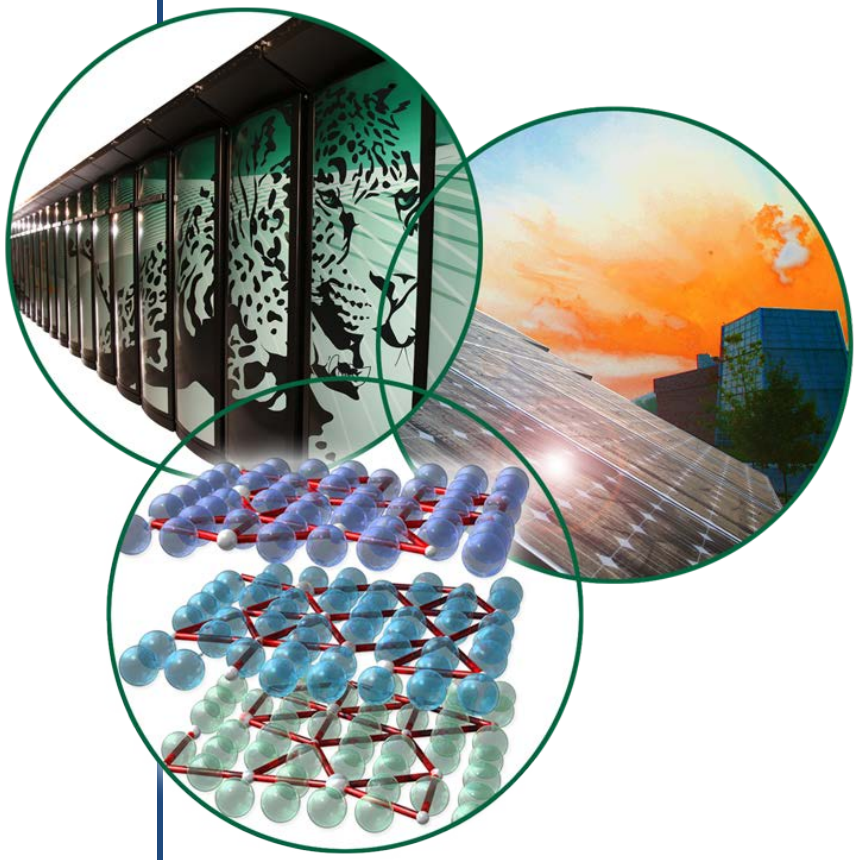
A Global View Programming Abstraction for Transitioning MPI Codes to PGAS Languages



Tiffany M. Mintz, Oscar Hernandez, and David E. Bernholdt

U.S. DEPARTMENT OF ENERGY

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Current HPC Standards

Software: Sequential Language + MPI

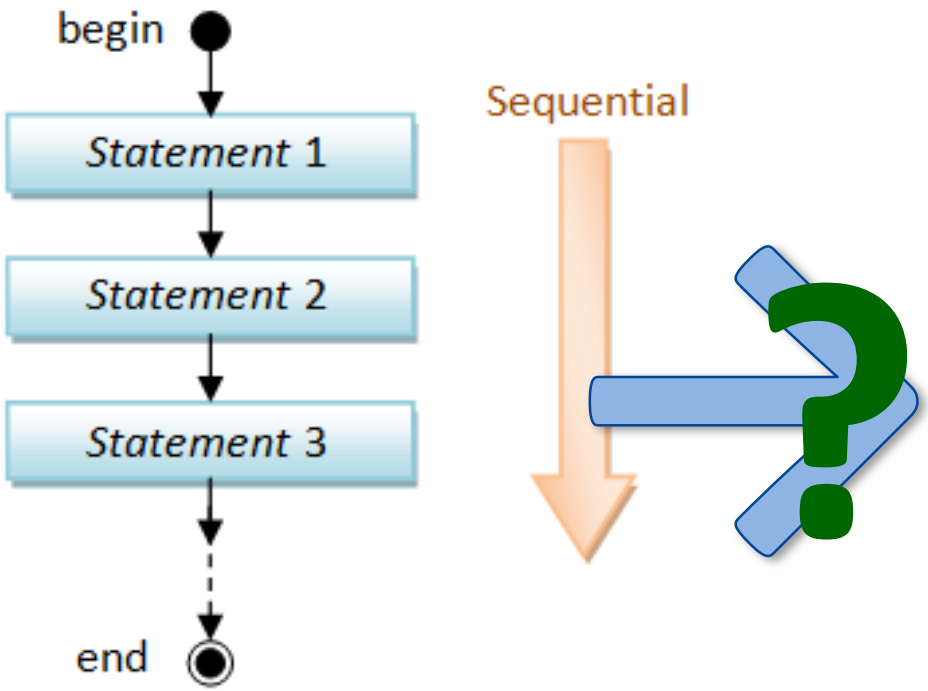


Image Source: http://www3.ntu.edu.sg/home/ehchua/programming/java/images/Construct_Sequential.png

