L15: Design Review, Midterm Review and 6-Function MPI

Administrative

- Design Review April 1
- Midterm April 3, in class
- Organick Lectures this week:
  - Peter Neumann, SRI International
    - Known for his work on Multics in the 1960s
  - "A Personal History of Layered Trustworthiness" Tue Mar 26 @ 7:00 PM, 220 Skaggs Biology
  - "Clean-Slate Formally Motivated Hardware and Software for HighlyTrustworthy Systems" Wed Mar 27 @ 3:20 PM
  - Roundtable, Wed Mar 27 @ 1:30. WEB 1248

Design Reviews

- Goal is to see a solid plan for each project and make sure projects are on track
  - Plan to evolve project so that results guaranteed
  - Show at least one thing is working
  - How work is being divided among team members
- Major suggestions from proposals
  - Project complexity – break it down into smaller chunks with evolutionary strategy
  - Add references – what has been done before? Known algorithm? GPU implementation?

Design Reviews

- Oral, 10-minute Q&A session (April 1 in class, plus office hours if needed)
  - Each team member presents one part
  - Team should identify "lead" to present plan
- Three major parts:
  I. Overview
  - Define computation and high-level mapping to GPU
  II. Project Plan
  - The pieces and who is doing what.
  - What is done so far? (Make sure something is working by the design review)
  III. Related Work
  - Prior sequential or parallel algorithms/implementations
  - Prior GPU implementations (or similar computations)
- Submit slides and written document revising proposal that covers these and cleans up anything missing from proposal.

Design Review

Title	Team
Overview	
Project Plan	
Related Work	
Implementation Status	
Visual interest	
Oral Presentation	

Final Project Presentation

- Dry run on April 22
  - Easels, tape and poster board provided
  - Tape a set of Powerpoint slides to a standard 2'x3' poster, or bring your own poster.
- Poster session during class on April 24
  - Invite your friends, profs who helped you, etc.
- Final Report on Projects due May 1
  - Submit code
  - And written document, roughly 10 pages, based on earlier submission.
  - In addition to original proposal, include
    - Project Plan and How Decomposed (from DR)
    - Description of CUDA implementation
    - Performance Measurement
    - Related Work (from DR)

Let's Talk about Demos

- For some of you, with very visual projects, I encourage you to think about demos for the poster session
- This is not a requirement, just something that would enhance the poster session
- Realistic?
  - I know everyone's laptops are slow …
  - … and don't have enough memory to solve very large problems
- Creative Suggestions?
  - Movies captured from run on larger system

Message Passing and MPI

- Message passing is the principle alternative to shared memory parallel programming, predominant programming model for supercomputers and clusters
  - Portable
  - Low-level, but universal and matches earlier hardware execution model
- What it is
  - A library used within conventional sequential languagess (Fortran, C, C++)
  - Based on Single Program, Multiple Data (SPMD)
  - Isolation of separate address spaces
    - + no data races, but communication errors possible
    - + exposes execution model and forces programmer to think about locality, both good for performance
    - - Complexity and code growth!

Like OpenMP, MPI arose as a standard to replace a large number of proprietary message passing libraries.

Message Passing Library Features

- All communication, synchronization require subroutine calls
  - No shared variables
  - Program runs on a single processor just like any uniprocessor program, except for calls to message passing library
- Subroutines for
  - Communication
    - Pairwise or point-to-point: A message is sent from a specific sending process (point a) to a specific receiving process (point b).
    - Collectives involving multiple processors
      - Move data: Broadcast, Scatter/gather
      - Compute and move: Reduce, AllReduce
  - Synchronization
    - Barrier
    - No locks because there are no shared variables to protect
  - Queries
    - How many processes? Which one am I? Any messages waiting?

MPI References

- The Standard itself:
  - at http://www.mpi-forum.org
  - All MPI official releases, in both postscript and HTML
- Other information on Web:
  - at http://www.mcs.anl.gov/mpi
  - pointers to lots of stuff, including other talks and tutorials, a FAQ, other MPI pages

Slide source: Bill Gropp

Compilation

wrapper script to compile

source file

```
mpicc  -g  -Wall  -o  mpi_hello  mpi_hello.c
```

produce debugging information

create this executable file name (as opposed to default a.out)

turns on all warnings

Execution

```
mpiexec  -n  <number of processes>   <executable>
```

```
mpiexec  -n  1  ./mpi_hello
```

run with 1 process

```
mpiexec  -n  4  ./mpi_hello
```

run with 4 processes

Hello (C)

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf( "Greetings from process %d of
              %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Hello (C++)

```
#include "mpi.h"
#include <iostream>

int main( int argc, char *argv[] )
{
    int rank, size;
    MPI::Init(argc, argv);
    rank = MPI::COMM_WORLD.Get_rank();
    size = MPI::COMM_WORLD.Get_size();
    std::cout << "Greetings from process " << rank << "
        of " << size << "\n";
    MPI::Finalize();
    return 0;
}
```

