Porting and Optimizing Performance of Global Arrays Toolkit on the Cray X1

Vinod Tipparaju, Manojkumar Krishnan, Bruce Palmer, Jarek Nieplocha
Computational Sciences and Mathematics Department
Pacific Northwest National Laboratory
Richland, WA 99352

Outline

- Overview
- Global Array programming model
- GA Core capabilities
- X1 architecture, - a nice fit for GA model
- Latency/Bandwidth numbers and application performance

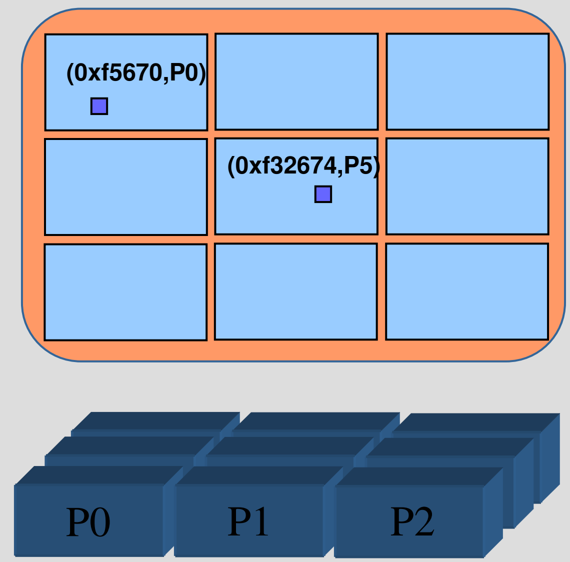
Overview

- X1 for us represents a shared memory architecture
- Programming models supported by Cray present it to applications as a distributed memory or a GAS system
- With GA, the programmer can view distributed data structure as a single object and access it as if it resided in shared memory.
- This approach helps to raise the level of abstraction and program composition as compared to the programming models with a fragmented memory view (e.g., MPI, Co-Array Fortran, SHMEM, and UPC).
- In addition to other application areas, GA is a widely used programming model in computational chemistry.
- I will describe how the GA toolkit is implemented on the Cray X1, and how the performance of its basic communication primitives were optimized

Distributed Data vs Shared Memory

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.
- Distributed computing is typically implemented with message passing (e.g. MPI)
- There is a potential to utilize extra processing capability on the nic to assist in moving data

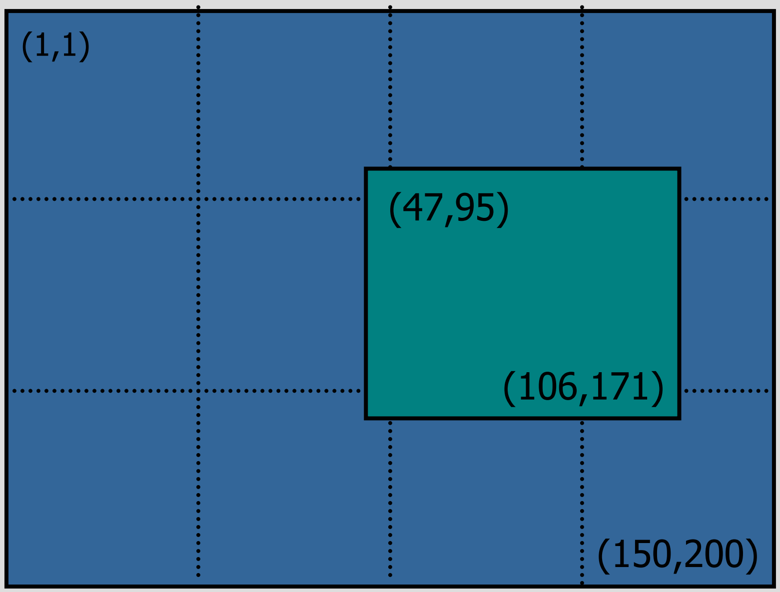


Distributed Data vs Shared Memory (Cont).

Shared Memory:

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

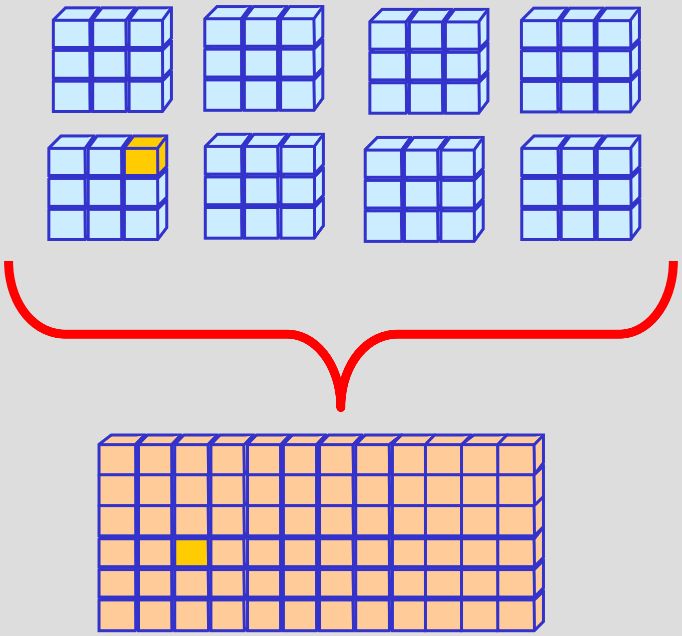
- Data is mapped out in a natural manner (usually corresponding to the original problem) and access is easy.
- Information on data locality can be obscured and leads to loss of performance.



Global Arrays

Distributed dense arrays that can be accessed in a shared memory-like style

Physically distributed data



Global Address Space

single, shared data structure/ global indexing

e.g., access A(4,3) rather than buf(7) on task 2

Global Arrays (cont.)

