

MPI and Hybrid Programming Models

William Gropp

www.cs.uiuc.edu/homes/wgropp



What is a Hybrid Model?

- Combination of several *parallel* programming models 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



Why a Hybrid Model?

- Note that in some ways MPI is already a hybrid programming model (MPI + C; MPI + Fortran)
 - ◆ Adding a third programming model is not a major change...
- Also note that many *applications* are multilingual, built from pieces in C, C++, Python, Matlab, ...
 - ◆ Developers are used to using the best tool for each part of their program
- 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
- Example of a Petascale Machine: Blue Waters...



Blue Waters

- NSF Supported Sustained PetaFLOP facility
 - ◆ www.ncsa.uiuc.edu/BlueWaters
 - All information here is available there
 - ◆ >200K cores (POWER7)
 - ◆ > 10PB User Storage
 - ◆ New Facility (provided by the University of Illinois)
 - ◆ Time primarily allocated by NSF through peer-reviewed proposals
 - ◆ Blue Waters is a capability machine
 - Will support a relatively small number of application groups



Location, Location, Location



Blue Waters Nodes

- Support Simultaneous MultiThreading (SMT)
- Provides VMX (sometimes called Altivec) (short vectors)
- Private L1 and L2 cache
- Shared L3
- At least 32 GB memory per SMP
- Interconnect fabric
 - ◆ Low latency, high bandwidth (of course)
 - ◆ Can overlap computation and communication
 - ◆ Supports RDMA



Blue Waters System Software

- Standard software, including
 - ◆ C/C++/Fortran
 - MPI (MPI-1 and MPI-2)
 - OpenMP
 - ◆ Partitioned Global Address Space (PGAS) languages
 - Unified Parallel C (UPC)
 - CoArray Fortran (CAF)
 - ◆ Eclipse-based development environment
 - ◆ GPFS Parallel File System
- Low-level active messaging layer
- Faults will not be a serious problem



Benchmark Applications on Blue Waters

- Three Benchmark Applications (Required by NSF)
 - ◆ Homogeneous Turbulence
 - Global FFTs
 - ◆ Lattice QCD
 - Many messages, one key operation is fast Allreduce
 - ◆ Molecular Dynamics
 - FFT (P-M Ewald) + pair forces
- Challenges include
 - ◆ Scaling
 - Concurrent threads
 - Load balance
 - Latency and Overlap
 - ◆ Maximizing use of resources
 - Overlapping communication and computation
 - Remember, any significant machine is hundreds to thousands of clock cycles across



Real Applications on Blue Waters

- No one algorithm or application area will dominate
- Likely be adaptive
 - ◆ Load balancing required, irregular data distributions
- May be hierarchical (in groups of processes)
 - ◆ Scalable definition of collections of processes (teams in some models) that are smaller than MPI_COMM_WORLD
- Likely to be multi-model
 - ◆ Multiple code modules, one for each model
 - ◆ For better overall scalability, each model may run on a (very large) subset of cores
- Likely to be multi-scale
 - ◆ In structure, often like multi-model
- Real applications do I/O
 - ◆ Zillion files per job a significant burden on everyone (system and users)
 - ◆ Few parallel programming models *except* MPI have efficient, scalable I/O capability



Programming Models for Petascale Systems

- Many (most?) Petascale applications have *already* been written.
- All MPI
 - ◆ Requires > 200k processes
 - ◆ Does not exploit SMP
 - ◆ MPI overheads
- All CAF or All UPC (PGAS)
 - ◆ Little to no application code
 - To be awarded time on leading systems such as Blue Waters, applications must demonstrate that they are ready to run
 - ◆ Virtually all parallel software libraries use MPI (distributed memory) or Threads/OpenMP (single SMP)
- Other possibilities for petascale systems in general include Distributed Memory OpenMP
 - ◆ Still a work in progress



Other Options: Hybrid Models

- MPI Everywhere *will* work on this system
 - ◆ Actual size of MPI library code loaded within the application is modest
 - Even if statically linked, the code can be kept small. Proven on IBM Blue Gene/L.
 - ◆ Most MPI routines are scalable by design
 - Exceptions include Alltoall, Allgather, Allscatter, Graph_create
 - Internal data structures are not *required* to have an allocated entry for every process (the implementation need only ensure that MPI works)
- Given the SMP nodes, exploiting those nodes with the MPI + OpenMP or MPI + Threads model is natural
 - ◆ But is it a good idea?