Hardware: Multi-Core NUMA Architectures

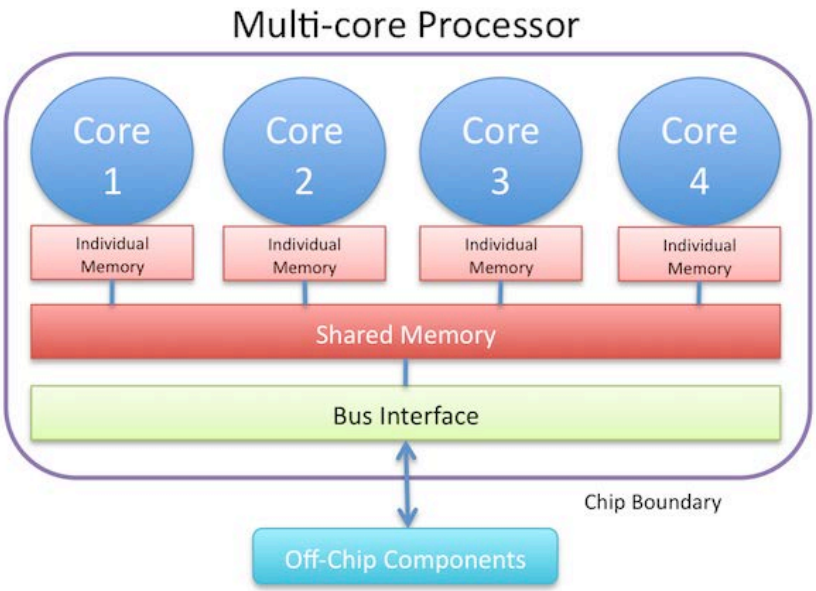


Image Source: http://www.cs.wustl.edu/~jain/cse567-11/ftp/multcore/fig1.png

Compatible Programming Paradigm

- Partitioned Global Address Space (PGAS)
  - Some Languages: Fortress, X10, Chapel

Global View of Data

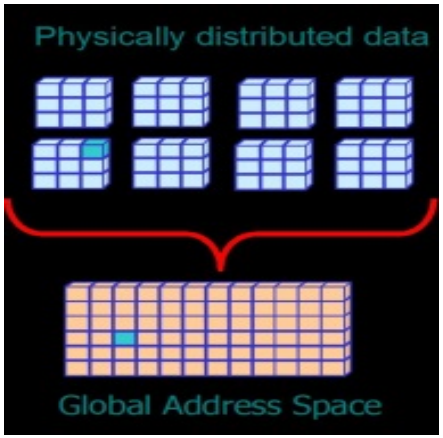


Image Source: http://www.pnl.gov/science/images/highlights/computing/arrays.jpg

Data Distributions

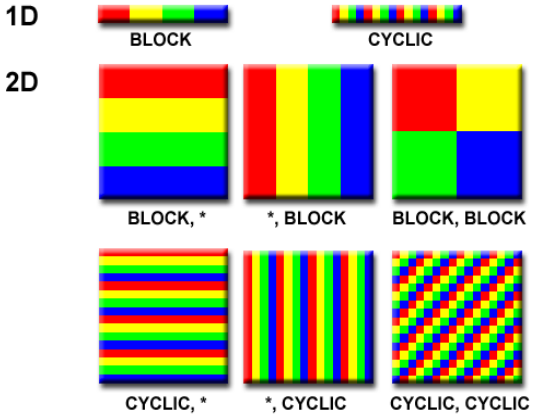


Image Source: https://computing.llnl.gov/tutorials/parallel_comp/images/distributions.gif