- Shared memory model in context of distributed dense arrays
- Level of abstraction that makes it simpler than message-passing to program in with out loss in performance
- Complete environment for parallel code development
- Compatible with MPI
- Data locality control similar to distributed memory/message passing model
- Extensible
- Scalable

Core Capabilities

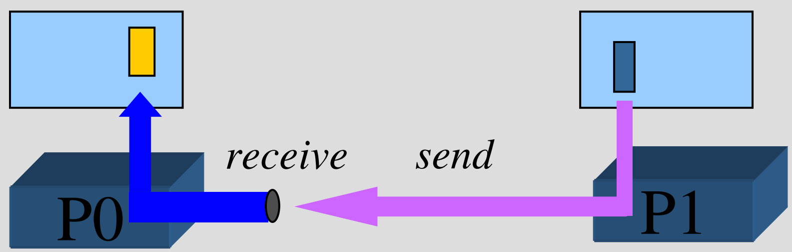
- Distributed array library
  - dense arrays 1-7 dimensions
  - four data types: *integer*, *real, double precision*, *double complex*
  - global rather than per-task view of data structures
  - user control over data distribution: regular and irregular
- Collective and shared-memory style operations
  - ga_sync, ga_scale, etc
  - ga_put, ga_get, ga_acc
  - nonblocking ga_put, ga_get, ga_acc
- Interfaces to third party parallel numerical libraries
  - PeIGS, Scalapack, SUMMA, Tao
    - example: to solve a linear system using LU factorization

```
call ga_lu_solve(g_a, g_b)
```

instead of

```
call pdgetrf(n,m, locA, p, q, dA, ind, info)
call pdgetrs(trans, n, mb, locA, p, q, dA,dB,info)
```

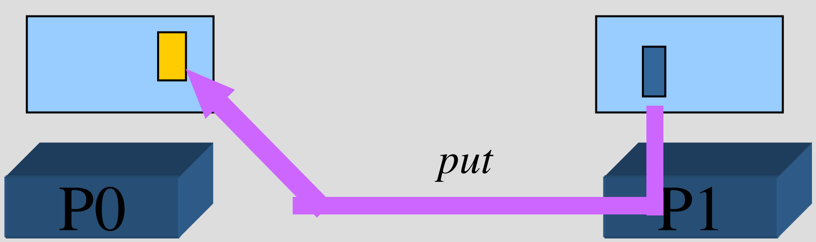
One-sided Communication



message passing
MPI

Message Passing:

Message requires cooperation on both sides. The processor sending the message (P1) and the processor receiving the message (P0) must both participate.



one-sided communication
SHMEM, ARMCI, MPI-2-1S

One-sided Communication:

Once message is initiated on sending processor (P1) the sending processor can continue computation. Receiving processor (P0) is not involved.

Remote Data Access in GA

Message Passing:

identify size and location of data blocks

loop over processors:
    if (me = P_N) then
        pack data in local message buffer
        send block of data to message buffer on P0
    else if (me = P0) then
        receive block of data from P_N in message buffer
        unpack data from message buffer to local buffer
    endif
end loop

copy local data on P0 to local buffer

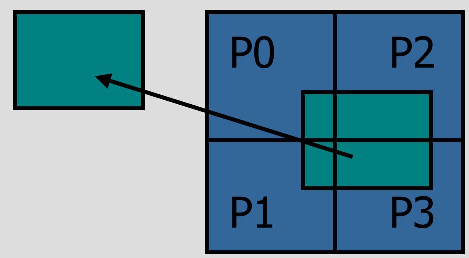
Global Arrays:

NGA_Get(g_a, lo, hi, buffer, ld);

Global Array handle

Global upper and lower indices of data patch

Local buffer and array of strides



Data Locality

What data does a processor own?

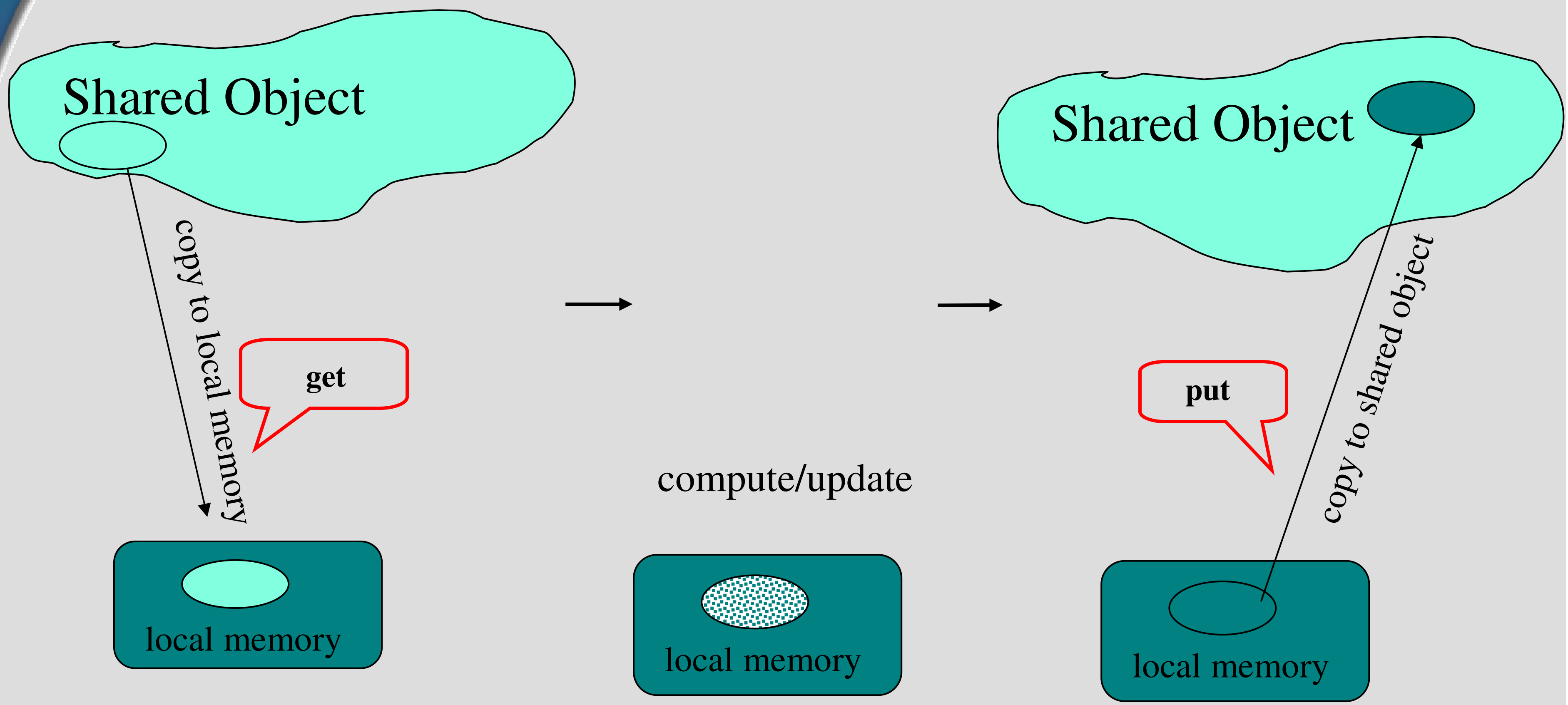
NGA_Distribution(g_a, iproc, lo, hi);

Where is the data?