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 to 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
- See *Using OpenMP*, 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; following pages with some results.



Some Things to Watch for in OpenMP

- No standard way to manage memory affinity
 - ◆ “First touch” (have intended “owning” thread perform first access) provides initial static mapping of memory
 - Next touch, mentioned yesterday, could help
 - ◆ No portable way to reassign affinity - reduces the effectiveness of OpenMP when used to improve load balancing.
- Memory model can require explicit “memory flush” operations
 - ◆ Defaults allow race conditions
 - ◆ Humans notoriously poor at recognizing all races
 - It only takes one mistake to create a hard-to-find bug



Some Things to Watch for in MPI + OpenMP

- No interface for apportioning resources between MPI and OpenMP
 - ◆ On an SMP node, how many MPI processes and how many OpenMP Threads?
 - Note the static nature assumed by this question
 - ◆ Note having more threads than cores important for hiding latency
 - Requires very lightweight threads
- Competition for resources
 - ◆ Particularly memory bandwidth and network access
 - ◆ This morning's talks by Rolf Rabenseifner's - apportionment of network access between threads and processes



Where Does the MPI + OpenMP Hybrid Model Work Well?

- Compute-bound loops
- Fine-grain parallelism
- Load balancing
- Memory bound loops



The Details

- Compute-Bound Loops
 - ◆ This can happen in some kinds of matrix assembly, for example.
- Fine-grain parallelism
 - ◆ E.g., in blocked preconditioners, where fewer, larger blocks, each managed with OpenMP, as opposed to more, smaller, single-threaded blocks in the all-MPI version, gives you an *algorithmic* advantage (e.g., fewer iterations in a preconditioned linear solution algorithm).
- Load Balancing
 - ◆ Where the computational load isn't exactly the same in all threads/processes; this can be viewed as a variation on fine-grained access.
- Memory bound loops
 - ◆ Where read data is shared, so that cache memory can be used more efficiently.



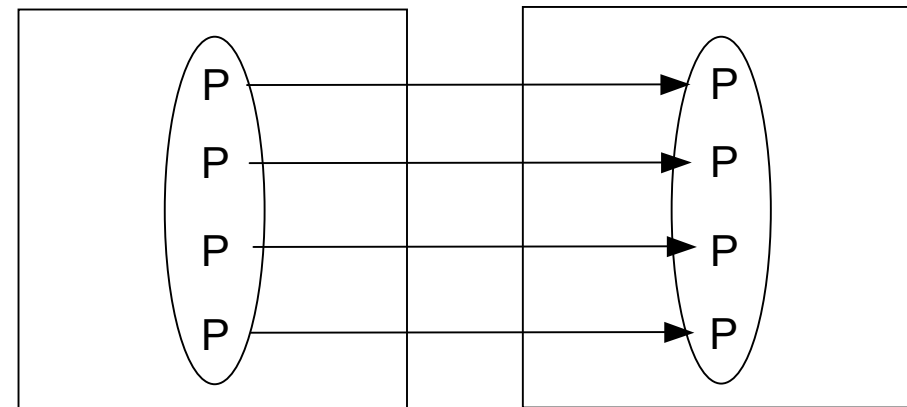
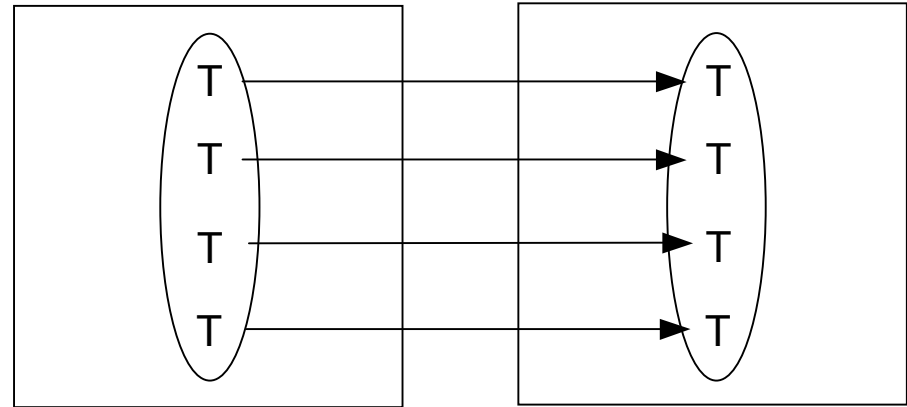
Where is Pure MPI Better?