Locality for Data Movement

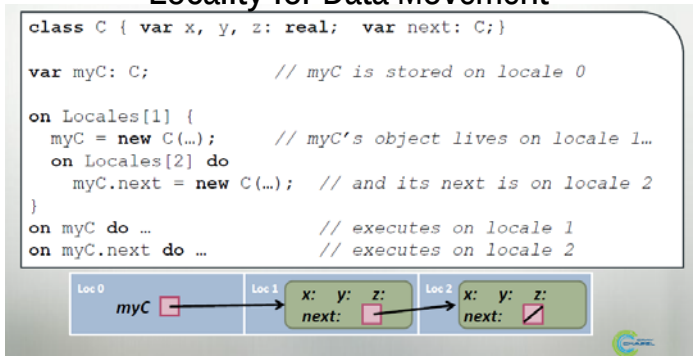


Image Source: http://chapel.cray.com/tutorials/Bergen/SC12-5-Locales.pdf

Mechanism for Transitioning Codes

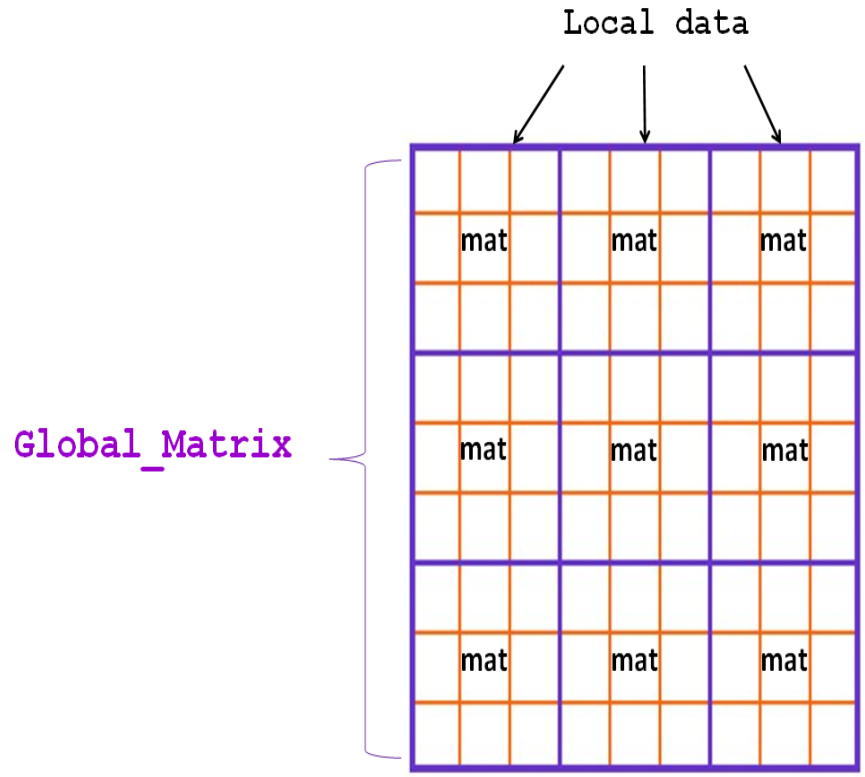
- Global View Directives
  - Data Distributions
  - Processor Groups and Arrangements
  - Data Movement

```
for (i=0; i<NRA; i++)
  for (j=0; j<NCA; j++)
    a[i][j]= i+j;

for (dest=1; dest<numworkers; dest++) {
  rows = (dest <= extra) ? averow+1 : averow;
  MPI_Send(a,N, MPI_DOUBLE, p, q,comm);
  offset = offset + rows;
}
rows = (extra > 0) ? averow+1 : averow;
```

```
#pragma global  forall  …

for (dest=1; dest<numworkers; dest++) {
  rows = (dest <= extra) ? averow+1 : averow;
  MPI_Send(a,N, MPI_DOUBLE, p, q,comm);
  offset = offset + rows;
}
rows = (extra > 0) ? averow+1 : averow;
```

Data Distributions

- BLOCK, CYCLIC, BLOCK-CYCLIC
- Directive: data_map
- Clauses:
  - local_data
  - global_domain
  - distribution
  - expand

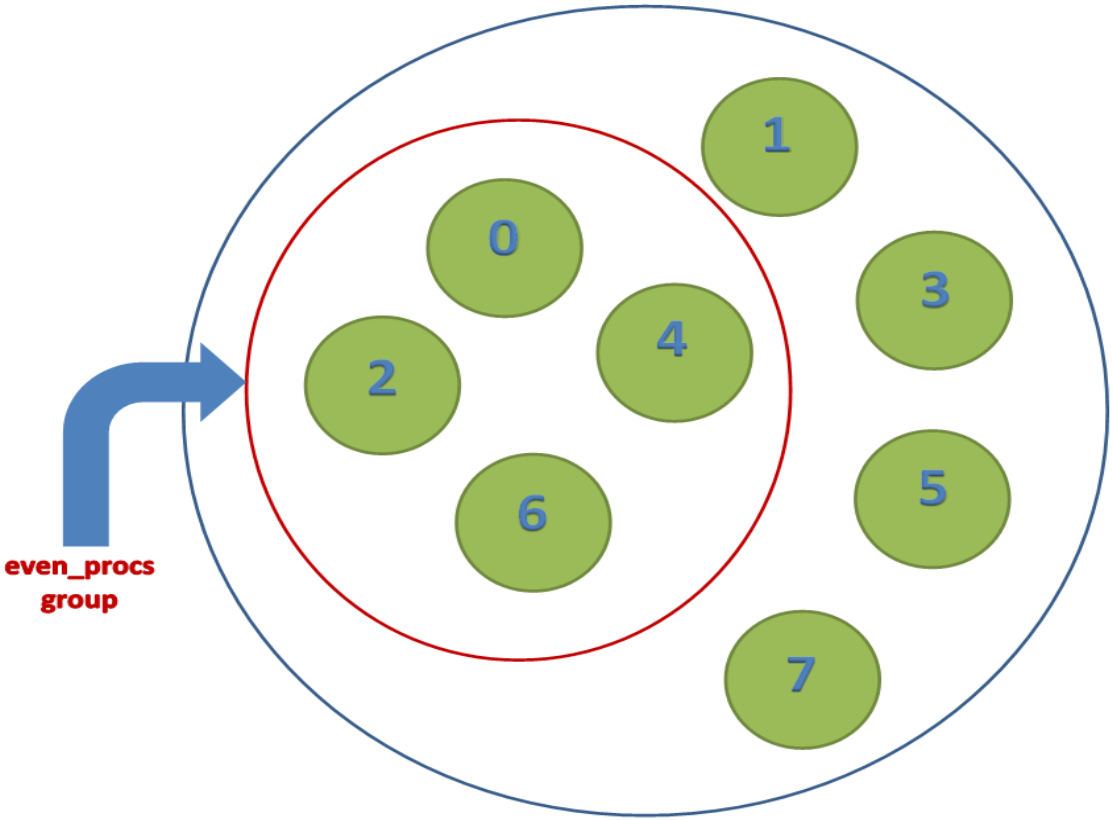



```
#pragma global data_map local_data(mat<XPLINES,XPCOLS>:[1..plines, 1..pcols]) \
  global_domain(Global_Matrix:[1..(plines)*proc_y, 1..(pcols)*proc_x]) \
  distribution(BLOCK) expand(mat:[1,1]:MIRRORED)
```

