COMS/CPRE 425
Spring 2005
Lecture 26

Ricky A. Kendall
rickyk@cs.iastate.edu
Logistics

• **Homework #5 due date will not be changed.**
  - Do NOT wait until the last minute.
  - Posted last night and officially assigned today!

• **Any questions about MPI**
  - We move to other models now.

• **C++ now works with the IBM SP compilers**
  - I have updated the simple codes to reflect this.

• **Compilers on the IBM SP**
  - C is mpcc_r
  - C++ mpCC_r -cpp
Coordination Models

• A coordination model describes the interaction between autonomous tasks and deals with communication, synchronization, and task management.
  ➢ Focus is on approaches in which the computation and interaction aspects are strictly separated.
  ➢ Cooperating tasks are anonymous to each other.
  ▪ Data movement is not coordinated
    ♦ The sender does not require the receiver to know about data movement (puts the data in the right place).
    ♦ The receiver does not require the sender to know about data movement (gets the data from the right place).
Advantages of Coordination Models

• Separation between computation and coordination
  ➢ Helps in structuring programs (modularity)
    ▪ Reuse of both computational and coordination modules
    ▪ Modules can interact in multiple / different ways.
    ▪ Modularity in general supports
      ♦ Collaborative software development
      ♦ Verification
      ♦ Maintenance

➢ Supports heterogeneity and interoperability
  ▪ Heterogeneity with respect to implementation and software details as well as architectures.
  ▪ Computational modules can work with different interaction modules.
Advantages of Coordination Models [2]

- **Anonymity supports**
  - Dynamic applications
    - Runtime cooperative tasks
  - Code mobility.
    - Code/Data can be constructed to move if necessary
One particular Coordination Model

- **Distributed Shared Memory (DSM)**
  - Physically distributed memory that is used logically as shared memory
    - Hardware or software support for this is required.
- **Common DSM mechanisms**
  - Page Level
    - E.g., Kernel extensions to moving virtual memory to other partner “kernels” in addition to Swap space
  - Data Models
    - Data Variables or objects are symmetrically identified as logically shared but physically distributed.
      - Objects have methods of access and updates attached 😊
Page Level DSM

• Requires modification of the virtual memory mechanisms.
  - OS source may not be available or modifiable.
    ▪ If not then yet another layer of indirection needed
  - Kernel interaction from one node to another
    ▪ SMP adds yet another layer at each node
  - Consistency model may become complex
    ▪ Is the page change local or global to the SMP image
  - Must move the whole page even if only one variable is needed
    ▪ False sharing at the page level instead of the cache level is a serious performance bottleneck
  - Data movement is “transparent” to the user
Data variable/object mechanisms

- **Programmer must identify variables that are shared**
  - Not like OpenMP where the default is everything is shared unless specified.
- **Operations that are separated among shared and local variables can be tricky**
  - Care must be taken to make sure that the consistency is appropriate for the algorithm.
- **Data movement is not automatic nor transparent.**
  - Either explicit or implicit in the library interface that accesses/moves data
Tuple Based Coordination Models

- A model where the process coordination is via a shared data structure.
- Often looks like a single homogenous entity.
- Multiple processes can access the shared data structure.
- A tuple is an ordered collection of information.
  - Actual details of the data structure are usually hidden from the programmer.
- Tuples are written by a process and read by any other of the cooperating processes.
One popular Implementation

- **Linda Language or Library Model**
  - Allows
    - `out (put),`
    - `rd(get),`
    - `in(get/delete),`
    - `eval (get, operate etc)`

- **Linda is used commercially for mostly distributed computing algorithms.**
  - Used by Gaussian a quantum chemistry code for parallel execution on MIMD systems.
    - Scales poorly because of overheads to read/write from/to tuple space.
Global Arrays

- The Global Arrays (GA) toolkit provides a shared memory style programming environment in the context of distributed array data structures
  - So called "global arrays"
  - From the user perspective, a global array can be used as if it was stored in shared memory.
  - All details of the data distribution, addressing, and data access are encapsulated in the global array objects.
  - Information about the actual data distribution and locality can be easily obtained and taken advantage of whenever data locality is important.

