Introduction to Global Arrays

Chao Mei
03/16/2006

Many thanks to Bruce Palmer's slides
Global Arrays (GAs)

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

- **Characteristics:**
  - Shared memory model in context of distributed dense arrays
  - **Much** simpler than message-passing for many applications
  - Compatible with MPI
  - Data locality control similar to distributed memory/message passing model
  - Scalable
Remote Data Access (1)

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 P_0;
    else if(me = P_0) then
        receive block of data from P_N in message buffer;
        unpack data from message buffer to local buffer;
    end if
end loop
Remote Data Access (2)

GAs (C interface):

\texttt{NGA\_Get}\((\text{int g\_a, int lo[\]}, \text{int hi[\]}, \text{void }* \text{buffer, int ld[]});\)

- \texttt{g\_a}: global array handle
- \texttt{ndim}: number of dimensions of the global array
- \texttt{lo[ndim], hi[ndim]}: array of starting/ending indices for global array section
- \texttt{buffer}: pointer to the local buffer array where the data goes
- \texttt{ld[ndim-1]}: array specifying leading dimensions/strides/widths for buffer array

Example:
For \texttt{ga\_get} operation transferring data from the \([10:14,0:4]\) section of 2-dimensional 15x10 global array into local buffer 5x15 array we have:
\begin{align*}
\text{lo} &= \{10,0\}, \text{hi} = \{14,4\}, \text{ld} = \{15\}
\end{align*}
Data Locality

- What data a processor own?
  - `NGA_Distribution(g_a, iproc, lo, hi)`

- Where is the data locally?
  - `NGA_Access(g_a, lo, hi, ptr, ld)`

Notice:
- A call this function normally follows a call to `NGA_Distribution`
- As data is shared, consider issues of mutual exclusion
- Use this information to organize calculation so that maximum use is made of locally held data
Computation Model

Shared Object

- get
- copy to local memory
- local memory

compute/update

Shared Object

- put
- copy to shared object
- local memory
One-sided Communication

• 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:
  - One message is initiated on sending processor (P1) the sending processor can continue computation. Receiving processor (P0) is not involved.
Non-Blocking Communication

- New functionality in GA version 3.3
- Allows overlapping of data transfers and computations
- 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
Structure of GA

Application programming language interface

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

- Fortran 77
- C
- C++
- Python
- Babel

distributed arrays layer
  memory management, index translation

Message Passing
  Global operations

ARMCI
  portable 1-sided communication
  put, get, locks, etc

system specific interfaces
  LAPI, GM/Myrinet, threads, VIA, etc.
Core capabilities

- Distributed array library
  - Dense arrays 1-7 dimensions
  - 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
  - (nonblocking/blocking) gaPu, gaGet, gaAcc

- Interfaces to third party parallel numerical libraries
  - Scalapack, etc
Comparison with Similar Programming Models

- Co-Array Fortran
  - Distributed arrays
  - One-sided communication
  - No global view of data

- UPC
  - Only applicable to C programs
  - Global shared pointers could be used to implement GA functionality

- High level functionality in GA is missing from these systems
Suitability of GA for Applications (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 and/or for out-of-core array-based algorithms

- **Usability Considerations**
  - Data locality must be explicitly available
  - When coding in message passing becomes too complicated
  - When portable performance is important
  - Need object orientation without the overhead of C++
Suitability of GA for Applications (When *not* to use GA)

- **Algorithmic Considerations**
  - For systolic, or nearest neighbor communications with regular communication patterns
  - When synchronization associated with cooperative point-to-point message passing is needed

- **Usability Considerations**:
  - When interprocedural analysis and compiler parallelization is more effective
  - A parallel language support is sufficient and robust compilers available