Programming Models for Parallel and Distributed Systems

Panel Session

ASPLOS X, Oct. 2002

Eric Brewer, UC Berkeley, Inktomi

Mary Hall, USC ISI

Maurice Herlihy, Brown Univ.

Kathy Yelick, UC Berkeley

Michael Scott, Univ. of Rochester (moderator)

How did we get here?

- Tons of work on parallel languages and models in the late 70s and 80s
- Some of it bad, but much of it good, from a conceptual point of view
- But nobody offered a *really* big win, and they often insisted on complete religious conversion
- We couldn't get good performance with good models, so we settled for good performance with poor models

And where exactly are we?

- To first approximation
  - nobody uses explicitly parallel languages (other than Java)
  - nobody uses parallelizing compilers
  - the tightly coupled and distributed worlds use completely different programming models
  - but they all take the form of library calls -- no significant language or compiler support

Moreover

- Programmers aren't getting any smarter
- The environment keeps getting messier
  - machines are growing more complex
  - e-commerce, the GRID, and technological trends are pulling parallel and distributed systems together
  - HPC is going mainstream

ASPLOS:
Architectural Sabotage of Programming Languages and Operating Systems

- ✓ relaxed memory models
- ✓ HW synchronization
- ✓ threaded and clustered processors
- ✓ deep memory hierarchies
- ✓ pefetching, self-invalidation, etc.
- ✓ communication /computation gap
- ✓ PIM
- ✓ sensor motes
- ✓ active disks
- ✓ user-level NICs
- ✓ programmable NICs
- ✓ etc., etc., etc.

Emerging Applications

- ✓ Simulation
  - weather, economics, biological systems, games
- ✓ Modeling and rendering
  - 3D photography, immersive virtual environments, augmented reality, telepresence, games
- ✓ Intelligent interfaces
  - vision / recognition / pattern matching / search
  - language and knowledge -- statistical speech models, modeling of user intent

→ These are HPC !

Belligerent opinions

- ✓ Pthreads are sort of ok
- ✓ MPI, OpenMP, and the M4 macros are not!
  - too hard to use
  - not applicable to non-HPC-style systems
- ✓ RPC/RMI is sort of ok
- ✓ Sockets are not!

- ✓ Parallel programming is (still) 'way, 'way harder than it "ought" to be.

Recommendation #1

- ✓ Recognize that both shared memory and message passing have a place
  - Shared memory good for passive communication
  - Message passing good for active communication
  - Much of the time you can use either (matter of taste)
  - Sometimes you *need* one or the other; witness
    - events in shared memory systems
    - put() and get() in message passing systems
  - → Many applications would benefit from *both*

Recommendation #2

- ✓ Make the easy stuff easy
  - coherent shared memory (CC-NUMA, S-DSM, or DSS) for
    - fast prototyping
    - non-performance critical code
  - global address space with put() and get() for performance tuning of shared memory code
  - transactions for atomic update (Maurice)
  - 2-ended message passing for active communication

  - These models *can* comfortably co-exist within a single application

Recommendation #3

- ✓ Leverage the compiler
  - parallelize what you can (Mary)
  - provide language support for the explicitly parallel stuff (Eric)
    - typesafe communication
    - invariant checking
    - exception handling
    - transactions
    - data placement when necessary
    - parallel operations (e.g. loops)

Recommendation #4

- ✓ Don't embed HPC assumptions (Kathy)
  - can't afford to assume
    - fixed number of processes
    - single process per processor
    - uniprogrammed workloads
    - homogeneous hardware
  - must accommodate
    - availability / replication
    - fault tolerance / recovery
    - language and machine heterogeneity
    - distribution / mobility
    - persistence
  - potential big wins for grid computing *today*; enormous wins for pervasive / ubiquitous computing tomorrow

Closing Thoughts

- ✓ HPC is going mainstream
- ✓ The applications will be *really* exciting
- ✓ We need programming models that merge
  - explicit and implicit parallelism
  - shared memory and message passing
  - data-parallel and distributed / mobile components
  - automatic and manual locality management
- ✓ These ideas are mutually compatible