- The primary target architectures for which GA was developed are massively-parallel distributed-memory or scalable shared-memory systems.
  - Loosely based on Linda Tuple model.
Global Arrays [2]

- GA divides logically shared data structures into local and remote portions and recognizes variable data transfer costs required to access the data depending on the proximity attributes.
  - A "local" portion of the “logically shared memory” is assumed to be faster to access
  - The remainder ("remote" portion of “logically shared memory”) is considered slower to access.
  - These differences do not hinder the ease-of-use since the library provides uniform access mechanisms for all the shared data regardless where the referenced data is located.
    - Detailed understanding of the distribution can lead to performance improvements.
  - In addition, any processes can access a local portion of the shared data directly/in-place like any other portion of local memory.
    - Access to other portions of the shared data must be done through the GA library calls.
Global Arrays [3]

- GA was designed to complement rather than substitute the message-passing model, and it allows the user/developer to combine shared-memory and message-passing styles of programming in the same program.
  - GA works with MPI, TCGMSG, PVM message passing libraries.
- GA inherits an execution environment from a message-passing library (w.r.t. processes, file descriptors etc.) that started the parallel program.
  - The message-passing library starts the parallel program for GA.
- GA is implemented as a library with C and Fortran-77 bindings
  - Python and C++ interfaces developed but not supported.
Global Arrays [4]

- Explicit library calls are required to use the GA model in a parallel C/Fortran program.
- A disk extension of the Global Array library is supported by its companion library called Disk Resident Arrays (DRA).
  - DRA maintains array objects in secondary storage and allows transfer of data to/from global arrays.
  - We won’t do anything with DRAs in the course 😞
- Basic Functionality
  - The basic shared memory operations supported include
    - get, put, scatter and gather.
    - They are complemented by the atomic read-and-increment, accumulate
      - reduction operations that combines data in local memory with data in the “logically shared memory” locations,
    - lock operations.
Global Arrays [5]

• However, these operations can only be used to access data in global arrays rather than arbitrary memory locations.
  ➢ At least one global array has to be created before data transfer operations can be used.
  ➢ These operations are truly one-sided/unilateral and will complete regardless of actions taken by the remote process(es) that own(s) the referenced data.
  ➢ In particular, GA does not offer or rely on a polling operation or require inserting other library calls on the remote side to assure communication progress.

• A programmer in the GA program has full control over the distribution of global arrays.
  ➢ Both regular and irregular distributions are supported.
Global Arrays [6]

• The GA data transfer operations use an array index-based interface rather than addresses of the shared data.
  - Unlike other systems based on global address space that support remote memory (put/get) operations.
    - E.g., SHMEM
  - GA does not require the user to specify the target process(es) where the referenced shared data resides.
    - It simply provides a global view of the data structures.
    - It “knows” where the data is without you telling it.
  - The higher level array oriented API makes GA easier to use
    - Does not compromising data locality control.
  - The library internally performs global array index-to-address translation and then transfers data between appropriate processes.
Global Arrays [7]

• If necessary, the programmer is always able to inquire:
  - where and an element or array section (e.g., patch) is located, and
  - which process(es) own data in the specified array section.

• The GA toolkit supports three data types:
  - integer,
  - double precision, and
  - double complex.

