CS194-15 Fall 2016 Midterm 2

Name: _____

Student Id: _____

Person to your left: _____

Person to your right: _____

Q1: _____ (25 points)

Q2: _____ (55 points)

Q3: _____ (20 points)

Read all of the following information before starting the exam:

- Do not write on the back of any page, very close to the edge of a sheet, or unstaple the pages.
- Show all work, clearly and in order, if you want to get full credit.
- Circle or otherwise indicate your final answers.
- Please keep your written answers brief; be clear and to the point. I will take points off for rambling and for incorrect or irrelevant statements.
- This test has 3 problems. Check to make sure that your test booklet has all pages!
- Good luck!

Problem 1: *(25 points)* **Pattern Language**

(a) *(5 points)* Your friend is implementing a text-analysis algorithm in which she constructs an *m×n* matrix A which has one row for every word in the English language, and one column for each of a set of topics (e.g. news, sports, science, etc). That is, there are *m* words and *n* topics. To decide what topics appear in a document D, your friend constructs an m-vector *b* with a 1 for every word that appears in D, and 0 for words that do not appear. To find an *n*-vector *x* representing the topics present in the article, your friend solves the system of equations $A^T Ax = A^T b$ via Gaussian Elimination. For this solution, give the:

Computational Pattern: _____ *Parallel Algorithm Strategy*: _____

Short Justification: (Note the pattern language is given on the last page of the exam)

(b) *(5 points)* Your friend is implementing a fluid-dynamics solver in order to produce a pre-computed special-effect for video games. Her solver will generate *T* timesteps of an *N × N × N* array, where each element represents the pressure in one voxel (a pixel in 3 dimensional space) at a given point in time. *N* and *T* are large enough that, to run efficiently, the computation must be executed across *P* nodes in a distributed-memory cluster. Each node owns N^3/P elements of the array, and they are divided so that each node's chunk of the array is spatially contiguous. The fluid-dynamics solver requires the computation of spatial derivatives, and time-stepped update of the *N × N × N* array. The solver is based on a finite-difference scheme: derivatives are computed by taking differences of contiguous grid elements in a "stencil" pattern.

Computational Pattern: _____ *Parallel Algorithm Strategy*: _____

Short Justification:

(c) *(15 points)* Fill out the table using the given applications below and draw out a little sketch of what the structural pattern looks like from a high level. Every row should be filled with one application example per pattern. Choose the one that fits best. You can use the OPL reference at the back of the exam, for help.

Applications: *Instant Messaging, Complex Nuclear Reactor Simulation, Support Vector Machine, Compiler Optimization, Thermostat system, Ruby on Rails web app, OSI Network Protocol, Computer Game architecture, Compiler, Training Support Vector Machine classifier*

Structural Pattern	Algorithm Example	Parallel Algorithm Strategy	Drawing of structural pattern (dataflow graph)
Pipe-and-filter			
Agent-and-repository			
Process control			
Event-based			
Model-view-controller			
Iterative refinement			
Map-reduce			
Layered-systems			
Puppeteer			
Static task graph			

Problem 2: *(55 points)* **Data Parallel Implementations**

```
int compact(int *in, int *out, int n)
{
  int idx=0;
  for(int i = 0; i < n; i++)
    if(in[i]!=0)
      out[idx++] = in[i];
  return idx;
}
```

Listing 1: Serial Array Compaction

(a) *(15 points)* Implement the serial code shown above using data-parallel primitives. You may create temporary arrays, if needed. Make sure your algorithm modifies the final array and returns the same result as the serial implementation. The following are the data-parallel primitives you may use:

- **A = Map(f, X, Y, Z, …)** For each element of *X, Y, Z,* compute $f(X_i, Y_i, Z_i, \ldots)$ and store the result in A_i. Feel free to use pseudo functions in f.
- **b = Sum(X)** Returns the sum of all values in *X*.
- **C = Scan(X)** Computes the cumulative sum of *X*, such that $C_i = \sum_{j=0}^{i} X_j$.
- **d = Max(X)** Returns the maximum value in X.

