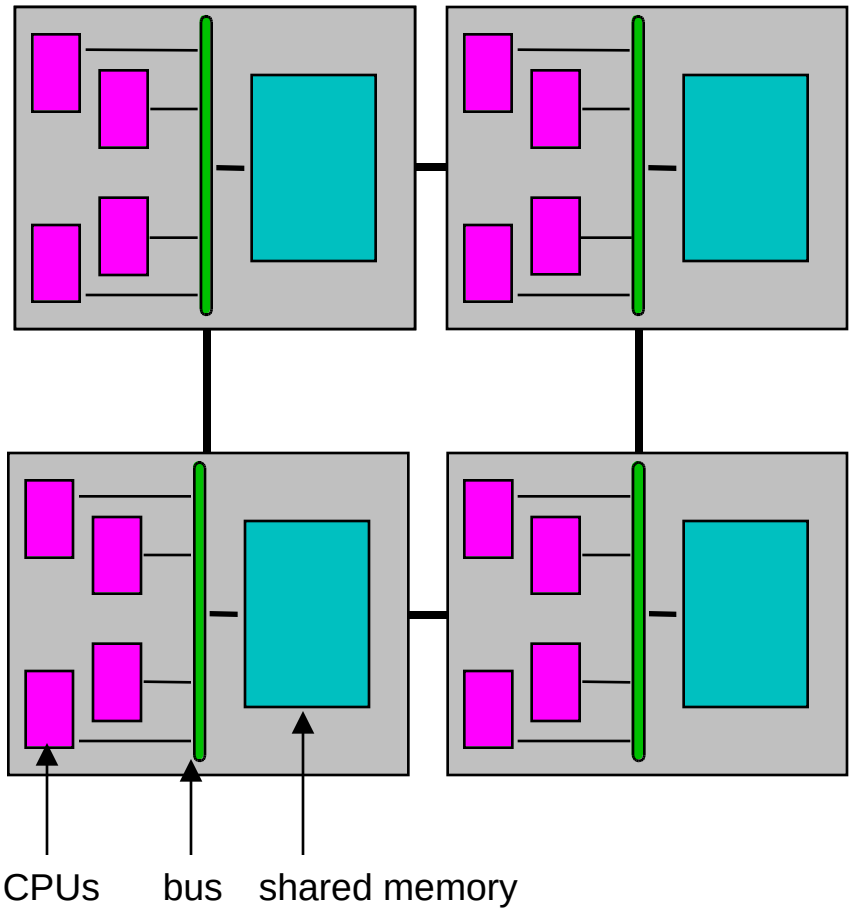
Experiences with Mixed MPI and Threaded Programming Models

John M. May and Bronis R. de Supinski

Center for Applied Scientific Computing

What is the best programming model for shared memory clusters?



- How do different programming models affect application performance?
- How do hardware characteristics affect application performance?
- How much work does it take to move to a new programming model?

Outline

- Hardware environment
- Mixed programming models
- Three mixed-model codes
  - Ares: regular-grid hydrodynamics
  - Ardra: neutral particle transport
  - JEEP: molecular dynamics
- Observations on mixed models

Hardware environment

- IBM “Tech Refresh” system
  - 168 four-way SMP nodes
  - Message passing can use “User Space” or “IP”
  - No global shared memory
- DEC cluster
  - Typical configuration has 80 CPUs divided (not equally) among 8 nodes
  - Message passing can span all CPUs, although most runs are not over the full system (capacity, not capability)

Programming models: Pure MPI

- Each CPU runs a separate MPI job
- Requires no changes for existing MPI codes
- Does not take advantage of shared memory
- Message passing is theoretically less efficient than shared memory
- Until recently, IBM didn’t support multiple user-space tasks on the same node