- 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.
 - ◆ Note that threads and processes are not equal - see next slides

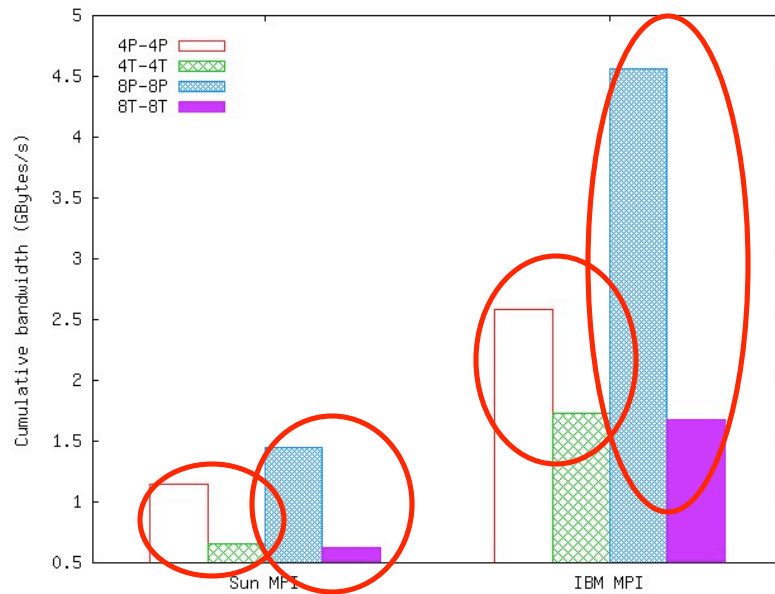
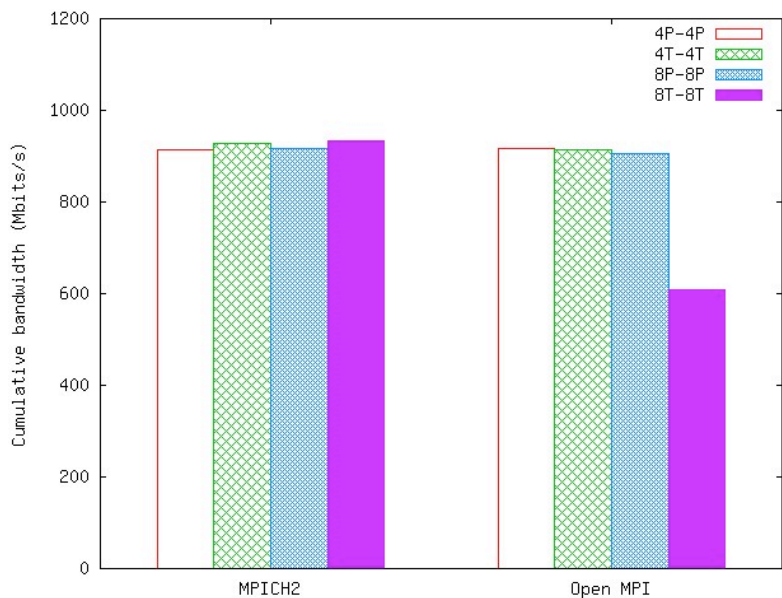


