A Manifesto for Supercomputing Research

Marc Snir

The Successes

- Peak performance of top supercomputers has grown faster than predicted by Moore's Law
  - average level of parallelism has increased
- Some important applications have found ways to leverage this performance
- A software infrastructure was developed to support programming on large scale parallel systems

The Failures

- No radically new HPC architecture idea made it into a real system in last 20 years
- Programming large scale parallel systems continues to be a pain in the neck, and is becoming harder
- Certain key apps are not well supported on current platforms
- parallel systems have become more loosely coupled and more complex
  - IO connected clusters of SMP's
  - relative internode bandwidth and latency are getting worse
  - deep memory hierarchies
  - complex languages and runtimes
  - parallel software environments provide no real support for managing this complexity

HPC Advocacy

- Need to repeatedly argue the need for HPC systems
- Many do not believe that computational scientists are constrained by lack of performance
- Many do not believe that there are scientific/engineering problems that cannot be mapped on current cluster of SMPs supercomputer architecture
- Research community prefers the challenge of programming a complex architecture (e.g. grid), rather than building hw that reduces the challenge

The Goal of HPC Research

- It's not about building Petaflop/s systems, it's about having them earlier, cheaper, and easier to use
- It's not only about new nifty architecture ideas, it's about a sustainable process to translate those into viable products
  - The lack of a viable conduit for moving innovation to products is the main impediment to HPC research!

The Revolutionary Approach

- Start with a “clean sheet of paper” design
- Seed supercomputer companies to implement nifty ideas and let them swim or sink on their own (e.g., Thinking Machines)
- They sink…
  - optimistic, hardware centric view of HPC market
    - flops/ops vs. time to solution
  - tendency to become stuck in perpetual revolution
  - NO KILLER APP

The Evolutionary Approach

- Pay a server company (IBM, Compaq) to “stretch” their server models
- They build supercomputers that look liked stretched commercial servers…
  - done by company for prestige, at much lower ROI than main product line
  - heavy sharing of development with server product, with no proper accounting of real cost & lost opportunities
  - ➪ shaky foundation and strong pressure to dumb down product

Needed

- Sw compatibility and rough “performance compatibility”
- Heavy use of “COTS” technology (hw and sw)
  - Vendor microprocessors (or cores)
  - Vendor IO subsystems
  - Open source and boutique vendor HPC sw
- System integrator that focuses on few HPC unique technologies and on integration
- Long-term contracts

Non HPC unique technologies

- How to use 1B (or 10B) transistors on a chip
- How to overcome the memory wall
- How to reduce power consumption
- PIM technology applies to each of these problems
  - PIM technology will mature faster if focused on these problems
  - “killer app” will not be high-end servers…nor supercomputers

HPC Unique Problems

- Tight coupling at the level of > 10,000 execution threads
  - for problems that require it
  - in order to simplify programming model
- A programming model that is better than herding 10,000 cats

Communication Requirements of Problems

- Theoretical “surface to volume” argument
  - FFT: n communication for V(n) = n log n computation
  - Blas3: n communication for V(n) = $n^{3/2}$ computation
  - “ASCI class problems” V(n) = $n^{4/3}$?
    - B – module external communication rate (bandwidth)
    - M – module storage
    - C – module internal compute rate
- **Balanced system**: V(M)/C = M/B (C/B = V(M)/M)
  - Fixed “byte per flop” rule of thumb makes sense only if assume fixed M or assume linear complexity
  - PIM theoretically reduces off chip bw requirements

Communication requirements of problems

- Time and space granularity of communication
  - space granularity (degree of comm graph; scatter-gather)
    - trade-off with communication (multi-stage permutation)
  - time granularity (latency & synchronization)
    - trade-off with parallel slack
- Predictability of communication
  - “communication prediction” can reduce latency
- Topology of communication (locality)
  - Need to manage locality algorithmically
  - Need to hide specific machine topology
  - ➡ Need programming notation that does both

Communication Requirements of Problems

- Recursive subdivision algorithms achieve good locality at each level of the system; but
- not easy to express using current programming models
  - because communications at different system levels are expressed differently
  - because recursion is not natural idiom
- need compiler support to flatten recursion and match division steps to machine thresholds
- probably need more than one idea…

Structured Parallel Programming

- Original argument (Goto Considered Harmful):

  Need programming notation where state of execution can be simply expressed

  - True for implicitly parallel models (e.g., OpenMP)
  - True for SIMD
  - False for MIMD
- Is efficient structured parallel programming possible?