Processor Groups

- **Directive:** group
- **Clauses:** name, processors

```
#pragma global group name("even_procs") processors(0:nprocs:2)
```



Processor Arrangements

- Directive: arrange_processors

Arrangement	Clauses	Relationships in Arrangement
GRID <1D|2D|3D>	size_x_axis size_y_axis size_z_axis	North Neighbor South Neighbor East Neighbor West Neighbor
MASTER-WORKER	master_processor*	Master Worker Master_Worker
TREE	root* order*	Parent Sibling Children Root Leaves Depth (tree and positional)
RING	direction*	Previous Next
LIST	direction*	Previous Next Head Tail

*indicates optional clauses that have some default behavior defined when not in use

Data Movement

- Global forall loops
  - Directive: forall
  - Clauses: index_var, domain, expression

Global forall example:

```
#pragma global  forall  index_var(i, j) \
      domain(global_a) expression(global_a[i][j] = i+j)
```

- Global updates
  - Directive: update
  - Clauses: update_domain, update_mirror, on
  - Translated to OpenSHMEM

Global update example:

```
#pragma global update update_domain(global_c) on(MASTER)
```


Managed by UT-Battelle
for the U.S. Department of Energy

Preliminary Experiment: Jacobi Algorithm

Expressing Block Distribution & 2D Grid

Directive Assertion of Matrix Block Distribution

```
#pragma comm data_map local_data(mat<N,M>:[1..lines, 1..cols]) \
  global_handle(Global_Matrix:[1..(lines)*proc_y, 1..(cols)*proc_x]) \
  distribution(BLOCK)
```

Directive Assertion of 2D Grid Processor Arrangement

```
#pragma global arrange_processors arrangement(GRID:2D) \
     size_x_axis(proc_x)  size_y_axis(proc_y)
```

C code to mimic 2D Processor Grid

```
x = myid % proc_x;
y = myid / proc_x;

if( y == 0 )
  north = BORDER;
else
  north = x + (y - 1) * proc_x;
if( y == proc_y - 1 )
  south = BORDER;
else
  south = x + (y + 1) * proc_x;
if( x == 0 )
  west = BORDER;
else
  west = myid - 1;
if( x == proc_x - 1 )
  east = BORDER;
else
  east = myid + 1;
```

Global update to stencil ghost cells

Directive Assertion for Updating Neighbor Data

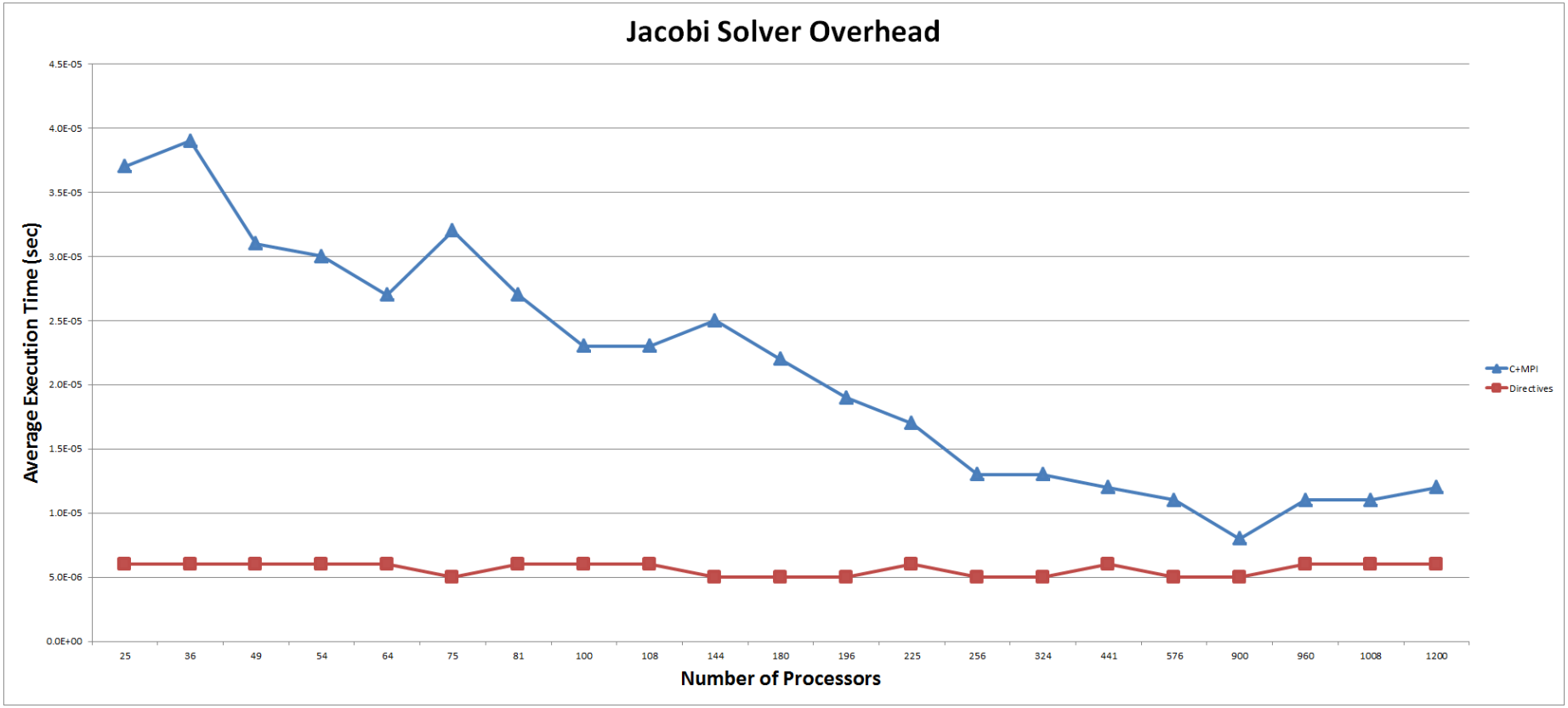
```
#pragma global update update_mirror(Global_Matrix) on(ALL_NEIGHBORS)
```

