits
  - ♦ components/tools can't be research prototypes and by definition buggy
  - ♦ must be fairly painless to use
  - ♦ must allow incremental adoption
- Offer early access or lots of time to innovative hw
  - ♦ If you want to be one of the first to get access to an advanced system, show that you can make good use of it (examples - RoadRunner and ASC early science runs)

Carrot Approaches, continued

- Bribery - offer to fund students and post-docs
- Appeal to our competitive nature – design a public competition
  - For example a competition in which we reward recasting memory intensive algorithms/modules/apps into compute intensive sw while maintaining fidelity
- Find a leader and pay/coerce/cajole into being an evangelist
  - Can we identify such leaders?

Stick Approaches

- Scare them with analyses of our likely futures
  - For example telling ASC code teams they will not be able to do bulk synchronous operations got their attention
- Publish the true costs of computing, i/o, and storage
  - For example telling users the costs of storing 1PB of data (~$100K per year) has motivated people to change their behavior
  - Continue to motivate by reporting each user's or code's costs on a continuous basis or embarrassing them with peers (report top ten power hogs)

Stick Approaches, continued

- Recast scheduling mechanisms away from core/node-hours to kw/hours, for example
- Direct app and system sw teams to evaluate risky options - I find this rarely works unless program directors are on the ball and persistent


1867

Potential Technologies

- What examples do we have of technologies that might have revolutionary impact?
  - This can't be an exhaustive list; rather an "existence proof"
  - Some technologies may be well-known but not widely applied. In such cases, address why that might change

Potential Revolutionary Technologies
Subgroup Discussions

William Gropp, UIUC

Thomas Sterling, IU

Needs for Revolutionary HW/SW Technologies

- Performance = efficiency * scalability * availability * unit-speed
- Power bounds and energy
- Resilience
- Productivity
  - ♦ Generality
  - ♦ Portability
  - ♦ Programmability

Strategic Technologies

- Paradigm shift and Execution Model
- Programming model(s)
- Runtime system software
- Processors with global semantics/mechanisms
- Packaging and interconnect
- Commodity component
- New algorithms for parallelism, resource allocation, locality
- New classes of algorithms (e.g., stochastic communication avoiding, extreme hierarchical parallelism)

Some Anticipated HW Technologies

- Stacked dies
- Closer-in optics
  - Maybe socket to socket
  - WDM?
- Power Efficient Cores (PEC)
- Single thread optimized cores for Amdahl
- Multi/Many core
- Heterogeneity
- Torus topology but depends on workflow?

Some Detailed Technologies

- Global address space
  - e.g., E-registers
  - Fine grain psuedo access; rapid load
  - More than PGAS – must be able to move virtual data in physical space without address change
- Message-driven computing
  - E.g., active messages
- New programming abstractions
  - Expressing parallelism of algorithms
  - Expressing locality in codes
  - Expressing other attributes to inform system about:
    - Energy, fault response, granularity (of locality)
- (Many) Lightweight user threads
  - Low overhead context switching, suspension, instantiation, termination
  - HW support for user thread signaling for "trampolining"
- Rich semantics lightweight synchronization for finer grain control and continuation migration.