Hybrid Programming: Preparing for Exascale

William Gropp
www.cs.illinois.edu/~wgropp

What This Talk is Not

- A tutorial on using the Hybrid Model
- A comprehensive discussion of MPI and OpenMP issues (I will use MPI+OpenMP to *illustrate* the issues)
- A pitch for a new programming model (even though there are cool things in MPI-3)

Rather, this talk is about

- Why hybrid models are important for HPC
- Opportunities and issues with hybrid programming
- What you should start doing (if you haven't already)

What is a Hybrid Model?

- Combination of several *parallel* programming models or systems in the same program
  - May be mixed in the same source
  - May be combinations of components or routines, each of which is in a single parallel programming model
- MPI + Threads or MPI + OpenMP is the most familiar hybrid model that involves MPI
  - There are other interesting choices for which we should prepare, including combinations of so-called domain specific languages

Why a Hybrid Model: The Hardware

- Scale of machines to come encourage the use of different programming models to address issues such as
  - Declining memory per core
  - Multiple threads/core
  - Load balance
  - Algorithmic issues
- Hardware will be specialized for cost/power/reliability reasons
  - No evidence that we can pretend a system is uniform and still get good performance from it
- Hardware will be (roughly) hierarchical
  - Number of “nodes” similar to current (10-100k)
  - Multiple levels of hierarchy (“sea of functional units”)
  - Number of “cores” per node will be 1k-100k

Why a Hybrid Model: The Software

- Already common and effective
    - ♦ MPI is already a hybrid programming model (MPI + C; MPI + Fortran)
        - Adding a third programming model is not a major change…
    - ♦ Many applications are multilingual, built from pieces in C, C++, Python, Matlab, …
- Developers use the best tool for each part of their program
- Complexity (if well designed) is additive
    - ♦ Putting everything in one model either limits capability or has *greater* complexity (multiplicative).

Why We Can't Pretend Everything is Simple

- It would be nice to adopt a simple homogenous abstraction, even though the hardware is more complex, and let the "system" handle the details, and let the scientists concentrate on the science.
- Unfortunately, we don't know how to do this. Worse: We ***know*** that we don't know – in much simpler situations, we have *given up already*
  - BLAS – why are there *any* optimized BLAS? Can't the compiler handle them?
  - The answer, terrifyingly, is **no**
- We must make virtue of necessity – can use use a compositional/hybrid approach to help *solve* these problems

Myths About MPI

- MPI is a programming model
  - No. Message passing is a programming model. MPI is a programming system that implements message passing and other programming models
- MPI is a bulk synchronous programming model (or system)
  - No. This was never true. However, data parallel and bulk synchronous programming are one route to high productivity programming (just look at MapReduce)
- Asynchronous Put/Get is something that MPI doesn't have
  - No. Defined in MPI 2.0; significantly extended in MPI 3.0. Unlike some put/get systems, MPI's has well-defined semantics

Myths About the MPI + OpenMP Hybrid Model

1. Never works
   - Examples from FEM assembly, others show benefit
2. Always works
   - Examples from NAS, EarthSim, others show MPI everywhere often as fast (or faster!) as hybrid models
3. Requires a special thread-safe MPI
   - In many cases does not; in others, requires a level defined in MPI-2
4. Harder to program
   - Harder than what?
   - Really the classic solution to complexity - divide problem into separate problems
     - 10000-fold coarse-grain parallelism + 100-fold fine-grain parallelism gives 1,000,000-fold total parallelism

Special Note

- Because neither 1 nor 2 are true, and 4 isn't entirely false, it is important for applications to engineer codes for the hybrid model. Applications must determine their:
  - Memory bandwidth requirements
  - Memory hierarchy requirements
  - Load Balance
- Don't confuse problems with getting good performance out of OpenMP with problems with the Hybrid programming model ("Use MPI + OpenMP *well*")
- See *Using OpenMP* by Barbara Chapman, Gabriele Jost and Ruud van der Pas, Chapters 5 and 6, for programming OpenMP for performance
  - See pages 207-211 where they discuss the hybrid model

Where Does the MPI + OpenMP Hybrid Model Work Well?

