SOS 13 Algorithms Panel

H. Carter Edwards, SNL

Ralf Gruber, STI-EPFL

Mike Heroux, SNL

Bob Numrich, U of Minnesota

Trey White, ORNL

Questions

1. What classes of algorithms (if any) can easily scale to effectively use extreme scale computers?
2. What new classes of algorithms (if any) promise qualitative advances via the use of extreme scale computers?
3. What algorithm research efforts are most needed to prepare for extreme scale computers? (How will we succeed?)
4. What other challenges do we face toward effective use of extreme scale computers? (How will we fail?)
5. What should computer system designers know about trends in algorithms?

Carter Edward:
Extreme Scale HPC: Childhood’s End for Parallel Applications

- **Give up naïve perspectives:**
  - Global static machine: it is a dynamic environment with failing nodes
  - Simple local node: it is manycore with shared memory hierarchy
  - Global SIMD algorithms: MPI_COMM_WORLD was a fatal *cultural* flaw
    - Frequent global synchronizations limited by max-loaded node / OS noise
- **Vision for future strategies:**
  - Proactive and reactive fault tolerance (NO SILENT ERRORS):
    - Cannot rely solely on checkpoint-to-disk + re-queue application
    - Proactive: query probable future state of machine, adapt accordingly
    - Reactive: in-memory checkpoint + live restart on subset of machine
      - Allowed to fail only when two critical-path nodes concurrently fail
  - New perspectives for multilevel parallelism:
    - MIMD + SIMD: Global ensembles of cooperating SIMD components; concurrent evaluations for UQ/Optimization, multiphysics, multiscale, multilevel solvers, post processing, I/O, …
  - Design in separation of concerns (better engineering of our software):
    - Lean and clean mathematical models: *just compute the equations!*
    - Separate and abstract: management of memory + manycore parallelism

Ralf Gruber

- BLAS2-based HPL would give a completely different TOP 500.

Mike Heroux

- CSE apps should deliver an (optimal) answer, plus error bars.
  - True exascale not needed, cluster of petascale instead.
- Mixed precision computation: one-time fix, perpetual gain.
- We will use GPUs.
- Threading under MPI:
  - Will happen, lots of work, only getting started.
  - Reduces MPI demands, memory/core.
  - 1M cores: 10K nodes, 100 cores/node.
  - Sequential consistency a big issue.
- Pushing reliability onto apps:
  - We may push back, or retreat altogether (no thanks).
  - GMRES: 1 bad value in 350M FLOPS: completely wrong result.
  - Separation of concerns: Algorithms expression vs. correct values.
- Ways to fail:
  - Application resilience is major issue if systems get too flaky.
  - Number of true exascale apps is small.

Bob Numrich

- "Main memory design was the controlling factor in determining the rest of the machine's design." von Neumann and Goldstine ~1946
  - If memory is, and always has been, the problem, why do we try to solve the problem by increasing the peak flop/s while making the memory system more and more complicated and less and less controllable?
  - Why not a single core with two levels of memory that the programmer can control: one on-chip (not cache) balanced with the functional-unit pipelines; one off-chip, with instructions that transfer data, asynchronously, between the two.
  - High-performance, single-core algorithms explicitly control memory management and satisfy:
    - » f_1 flop/word in algorithm
    - » f_2 flop/word in hardware: too big on current machines!
    - » f_2=(2 Gflop/s)/(1 Gbyte/s/(8 byte/word))=16 flop/word
    - » n_L words loaded from memory
    - » n_S words stored to memory
    - » $n_L >> n_S \rightarrow f_1 > 16$ flop/word
- Programming languages are not the problem.
  - You can write good code (or bad code) in any language if you can control the hardware.
  - Uncontrollable hardware makes it very difficult to write good code in any language.
  - Compiler technology should be able to produce more than 5-10% of single-core performance.
  - Is there something wrong with the way compilers are developed?
  - New languages with an excess of abstraction make the problem worse not better.
  - What part of a new algorithm is "new" when it must be forced onto the 25 year "old" message-passing model?
- If vendors can't be profitable building HPC-specific systems, then we need to treat HPC-specific systems as a strategic national investment for a higher good (climate, energy, materials?).
- Are solutions for these computational problems important enough to pay for or not?

Trey White

- Algorithms that scale
  - The very few that still need to grow their physical domain
- New algorithms for qualitative advances
  - Growing from predictive simulation to optimization (design, decision)
- Algorithm research and development
  - Trade off flops for everything else
  - Minimize **number** of communication messages
  - Hierarchical generalizations to checkpoint/restart
  - Built-in analysis, diagnostics, verification
- Other challenges
  - Faults of all kinds: hardware, software, cosmic rays, race conditions
  - Dot products: growing cost, irreproducible, hardware solution?
  - Scaling software for pre- and post-processing
- Implications for system designers
  - Since when do algorithms influence system designers?
  - Accelerated, reproducible dot products (allreduce)
  - Parallel **asynchronous** I/O