

Techniques for Petascale Programming Models

Daniel Chavarría-Miranda, Jarek Nieplocha,
Vinod Tipparaju, Manoj Krishnan, Bruce Palmer

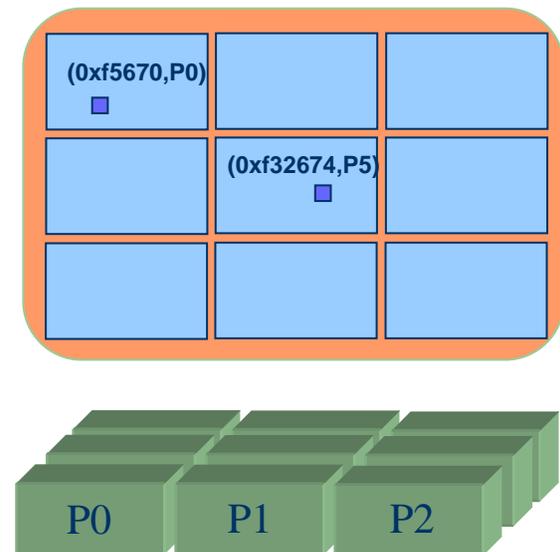
Pacific Northwest National Laboratory (PNNL)

Distributed Memory Programming Models

Distributed Data:

Data is explicitly associated with each processor, accessing data requires specifying the location of the data on the processor and the processor itself.

Data locality is explicit but data access is complicated.

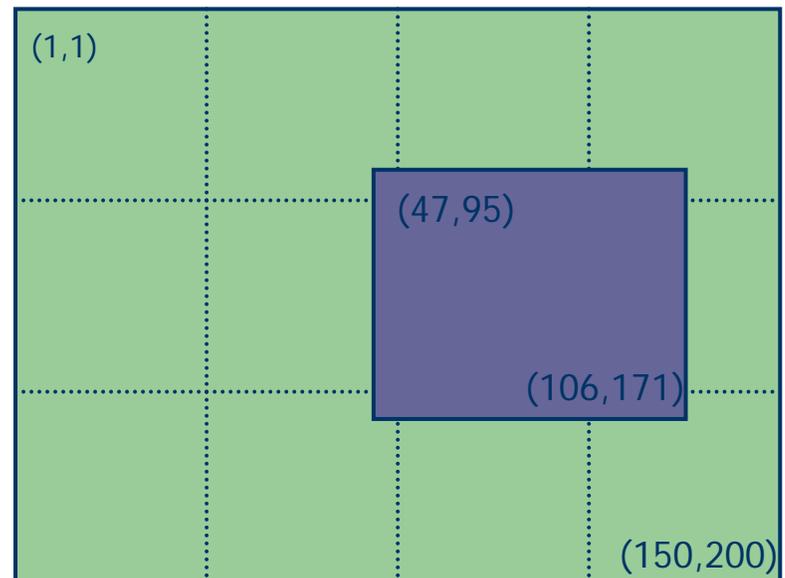


Shared Memory Programming Models

Shared Memory:

Data is in a globally accessible address space, any processor can access data by specifying its location using a global index

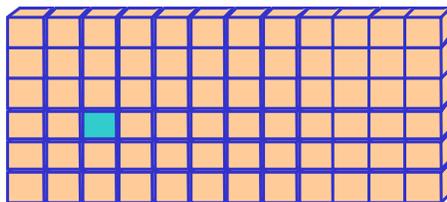
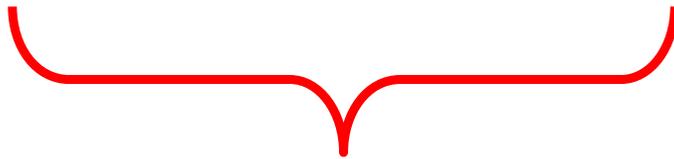
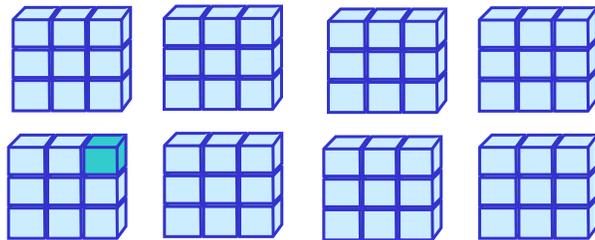
Data is mapped out in a natural manner (usually corresponding to the original problem) and access is easy. Information on data locality is obscured and can lead to loss of performance.