- Compute-bound loops
  - Many operations per memory load
- Memory bound loops
- Fine-grain parallelism
  - (New) Algorithms that are latency-sensitive
- Load balancing
  - Similar to fine-grain parallelism; ease of moving data/tasks + overdecomposition

Implications for Exascale Hybrid Programming Systems

- Off-node programming system between nodes.
  - ♦ Focus on scaling, locality, RDMA
- On-node programming system within node/sea of functional units
  - ♦ Focus on exploiting memory, ILP, direct hardware access to resources
- Challenges include
  - ♦ Hybrid models must work well together (sharing resources)
  - ♦ Managing user data structures
    - Most complaints about MPI usability are about what MPI *doesn't* have: support for distributed data structures

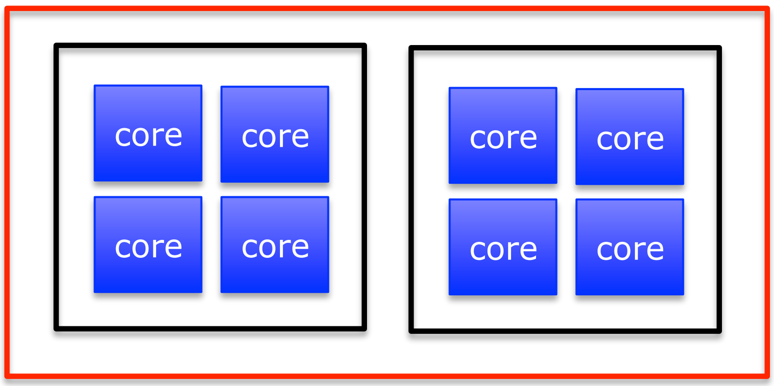
Where is Pure MPI Better?

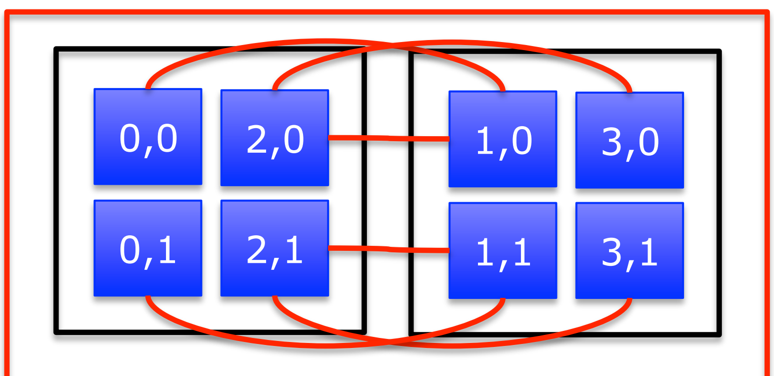
- Trying to use OpenMP + MPI on very regular, memory-bandwidth-bound computations is likely to lose because of the better, programmer-enforced memory locality management in the pure MPI version.
- Another reason to use more than one MPI process - if a single process (or thread) can't saturate the interconnect - then use multiple communicating processes or threads.
- Another option: MPI-3 with shared memory
  - MPI 3 permits processes to share memory directly; allows load/store access to data
  - This is ***still a hybrid model*** – just implemented within a single programming system (MPI-3)

Locality is Critical

- Placement of processes and threads is critical for performance
  - ♦ Placement of processes impacts use of communication links; poor placement creates more communication
  - ♦ Placement of threads within a process on cores impacts both memory and intranode performance
    - Threads must bind to preserve cache
    - In multi-chip nodes, some cores closer than others – same issue as processes
- MPI has limited, but useful, features for placement

Importance of ordering processes/ threads within a multichip node





- 2x4 processes in a mesh
- How should they be mapped onto this single node?
- Round robin (by chip)?
  - Labels are coordinates of process in logical computational mesh
  - Results in 3x interchip communication than the natural order
  - Same issue results if there is 1 process with 4 threads on each chip, or 1 process with 8 threads on the node

Challenges for Programming Models

- Parallel programming models need to provide ways to <u>coordinate</u> resource allocation
  - Numbers of cores/threads/functional units
  - Assignment (affinity) of cores/threads
  - Intranode memory bandwidth
  - Internode memory bandwidth
- They must also provide clean ways to share data
  - Consistent memory models
  - Decide whether its best to make it easy and transparent for the programmer (but slow) or fast but hard (or impossible, which is often the current state)
- Remember, parallel programming is about performance
  - You will always get higher programmer productivity with a single threaded code

Challenges for Developers

- Performance issues cannot be ignored
  - ♦ Must deal at least with an abstraction of a hierarchical or sea of functional units system
  - ♦ Model and algorithm must be chosen with awareness of the impact on performance
    - Make tradeoffs here, but know that you do
- Immature systems require dialog with developers and standard community
  - ♦ A good time to talk to OpenMP, MPI committees
- Growing complexity of code will require adopting approaches that distance you from the final code
  - ♦ Source to source transformation system
  - ♦ Abstract Data Structure Specific Languages (the name that should be used for DSL)

Conclusions

- Hybrid programming models exploit complementary strengths
  - In many cases today, can use OpenMP or OpenACC
  - Algorithms will need to (approximately) match hardware capabilities
- Evolutionary Path to Hybrid Models
  - Short term - better support for resource sharing
  - Medium term - better support for interoperating components
    - We need to ensure that communication infrastructures can cooperate
    - Consider extensions to make implementations aware that they are in a hybrid model program
  - Long term - Generalized model, efficient sharing of communication and computation infrastructure