Logistics

• **Homework #5**
  ➢ Any issues or Questions?

• **Project**
  ➢ Presentations and written reports send them to me via email.
    - Presentations due the day before you give your talk
      ♦ Schedule will be up this week.
      ♦ Thanks to volunteers.
    - Written reports due on the 15th (2 weeks from this Friday!).
      ♦ Seth time (aka Midnight).
Collective Operations on GAs

• GA provides functions for collective array operations, targeting both whole arrays and patches (portions of global arrays).
• Collective operations require all the processes to make the call. In the underlying implementation, each process deals with its local data. These functions include:
  ➢ basic array operations,
  ➢ linear algebra operations, and
  ➢ interfaces to third party software packages.
Collective Operations on GAs [2]

- GA provides several mechanisms to manipulate contents of the arrays.
  - One can set all the elements in an array/patch to a specific value, or as a special case set to zero.
  - Since GA does not explicitly initialize newly created arrays, these calls are useful for initialization of an array/patch.
    - To fill the array with different values for each element, one can choose the one sided operation put or
    - each process can initialize its local portion of an array/patch like ordinary local memory.
  - One can also scale the array/patch by a certain factor, or copy the contents of one array/patch to another.
Collective Operations on GAs [3]

• These functions apply to the entire array.
  ➢ The function void GA_Zero(int g_a)
    ▪ sets all the elements in the array to zero.
  ➢ To assign a single value to all the elements in an array, use the function void GA_Fill(int g_a, void *val)
    ▪ It sets all the elements in the array to the value val.
    ▪ The val must have the same data type as that of the array.
  ➢ The function void GA_Scale(int g_a, void *val)
    ▪ scales all the elements in the array by factor val.
    ▪ Again the val must be the same data type as that of the array itself.
Collective Operations on GAs [4]

• The following function copies data between two arrays.
  ➢ The function void GA_Copy(int g_a, int g_b)
    ▪ copies the contents of one array to another.
    ▪ The arrays must be of the same data type and have the same number of elements.

• GA provides a set of operations on segments of the global arrays, namely patch operations.
  ➢ These functions are more general, in a sense they can apply to the entire array(s) or just portions of a GA.
  ➢ As a matter of fact, many of the GA collective operations are based on the patch operations,
    ▪ for instance, the GA_Print is only a special case of NGA_Print_patch, called by setting the bounds of the patch to the entire global array.
Linear Algebra Interfaces

- **GA provides three linear algebra operations:**
  - addition, multiplication, and dot product.
  - There are both patch and entire array functions

- **The function**
  - `void GA_Add(void *alpha, int g_a, void *beta,int g_b, int g_c)`
    - adds two arrays, `g_a` and `g_b`, and saves the results to `g_c`.
    - The two source arrays can be scaled by certain factors.
    - This operation requires the two source arrays have the same number of elements and the same data types, but the arrays can have different shapes or distributions.
    - `g_c` can also be `g_a` or `g_b`, aliasing is allowed.
  - It is encouraged to use this function when the two source arrays are identical in distributions and shapes, because of its efficiency.
    - It would be less efficient if the two source arrays are different in distributions or shapes.

- **C = alpha*A + beta*B**
Matrix Multiplication

- **The function**
  - void GA_Dgemm(char ta, char tb, int m, int n, int k, double alpha, int g_a, int g_b, double beta, int g_c)
  - Performs one of the matrix-matrix operations:
    - $C := \alpha \times \text{op}(A) \times \text{op}(B) + \beta \times C$,
    - where $\text{op}(X)$ is one of $\text{op}(X) = X$ or $\text{op}(X) = X'$,
    - $\alpha$ and $\beta$ are scalars, and
    - $A$, $B$ and $C$ are matrices,
      - with $\text{op}(A)$ an $m$ by $k$ matrix,
      - $\text{op}(B)$ a $k$ by $n$ matrix and
      - $C$ an $m$ by $n$ matrix.
    - On entry, $ta$ specifies the form of $\text{op}(A)$ to be used in the matrix multiplication as follows:
      - $ta = 'N'$ or 'n', $\text{op}(A) = A$.
      - $ta = 'T'$ or 't', $\text{op}(A) = A'$. 