Tests with Multiple Threads versus Processes

- Consider these two cases:
 - ◆ Nodes with 4 cores
 - ◆ 1 process with four threads sends to 1 process with four threads, each thread sending, or
 - ◆ 4 processes, each with one thread, sending to a corresponding thread
- User expectation is that the performance is the same
- Results are joint work with Rajeev Thakur (Argonne)



Concurrent Bandwidth Test



Lesson: Its hard to provide full performance from threads



PGAS Programming Models

- Partitioned Global Address Space (PGAS) languages provide an alternative parallel programming model
 - ◆ Provide a global name space for objects
 - Acknowledge that local and non-local objects have different performance behavior that should be visible in the language
 - Demonstrated performance benefits in some cases
 - E.g., "Productivity and Performance Using Partitioned Global Address Space Languages", Yelick et al
 - ◆ UPC (Unified Parallel C) adds "shared" pointers to C
 - ◆ CAF (CoArray Fortran) adds "co arrays" - explicitly distributed arrays
 - Use the "co-array" dimension to reference the part of the co-array on a specific process
- One weakness in the model is that data is either local or global
 - ◆ No "regional" notion
 - ◆ Some support for subsets of processes, but not fully integrated into the language
- Can we combine these in a hybrid model with MPI?



Some Historical Background

- MPI + HPF (High Performance Fortran)
 - ◆ Let MPI make use of HPF arrays
 - ◆ See “MPI as a coordination layer for communicating HPF tasks”, Ian T. Foster, David R. Kohr, Rakesh Krishnaiyer, and Alok Choudhary
 - ◆ Points out one of the first problems in these hybrid models: what does MPI do with an HPF distributed array?