C+MPI Code to Exchange Data Across Neighboring Processors

```
if( west != BORDER ){
  for( i = 0; i < lines; i++ )
    *(awest + i) = MAT(i, 1);
  MPI_Send(awest,lines,MPI_DOUBLE,west,MSGTYPE,MPI_COMM_WORLD);
}

if( east != BORDER ){
  for( i = 0; i < lines; i++ )
    *(aeast + i) = MAT(i, cols - 2);
  MPI_Send(aeast,lines,MPI_DOUBLE,east,MSGTYPE,MPI_COMM_WORLD);
}

if( north != BORDER )
  MPI_Send(mat+cols,cols,MPI_DOUBLE,north,MSGTYPE,MPI_COMM_WORLD);

if( south != BORDER )
  MPI_Send(mat+(lines-2)*cols,cols,MPI_DOUBLE,south,MSGTYPE,MPI_COMM_WORLD);


if( west != BORDER ){
  MPI_Recv(awest,lines,MPI_DOUBLE,west,MSGTYPE,MPI_COMM_WORLD);
    for( i = 0; i < lines; i++ )
      MAT(i, 0) = *(awest + i);
}

if( east != BORDER ){
  MPI_Recv(aeast,lines,MPI_DOUBLE,east,MSGTYPE,MPI_COMM_WORLD);
      for( i = 0; i < lines; i++ )
        MAT(i, cols - 1) = *(aeast + i);
}

if( north != BORDER )
  MPI_Recv(mat,cols,MPI_DOUBLE,north,MSGTYPE,MPI_COMM_WORLD);

if( south != BORDER )
  MPI_Recv(mat+(lines-1)*cols,cols,MPI_DOUBLE,south,MSGTYPE,MPI_COMM_WORLD);
```

