L17: Introduction to "Irregular" Algorithms and MPI, cont.

November 8, 2011

Administrative

- Class cancelled, Tuesday, November 15
- Guest Lecture, Thursday, November 17, Ganesh Gopalakrishnan
- CUDA Project 4, due November 21
  - Available on CADE Linux machines (lab1 and lab3) and Windows machines (lab5 and lab6)
  - You can also use your own Nvidia GPUs

Outline

- Introduction to irregular parallel computation
  - Sparse matrix operations and graph algorithms
- Finish MPI discussion
  - Review blocking and non-blocking communication
  - One-sided communication
- Sources for this lecture:
  - http://mpi.deino.net/mpi_functions/
  - Kathy Yelick/Jim Demmel (UC Berkeley): CS 267, Spr 07 • http://www.eecs.berkeley.edu/~yelick/cs267_sp07/lectures
  - "Implementing Sparse Matrix-Vector Multiplication on Throughput Oriented Processors," Bell and Garland (Nvidia), SC09, Nov. 2009.

Motivation: Dense Array-Based Computation

- Dense arrays and loop-based data-parallel computation has been the focus of this class so far
- Review: what have you learned about parallelizing such computations?
  - Good source of data parallelism and balanced load
  - Top500 measured with dense linear algebra
    - How fast is your computer?" = "How fast can you solve dense Ax=b?"
  - Many domains of applicability, not just scientific computing
    - Graphics and games, knowledge discovery, social networks, biomedical imaging, signal processing
- What about "irregular" computations?
  - On sparse matrices? (i.e., many elements are zero)
  - On graphs?
  - Start with representations and some key concepts

Sparse Matrix or Graph Applications

- Telephone network design
  - Original application, algorithm due to Kernighan
- Load Balancing while Minimizing Communication
- Sparse Matrix times Vector Multiplication
  - Solving PDEs • N = {1,…,n}, (j,k) in E if A(j,k) nonzero, •
  - WN(j) = #nonzeros in row j, WE(j,k) = 1
- VLSI Layout
  - N = {units on chip}, E = {wires}, WE(j,k) = wire length
- Data mining and clustering
- Analysis of social networks
- Physical Mapping of DNA

Dense Linear Algebra vs. Sparse Linear Algebra

Matrix vector multiply:

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        a[i] += c[j][i]*b[j];
```

- What if n is very large, and some large percentage (say 90%) of c is zeros?
- Should you represent all those zeros? If not, how to represent "c"?

Sparse Linear Algebra

- Suppose you are applying matrix-vector multiply and the matrix has lots of zero elements
  - Computation cost? Space requirements?
- General sparse matrix representation concepts
  - Primarily only represent the nonzero data values
  - Auxiliary data structures describe placement of nonzeros in "dense matrix"

Some common representations

$$A = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\text{data} = \begin{bmatrix} * & 1 & 7 \\ * & 2 & 8 \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad \text{offsets} = [-2\ 0\ 1]$$

DIA: Store elements along a set of diagonals.

ptr = [0 2 4 7 9]
indices = [0 1 1 2 0 2 3 1 3]
data = [1 7 2 8 5 3 9 6 4]

Compressed Sparse Row (CSR): Store only nonzero elements, with "ptr" to beginning of each row and "indices" representing column.

$$\text{data} = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad \text{indices} = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix}$$

ELL: Store a set of K elements per row and pad as needed. Best suited when number non-zeros roughly consistent across rows.

row = [0 0 1 1 2 2 2 3 3]
indices = [0 1 1 2 0 2 3 1 3]
data = [1 7 2 8 5 3 9 6 4]

COO: Store nonzero elements and their corresponding "coordinates".

THE UNIVERSITY OF UTAH

Connect to dense linear algebra

Dense matvec from L15:

```
for (i=0; i<n; i++) {
   for (j=0; j<n; j++) {
      a[i] += c[j][i] * b[j];
   }
}
```

Equivalent CSR matvec:

