Administrative

- Guest Lecture, November 18, Matt Might
- Project proposals: Reviewed all of them, will mail comments to CADE account during office hours
- CUDA Projects status:
  - Available on CADE Linux machines (lab1 and lab3) and Windows machines (lab5 and lab6)
  - Windows instructions hopefully today
  - Still do not have access to "cutil" library, where timing code resides, but expect that later today

Programming Assignment #3: Simple CUDA
Due Monday, November 18, 11:59 PM

Today we will cover Successive Over Relaxation. Here is the sequential code for the core computation, which we parallelize using CUDA:

```c
for(i=1;i<N-1;i++) {
    for(j=1;j<N-1;j++) {
    }
}
```

You are provided with a CUDA template (sor.cu) that (1) provides the sequential implementation; (2) times the computation; and (3) verifies that its output matches the sequential code.

Programming Assignment #3, cont.

Your mission:
- Write parallel CUDA code, including data allocation and copying to/from CPU
- Measure speedup and report
- 45 points for correct implementation
- 5 points for performance
- Extra credit (10 points): use shared memory and compare performance
Programming Assignment #3, cont.

• You can install CUDA on your own computer
  - http://www.nvidia.com/cudazone/
• How to compile under Linux and MacOS
  - Version 2.x:
    nvcc -I/Developer/CUDA/common/inc \ 
    -L/Developer/CUDA/lib sor.cu -lcutil
  - Version 3.x:
    nvcc -I/Developer/GPU\ Computing/C/common/inc -L/ 
    Developer/GPU\ Computing/C/lib sor.cu -lcutil
• Turn in
  - Handin cs4961 p03 <file> (includes source file and 
    explanation of results)

One-Sided Communication

Outline

• Finish MPI discussion
  - Review blocking and non-blocking communication
  - One-sided communication
• Irregular parallel computation
  - Sparse matrix operations and graph algorithms
• 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/
  - "Implementing Sparse Matrix-Vector Multiplication on 
    Throughput Oriented Processors," Bell and Garland (Nvidia), 
    SC09, Nov. 2009.
  - Slides accompanying textbook "Introduction to Parallel 
    Computing" by Grama, Gupta, Karypis and Kumar
  - http://www-users.cs.umn.edu/~karypis/parbook/

MPI One-Sided Communication or Remote Memory Access (RMA)

• Goals of MPI-2 RMA Design
  - Balancing efficiency and portability across a wide class of 
    architectures
  - shared-memory multiprocessors
  - NUMA architectures
  - distributed-memory MPP’s, clusters
  - Workstation networks
• Retaining “look and feel” of MPI-1
• Dealing with subtle memory behavior issues: cache 
  coherence, sequential consistency
**MPI Constructs supporting One-Sided Communication (RMA)**

- **MPI_Win_create** exposes local memory to RMA operation by other processes in a communicator
  - Collective operation
  - Creates window object
- **MPI_Win_free** deallocates window object
- **MPI_Put** moves data from local memory to remote memory
- **MPI_Get** retrieves data from remote memory into local memory
- **MPI_Accumulate** updates remote memory using local values

```c
i = MPI_Alloc_mem(200 * sizeof(int), MPI_INFO_NULL, &A);
i = MPI_Alloc_mem(200 * sizeof(int), MPI_INFO_NULL, &B);
if (rank == 0) {
    for (i = 0; i < 200; i++)
        A[i] = B[i] = i;
    MPI_Win_create(NULL, 0, 1, MPI_INFO_NULL, MPI_COMM_WORLD, &win);
    MPI_Win_start(group, 0, win);
    for (i = 0; i < 100; i++)
        MPI_Put(A+i, 1, MPI_INT, 1, i, 1, MPI_INT, win);
    for (i = 0; i < 100; i++)
        MPI_Get(B+i, 1, MPI_INT, 1, 100+i, 1, MPI_INT, win);
    MPI_Win_complete(win);
    for (i = 0; i < 100; i++)
        if (B[i] != (-4)*(i+100)) {
            printf("Get Error: B[%d] is %d, should be %d\n", B[i], (-4)*(i+100));
            fflush(stdout);
            errs++;
        }
}
```