4/1/2005  ComS 425  Spring 2005  8 of 36: Lecture 28
Dot Product

• The functions
  - long GA_Idot(int g_a, int g_b)
  - double GA_Ddot(int g_a, int g_b)
  - DoubleComplex GA_Zdot(int g_a, int g_b)

• Compute the element-wise dot product of two arrays.
  - Works for N-dimensional Arrays
  - Result = Sum over(I,J) [A(I,J)*B(I,J)]

• A separate function for each datatype
Other functions

• The function
  ➢ void GA_Symmetrize(int g_a)
  ➢ symmetrizes matrix A represented with handle g_a:
    ▪ A = .5 * (A+A').

• The function
  ➢ void GA_Transpose(int g_a, int g_b)
  ➢ transposes a matrix:
    ▪ B = A'.
Patch routine equivalents

• **Exist for most of these routines.**
  ➢ You should look at the API page for explicit details of the patch interface.

• **The matrix multiplication is the only operation on array patches that is restricted to the two dimensional domain, because of its nature.**
Interfaces to Third Party Software Packages

- There are many existing software packages designed for solving engineering problems.
- Global Arrays provide interfaces to several of these packages.
- They are specialized in one or two problem domains, such as solving
  - Linear systems,
  - eigen-vectors, and
  - differential equations,
Scalapack

- Scalapack is a well known software library for linear algebra computations on distributed memory computers.
- Global Arrays uses this library to solve systems of linear equations and also to invert matrices.
- The function
  - `int GA_Solve(int g_a, int g_b)`
  - solves a system of linear equations $A \times X = B$.
  - It first will call the Cholesky factorization routine and, if successful, will solve the system with the Cholesky solver.
  - If Cholesky is not able to factorize $A$, then it will call the LU factorization routine and will solve the system with forward/backward substitution.
  - On exit $B$ will contain the solution $X$. 
Scalapack [2]

- The function
  - `int GA_Llt_solve(int g_a, int g_b)`
  - also solves a system of linear equations \( A \times X = B \), using the Cholesky factorization of an \( N \times N \) double precision symmetric positive definite matrix \( A \).
  - On successful exit \( B \) will contain the solution \( X \).

- The function
  - `void GA_Lu_solve(char trans, int g_a, int g_b)`
  - solves the system of linear equations \( \text{op}(A)X = B \) based on the LU factorization.
  - \( \text{op}(A) = A \) or \( A' \) depending on the parameter \( \text{trans} \).
  - Matrix \( A \) is a general real matrix.
  - Matrix \( B \) contains possibly multiple rhs vectors. The array associated with the handle \( g_b \) is overwritten by the solution matrix \( X \).
Scalapack [3]

• The function
  - `int GA_Spd_invert(int g_a)`
  - Computes the inverse of a double precision matrix using the Cholesky factorization of a NxN double precision symmetric positive definite matrix A stored in the global array represented by g_a.
  - On successful exit, A will contain the inverse.
PeIGS

- The PeIGS library contains subroutines for solving standard and generalized real symmetric eigensystems.
- All eigenvalues and eigenvectors can be computed.
- The library is implemented using a message-passing model and is portable across many platforms.
  - For more information and availability send a message to gi_fann@pnl.gov.
- GA uses this library to solve eigen-value problems.
- The function
  - void GA_Diag(int g_a, int g_s, int g_v, void *eval)
  - solves the generalized eigen-value problem returning all eigen-vectors and values in ascending order. The input matrices are not overwritten or destroyed.
PeIGS [2]

- The function
  - `void GA_Diag_reuse(int control, int g_a, int g_s, int g_v, void *eval)`
  - solves the generalized eigen-value problem returning all eigen-vectors and values in ascending order.
  - Recommended for REPEATED calls if g_s is unchanged.
- The function
  - `void GA_Diag_std(int g_a, int g_v, void *eval)`
  - solves the standard (non-generalized) eigenvalue problem returning all eigenvectors and values in the ascending order.
  - The input matrix is neither overwritten nor destroyed.
Global Arrays