NGA_Access(g_a, lo, hi, ptr, ld)

Use this information to organize calculation so that maximum use is made of locally held data

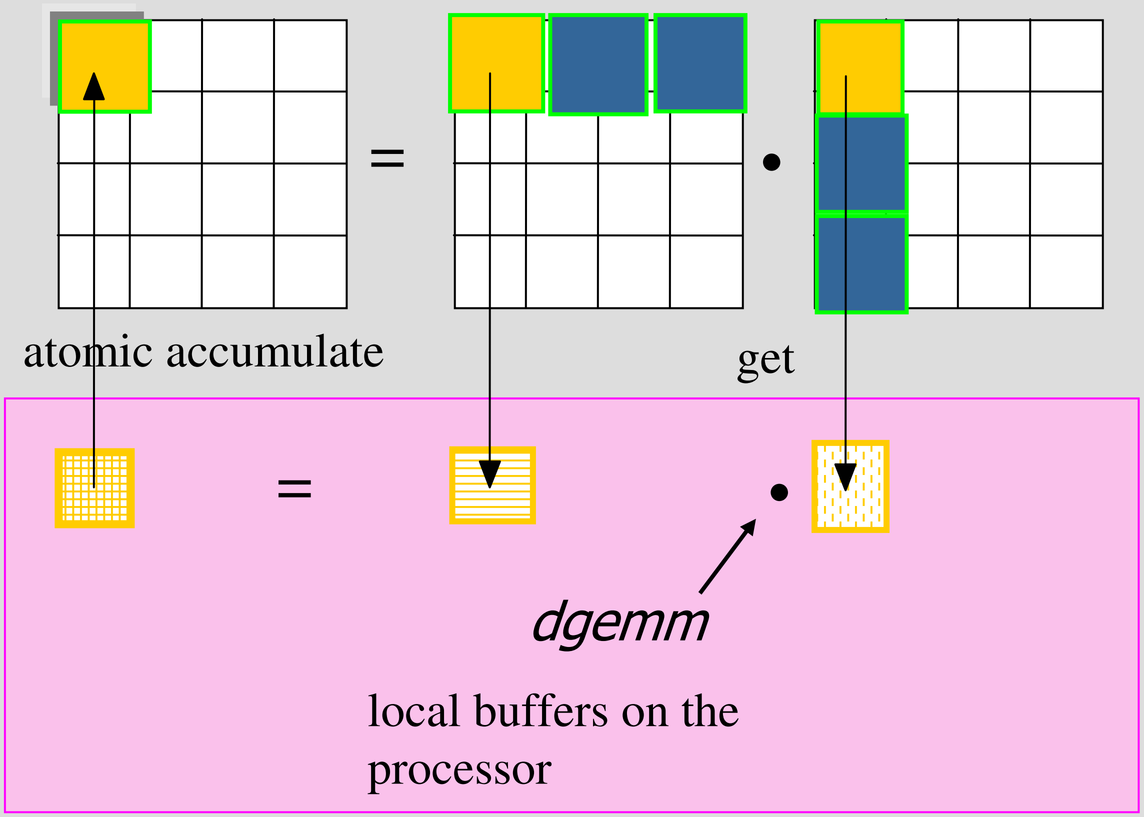
Global Array Model of Computations



Non-Blocking Communication

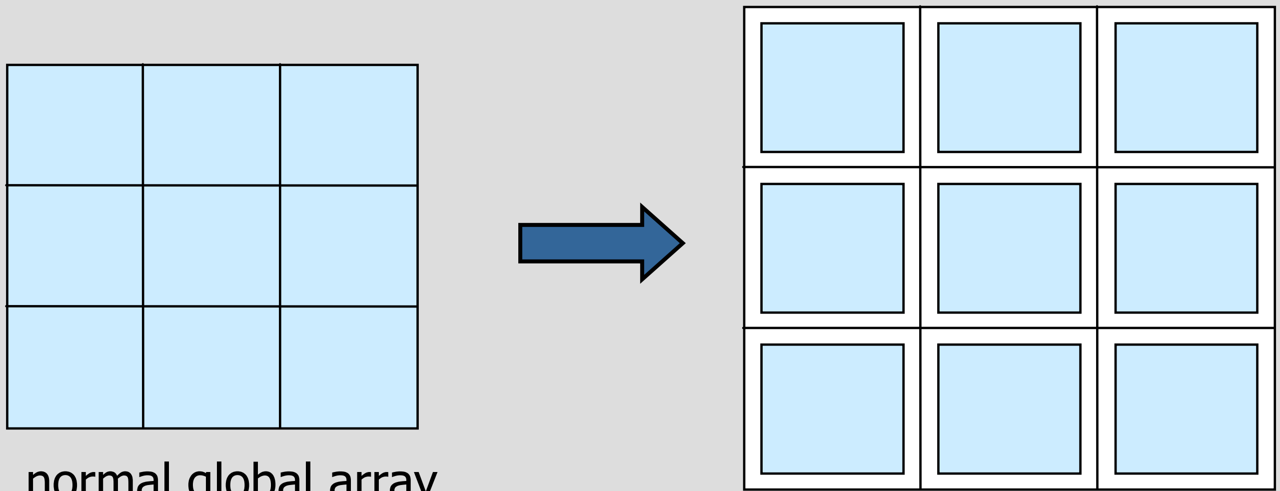
- New functionality in GA version 3.3
- Allows overlapping of data transfers and computations
  - Technique for latency hiding
- Nonblocking operations initiate a communication call and then return control to the application immediately
- operation completed locally by making a call to the *wait* routine

Matrix Multiply
(a better version)



more scalable!
(less memory,
higher parallelism)

Ghost Cells



normal global array

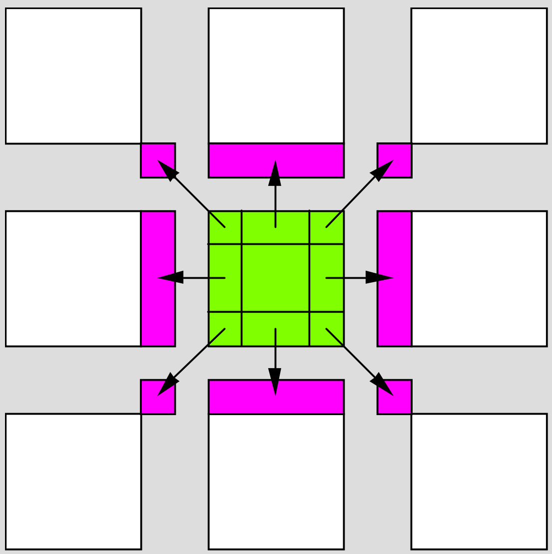
global array with ghost cells

Operations:

NGA_Create_ghosts - creates array with ghosts cells
GA_Update_ghosts - updates with data from adjacent processors
NGA_Access_ghosts - provides access to “local” ghost cell elements
NGA_Nbget_ghost_dir - nonblocking call to update ghosts cells

Ghost Cell Update

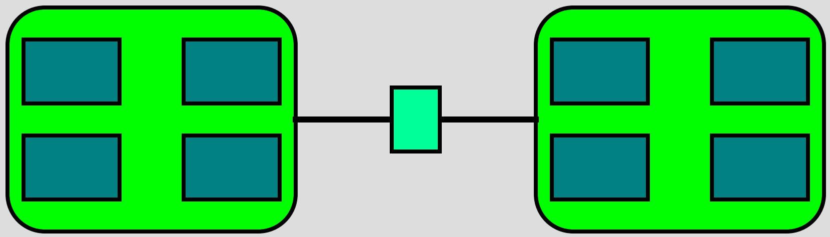
Automatically update ghost cells with appropriate data from neighboring processors. A multiprotocol implementation has been used to optimize the update operation to match platform characteristics.

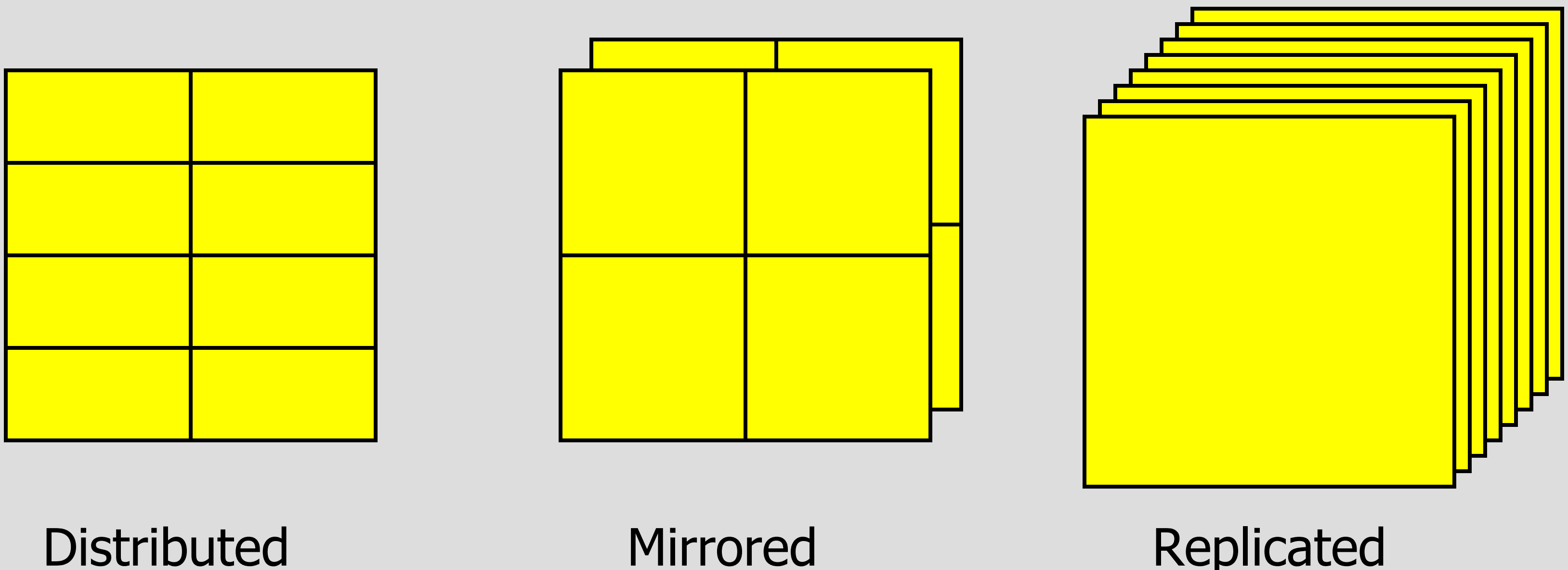


Mirrored Arrays

- Create Global Arrays that are replicated between SMP nodes but distributed within SMP nodes
- Aimed at fast nodes connected by relatively slow networks (e.g. Beowulf clusters)
- Use memory to hide latency
- Most of the operations supported on ordinary Global Arrays are also supported for mirrored arrays
- Global Array toolkit augmented by a merge operation that adds all copies of mirrored arrays together
- Easy conversion between mirrored and distributed arrays

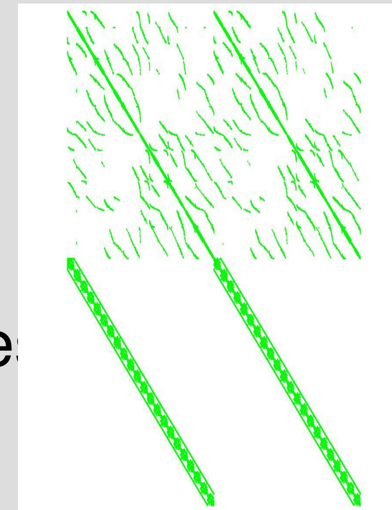
Mirrored Arrays (cont.)

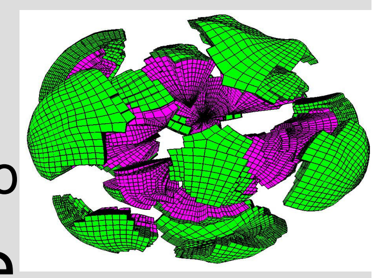




Sparse data managment

- Sparse arrays can be implemented with
  - 1-dimensional global arrays
    - Nonzero elements, row and/or index arrays
  - Set of new operations follow Thinking Machine
    - Enumerate
    - Pack/unpack
    - Binning (NxM mapping)
    - 2-key binning/sorting functions
    - Scatter_with_OP, where OP={+,min,max}
    - Segmented_scan_with_OP, where OP={+,min,max,co