**MPI Critique (Snyder)**

- Message passing is a very simple model
- Extremely low level; heavy weight
  - Expense comes from \( \lambda \) and lots of local code
  - Communication code is often more than half
  - Tough to make adaptable and flexible
  - Tough to get right and know it
  - Tough to make perform in some (Snyder says most) cases
- Programming model of choice for scalability
- Widespread adoption due to portability, although not completely true in practice
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: \( N = \{1,...,n\}, \quad (j,k) \in E \text{ if } A(j,k) \text{ nonzero}, \quad W(j) = \text{#nonzeros in row } j, \quad WE(j,k) = 1 \)
• VLSI Layout
  - \( N = \{\text{units on chip}\}, \quad E = \{\text{wires}\}, \quad WE(j,k) = \text{wire length} \)
• Data mining and clustering
• Analysis of social networks
• Physical Mapping of DNA

Dense Linear Algebra vs. Sparse Linear Algebra

Matrix vector multiply:

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

- **DIA:** Store elements along a set of diagonals.

<table>
<thead>
<tr>
<th>A</th>
<th>1 7 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 8 0</td>
<td></td>
</tr>
<tr>
<td>5 0 3 9</td>
<td></td>
</tr>
<tr>
<td>0 6 0 4</td>
<td></td>
</tr>
</tbody>
</table>

- **CSR:** Store only nonzero elements, with "ptr" to beginning of each row and "indices" representing column.

<table>
<thead>
<tr>
<th>ptr</th>
<th>[0 2 4 7 9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>[0 1 1 2 0 2 3 1 3]</td>
</tr>
<tr>
<td>data</td>
<td>[1 7 2 8 5 3 9 6 4]</td>
</tr>
</tbody>
</table>

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

| data | [1 7 * 2 8 * 5 3 9 6 4 *] |
| indices | [0 1 * 1 2 * 0 2 * 3 1 * 3] |

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

| row | [0 0 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] |

**CSR Example (UPDATE)**

```c
for (j=0; j<nr; j++) {
    for (k = ptr[j]; k<ptr[j+1]-1; k++)
        t[j] = t[j] + data[k] * x[indices[k]];
}
```

**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)

**Basic Definitions: A Quiz**

- **Graph**
  - undirected, directed

- **Path, simple path**

- **Forest**

- **Connected component**
Representing Graphs

- An undirected graph and its adjacency matrix representation.

Common Challenges in Graph Algorithms

- Localizing portions of the computation
  - How to partition the workload so that nearby nodes in the graph are mapped to the same processor?
  - How to partition the workload so that edges that represent significant communication are co-located on the same processor?

- Balancing the load
  - How to give each processor a comparable amount of work?
  - How much knowledge of the graph do we need to do this since complete traversal may not be realistic?

All of these issues must be addressed by a graph partitioning algorithm that maps individual subgraphs to individual processors.

Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
  - $N$ = nodes (or vertices),
  - $W_N$ = node weights
  - $E$ = edges
  - $W_E$ = edge weights

- Ex: $N$ = {tasks}, $W_N$ = {task costs}, edge $(j,k)$ in $E$ means task $j$ sends $W_E(j,k)$ words to task $k$

- Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that
  - The sum of the node weights in each $N_j$ is “about the same”
  - The sum of all edge weights of edges connecting all different pairs $N_j$ and $N_k$ is minimized

- Ex: balance the work load, while minimizing communication

- Special case of $N = N_1 \cup N_2$: Graph Bisection

Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
  - $N$ = nodes (or vertices),
  - $W_N$ = node weights
  - $E$ = edges
  - $W_E$ = edge weights

- Ex: $N$ = {tasks}, $W_N$ = {task costs}, edge $(j,k)$ in $E$ means task $j$ sends $W_E(j,k)$ words to task $k$

- Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that
  - The sum of the node weights in each $N_j$ is “about the same”
  - The sum of all edge weights of edges connecting all different pairs $N_j$ and $N_k$ is minimized (shown in black)

- Ex: balance the work load, while minimizing communication

- Special case of $N = N_1 \cup N_2$: Graph Bisection
Summary of Lecture

• Summary
  - Regular computations are easier to schedule, more amenable to data parallel programming models, easier to program, etc.
  - Performance of irregular computations is heavily dependent on representation of data
  - Choosing this representation may depend on knowledge of the problem, which may only be available at run time

• Next Time
  - Introduction to parallel graph algorithms
  - Minimizing bottlenecks