- **Remember GA thinks in terms of FORTRAN storage.**
  - As long as you
    - compute dense patches
    - address them linearly and similarly in each routine
    - Compute the leading dimension as the column dimension
      - $A(\text{rows,columns})$
  - They will work in C-land.
  - If you use GA_print it will look funny
    - Thinks in fortran.
Locality Information

- For a given global array element, or a given patch, sometimes it is necessary to find out who owns this element or patch.
- The function
  - `int NGA_Locate(int g_a, int subscript[])`
  - tells who (process id) owns the elements defined by the array subscripts.
Locality Information [2]

• The function
  
  ```c
  int NGA_Locate_region(int g_a, int lo[], int hi[], int *map[], int procs[])
  ```

  returns a list of GA process IDs that 'own' the patch.

  The Global Arrays support an abstraction of a distributed array object. This object is represented by an integer handle. A process can access its portion of the data in the global array.

  To do this, the following steps need to be taken:
  - find the distribution of an array, which part of the data the calling process own
  - access the data
  - operate on the data: read/write
  - release the access to the data
Locality Information [3]

- The function
  - void NGA_Distribution(int g_a, int iproc, int lo[], int hi[])
  - finds out the range of the global array g_a that process iproc owns. iproc can be any valid process ID.

- The function
  - void NGA_Access(int g_a, int lo[], int hi[], void *ptr, int ld[])
  - provides access to local data in the specified patch of the array owned by the calling process.
  - The C interface gives the pointer to the patch.
  - The Fortran interface gives the patch address as the index (distance) from the reference address (the appropriate MA base addressing array).
Locality Information [4]

• The function
  ➢ void NGA_Release(int g_a, lo[], int hi[])
  ▪ and
  ➢ void NGA_Release_update(int g_a, int lo[], int hi[])
  ➢ releases access to a global array. The former set is used when the data was read only and the latter set is used when the data was accessed for writing.

• Global Arrays also provide a function to compare distributions of two arrays. It is
  ➢ void NGA_Compare_distr(int g_a, int g_b)
  ➢ Returns true or false
Process Information

• When developing a program, one needs to use characteristics of the parallel environment:
  ➢ process ID,
  ➢ how many processes are working together, and
  ➢ what the topology of processes look like.

• To answer these questions, the following functions can be used.
  ➢ The function: `int GA_Nodeid()`  
    ▪ returns the GA process ID of the current process.
  ➢ The function: `int GA_Nnodes()`  
    ▪ tells the number of computing processes.
  ➢ The function
  ➢ void NGA_Proc_topology(int g_a, int proc, int coordinates)
    ▪ determines the coordinates of the specified processor in the virtual processor grid corresponding to the distribution of array g_a.
Process Information

• **Example:** A GA is distributed on 9 processes.

• **The processes are numbered from 0 to 8.**

• **If one wants to find out the coordinates of process 7 in the virtual processor grid,**
  - by calling the function `ga_proc_topology`, the coordinates of (2,1) will be returned.
  - **Two dimensional grids only**
Memory Information

• Even though the memory management does not have to be performed directly by the user,

• Global Arrays provide functions to verify the memory availability.

• Global Arrays provide the following information:
  ➢ How much memory has been used by the allocated global arrays.
  ➢ How much memory is left for allocation of new the global arrays.
  ➢ Whether the memory in global arrays comes from the Memory Allocator (MA).
  ➢ Is there any limitation for the memory usage by the Global Arrays.
Memory Information [2]

• The function
  ➢ `size_t GA_Inquire_memory()`
  ➢ Returns the amount of memory (in bytes) used in the allocated global arrays on the calling processor.

• The function
  ➢ `size_t GA_Memory_avail()`
  ➢ Returns the amount of memory (in bytes) left for allocation of new global arrays on the calling processor.