```
for (i=0; i<nr; i++) {
   for (j = ptr[i]; j<ptr[i+1]-1; j++)
     t[i] += data[j] * b[indices[j]];
```

Other Representation Examples

- Blocked CSR
  - Represent non-zeros as a set of blocks, usually of fixed size
  - Within each block, treat as dense and pad block with zeros
  - Block looks like standard matvec
  - So performs well for blocks of decent size
- Hybrid ELL and COO
  - Find a "K" value that works for most of matrix
  - Use COO for rows with more nonzeros (or even significantly fewer)

Today's MPI Focus – Communication Primitives

- Collective communication
  - Reductions, Broadcast, Scatter, Gather
- Blocking communication
  - Overhead
  - Deadlock?
- Non-blocking
- One-sided communication

Quick MPI Review

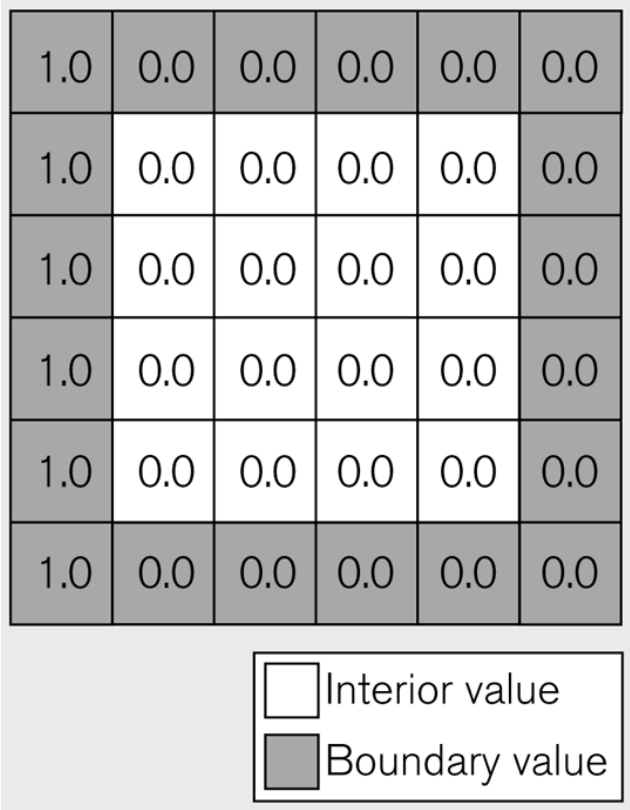
- Six most common MPI Commands (aka, Six Command MPI)
  - **MPI_Init**
  - **MPI_Finalize**
  - **MPI_Comm_size**
  - **MPI_Comm_rank**
  - **MPI_Send**
  - **MPI_Recv**
- Send and Receive refer to "point-to-point" communication
- Last time we also showed Broadcast communication
  - **Reduce**

More difficult p2p example: 2D relaxation

Replaces each interior value by the average of its four nearest neighbors.

Sequential code:

```
for (i=1; i<n-1; i++)
   for (j=1; j<n-1; j++)
      b[i,j] = (a[i-1][j]+a[i][j-1]+
            a[i+1][j]+a[i][j+1])/4.0;
```



MPI code, main loop of 2D SOR computation

```
  1  #define Top      0
  2  #define Left     0
  3  #define Right    (Cols-1)
  4  #define Bottom   (Rows-1)
  5
  6  #define NorthPE(i)       ((i)-Cols)
  7  #define SouthPE(i)       ((i)+Cols)
  8  #define EastPE(i)        ((i)+1)
  9  #define WestPE(i)        ((i)-1)
...
101  do
102  { /*
103    * Send data to four neighbors
104    */
105    if(row !=Top)                                  /* Send North */
106    {
107      MPI_Send(&val[1][1], Width-2, MPI_FLOAT,
108               NorthPE(myID), tag, MPI_COMM_WORLD);
109    }
110
111    if(col !=Right)                                /* Send East */
112    {
113      for(i=1; i<Height-1; i++)
114      {
115        buffer[i-1]=val[i][Width-2];
116      }
117      MPI_Send(buffer, Height-2, MPI_FLOAT,
118               EastPE(myID), tag, MPI_COMM_WORLD);
119    }
120
121    if(row !=Bottom)                               /* Send South */
122    {
123      MPI_Send(&val[Height-2][1], Width-2, MPI_FLOAT,
124               SouthPE(myID), tag, MPI_COMM_WORLD);
125    }
126
127    if(col !=Left)                                 /* Send West */
128    {
```

MPI code, main loop of 2D SOR computation, cont.