Global Arrays

Distributed dense arrays that can be accessed through a shared memory-like interface

Physically distributed data



Global Address Space

single, shared data structure/
global indexing

e.g., access $A(4,3)$ rather than
 $\text{buf}(7)$ on task 2

Global Arrays (cont.)

- *Global Arrays* (GA) implements a DSO (Distributed Shared Object) programming model
 - Implemented as a runtime library
 - Specialized for dense arrays
 - Efficient implementation on top of RDMA hardware on clusters
 - Uses one-sided communication (put/get)
 - Locality awareness is part of the programming model
 - Programs can efficiently determine which processor(s) owns a given array section, before issuing any remote operations
- Compatible & interoperable with MPI
 - A program can use GA operations as well as message passing
- Scalable to thousands of processors

GA Programming Model

- GA's programming model abstractions naturally lead to data-parallel programming
 - Each processor executes equivalent computation on different array sections
- GA currently uses the same process model as MPI (compatibility)
 - Heavy-weight UNIX processes are launched on processors on cluster nodes
 - Processes on the same physical node share memory through OS mechanisms
 - It's simpler to run the same executable on the processors
 - Naturally leads to Single Program Multiple Data (SPMD) programming

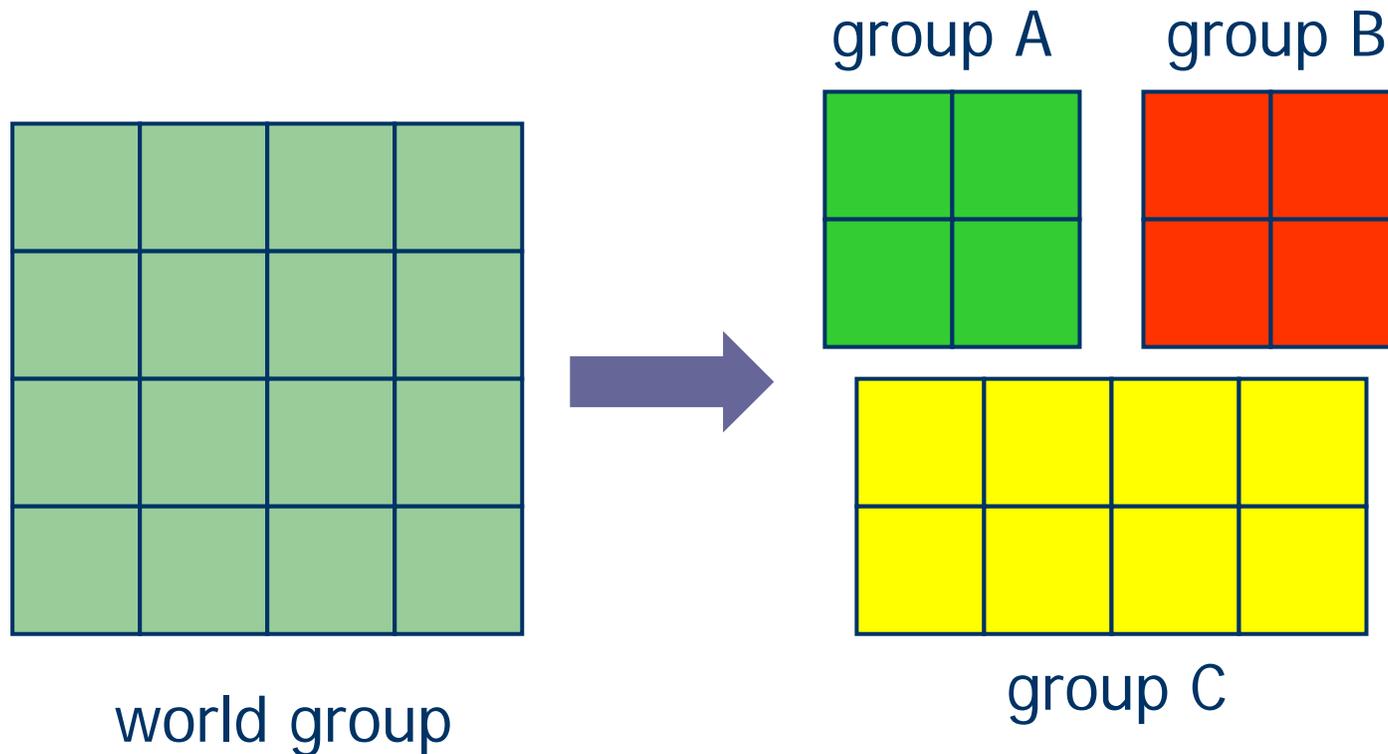
GA Programming Model (cont.)