Execution

```
mpiexec -n 1 ./mpi_hello

    Greetings from process 0 of 1 !
```

```
mpiexec -n 4 ./mpi_hello

    Greetings from process 0 of 4 !
    Greetings from process 1 of 4 !
    Greetings from process 2 of 4 !
    Greetings from process 3 of 4 !
```

MPI Components

- MPI_Init
  - Tells MPI to do all the necessary setup.

```
int MPI_Init(
      int*      argc_p  /* in/out */,
      char***   argv_p  /* in/out */);
```

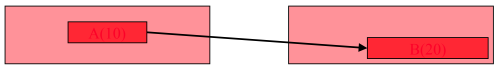
- MPI_Finalize
  - Tells MPI we're done, so clean up anything allocated for this program.

```
int MPI_Finalize(void);
```

Basic Outline

```
. . .
#include <mpi.h>
. . .
int main(int argc, char* argv[]) {
   . . .
   /* No MPI calls before this */
   MPI_Init(&argc, &argv);
   . . .
   MPI_Finalize();
   /* No MPI calls after this */
   . . .
   return 0;
}
```

MPI Basic (Blocking) Send

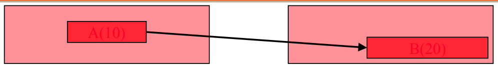


MPI_Send(A, 10, MPI_DOUBLE, 1, …) MPI_Recv(B, 20, MPI_DOUBLE, 0, …)

MPI_SEND(start, count, datatype, dest, tag, comm)

- The message buffer is described by (**start, count, datatype**).
- The target process is specified by **dest**, which is the rank of the target process in the communicator specified by **comm**.
- When this function returns, the data has been delivered to the system and the buffer can be reused. The message may not have been received by the target process.

MPI Basic (Blocking) Receive



MPI_Send(A, 10, MPI_DOUBLE, 1, …) MPI_Recv(B, 20, MPI_DOUBLE, 0, …)

MPI_RECV(start, count, datatype, source, tag, comm, status)

- Waits until a matching (both **source** and **tag**) message is received from the system, and the buffer can be used
- **source** is rank in communicator specified by **comm**, or **MPI_ANY_SOURCE**
- **tag** is a tag to be matched on or **MPI_ANY_TAG**
- receiving fewer than **count** occurrences of **datatype** is OK, but receiving more is an error
- **status** contains further information (e.g. size of message)

MPI Datatypes

- The data in a message to send or receive is described by a triple (address, count, datatype), where
- An MPI datatype is recursively defined as:
  - predefined, corresponding to a data type from the language (e.g., MPI_INT, MPI_DOUBLE)
  - a contiguous array of MPI datatypes
  - a strided block of datatypes
  - an indexed array of blocks of datatypes
  - an arbitrary structure of datatypes
- There are MPI functions to construct custom datatypes, in particular ones for subarrays

A Simple MPI Program

```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[])
{
  int rank, buf;
  MPI_Status status;
  MPI_Init(&argv, &argc);
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );

  /* Process 0 sends and Process 1 receives */
  if (rank == 0) {
    buf = 123456;
    MPI_Send( &buf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  }
  else if (rank == 1) {
    MPI_Recv( &buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
              &status );
    printf( "Received %d\n", buf );
  }

  MPI_Finalize();
  return 0;
}
```

Slide source: Bill Gropp

Six-Function MPI

- Most commonly used constructs
- A decade or more ago, almost all supercomputer programs only used these
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_Size
  - MPI_Comm_Rank
  - MPI_Send
  - MPI_Recv
- Also very useful
  - MPI_Reduce and other *collectives*
- Other features of MPI
  - Task parallel constructs
  - Optimized communication: non-blocking, one-sided

MPI_Reduce

```
int MPI_Reduce(
      void*          input_data_p   /* in   */,
      void*          output_data_p  /* out  */,
      int            count          /* in   */,
      MPI_Datatype   datatype       /* in   */,
      MPI_Op         operator       /* in   */,
      int            dest_process   /* in   */,
      MPI_Comm       comm           /* in   */);
```

```
MPI_Reduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);
```

```
double local_x[N], sum[N];
. . .
MPI_Reduce(local_x, sum, N, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);
```

Questions from Previous Exams

Short answer questions from last year:

- Describe one mechanism we discussed for eliminating shared memory bank conflicts in code that exhibits these bank conflicts. Since the occurrence of bank conflicts depends on the data access patterns, please explain any assumptions you are making about the original code with bank conflicts.
- Give one example of a synchronization mechanism that is *control-based*, meaning it controls thread execution, and one that is *memory-based*, meaning that it protects race conditions on memory locations.
- What happens if two blocks assigned to the same streaming multiprocessor each use more than half of either registers or shared memory? How does this affect scheduling of warps? By comparison, what if the total register and shared memory usage fits within the capacity of these resources?

