ADVANCES, APPLICATIONS AND PERFORMANCE OF THE GLOBAL ARRAYS SHARED MEMORY PROGRAMMING TOOLKIT

Presented by: Alexandros Papados (AMSC)

University of Maryland, College Park:
CMSC 714: High Performance Computing

March 1, 2021

Table of Contents

Table of Contents

Introduction

- Shared memory and distributed memory models have advantages and shortcomings
- **Shared Memory**:
  - Much easier to use but it ignores data locality/placement
- **Distributed Memory**
  - Offer performance and scalability but they are difficult to program
- Global Arrays toolkit (Nieplocha, Harrison, and Littlefield 1994, 1996; Nieplocha et al. 2002a) attempts to offer the best features of both models

Table of Contents

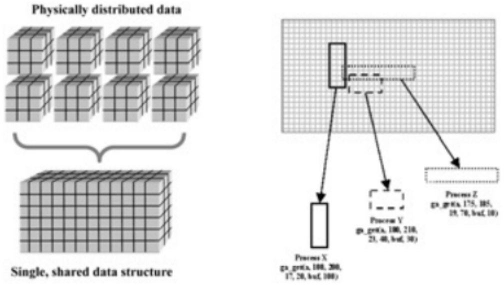
Global Array Tool Kit

- Created to provide application programmers with an interface that allows them to distribute data while maintaining the type of global index space and programming syntax when using only one processor
- **Goal:** Free the programmer from the low level management of communication and allow them to deal with their problems at the level at which they were originally formulated
- Compatibility of GA with MPI enables the programmer to take advantage of the existing MPI software/libraries when available and appropriate

GA Data Structure

- Implements shared-memory programming model in which data locality is managed by the programmer
- Management is achieved by calls to functions that transfer data between a global address space (a distributed array) and local storage

GA Data Structure



Physically distributed data
Single, shared data structure
Process X
ga_get(a, 100, 200, 17, 20, buf, 100)
Process Y
ga_get(a, 180, 210, 23, 40, buf, 30)
Process Z
ga_get(a, 175, 185, 19, 70, buf, 10)

Fig. 1 **Dual view of GA data structures (left). Any part of GA data can be accessed independently by any process at any time (right).**

Languages

- Fortran
- C/C++
- Python

Table of Contents

Global Array Models

- The shared memory model based on Global Arrays combines the advantages of a distributed memory model with the ease of use of shared memory
- Exploits SMP locality and deliver peak performance within the SMP by placing user's data in shared memory and allowing direct access rather than through a message-passing protocol
- Allows users to make use of the fact that remote data is slower to access than local data and to optimize data reuse and minimize communication in their algorithms
- **Advantage:** Optimizing and moving only the data requested by the user avoids issues such as false sharing, data coherence overheads, and redundant data transfers present in some software-based distributed shared memory

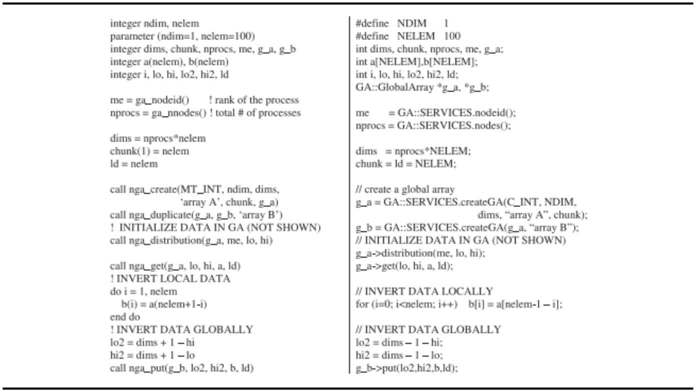
Table of Contents

Global Array Toolkit Functionality

- Basic components of the Global Arrays toolkit are function calls to create global arrays, copy data to, from, and between global arrays, and identify and access the portions of the global array data that are held locally
- ***nga_create*** creates new global arrays
- ***nga_get*** can be used to move a block of distributed data from a global array to a local buffer and has a relatively simple argument list
- ***nga_distribution*** takes a processor ID and an array handle as its arguments and returns a set of lower and upper indices in the global address space representing the local data block
- ***nga_access*** returns an array index and an array of strides to the locally held data
- ***nga_copy_patch*** can be used to move a patch, identified by a set of lower and upper indices in the global index space, from one global array to a patch located within another global array