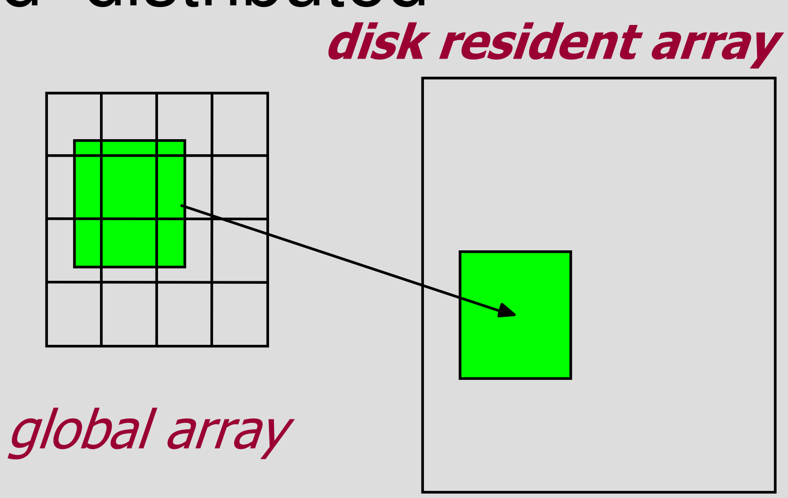


- Adopted in NWPhys/NWGrid AMR package

  http://www.emsl.pnl.gov/nwgrid

Disk Resident Arrays

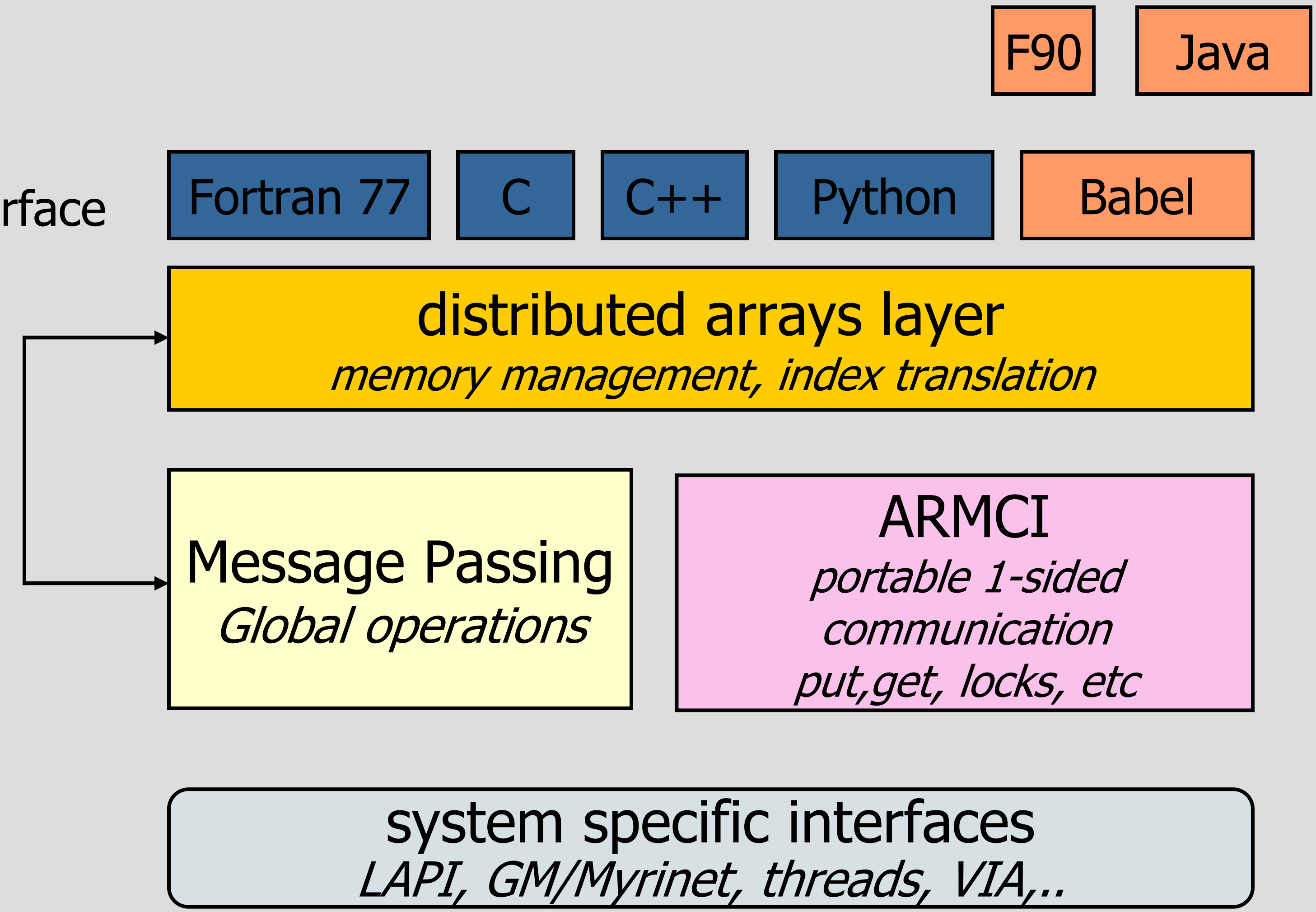
- Extend GA model to disk
  - system similar to Panda (U. Illinois) but higher level APIs
- Provide easy transfer of data between N-dim arrays stored on disk and distributed arrays stored in memory
- Use when
  - Arrays too big to store in core
  - checkpoint/restart
  - out-of-core solvers



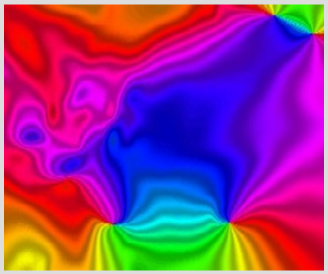
Structure of GA

Application programming language interface

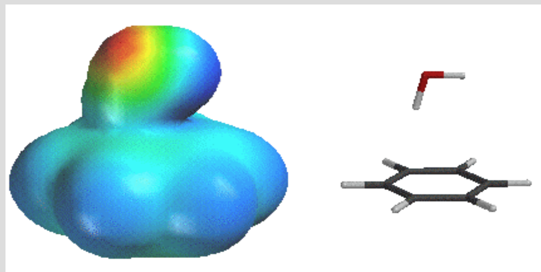
Global Arrays and MPI are completely interoperable. Code can contain calls to both libraries.