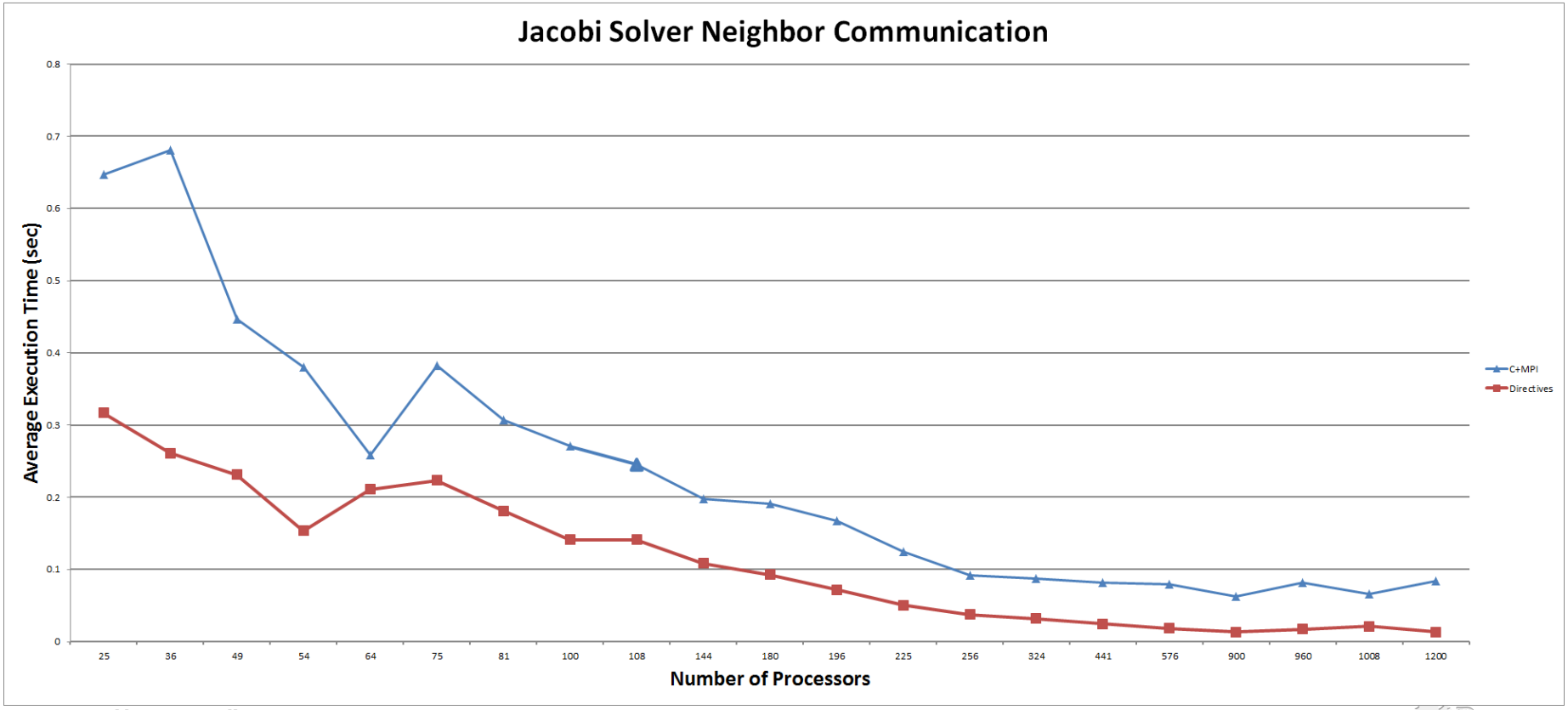
Jacobi Overhead

- C+MPI overhead ~3.7x greater that directive



Jacobi Communication

- Directive translated using OpenSHMEM strided put
- ~2x average speedup using directive update



Preliminary Experiment: Matrix Multiply

C + MPI Matrix-Matrix Multiply

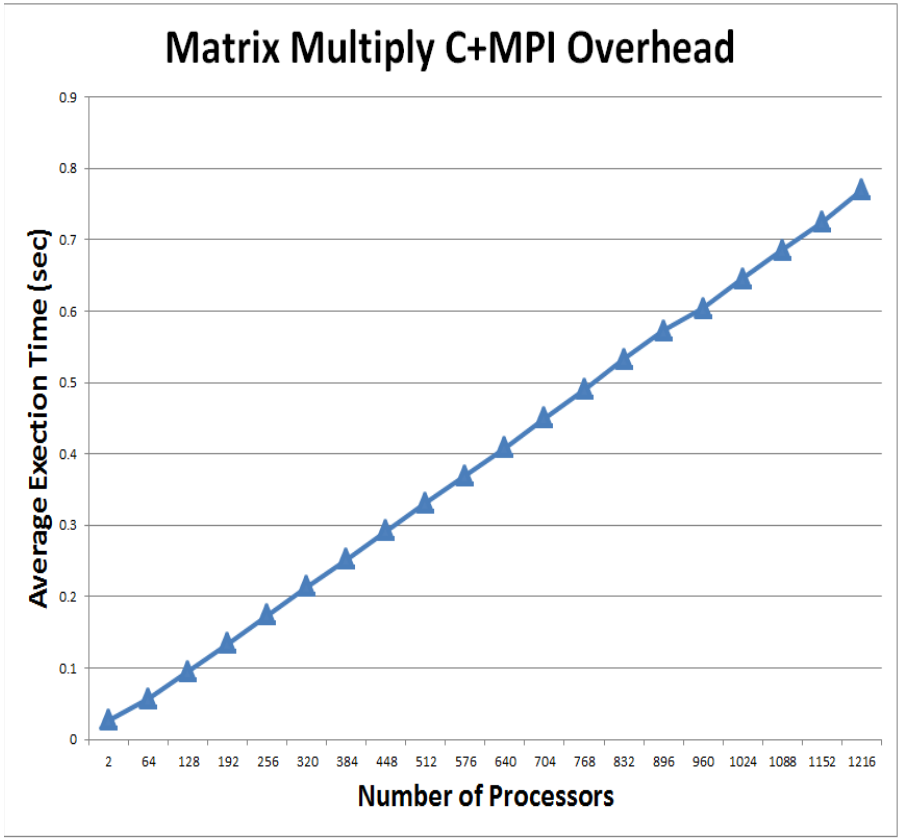
```
if (taskid == MASTER) {
   for (i=0; i<NRA; i++)
      for (j=0; j<NCA; j++)
         a[i][j]= i+j;

   offset = (extra > 0) ? averow+1 : averow;
   for (dest=1; dest<numworkers; dest++) {
      rows = (dest <= extra) ? averow+1 : averow;
      MPI_Send(&offset, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
      MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, dest, FROM_MASTER, MPI_COMM_WORLD);
      offset = offset + rows;
   }
   rows = (extra > 0) ? averow+1 : averow;

   for (k=0; k<NCB; k++)
      for (i=0; i<rows; i++) {
         c[i][k] = 0.0;
         for (j=0; j<NCA; j++)
            c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }
   if(extra > 0) {
     list_offsets[0] = 0;
     row_cnts[0] = (averow+1)*NCB;
     for (dest=1; dest<numworkers; dest++) {
        rows = (dest <= extra) ? averow+1 : averow;
        row_cnts[dest] = rows*NCB;
        list_offsets[dest] = list_offsets[dest-1]+row_cnts[dest-1];
     }
     MPI_Gatherv(MPI_IN_PLACE,0,MPI_DATATYPE_NULL,&c,row_cnts,list_offsets,MPI_DOUBLE,MASTER,MPI_COMM_WORLD);
   }else
     MPI_Gather(MPI_IN_PLACE,0,MPI_DATATYPE_NULL,&c,rows*NCB,MPI_DOUBLE,MASTER,MPI_COMM_WORLD);
}

if (taskid > MASTER) {
   MPI_Recv(&offset, 1, MPI_INT, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
   MPI_Recv(&rows, 1, MPI_INT, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
   MPI_Recv(&a, rows*NCA, MPI_DOUBLE, FROM_MASTER, mtype, MPI_COMM_WORLD, &status);

   for (k=0; k<NCB; k++)
      for (i=0; i<rows; i++) {
         c[i][k] = 0.0;
         for (j=0; j<NCA; j++)
            c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }
   if(extra > 0)
     MPI_Gatherv(&c,rows*NCB,MPI_DOUBLE,&c,row_cnts,list_offsets,MPI_DOUBLE,MASTER,MPI_COMM_WORLD);
   else
     MPI_Gather(&c,rows*NCB,MPI_DOUBLE,&c,rows*NCB,MPI_DOUBLE,MASTER,MPI_COMM_WORLD);
}
```