- Differences between data parallelism and SPMD
 - Data parallelism implies that parallel execution is derived from distributing equivalent computation over different data sections
 - Coarse-grained SIMD (??)
 - SPMD implies that data is distributed onto multiple processors, each processor can take its own execution path through the code
 - A single executable image is used for all processors

GA Programming Model (cont.)

- Array-based data parallelism is a powerful mechanism for structuring parallel programs
 - However, there are limitations to scaling data parallel operations
 - In many domains there are limitations on how large can arrays be
 - Higher resolution meshes might not make physical/mathematical sense
 - There are mathematical constraints on how large can matrices be
- GA has introduced a mechanism to create *processor groups*
 - Processor groups are subsets of the number of processors on which the program is executing
 - Enhances non-data parallel capabilities while still maintaining SPMD advantages
 - Global array objects can now be universal or local to a processor group

Processor Groups

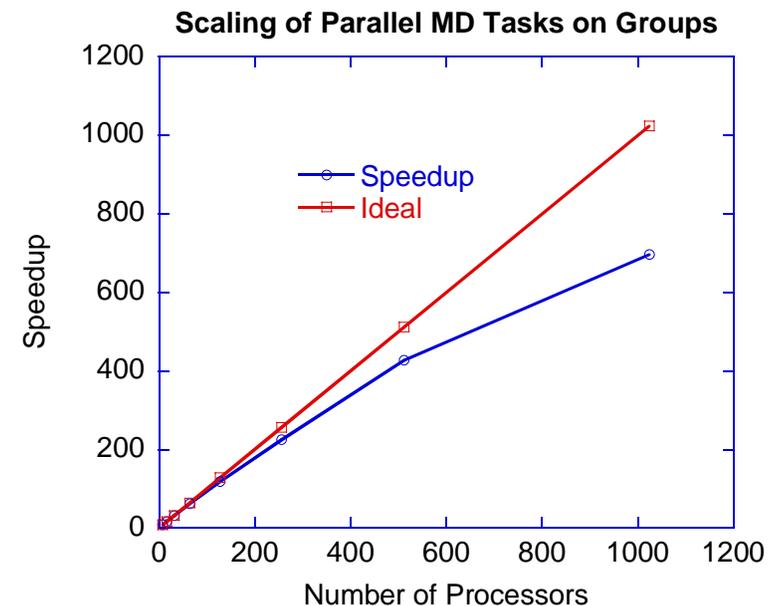
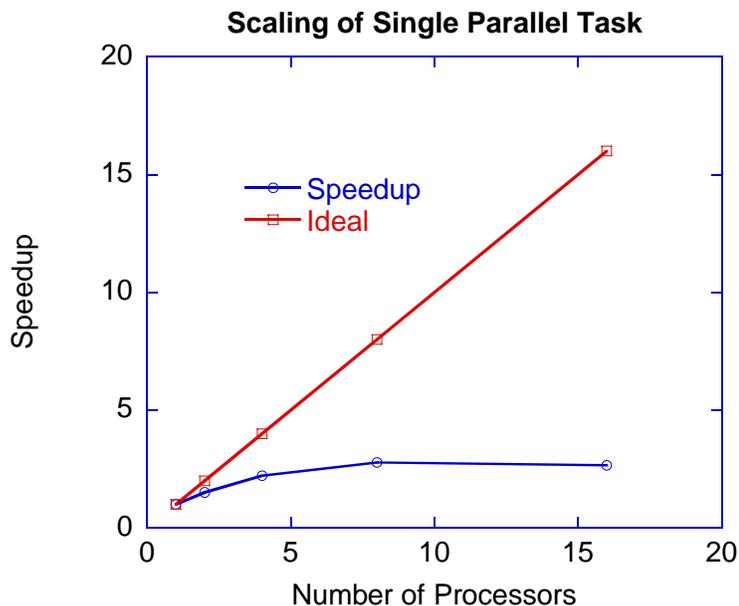