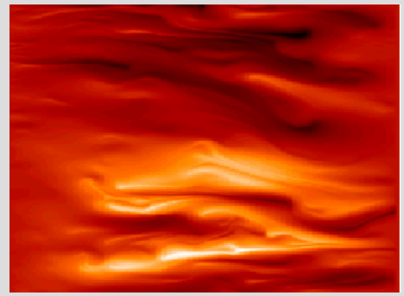
Application Areas



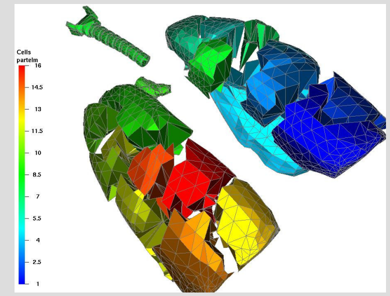
Visualization and image analysis



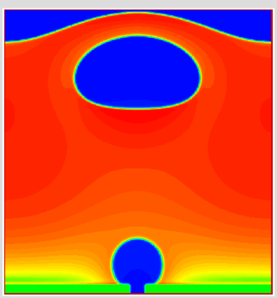
electronic structure



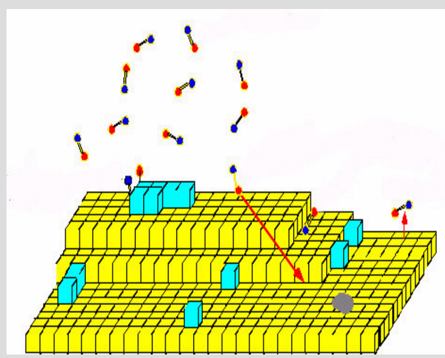
glass flow simulation



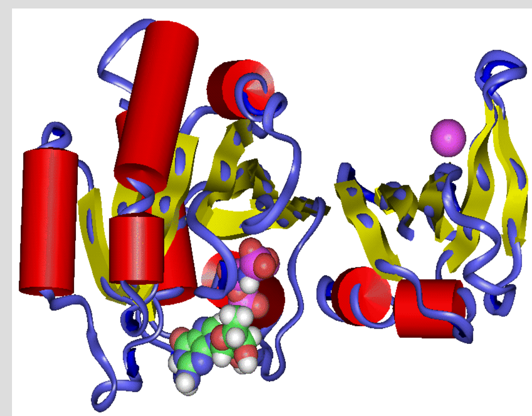
biology



thermal flow simulation



material sciences



molecular dynamics

Others: financial security forecasting, astrophysics, geosciences

Interoperability and Interfaces

- Language interfaces to Fortran, C, C++, Python
- Interoperability with MPI and MPI libararies
  - e.g., PETSC, CUMULVS
- Explicit interfaces to other systems that expand functionality of GA
  - ScaLAPACK-scalable linear algebra software
  - Peigs-parallel eigensolvers
  - TAO-advanced optimization package

GA on X1

- GA uses ARMCI for communication
- At ARMCI level, all data movements on X1 are done as loads and stores
- After initial port, Latency was terrible (24Microseconds)
- Code is mostly in C and has many small loops inside macros
- Explicit pragmas are needed for each of these small loops that loop over dimensions so that they are not vectorized

GA on X1

```
size = GA[handle].elemsize;
ndim = GA[handle].ndim;

gam_CountElems(ndim, lo, hi, &elems);
GAbytes.puttot += (double)size*elems;
GAstat.numput++;
GAstat.numput_procs += np;
```

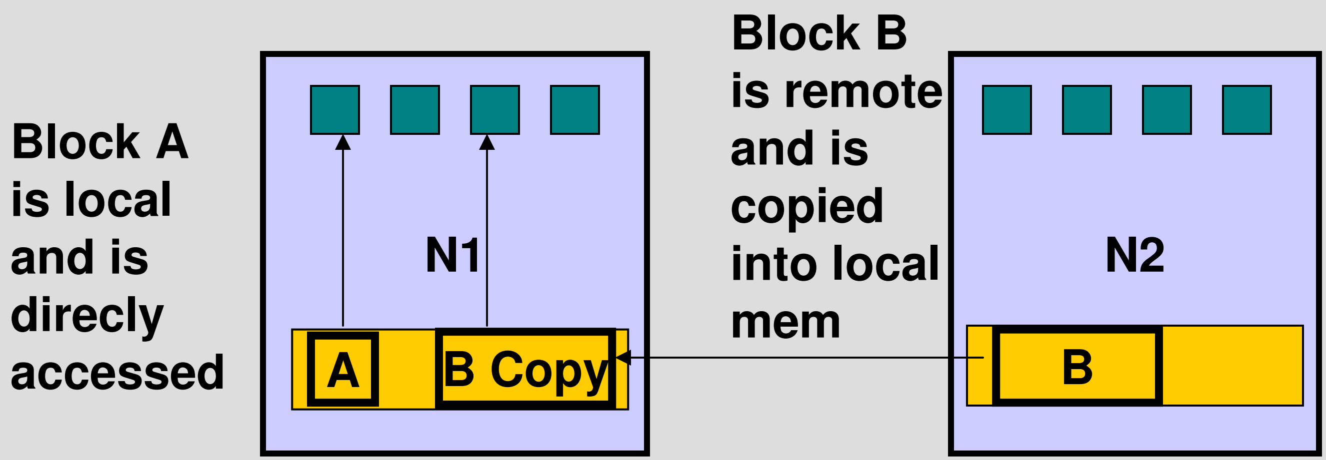
```
size = GA[handle].elemsize;
ndim = GA[handle].ndim;

MV->     gam_CountElems(ndim, lo, hi, &elems);
GAbytes.puttot += (double)size*elems;
GAstat.numput++;
GAstat.numput_procs += np;
```

```
size = GA[handle].elemsize;
ndim = GA[handle].ndim;

gam_CountElems(ndim, lo, hi, &elems);
/*GAbytes.puttot += (double)size*elems;
GAstat.numput++;
GAstat.numput_procs += np;*/
```

```
size = GA[handle].elemsize;
ndim = GA[handle].ndim;

D->     gam_CountElems(ndim, lo, hi, &elems);
/*GAbytes.puttot += (double)size*elems;
GAstat.numput++;
GAstat.numput_procs += np;*/
```