Global View Matrix-Matrix Multiply with Directives

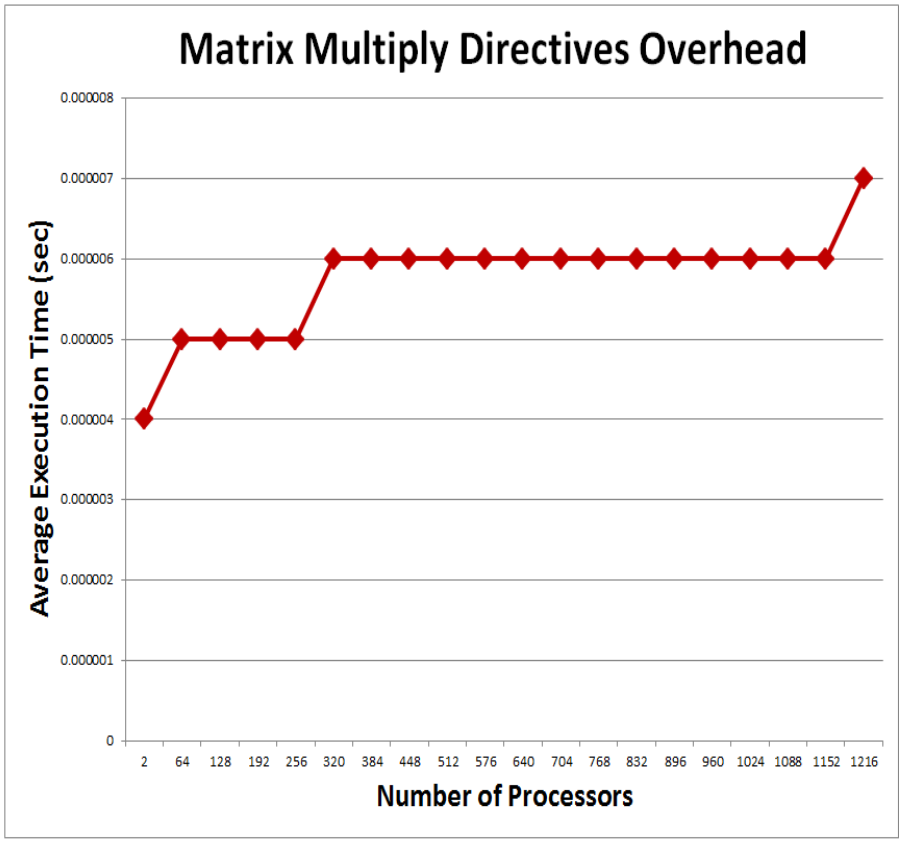
```
#pragma global arrange_processors arrangement(MASTER_WORKER, MASTER_SHARES_WORK)

#pragma global data_map local_data(a<nrows,NCA>:[0..nrows-1,0..NCA-1])  \
     global_domain(global_a:[0..NRA-1,0..NCA-1]) distribution(BLOCK, NONE)

#pragma global data_map local_data(c<nrows, NCB>:[0..nrows-1,0..NCB-1])  \
     global_domain(global_c:[0..NRA-1,0..NCB-1]) distribution(BLOCK, NONE)

#pragma global  forall  index_var(i, j)  domain(global_a)  \
     expression(global_a[i][j] = i+j)

#pragma global forall  index_var(i, j)  domain(global_c)  \
     expression(global_c[i][j] = 0.0)

for(k=0; k<NCB; k++)
  #pragma global forall index_var( i, j) domain( global_a)  \
       expression(global_c[i][k] = global_c[i][k] + global_a[i][j] * b[j][k])

#pragma global update update_domain(global_c) on(MASTER)
```