• The supported array dimensions
  - range from one to seven.
  - This limit follows the Fortran convention.
  - If you are insane, the library can be reconfigured to support more than 7-dimensions but only through the C interface.
Global Arrays Programming Model

- The Global Arrays library supports two programming styles:
  - task-parallel and data-parallel.
- The GA task-parallel model of computations is based on the explicit remote memory copy:
  - The remote portion of shared data has to be copied into the local memory area of a process before it can be used in computations by that process.
  - Of course, the "local" portion of shared data can always be accessed directly thus avoiding the memory copy.
- The data distribution and locality control are provided to the programmer.
  - Data locality information for the shared data is available.
  - Operations for management of its data structures, one-sided data transfer operations, and supportive operations for data locality control and queries.
Global Arrays Programming Model [2]

• The GA shared memory consistency model is a result of a compromise between
  ➢ the ease of use and
  ➢ portable performance.
  ➢ The load and store operations are guaranteed to be ordered with respect to each other only if they target overlapping memory locations.
  ➢ The store operations (put, scatter) and accumulate complete locally before returning
    ▪ i.e., the data in the user local buffer has been copied out but not necessarily completed at the remote side.
  ➢ The memory consistency is only guaranteed for:
    ▪ multiple read operations (as the data does not change),
    ▪ multiple accumulate operations (as addition is commutative), and
    ▪ multiple disjoint put operations (as there is only one writer for each element).
Global Arrays Programming Model [3]

- The application can manage consistency of its data structures in other cases by using
  - Lock (mutixes),
  - barrier, and
  - fence operations.
- The data-parallel model is supported by a set of (collectively called) functions that operate on global arrays or their portions.
- Underneath, if any interprocessor communication is required, the library uses
  - remote memory copy or
  - collective message-passing operations.
GA the pictorial view
Application Guidelines

• When to use GA:
  ➢ Algorithmic Considerations
    ▪ applications with dynamic and irregular communication patterns
    ▪ for calculations driven by dynamic load balancing
    ▪ need 1-sided access to shared data structures
    ▪ need high-level operations on distributed arrays for out-of-core array-based algorithms (GA + DRA)
  ➢ Usability Considerations
    ▪ data locality important
    ▪ when coding in message passing becomes too complicated
    ▪ when portable performance is important
    ▪ need object orientation without the overhead of C++
Application Guidelines [2]

• When not to use GA:

➤ Algorithmic Considerations

▪ for systolic communications, or
  ♦ Shifting B in Fox’s algorithm is a systolic operation.
▪ nearest neighbor communications
▪ regular communication patterns
▪ when synchronization associated with cooperative point-to-point message passing is needed
  ♦ e.g., Cholesky factorization in Scalapack

➤ Usability Considerations

▪ when interprocedural analysis and compiler parallelization is more effective
▪ existing language support is sufficient and robust compilers are available!
Portable Performance

- **Supported Platforms**
  - IBM SP, CRAY T3E/J90/SV1, SGI Origin, Fujitsu VX/VPP
  - Cluster of workstations: Solaris, IRIX, AIX, HPUX, Digital/Tru64 Unix, Linux, NT
  - Standalone uni- or multi-processor workstations or servers
  - Standalone uni- or multi-processor Windows NT workstations or servers

- **Older versions of GA supported some additional (now obsolete) platforms such as:**
  - IPSC, KSR, PARAGON, DELTA, CONVEX.
  - They are not supported in the newer (>3.1) versions because we do not have access to these systems.
    - We recommend using GA 2.4 on these platforms.
### Portable Performance 32/64

<table>
<thead>
<tr>
<th>Platform</th>
<th>32-bit Target</th>
<th>64-bit Target</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Ultra</td>
<td>SOLARIS</td>
<td>SOLARIS64</td>
<td></td>
</tr>
<tr>
<td>IBM RS/6000</td>
<td>IBM</td>
<td>IBM64</td>
<td></td>
</tr>
<tr>
<td>IBM SP</td>
<td>LAPI</td>
<td>N/A</td>
<td>no 64-bit user-space</td>
</tr>
<tr>
<td>Compaq</td>
<td>N/A</td>
<td>DECOSF</td>
<td></td>
</tr>
<tr>
<td>HP PA-RISC</td>
<td>HPUX</td>
<td>HPUX64</td>
<td></td>
</tr>
<tr>
<td>Linux (x85, ultra, PPC)</td>
<td>LINUX</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Linux Alpha</td>
<td>N/A</td>
<td>LINUX64</td>
<td>Compaq compilers</td>
</tr>
<tr>
<td>CRAY T3E</td>
<td>N/A</td>
<td>CRAY-T3E</td>
<td></td>
</tr>
<tr>
<td>Cray J90 of SV1</td>
<td>N/A</td>
<td>CRAY-YMP</td>
<td></td>
</tr>
<tr>
<td>SGI IRIX</td>
<td>SGI_N32, SGI</td>
<td>SGITFG</td>
<td></td>
</tr>
<tr>
<td>Fujitsu-VPP</td>
<td>FUJITSU-VPP</td>
<td>FUJITSU-VPP64</td>
<td></td>
</tr>
</tbody>
</table>