GA on X1

Copying the data when repeatedly accessed locally



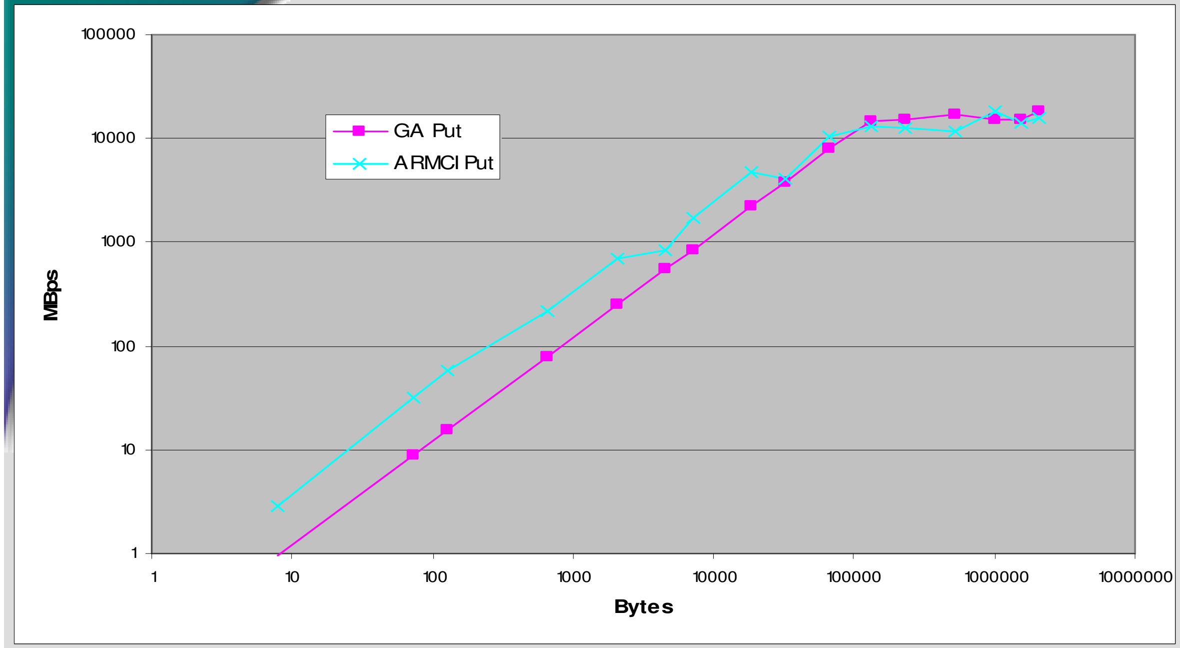
GA on X1

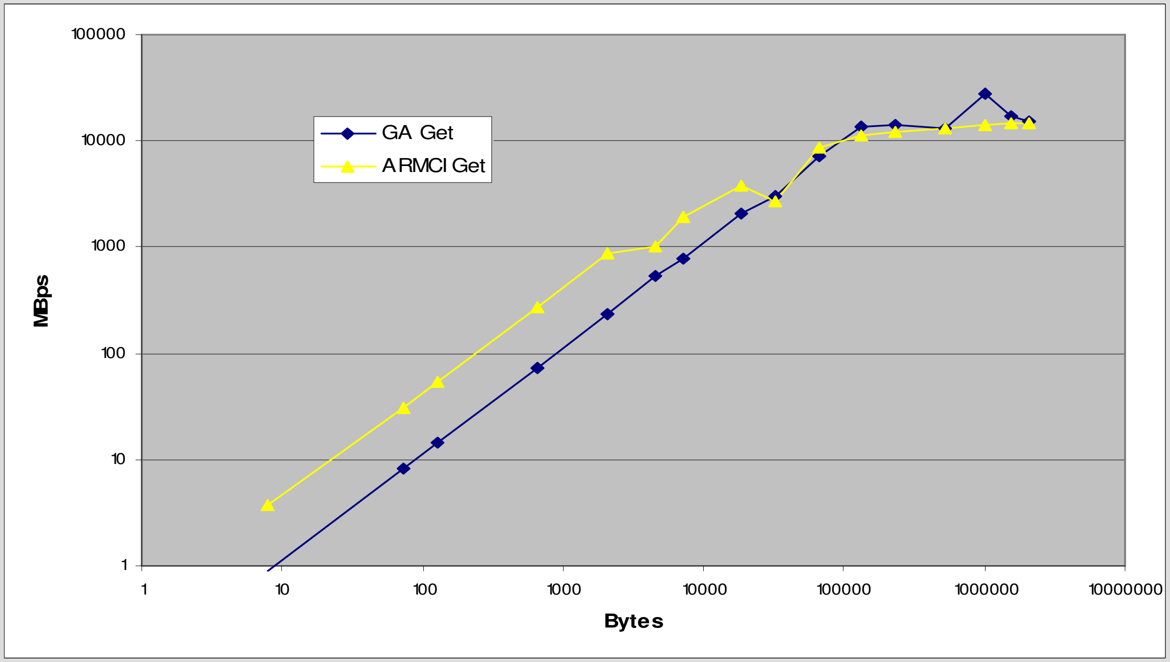
- Copy of global variable's in a few functions was reducing latency
- Sometimes using a local copy of the pointer to a global variable is making a difference in latency
- Entirely eliminating streaming was getting the lateny down from 24 to 19 microseconds. Selective "de-streaming" along with making local copies of a few global variables reduced it to 8.4 micro seconds
- Still looking into issues with Nwchem performance
- Cray is also looking these issues

GA on X1

- GA is being modified to utilize memory hierarchy (caching) to attain better performance.
- Some kernels like matmul have already been modified to take advantage of this
- Some of the GA kernels use algorithms to avoid memory contention in shared memory machines