• Memory Allocator(MA) is a library of routines that comprises a dynamic memory allocator for use by C, Fortran, or mixed-language applications.
  ➢ Fortran- 77 applications require such a library because the language does not support dynamic memory allocation.
  ➢ C (and Fortran-90) applications can benefit from using MA instead of the ordinary malloc() and free() routines because of the extra features MA provides.
Memory Information [3]

• The function
  ➢ `int GA_Uses_ma()`
  ➢ tells whether the memory in Global Arrays comes from the Memory Allocator (MA) or not.

• The function
  ➢ `int GA_Memory_limited()`
  ➢ Indicates if a limit is set on memory usage in Global Arrays on the calling processor.
MPL Wrappers

• GA provides convenient operations for broadcast or reduce regardless of the MPL.
• The function
  ➢ void GA_Brdcst(void *buf, int lenbuf, int root)
    ▪ broadcasts from process root to all other processes.
  ➢ The functions
    ▪ void GA_Igop(long x[], int n, char *op)
    ▪ void GA_Dgop(double x[], int n, char *op)
    ▪ 'sum' elements of X(1:N) (a vector present on each process) across all nodes using the communicative operator op,
    ▪ The result is broadcasted to all nodes.
    ▪ Supported operations include
      ♦ +, *, Max, min, Absmax, absmin
      ♦ The integer version also includes the bitwise OR operation.
    ▪ These operations unlike ga_sync, do not include embedded ga_fence operations.
Other Utility functions

- There are some other useful functions in Global Arrays.
  - One group is about inquiring the array attributes.
  - Another group is about printing the array or part of the array.
Inquire

• A global array is represented by a handle.
• Given a handle, one can get the array information, such as the array name, memory used, array data type, and array dimension information, with the help of following functions.
  > void NGA_Inquire(int g_a, int *type, int *ndim, int dims[])
  > returns
    ▪ The data type of the array,
    ▪ The number of dimensions,
    ▪ And the dimensions of the array.

• The function
  > char* GA_Inquire_name(int g_a)
  > finds out the name of the array.
Print

- Global arrays provide functions to print
  - content of the global array
  - content of a patch of global array
  - the status of array operations
  - summary of allocated arrays

- The function
  - void GA_Print(int g_a)
    - prints the entire array to the standard output. The output is formatted.
  - void NGA_Print_patch(int g_a, int lo[], int hi[], int pretty)
    - Print a patch of the array.
    - One can either specify a formatted output
      - (set pretty to one) where the output is formatted and rows/ columns are labeled, or
      - (set pretty to zero) just dump all the elements of this patch to the standard output without any formatting.
Print

• The function

  ➢ void GA_Print_stats()
  ➢ prints the global statistics information about array operations for the calling process, including
    ▪ number of calls to the GA
      ♦ create/duplicate,
      ♦ destroy,
      ♦ get, put,
      ♦ scatter, gather, and
      ♦ read_and_inc operations
    ▪ total amount of data moved in the GA primitive operations
    ▪ amount of data moved in GA primitive operations to logically remote locations
    ▪ maximum memory consumption in global arrays, the "high-water mark"
Print [2]

• The function
  ➢ void GA_Print_distribution(int g_a)
  ➢ prints the global array distribution.
  ➢ It shows mapping array data to the processes.

• The function
  ➢ void GA_Summarize(int verbose)
  ➢ prints info about allocated arrays. verbose can be either one or zero.
The function

```c
void GA_Check_handle(int g_a, char *string)
```

- checks if the global array handle `g_a` represents a valid array.
- The string is the message to be printed when the handle is invalid.
New functionality

- Non-blocking puts and gets.
  - Similar to isend/irecv in MPI
  - Overlap computation and communication events
    - Initiate an nbput or nbget
    - Do some work that does not require those buffers
    - Initiate a wait
  - Logically the same as the MPI equivalents but …
    - Still one-sided
      - Left hand doesn’t need to know that the right hand exists !!!!!
New functionality [2]

• **Automatic Ghost Cells or Halos**
  - Typically used in algorithms that compute data in a periodic or standard multidimensional grid.
    - Nearest neighbor communication patterns.
  - Size of the Halo determined by user when GA is created.
  - Updates to/from halo data happen automatically or scheduled by the user.