Questions from Previous Exams

Short answer questions from 2011:

- Describe how you can exploit spatial reuse in optimizing for memory bandwidth on a GPU. (Partial credit: what are the memory bandwidth optimizations we studied?)
- Given examples we have seen of control flow in GPU kernels, describe ONE way to reduce divergent branches for ONE of the following: consider tree-structured reductions, even-odd computations, or boundary conditions.
- What happens if two threads assigned to different blocks write to the same memory location in global memory?

Questions from Previous Exams

Short answer questions from 2009 and 2010:

- Select ONE of the following aspects of the NVIDIA architecture warp execution and describe briefly (in a couple sentences) how it works:
  - selecting a warp to be scheduled for execution;
  - executing a branch operation in a warp;
  - scheduling global memory accesses for a warp;
  - allocating registers to a thread.
- List a specific constraint on either parallelism (threads, blocks, dimensionality of each) or memory capacity (for one specific part of the GPU memory hierarchy), and in one sentence, describe how this impacts your GPU program as compared to a sequential CPU program.

Questions from Previous Exams

- Problem Solving: Data placement in the memory hierarchy

Given the following CUDA code, for each data structure in the thread program, what is the most appropriate portion of the memory hierarchy to place that data (constant, global, shared, or registers) and why. Feel free to give multiple answers for some data in cases where there are multiple possibilities that are all appropriate. In each case, explain how you would modify the CUDA code to use that level of the memory hierarchy.

```
#define N 512
float a[N], b[N], c[N][N], d[N];
int   e[N];
__global compute(float *a, float *b, float *c, float *d, int *e) {
float temp;
int index = blockIdx.x*blockDim.x + threadIdx.x

for (j =0; j<N; j++) {
   temp = (c[index][j] + b[j])*d[e[index]];
   a[index] = a[index] - temp;
 }
}
```

Questions from Previous Exams

- Given the following CUDA code, add synchronization to derive a correct implementation that has no race conditions. (Hint: You should be able to simply insert __synchthreads() calls.)

```
__global__ compute (float *a, float *b, int BLOCKSIZE) {
   __shared__ s_a[128], s_b[128];
   /* copy portion of input data into shared memory */
   s_a[threadIdx.x] = a[blockIdx.x*BLOCKSIZE + threadIdx.x];

   /* Time step loop */
    for (int t = 0; t<MAX_TIME; t++) {
        int boundary = min((blockIdx.x+1)*BLOCKSIZE-1,
                                    blockDim.x*BLOCKSIZE-1,threadIdx.x+2);
        /* alternate inputs and outputs on even/odd time steps */
        if (t % 2 == 0) {
           s_b[threadIdx.x] = s_a[threadIdx.x] + s_a[boundary];
        }
        else /* (t%2 == 1) */ {
           s_a[threadIdx.x] = s_b[threadIdx.x] + s_b[boundary];
       }
   }
/* Result is in s_b, and must be copied to b */
b[blockIdx.x*BLOCKSIZE + threadIdx.x] = s_b[threadIdx.x];
}
```

- Without writing out the CUDA code, consider a CUDA mapping of the LU Decomposition sequential code below. Answer should be in three parts, providing opportunities for partial credit: (i) where are the data dependences in this computation? (ii) how would you partition the computation across threads and blocks? (iii) how would you add synchronization to avoid race conditions?

```
float a[1024][1024];
for (k=0; j<1023; k++) {
  for (i=k+1; i<1024; i++)
     a[i][k] = a[i][k] / a[k][k];
  for (i=k+1; i<1024; i++)
     for (j=k+1; j<1024; j++)
        a[i][j] = a[i][j] – a[i][k]*a[k][j];
}
```

- Given a sparse matrix-vector multiplication, consider two different GPU implementations: (a) one in which the sparse matrix is stored in compressed sparse row format (see below code); and, (b) one which uses an ELL format for the sparse matrix because the upper limit on number of nonzeros per row is fixed. For (a), describe the mapping of this code to GPUs. How is it decomposed into threads, and where are t, data, indices and x placed in the memory hierarchy? For (b), how does the ELL representation affect your solution? Would you consider a different thread decomposition and data placement?

```
// Sequential sparse matrix vector multiplication using compressed sparse row
// CSR) representation of sparse matrix.  "data" holds the nonzeros in the sparse
// matrix


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]];
```

Questions from Previous Exams

Example essay questions:

Describe the features of computations that are likely to obtain high speedup on a GPU as compared to a sequential CPU.

Consider the architecture of the current GPUs and impact on programmability. If you could change one aspect of the architecture to simplify programming, what would it be and why? (You don't have to propose an alternative architecture.)

We talked about sparse matrix computations with respect to linear algebra, graph coloring and program analysis. Describe a sparse matrix representation that is appropriate for a GPU implementation of one of these applications and explain why it is well suited.

Describe three optimizations that were performed for the MRI application case study.