```
129       for(i=1; i<Height-1; i++)
130       {
131         buffer[i-1]=val[i][1];
132       }
133       MPI_Send(buffer, Height-2, MPI_FLOAT,
134                WestPE(myID), tag, MPI_COMM_WORLD);
135     }
136
137     /*
138      * Receive messages
139      */
140     if(row !=Top)                          /* Receive from North */
141     {
142       MPI_Recv(&val[0][1], Width-2, MPI_FLOAT,
143                NorthPE(myID), tag, MPI_COMM_WORLD, &status);
144     }
145
146     if(col !=Right)                        /* Receive from East */
147     {
148       MPI_Recv(&buffer, Height-2, MPI_FLOAT,
149                EastPE(myID), tag, MPI_COMM_WORLD, &status);
150       for(i=1; i<Height-1; i++)
151       {
152         val[i][Width-1]=buffer[i-1];
153       }
154     }
155
156     if(row !=Bottom)                       /* Receive from South */
157     {
158       MPI_Recv(&val[0][Height-1], Width-2, MPI_FLOAT,
159                SouthPE(myID), tag, MPI_COMM_WORLD, &status);
160     }
161
162     if(col !=Left)                         /* Receive from West */
163     {
164       MPI_Recv(&buffer, Height-2, MPI_FLOAT,
165                WestPE(myID), tag, MPI_COMM_WORLD, &status);
166       for(i=1; i<Height-1; i++)
167       {
168         val[i][0]=buffer[i-1];
169       }
170     }
171
172     delta=0.0; /* Calculate average, delta for all points */
173     for(i=1; i<Height-1; i++)
174     {
175       for(j=1; j<Width-1; j++)
176       {
```

MPI code, main loop of 2D SOR computation, cont.

```
177          average=(val[i-1][j]+val[i][j+1]+
178                   val[i+1][j]+val[i][j-1])/4;
179          delta=Max(delta, Abs(average-val[i][j]));
180          new[i][j]=average;
181        }
182      }
183
184      /* Find maximum diff */
185      MPI_Reduce(&delta, &globalDelta, 1, MPI_FLOAT, MPI_MIN,
186                 RootProcess, MPI_COMM_WORLD);
187      Swap(val, new);
188    } while(globalDelta < THRESHOLD);
```

Broadcast: Collective communication within a group

```
 1  int numCols;                /* initialized elsewhere */
 2
 3  void broadcast_example()
 4  {
 5    int **ranks;              /* the ranks that belong to each group */
 6    int myRank;
 7    int rowNumber;            /* row number of this process */
 8    int random;               /* value that we would like to broadcast */
 9    rowNumber=myRank/numCols;
10    MPI_Group globalGroup, newGroup;
11    MPI_Comm rowComm[numCols];
12
13    /* initialize ranks[][] array */
14    ranks[0]={0,1,2,3};  /* not legal C */
15    ranks[1]={4,5,6,7};
16    ranks[2]={8,9,10,11};
17    ranks[3]={12,13,14,15};
18
19    /* Extract the original group handle */
20    MPI_Comm_group(MPI_COMM_WORLD, &globalGroup);
21
22    /* Define the new group */
23    MPI_Group_incl(globalGroup, P/numCols, ranks[rowNumber], &newGroup);
24
25    /* Create new communicator */
26    MPI_Comm_create(MPI_COMM_WORLD, newGroup, &newComm);
27
28    random=rand();
29
30    /* Broadcast 'random' across rows */
31    MPI_Bcast(&random, 1, MPI_, rowNumber*numCols, newComm);
32  }
```

MPI_Scatter()

MPI_Scatter()