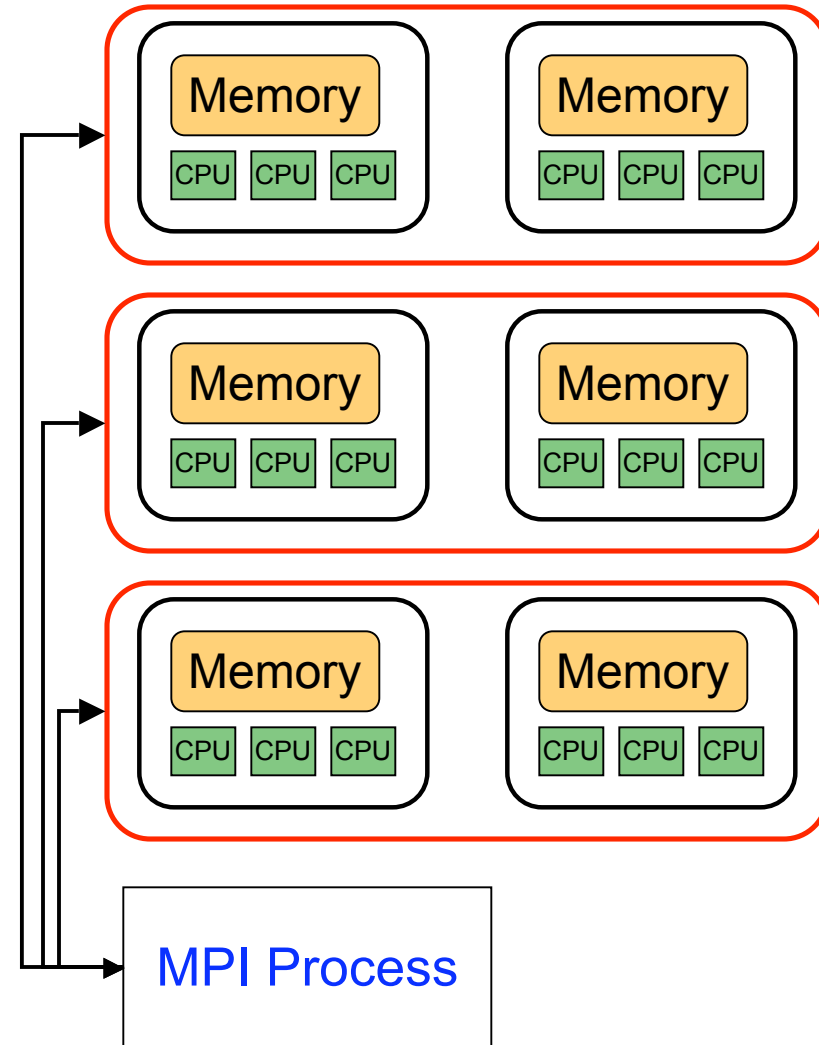
More Historical Background

- MPI on SMP
 - ◆ Typical implementation
 - Use regular processes for each MPI process, use special services to share memory between the processes
 - ◆ An alternative
 - Use each MPI process on the SMP is a thread that is part of a single operating system process
 - See “Optimizing threaded MPI execution on SMP clusters”, H. Tang and T. Yang
 - Must use a special compiler
 - Global variables in the user program must be thread-private by default
- Question: should we rethink the identification of MPI processes with OS Processes?



Generalized MPI Processes

- ◆ Let an MPI “Process” span multiple nodes
 - Solves the memory problem
 - Provides a way to address the local/global problem
- ◆ Issues
 - What does a send with a remote pointer mean?
 - What is the address space for an MPI process?
 - Initializing - who is in charge?
 - Programming model support for the MPI “Process”
 - Distributed OpenMP?
 - UPC? CAF?



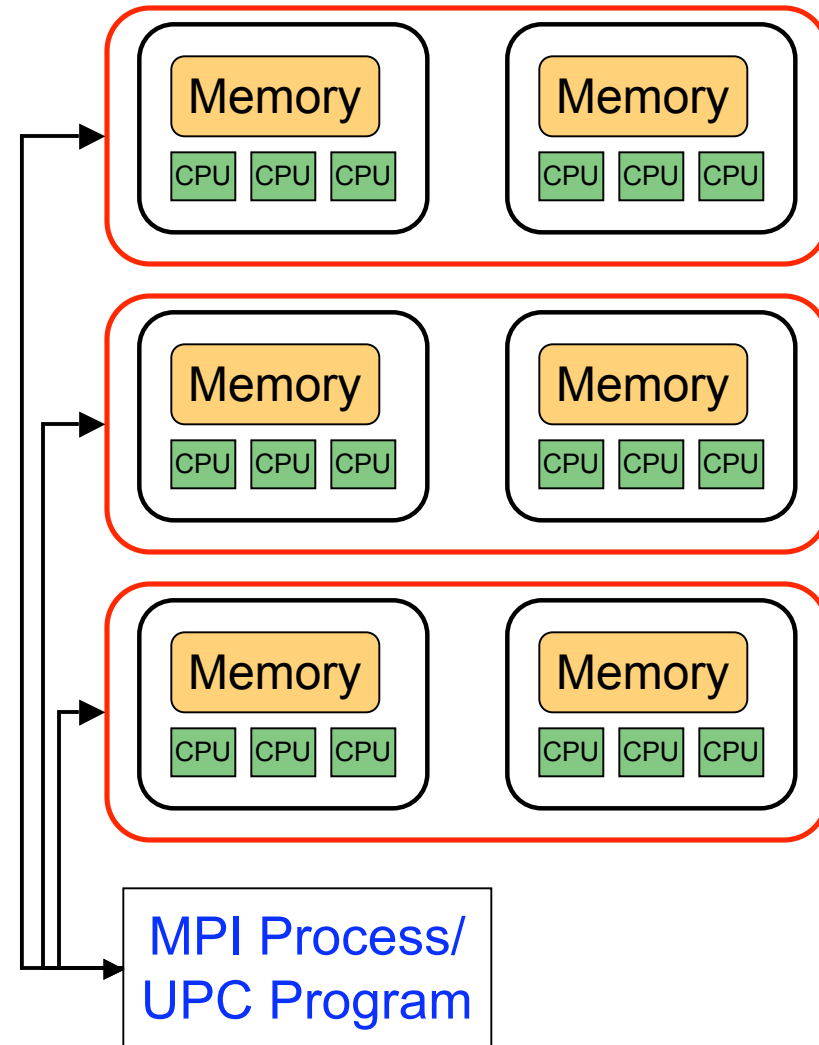
More General MPI Hybrid Programming Models