Matrix Multiply Overhead

- Directives provide several orders of magnitude overhead performance speedup



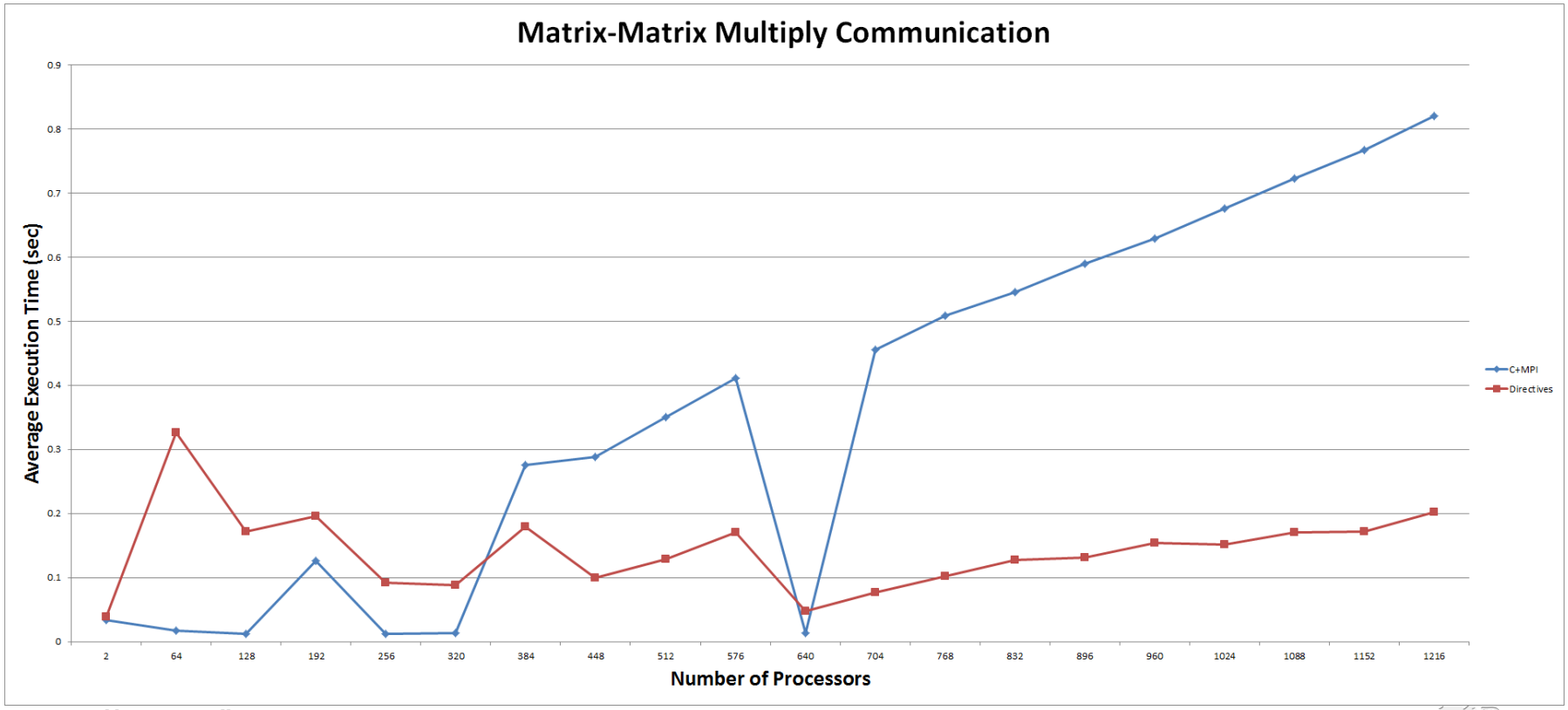
(a)



(b)

Matrix Multiply Communication

- update translated to OpenSHMEM put & barrier synchronization
- ~2.5x average speedup using update



Wrap Up

- Have an initial set of global view directives
- Preliminary experiments show promising results

Looking Forward

- Expand directive representation of PGAS language constructs
- Develop experiments with larger applications

Acknowledgements

- Co-Investigators: Oscar Hernandez and David Bernholdt
- Sponsors

Questions?