Portable Performance [2]

- To aid development of fully portable applications, in 64-bit mode Fortran integer datatype is 64-bits.
  - It is motivated by
    1. The need of applications to use very large data structures and
    2. Fortran INTEGER*8 not being fully portable.
  - The 64-bit representation of integer datatype is accomplished by using appropriate Fortran compiler flag.

- Because of limited interest in heterogenous computing among known GA users, the Global Array library still does not support heterogenous platforms.
  - This capability can be added if required by new applications.
Dependencies on other software

• GA requires
  - GNU make to build the software.
  - A message-passing library
  - MA (Memory Allocator), a library for management of local memory;
  - ARMCI, a one-sided communication library that GA uses as its run-time system;
  - BLAS library is required for the parallel eigensolver and ga_dgemm;
  - LAPACK library is required for the parallel eigensolver (a subset is included with GA, which is built into liblinalg.a);
Underlying Network

• Some cluster installations can be equipped with a high performance network which offer some special communication protocol

➢ To achieve high performance in GA, ARMCI must be built to use these protocols for one-sided communication.

➢ Ethernet
  ▪ TCP/IP SOCKETS workstation clusters

➢ Quadrics
  ▪ Elan/Shmem QUADRICS Linux (alpha), Compaq

➢ Myrinet
  ▪ GM GM Linux (x86, ultra)
  ▪ The Myrinet port has been partially optimized.

➢ Giganet cLAN
  ▪ VIA VIA Linux (x86)
  ▪ The Giganet/VIA port has not been optimized
    ♦ included on an experimental basis.
Program Skeleton for GA/MPI

/* initialize MPI space */
if (MPI_SUCCESS != (MPI_Init(&argc, &argv)))
  (void)fprintf(stderr,'"MPI_Init failed"
');

/*-------------- Initialize GA and MA----------------*/
GA_Initialize( );
stack = 100000; /* stack size per node*/;
heap = 100000; /* heap size per node*/;
if (!(MA_init(MT_F_DBL,stack,heap)))
  GA_Error("MA_init failed",(stack+heap));

… … doSomethingUseful( );

GA_Terminate( );
MPI_Finalize( );
GA and Memory Allocation.

• GA programs require message-passing and Memory Allocator (MA) libraries to work.
  ➢ GA internally does not allocate local memory from the operating system
    ▪ all dynamically allocated local memory comes from MA.

• GA uses a very limited amount of \textit{statically} allocated memory to maintain its data structures and state.
  ➢ Most of the memory is allocated dynamically as needed,
    ▪ primarily to store data in newly allocated global arrays or
    ▪ as temporary buffers internally used in some operations
      ♦ deallocated when the operation completes.
GA kinds of memory

- There are two flavors of dynamically allocated memory in GA: shared memory and local memory.
  - Shared memory is allocated from the operating system (UNIX and Windows) and is shared between user processes.
    - A process that attaches to a shared memory segment can access it as if it was local memory.
    - All the data in shared memory is directly visible to every process that attaches to that segment.
    - On shared memory systems and clusters of SMP (symmetric multiprocessor) nodes, shared memory is used to store global array data
      - Allocated by the Global Arrays run-time system called ARMCI.
  - On systems that do not offer shared-memory capabilities or when a program is executed in a serial mode, GA uses local memory to store data in global arrays.
Aggregate Remote Memory Copy Interface