Programming models: Threads

- Explicit thread libraries (pthreads)
  - Maximum control
  - Most difficult threading model
- Parallelization directives
  - Examples include OpenMP and IBM SMP
  - Implies loop-level parallelization
  - Works best when loop iterations are large and independent
- Threaded libraries
  - If code spends substantial time in standard function (e.g. Fourier transform), can look for threaded version
  - Easiest to use
  - Performance depends on time spent in library

Mixed programming models

- Use MPI for internode communication and a threaded model within nodes
- Requires a thread-safe MPI library
  - Not all threads may communicate, but MPI may call non-thread-safe functions (e.g. malloc)
- Allows (one hopes) efficient use of shared memory
- Other models for multilevel parallelism exist (e.g. KeLP) but they aren't used in our codes

- Three examples from LLNL: Ares, Ardra, JEEP

Ares: Regular grid hydrodynamics

- Domain overloading
  - Decomposes problem space into more domains than processes
  - Improves load balance and cache use
  - Makes threading straightforward
  - One thread handles all MPI communication
- Initial threading used pthreads
  - Thread pool kept threads alive between phases
  - Complicated to program
- Switched to parallelization directives when available
  - Much easier to manage
  - No significant performance difference

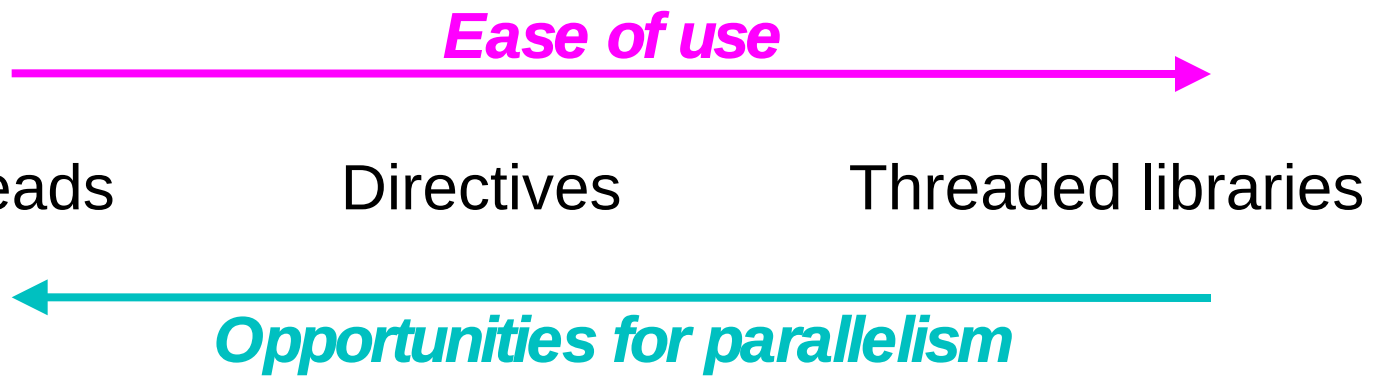
Ardra: Neutral particle transport

- Block-structured mesh
- About half the time spent in a “sweep” phase
- Programming model is data-driven for both message passing and threads
- Explicit threads scan work queue; compute each data point when its input dependencies are satisfied
- Initially used a thread pool; now spawn threads as needed
- Code is not well-suited for loop-based parallelism

JEEP: Molecular dynamics

- Computes molecular interactions at quantum-mechanical level
- Solves Schrödinger equation at each time step
- Message passing code is parallelized over electronic states
- Each state requires Fourier transform computations
- Fourier transform done with IBM's multithreaded ESSL code
- Easy way to get multithreading; speedup is limited

Conclusions



- **For loop-level parallelism, it's better to parallelize one outer loop than many inner loops, if possible**
- **Directive-based compilers are maturing**
- **No hard data yet on performance of threads vs. pure MPI; initial results are mixed**

Acknowledgements

- Thanks to code developers
  - Ares: Brian Pudliner
  - Ardra: Ulf Hanebutte
  - JEEP: François Gygi

- Work performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48, UCRL-MI-133179