- GA provides the notion of a *default active processor group*
 - All non-group specific GA operations will apply only to the processors in the active group
 - Simplifies application development with groups

Processor Groups (cont.)

- Partitioning the set of processors into groups enables the use of data-parallel methods inside the group
 - Collective operations and synchronization can also be group-local
 - Loosely-synchronous coordination can be used across groups

MD Application on Groups



Processor Groups (cont.)

- Processor groups enable a hybrid task/data parallel programming style
 - There is coarse-grained task parallelism between finer-grained data-parallel processor groups
- Multiple advantages:
 - Scalability for the loosely-coordinated coarse tasks can be better
 - The tightly-coupled data-parallel tasks can be mapped to physical nodes in a cluster in a way that enhances locality
 - By restricting communication to smaller neighborhoods of nodes in the physical machine
 - “Group-local” collective operations don’t have to span large sections of a petascale machine
- Disadvantages:
 - Conceptual parallelism model is more complex for the programmer

Rethinking HPC Process Models

- The traditional MPI/GA-style heavyweight process model is no longer well-matched to hardware characteristics
 - Advent of multicore processors changes the node architecture of a supercomputer
 - Nodes are starting to have multiple sockets with multiple cores each
- It should be better to spawn a single heavyweight process per separate *address space* (node)
 - Spawn threads on computational cores
 - Enables simpler resource sharing between computational tasks
 - Data sharing between threads should be under the control of the programming model to minimize nasty bugs
 - Data races
 - Memory overwrites (single protection domain)

HPC Process Models (cont.)

- A threaded model for HPC applications enables more dynamic forms of parallelism at the node level
 - Dynamic tasks could be activated on demand
 - Due to local (node) requests
 - Due to remote (off-node) requests
 - Granularity of the tasks and efficient task creation/destruction should be critical design & implementation issues
- The number of tasks (threads) could be matched to the physical capabilities of the machine
 - Number of available cores on a node
 - More importantly, the needed memory and network bandwidth per task
 - Throttle the number of active tasks to maximize the use of bandwidth *in spite of having inactive cores*

Global Arrays & Threads

- GA currently utilizes threads for auxiliary operations
 - There is a single *helper thread* per node to handle certain data-parallel operations
 - The helper thread handles data-parallel operations over local array sections
 - For example, multiplying elements of an array section by a scalar
- We would like to extend GA's use of threads and expose a more general interface to them at the programming model level
 - Challenges in maintaining compatibility with existing operations
 - How do we create an API that is simple and productive for the scientific programmer?
 - While maintaining high performance and scalability