TMC Panelist Intro

Matt Sottile // mott@galois.com
Background (academic)

- B.S. - Mathematics and Computer Science
- M.S. - Computer Science
- Ph.D. - Computer Engineering

- In all cases, focus on parallel computing.
- Focal points: computational science and parallel computer operating systems.
Background (professional)

- Los Alamos National Laboratory (8 yrs)
  - Parallel system design
  - Computational physics
- University of Oregon (3 years)
  - Computational neuroscience
- Galois, Inc. (2 years)
  - Language tools for computational science
Concurrency in Languages

• I’m very interested in concurrency and its implications at the language level.
Current activities

- Distributed memory parallel program skeleton generation for performance analysis.
- Program transformation tools for peta/exa-scale architectures.
- Concurrency features for Fortran 2008 and beyond.
- Studying parallel application performance perturbations due to system effects (aka, “OS Noise”)

- All are performed under DOE funding.
Multicore

• Vanilla multicore so far has been a familiar field, just on a single die.
  • Symmetric multiprocessors
  • Vector computing
  • Hierarchical memory systems
• Programming multicore feels like programming my trusty old Origin 2000.
  • Same tools (e.g., OpenMP)
Von Neumann’s Ghost

- Finding parallelism doesn’t scare me.
- The Von Neumann bottleneck does.

- Feeding parallel cores is hard.
Challenging trend: complicated memories

- Complex memory systems minus convenient hardware.
- STI Cell and GPUs first widespread taste of this.
- Multicore isn’t too bad. Manycore is hard.

- Distinction:
  - Multicore $\approx$ UMA SMP; NUMA possible soon
  - Manycore $\approx$ Partitioned memory hierarchies
Notes from GPUScA

• Recent meeting colocated with PACT’11
• Numerous talks showing performance graphs.
• A consistent theme: better performance came from better use of the memory system!
• Parallelism itself wasn’t hard to find - just how to partition and schedule it with respect to memory access.
Parallel memory usage is crucial

• If you can find parallelism,
• ...it is likely not embarassingly parallel,
• ...you’re going to have to feed the threads,
• ...they will have to work together,
• ...and the manner of feeding and collaborating decides performance.
• Case in point: GPU memory coalescing.
What we need

• Ability to reason about proximity of computations to the memory they use.

• Easier ways to deal with partitioned memory in languages designed with an assumed flat, globally addressable memory.

• Runtime systems.
RTS Is Very Important!

- Recent language successes in parallel space that are intimately dependent on an RTS:
  - OpenMP
  - Cilk
  - Parallel Haskell

- Common theme: runtime systems play critical role. (E.g., Cilk, Haskell use work stealing).