Table of Contents

Linear Algebra Example: $AB = C$



```fortran
integer ndim, nelem
parameter (ndim=1, nelem=100)
integer dims, chunk, nprocs, me, g_a, g_b
integer a(nelem), b(nelem)
integer i, lo, hi, lo2, hi2, ld

me = ga_nodeid()       ! rank of the process
nprocs = ga_nnodes() ! total # of processes

dims = nprocs*nelem
chunk(1) = nelem
ld = nelem

call nga_create(MT_INT, ndim, dims,
                    'array A', chunk, g_a)
call nga_duplicate(g_a, g_b, 'array B')
!  INITIALIZE DATA IN GA (NOT SHOWN)
call nga_distribution(g_a, me, lo, hi)

call nga_get(g_a, lo, hi, a, ld)
! INVERT LOCAL DATA
do i = 1, nelem
   b(i) = a(nelem+1-i)
end do
! INVERT DATA GLOBALLY
lo2 = dims + 1 – hi
hi2 = dims + 1 – lo
call nga_put(g_b, lo2, hi2, b, ld)
```

```cpp
#define   NDIM      1
#define   NELEM   100
int dims, chunk, nprocs, me, g_a;
int a[NELEM],b[NELEM];
int i, lo, hi, lo2, hi2, ld;
GA::GlobalArray *g_a, *g_b;

me       = GA::SERVICES.nodeid();
nprocs = GA::SERVICES.nodes();

dims   = nprocs*NELEM;
chunk = ld = NELEM;

// create a global array
g_a = GA::SERVICES.createGA(C_INT, NDIM,
                              dims, "array A", chunk);
g_b = GA::SERVICES.createGA(g_a, "array B");
// INITIALIZE DATA IN GA (NOT SHOWN)
g_a->distribution(me, lo, hi);
g_a->get(lo, hi, a, ld);

// INVERT DATA LOCALLY
for (i=0; i<nelem; i++)    b[i] = a[nelem-1 – i];

// INVERT DATA GLOBALLY
lo2 = dims – 1 – hi;
hi2 = dims – 1 – lo;
g_b->put(lo2,hi2,b,ld);
```

Fig. 2 Example Fortran (left) and C++ (right) code for transposing elements of an array.

Linear Algebra Example: $AB = C$

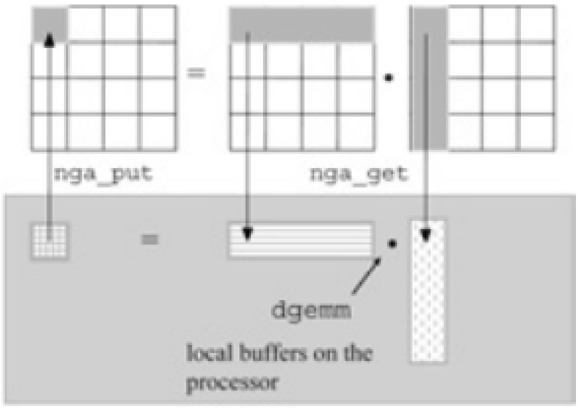


Fig. 3 Schematic representation of distributed matrix multiply, C = A·B.

Table of Contents

Ghost Cells and Periodic Boundary Conditions

- Many applications simulating physical phenomena defined on regular grids benefit from explicit support for ghost cells
- These capabilities have been added recently to Global Arrays, along with the corresponding update and shift operations that operate on ghost cell regions
- The update operation fills in the ghost cells with the visible data residing on neighboring processors.
- Local data on each processor contains the locally held "visible" data plus data from the neighboring elements of the global array, which has been used to fill in the ghost cells
- GA also allows ghost cell widths to be set to arbitrary values in each dimension, thereby allowing programmers to improve performance by combining multiple fields into one global array and using multiple time steps between ghost cell updates

Ghost Cells and Periodic Boundary Conditions

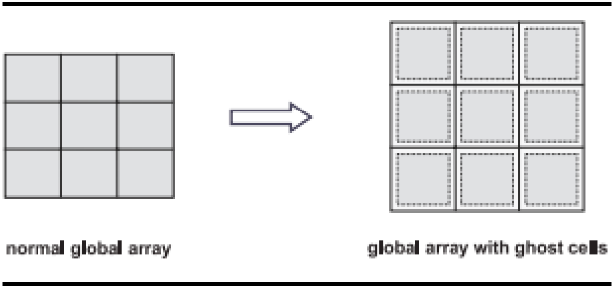


Table of Contents

Applications

- Computational Molecular Dynamics
- Lattice Boltzmann Simulations
- Density Functional Theory
- AMR-based Computational Physics

Table of Contents

References

[1] Nieplocha, Jarek, et al. "*Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit.*" The International Journal of High Performance Computing Applications, vol. 20, no. 2, 2006, pp. 203–31. Crossref, doi:10.1177/1094342006064503.