- Why consider the Hybrid Model with PGAS or other programming models?
 - ◆ Load balancing
 - ◆ Shared data (reduce memory pressure, particularly for processor-rich (and hence memory poor) nodes)
 - ◆ Component software (use the best programming model to implement a component)
 - ◆ OpenMP and MPI understood
 - ◆ What about others: MPI/UPC (or PGAS) interoperability
- The following is based on discussions at a “Workshop on collective communication primitives in PGAS and SPMD languages”, May 2008, IBM Hawthorne
- Possible combinations for MPI and UPC (or other PGAS) languages include:
 - ◆ MPI processes are UPC programs
 - ◆ MPI processes are UPC threads
 - ◆ UPC Programs are combined into MPI programs



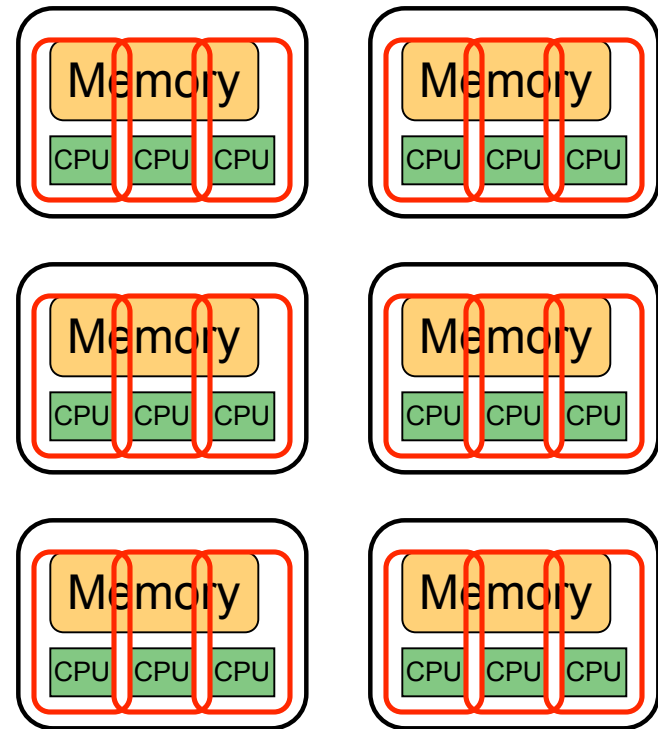
MPI Processes are UPC Programs

- MPI Processes are UPC programs (not threads), spanning multiple nodes. This model is the closest counterpart to the MPI+OpenMP model, using PGAS to extend the "process" beyond a single node. (An MPI process need not be an OS process).



