A C++ Implementation of the Co-Array Programming Model

Maria Eleftheriou  Siddhartha Chatterjee  José E. Moreira
IBM Thomas J. Watson Research Center

POHLL-02 Workshop
New York, NY

Outline

- Background
- Co-Array C++ library motivation
- Implementation of Co-Array C++
- Performance of Co-Array C++
- Future work
Programming Models

- Research on programming models for the Blue Gene/L project
- Message passing model
  - MPI - will be implemented in BG/L
- Global address space model
  - Titanium
  - Unified Parallel C (UPC) - under way
  - Global Arrays
  - Co-Array Fortran

Co-Array Fortran

- Language extension to F95 [Numrich and Reid]
  - Based on earlier F-- work
- Global address space model
  - Shared memory semantics + locality
  - `integer A(10) [*]`
    - Each node has a one-dimensional array of ten integers named `A`
- Two-level addressing
  - `A(offset)[image]`
  - `image` is the rank of the node
  - `offset` is the position of the local data
Co-Array Fortran (continued)

- Program directly stores and loads local and remote data
  - \( v(i) = A(\text{offset})[\text{image}] \)
  - \( A(\text{offset})[\text{image}] = v(i) \)

- Significantly higher level semantics than MPI
  - Subscripting implies communication between images
  - Compiler/RTE responsible for synthesizing and managing communication

Relevant Features of BG/L

- 65,536 dual processor nodes interconnected in torus topology
  - Processors are symmetric in access to memory/devices
  - Non-coherent shared memory on node

- Interconnection network
  - High bandwidth, low latency
  - Nodes can send and receive at aggregate rate of 2GB/s

- Preferred programming model
  - Dedicate one processor to handle inter-node communication
  - Dedicate other processor to run user application
  - Other models are possible
C++ Library for Co-Array Model

- C++ features allow most of Co-Array notation to be implemented naturally
  - Operator overloading (for [] and () operators)
  - Generic programming (CoArray<T>)
- Library implementation is faster to prototype and faster to deploy
- Easier to motivate users to experiment with new library than new language
- Portable across variety of systems
- We wanted to have some fun with C++

Example: Relaxation Code

- Grid represented as one-dimensional CoArray
- The elements of the CoArray are vectors of size "nrows"
- Each image has (nrows+2) elements
  - It "owns" the middle ncols
  - Left and right shadows
- Before relaxation step, image i has to update shadows of images i-1 and i+1
- Synchronize at the end of update
- After update, relaxation step is a strictly local operation
Co-Array Relaxation Code

typedef double vector_t[VECTOR_SIZE];

void laplace(int nrow, int ncol, CoArray<vector_t>& u)
{
    int me = this_image();
    int images = num_images();
    Array<vector_t> new_u(ncol+2);
    int left = me == 0 ? images-1 : me-1;
    int right = me == images-1 ? 0 : me+1;
    u[left](ncol+1) = u[i]; // communication (put)
    u[right](0) = u(ncol); // communication (put)
    sync_all(list_2);
    for (int i = 1; i < ncol+2; i++) {
        new_u[0][i] = u[0][i] + u[j][0] + u[j][1];
        for (int j = 1; j < nrow-2; j++)
            new_u[0][0] = u[0][0] + u[j][0] + u[j][j] + u[j][j+1];
    }
    for (int j = 0; j < ncol+2; j++)
        for (int i = 0; i < nrow-2; i++)
            u[i][j] = new_u[i][j] - 6.0 * u[i][j];
}

Co-Array C++ Implementation

- Computation and communication agents
- Co-Array declaration and operations
- Point-to-point communication
- Group synchronization
Implementing Computation and Communication Agents

- multithreading within a node
- MPI between nodes (prototype)

Communication Thread

Communication Thread

Communication Thread

Communication Thread

Implementing Co-Arrays

- Declare as a C++ object
  - CoArray<double> A(100);
- Collective operation
Implementing Co-Arrays

CoArray<T>::operator() implements access to elements of the local portion of Co-Array
- u = A(i);

CoArray<T>::operator[] implements access to elements of remote portions of Co-Array
- A[node] is a RemoteArray<T>
- A[node](i) is a RemotePtr<T>

RemotePtr<T>::operator= handles inter-image communication for put operation
- A[node](i) = u;

RemotePtr<T>::operator T() handles inter-image communication for get operation
- u = A[node](i);

Implementing Point-to-Point Communication

Local computation thread initiates
- remote write by enqueuing a PutRequest on its comp2comm queue
- remote read by enqueuing a GetRequest on its comp2comm queue

Local communication thread
- dequeues the request
- sends message to remote communication thread
  - Put message
  - Get message
Implementing Point-to-Point Communication (continued)

- Remote communication thread takes the following actions on receiving a message
  - Put, enqueues a `PutRequest` on its comm2comp queue and sends `Ack` message back to local communication thread
  - Get, enqueues a `GetRequest` on its comm2comp queue
- Remote computation thread dequeues requests and takes the following actions
  - Put, completes remote write
  - Get, reads specified memory location and sends `Put` request

Put Operation
Get Operation

Group Synchronization

- `sync_all();`
  - barrier among all images
- `sync_all(list of images);`
  - barrier among all images
  - not all images need to wait for all others!
- Synchronization is implemented primarily by communication thread
Synchronization Operation

Speedup of Jacobi Relaxation
Speedup of Matrix Transposition

Future Work

- Features
  - Array section operations
  - Multiple co-dimensions (process topologies)
- Program transformations with ROSE
- Scalability issues
  - Fill remote array information on demand
  - More scalable synchronization
- Improve performance
  - Eliminate copies (even with cache coherence problem)
  - Use basic packet operations in BG/L
- Better performance characterization
  - On BG/L simulator
  - Eventually on real hardware (2003)
Example: Matrix Transposition

typedef double block_t[ BLOCK_SIZE ][ BLOCK_SIZE ];
void MatrixTranspose (CoArray<block_t>& u, int nrow, int ncol )
    
        int me = u.this_image(), comm_size = u.num_images();
        CoArray<block_t> b(nrow, ncol);
        block_t temp;
        CoArray<block_t>:: sync_all();
        for (int i = 0; i < nrow; i++) {
            int j = (i+me*ncol)%nrow;
            for (int j = 0; j < ncol; j++) {
                transposeBlock(i, j, temp); // transpose local block
                b[i] = temp; //comm (put)
            }
        }
        CoArray<block_t>:: sync_all();
    
void transposeBlock (block_t & src, block_t & dst )
    
        for (int i = 0; i < BLOCK_SIZE; i++)
            for (int j = 0; j < BLOCK_SIZE; j++)
                dst[i][j] = src[j][i];
    