• ARMCI uses shared memory to optimize performance and avoid explicit inter-processor communication within a single shared memory system or an SMP node.
  - ARMCI allocates shared memory from the operating system in large segments
    ▪ Manages memory in each segment in response to the GA allocation (ga_create) and deallocation (ga_destroy) calls.
  - Each segment can hold data in many small global arrays.
  - ARMCI does not return shared memory segments to the operating system until the program terminates (calls ga_terminate).
GA and MA interactions

- Dynamically allocated local memory in GA comes from the Memory Allocator (MA) library.
- MA allocates and manages local memory using stack and heap disciplines.
  - Any buffer allocated and deallocated by a GA operation that needs temporary buffer space comes from the MA stack.
  - Memory to store data in global arrays (not in shared memory) comes from heap.
  - MA has additional features useful for program debugging such as:
    - left and right guards: they are stamps that detect if a memory segment was overwritten by the application (finds memory leaks)
    - named memory segments
    - memory usage statistics for the entire program.
  - What is the maximum amount of “dynamic” memory used in this particular run.
GA and MA interactions [2]

- Explicit use of MA by the application to manage its non-GA local data structures is not necessary but encouraged.
  - Because MA is used implicitly by GA, it has to be initialized before the first global array is allocated.
  - The MA_init function requires users to specify memory for heap and stack. This is because MA:
    - allocates from the operating system only one segment equal in size to the sum of heap and stack,
    - manages both allocation schemes using memory coming from opposite ends of the same segment, and
    - the boundary between free stack and heap memory is dynamic.
  - It is not important what the stack and heap size argument values are as long as the aggregate memory consumption by a program does not exceed their sum at any given time.
GA/MA Memory scope SM systems

Physical Memory

Static Data and Program Instructions

Stack

Heap

Shared Memory Segments

GA

GA

GA

GA

GA

GA

GA
GA/MA Memory scope

Physical Memory

Stack

Heap

Static Data and Program Instructions

GA

GA

GA

GA

GA

GA

GA

GA
What should MA Stack and Heap be?

- Depends on the run-time environment of the program including the availability of shared memory.
  - A part of GA initialization involves initialization of the ARMCI run-time library.
  - ARMCI dynamically determines if the program can use shared memory based on the architecture type and current configuration of the SMP cluster.
    - For example, on uni-processor nodes of the IBM SP shared memory is not used whereas on the IBMSP with SMP nodes it is.
  - This decision is made at run-time.
    - GA reports the information about the type of memory used with the function `ga_uses_ma()`.
      - returns false when shared memory is used and true when MA is used.
What should MA Stack and Heap be? [2]

- Based on this information a consideration of the amount of memory per single process needed to store data in global arrays.

  - The heap size argument value in ma_init.
    - Includes the GA space if GA uses MA Heap storage.
    - Does not include the GA space if Shared Memory is used.

  - The amount of stack space depends on the GA operations used by the program
    - The stack space is only used when a GA operation is executing and it is returned to MA when it completes.
How to have strict control of Memory

GA_Initialize( );
if (GA_Uses_ma( )) {
    status = MA_init(MT_F_DBL, stack, (heap+global) );
} else {
    status = MA_init(mt_dbl,stack,heap) ;
    GA_Set_memory_limit(MA_sizeof(MT_F_DBL, global,MT_BYTE));
}
if(!status) /*we got an error condition here */
    GA_Error("MA failed",911);
What does this mean?

• Three classifications of two kinds of “memory”
  ➢ Local memory per MPI task
    ▪ Stack
    ▪ Heap
  ➢ Global memory per MPI task
    ▪ Global

• Total Dynamic memory = Stack+Heap+Global
  ➢ Program dependent requirements
    ▪ Application memory
    ▪ Global Array utilization

• Set “reasonable defaults”
  ➢ Compile-time parameters for specific machines
    ▪ 10 to 20 % less than Physical Limit