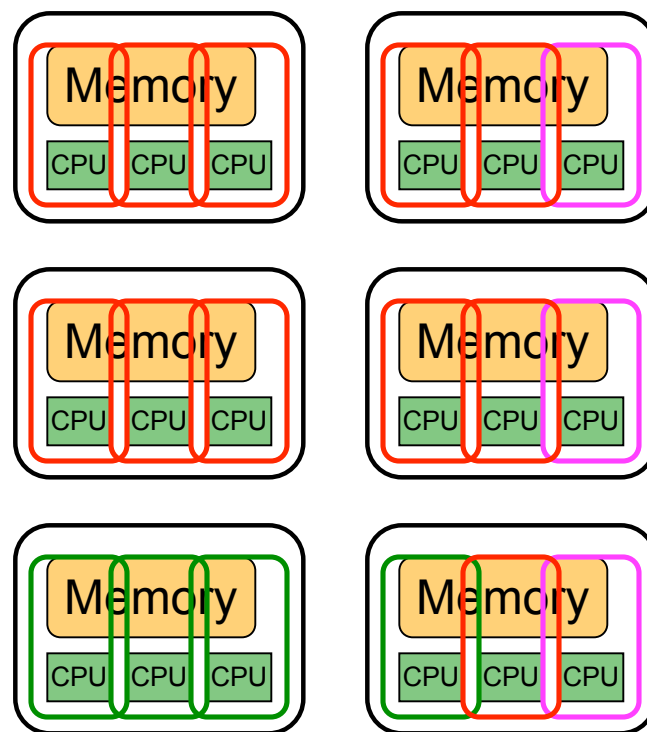
MPI Processes are UPC Threads

- The program starts as a single UPC program. Each UPC thread calls `MPI_Init` (or `MPI_Init_thread`). The process management system must permit UPC programs to use `MPI_Init` to also become MPI programs.
- The program starts as a single MPI program (started with `mpiexec`). UPC is initialized somehow
 - ◆ UPC initialized explicitly with a routine call
 - ◆ UPC initialized implicitly because UPC compiler knew this was an MPI + UPC program



MPI Processes are UPC Threads (con't)

- The MPI program is tiled with separate UPC programs. That is, every MPI process is also a UPC thread, but not all MPI processes belong to the same UPC program.
 - a) The UPC programs are created from MPI subcommunicators with an explicit call, e.g., add a `upc_init(MPI_Comm)` (proposed by Marc Snir)
 - b) The UPC programs are defined at startup through interaction with the process management system; e.g., an extension to `mpiexec` defines how the MPI processes are tiled with UPC programs.
 - c) Like (b), but not all MPI processes correspond to a UPC thread. This is like (a) if not all MPI processes were to call `upc_init`.



The Program is a Collection of UPC Programs

- The program starts as a collection of separate UPC programs.
 - ◆ Use `MPI_Comm_connect/accept` to become an MPI program on all threads
 - ◆ Use `MPI_Comm_connect/accept` to become an MPI program on a subset of UPC threads



Likely Hybrid Models

- For the BW benchmarks (part of the NSF criteria) that may want to use UPC FFTs, then the model is most likely to be an MPI program that will call `MPI_Init`, but will also be UPC.
- Within each of the different relationship between MPI and PGAS processes/threads described above, there is a (nearly) orthogonal question of the interaction between MPI and the PGAS language. Such things include:
 - ◆ Can MPI use nonlocal/shared data or are MPI calls restricted to local data?
 - ◆ Do MPI operations have any memory consistency side effects (e.g., should an `MPI_Barrier` include a `upc_barrier`? What about `MPI_Win_fence`)?



Interactions Between Models

- MPI provides routines to help the programmer map processes to processors
 - ◆ The MPI implementation can take advantage of this to make best use of the interconnect (both inter and intra node)
- But MPI has no way to know what other programming models are making use of the hardware
 - ◆ OpenMP thread, UPC threads, CAF processes, etc.
 - ◆ Rabenseifner's talk this morning gave some excellent examples of the burden that this places on the programmer

4 Neighbor Exchange	BG/L	BG/L VN	BG/P	BG/P VN
World	199	120	328	132 (177)
Even/Odd	114	64	327	84 (84)
Cart_create	218	201	581	132 (177)



Challenges for Programming Models

- Parallel programming models need to provide ways to coordinate resource allocation
 - ◆ Numbers of cores/threads
 - ◆ 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 Implementations

- Sharing of communication infrastructure
 - ◆ For example, the Berkeley UPC implementation makes use of GASNET, an efficient, portable communication layer
 - ◆ But GASNET does not provide all of the features required by an efficient, full MPI implementation
 - ◆ Similarly, the communication layer used by MPI implementations may not provide all of the features needed by UPC (see “Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations”, Dan Bonachea, Jason Duell)
- It is possible to build such infrastuctures
 - ◆ But current examples only address some of the issues.
 - ◆ Resource allocation and sharing not covered



Conclusions

- Hybrid programming models exploit complementary strengths
 - ◆ Though not discussed here, programming models for accelerators is another hybrid model that is likely to be important (e.g., Road Runner)
- Evolutionary Path to Hybrid Models
 - ◆ Short term - better support for resource sharing
 - We need to experiment with specifying additional information, e.g., through mpiexec
 - ◆ 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