```
int MPI_Scatter(                        // Scatter routine
  void          *sendbuffer,            // Address of the data to send
  int           sendcount,              // Number of data elements to send
  MPI_Datatype  sendtype,               // Type of data elements to send
  int           destbuffer,             // Address of buffer to receive data
  int           destcount,              // Number of data elements to receive
  MPI_Datatype  desttype,               // Type of data elements to receive
  int           root,                   // Rank of the root process
  MPI_Comm      *comm                   // An MPI communicator
);
```

Arguments:

- The first three arguments specify the address, size, and type of the data elements to send to each process. These arguments only have meaning for the root process.
- The second three arguments specify the address, size, and type of the data elements for each receiving process. The size and type of the sending data and the receiving data may differ as a means of converting data types.
- The seventh argument specifies the root process that is the source of the data.
- The eighth argument specifies the MPI communicator to use.

Notes:

This routine distributes data from the root process to all other processes, including the root. A more sophisticated version of the routine, `MPI_Scatterv()`, allows the root process to send different amounts of data to the various processes. Details can be found in the MPI standard.

Return value:

An MPI error code.

Distribute Data from input using a scatter operation

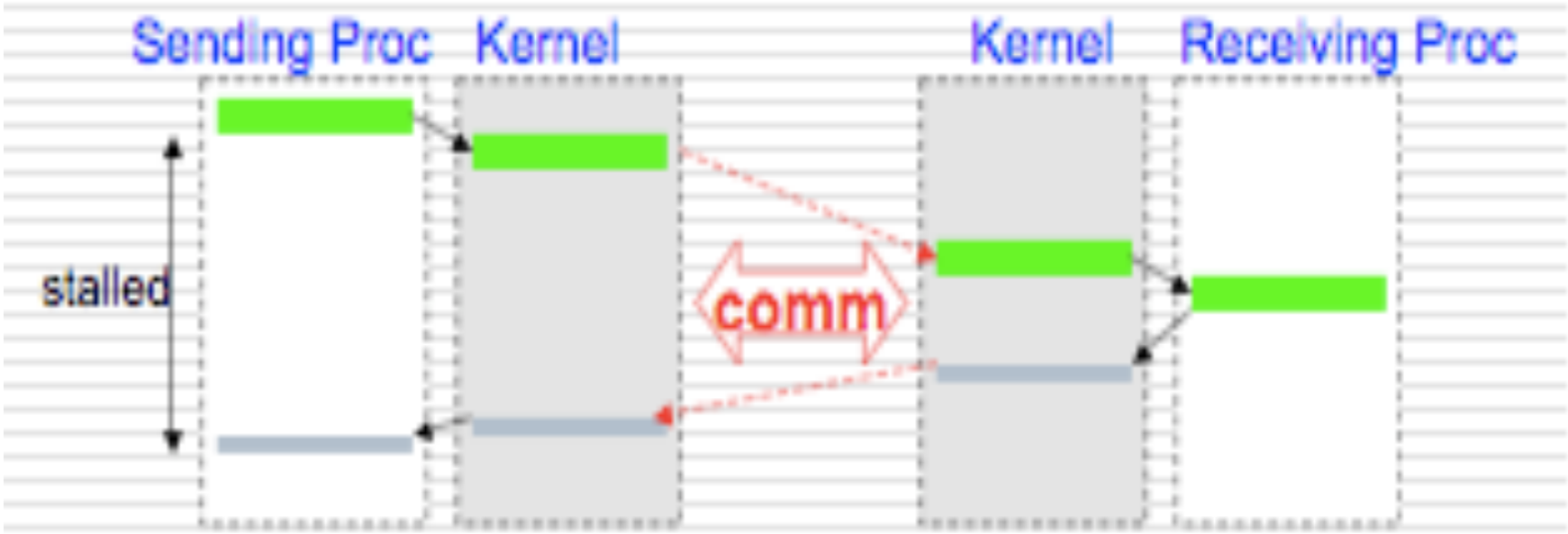
```
16  length_per_process=length/size;
17  myArray=(int *) malloc(length_per_process*sizeof(int));
18
19  array=(int *) malloc(length*sizeof(int));
20
21  /* Read the data, distribute it among the various processes */
22  if(myID==RootProcess)
23  {
24    if((fp=fopen(*argv, "r"))==NULL)
25    {
26      printf("fopen failed on %s\n", filename);
27      exit(0);
28    }
29    fscanf(fp,"%d", &length);        /* read input size */
30
31    for(i=0; i<length-1; i++)        /* read entire input file */
32    {
33      fscanf(fp,"%d", myArray+i);
34    }
35  }
36
37  MPI_Scatter(Array, length_per_process, MPI_INT,
38              myArray, length_per_process, MPI_INT,
39              RootProcess, MPI_COMM_WORLD);
```

Other Basic Features of MPI

- MPI_Gather
  - Analogous to MPI_Scatter
- Scans and reductions (reduction last time)
- Groups, communicators, tags
  - Mechanisms for identifying which processes participate in a communication
- MPI_Bcast
  - Broadcast to all other processes in a "group"

The Path of a Message

- A blocking send visits 4 address spaces



- Besides being time-consuming, it locks processors together quite tightly