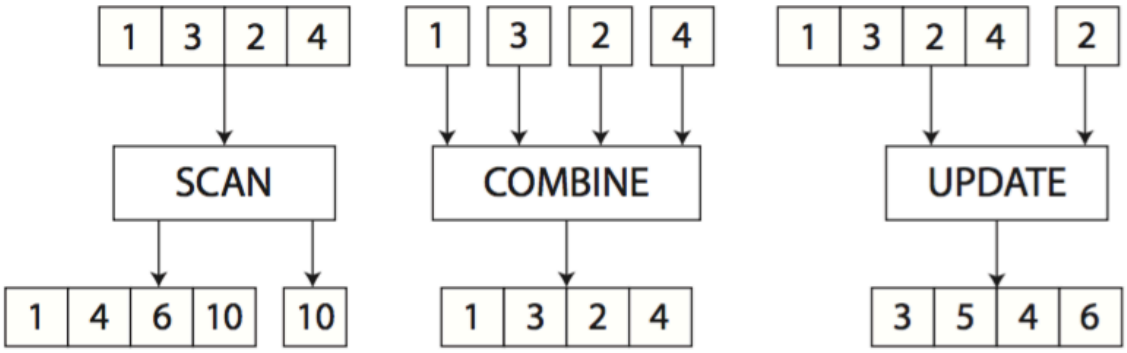


Figure 1: Primitive Hardware Blocks

(b) *(15 points)* You are tasked with creating a hardware implementation of a 16 element data-parallel scan. Luckily for you, your co-workers have completed three primitive blocks that they believe will make your job tolerable.

As shown in Figure 1:

- The SCAN block takes as input a 4-element array and outputs its cumulative sum, and the total.
- The COMBINE block takes as input 4 single numbers and generates an array.
- The UPDATE block takes as input a 4-element array, and a single number x, and outputs a 4-element array where x is added to each element in the input array.

Please show how your circuit (dataflow graph) works with the test input:

A = [1, 3, 4, 2, 5, 2, 3, 1, 1, 3, 2, 4, 5, 2, 2, 1]

```
/** in and out are arrays of size len
/* in contains num_zeroes 0's, this has been counted for you
/* out is pre-filled with 0's
/* e.g. [1, 5, 6, 0, 2, 0] -> [0, 0, 1, 5, 6, 2]
**/
void move_zeroes_to_front(int *in , int *out, int len, int num_zeroes) {
        int idx = num_zeroes;
        for (int i = 0; i < len; i++) {
                if (in[i] != 0) {
                        out[idx] = in[i];
                        idx++;
                }
        }
}
```

(c) *(15 points*) Use data-parallel primitives to move zeroes in an array to the front, while keeping all other elements in original order. The serial implementation is above. Some primitives that you may or may not have to use include the following. Those that return arrays, return an allocated copy instead of modifying input in-place.

- **A = Map(f, X, Y, Z, …)** For each element of *X, Y, Z,* compute $f(X_i, Y_i, Z_i, \ldots)$ and store the result in A_i. Feel free to use pseudo functions in f.
- **b = Reduce(f, X)** Collapses array to 1 value by successively applying $f(X_i, X_{j)}$.
- **C = Scan(X)** Computes the cumulative sum of *X*, such that $C_i = \sum^{i}_{j=0} X_j$.
- **Scatter_nonzeroes(in, out, idx)** *out[idx[i]] = in[i]* for all *i* where *input[i] != 0.*
- **Scatter(in, out, idx)** *out[idx[i]] = in[i]* for all *i.*

(d) *(2 points)* Now we'll run this on the GPU, which requires a call to `clEnqueueNDRangeKernel` with argument `cl_uint work_dim`. What should be passed in for this argument, and why?

(e) *(8 points)* Fill in the OpenCl kernel for `scatter_nonzeroes`:

```
/**
 * output[idx[i]] = input[i], for 0 <= i < n, if input[i] != 0
 * You may assume max(indices) < len(out)
 * No duplicates in indices,
 * aside from those that map to zeroes in the input array.
 * You may also assume that len(in) < n and work_size >= n.
 * all inputs are well-formed.
 */
__kernel scatter_nonzeroes(__global int *in, __global int *out,
                  __global int *indices, const unsigned int n) {









}
```

Problem 3: *(20 points)* **GPU and CPU Questions**

(a) *(3 points) TRUE/FALSE (No justification):* OpenCL is a platform for heterogeneous computing, which means it's a portable language for GPUs, CPUs, and other processors.

(b) *(3 points) TRUE/FALSE (No justification)* In GPU computing, branch divergence is when execution paths of threads in a work group diverge and both code paths execute serially, reducing the efficiency of our kernel.

(c) *(3 points) TRUE/FALSE (No justification)* GPUs have lots of cores running smaller computations at high clock frequencies, whereas CPUs have fewer cores running larger computations at relatively slower clock frequencies.