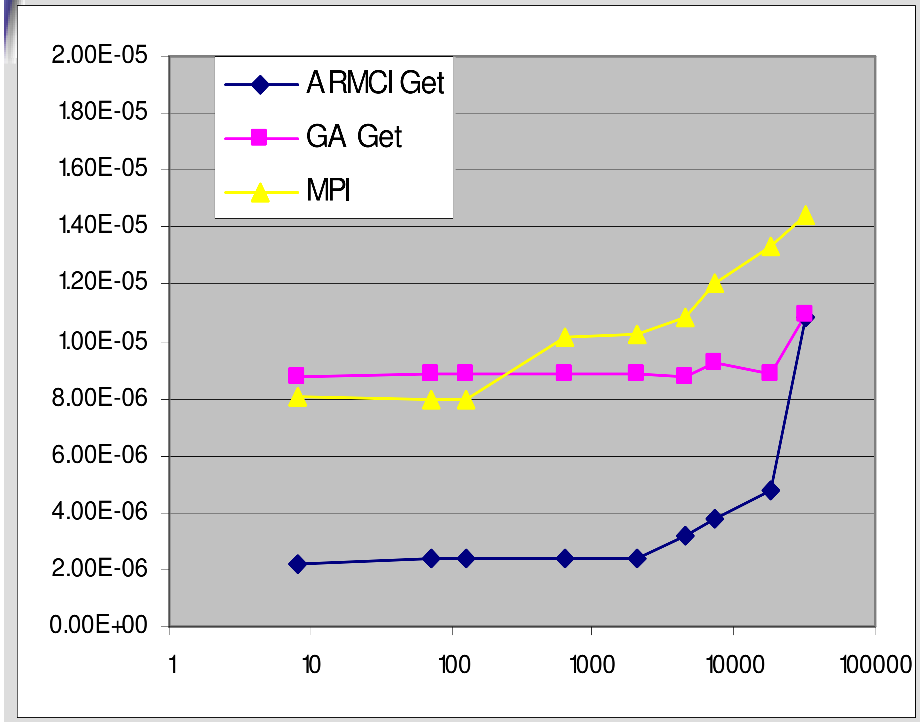
Experimental Results - Bandwidth

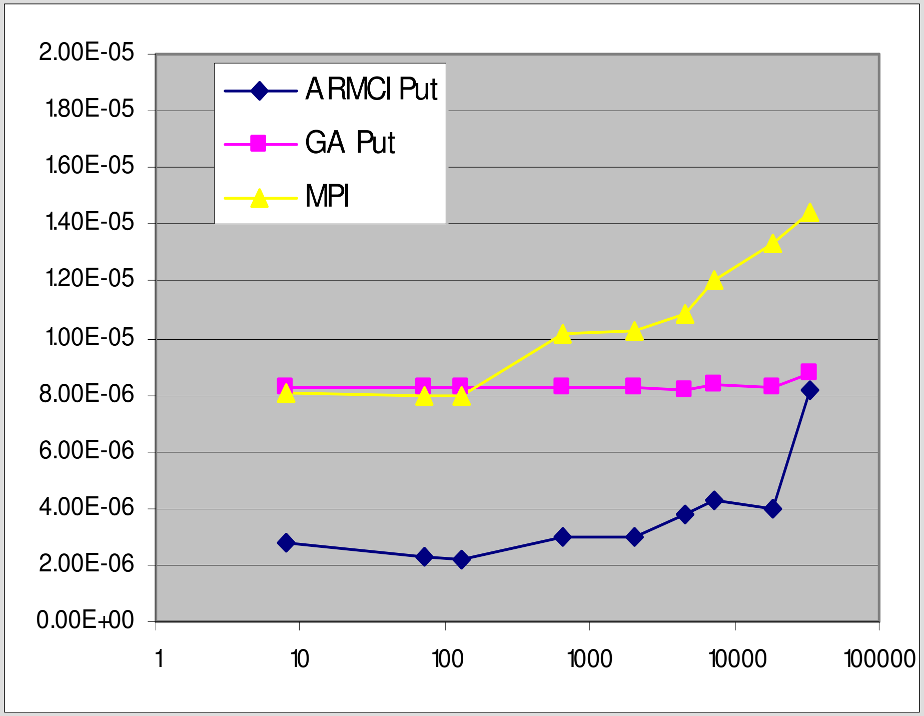




Battelle

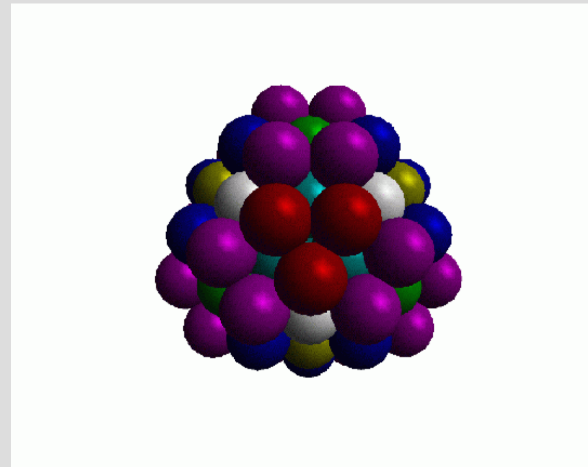
Experimental Results - Latency





Lennard-Jones Simulation (MD)

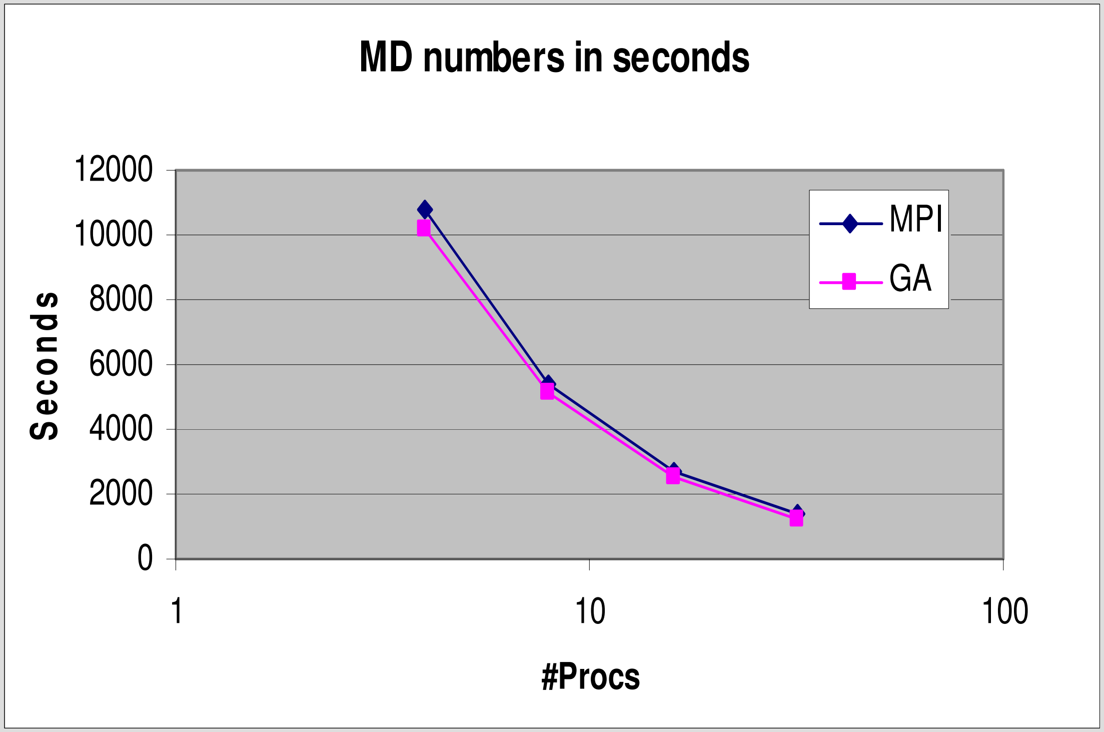
- Molecular Dynamics (MD) Simulation:
  - Simulates particle systems
    - Solids, liquids, gases
    - Biomolecules on Earth
    - Motion of stars, etc.
- GA Implementation:
  - Based on force decomposition
  - Dynamic Load Balancing



Lennard Jones Potential

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

MD Performance Results



Lennard Jones MD, Force Decomposition, MPI (steve Plimptons) and GA

Conclusion

- GA model fits well on X1
- Performance tuning by
  - selectively removing streaming from few small loops
  - exploiting locality information
  - Avoiding memory contention
- Issue with global variables needs to be understood