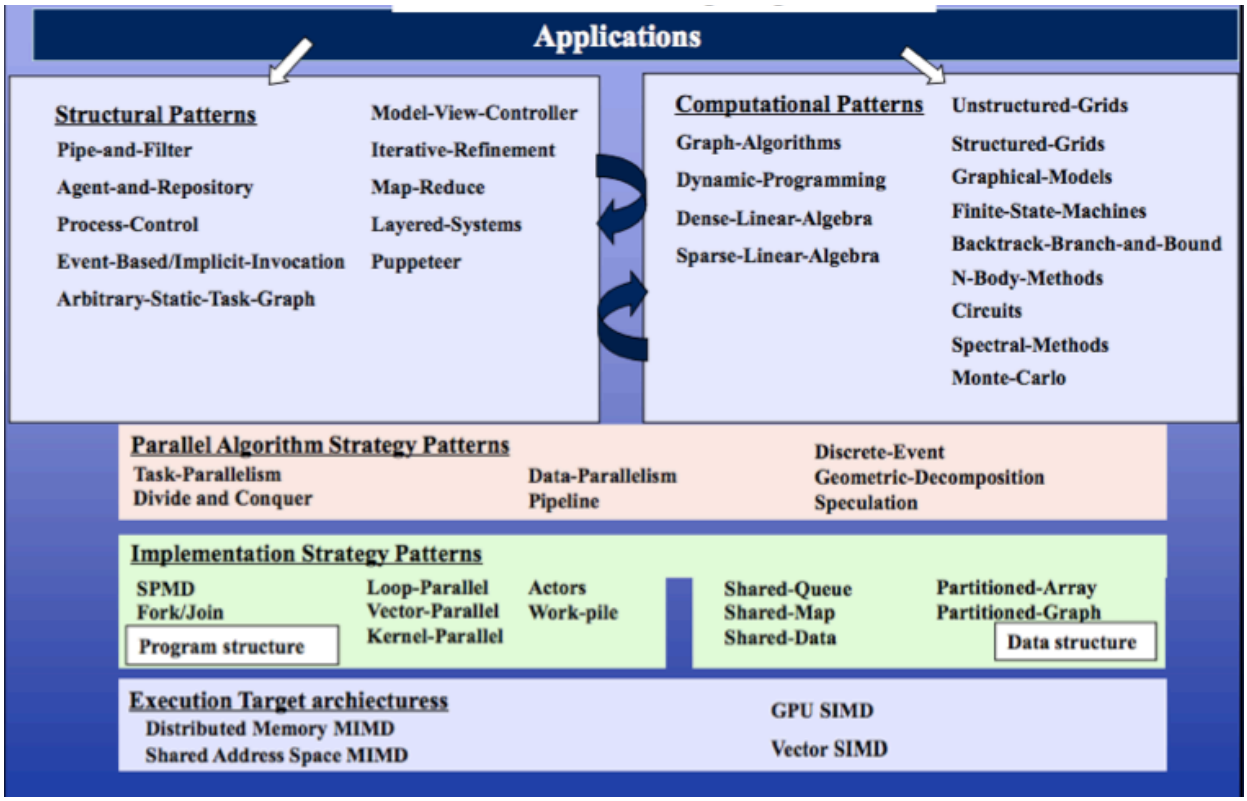
(d) *(4 points) TRUE/FALSE (No justification)* Is this kernel code branch divergent?

```
switch (get_global_id(0) ) {
   4 cases...
}
```

(e) *(4 points) TRUE/FALSE (No justification)* Is this kernel code branch divergent?

```
switch (get_group_id(0) ) {
   4 cases...
}
```

(f) *(3 points)* OpenCL has four memory types: `__private`, `__local`, `__constant`, and `__global`. Which are visible/accessible to the host? (No justification necessary)


Applications
Structural Patterns
Pipe-and-Filter
Agent-and-Repository
Process-Control
Event-Based/Implicit-Invocation
Arbitrary-Static-Task-Graph
Model-View-Controller
Iterative-Refinement
Map-Reduce
Layered-Systems
Puppeteer
Computational Patterns
Graph-Algorithms
Dynamic-Programming
Dense-Linear-Algebra
Sparse-Linear-Algebra
Unstructured-Grids
Structured-Grids
Graphical-Models
Finite-State-Machines
Backtrack-Branch-and-Bound
N-Body-Methods
Circuits
Spectral-Methods
Monte-Carlo
Parallel Algorithm Strategy Patterns
Task-Parallelism
Divide and Conquer
Data-Parallelism
Pipeline
Discrete-Event
Geometric-Decomposition
Speculation
Implementation Strategy Patterns
SPMD
Fork/Join
Program structure
Loop-Parallel
Vector-Parallel
Kernel-Parallel
Actors
Work-pile
Shared-Queue
Shared-Map
Shared-Data
Partitioned-Array
Partitioned-Graph
Data structure
Execution Target archiecturess
Distributed Memory MIMD
Shared Address Space MIMD
GPU SIMD
Vector SIMD

This page left intentionally blank for scratch.