Full Name: ____

Andrew Id: ____

CMU 15-418/618 Practice Exercise 2

Problem 1: Be a Parallel Processor Architect

You are hired to start the parallel processor design team at Yinzer Processors, Inc. Your boss tells you that you are responsible for designing the company's first shared address space multi-core processor, which will be constructed by cramming multiple copies of the company's best selling uniprocessor core on a single chip. He expects the project to yield at least a 5$\times$ speedup on the performance of the program given below. You are not allowed to change the program, and assume that:

- Each Yinzer core can complete one floating point operation per clock
- Cores are clocked at 1 GHz, and each have a 1 MB cache using FIFO replacement.
- All Yinzer processors (both single and multi-core) are attached to a 100 GB/s memory bus
- Memory latency is perfectly hidden (Yinzer processors have excellent pre-fetchers)

```
float A[N]; // let N = 100 million elements
float total = 0;

// ASSUME TIMER STARTS HERE ///////////////////////////////////

for (int i=0; i<N; i++)
   total += A[i];

for (int i=0; i<9; i++) {

   // made up syntax for brevity: 'parallel_for'
   // Assume iterations of this loop are perfectly partitioned
   // using blocked assignment among X pthreads each running on
   // one of the processor's X cores.
   parallel_for(int j=0; j<N; j++) {
      A[j] = A[j] / total;
   }
}

// ASSUME TIMER STOPS HERE ///////////////////////////////////
```

A. (3 pts) How do you respond to your boss' request? Do you believe you can meet the performance goal? If yes, how many cores should be included in the new multi-core processor? If no, explain why.

B. (4 pts) You tell your boss that if he just allowed you to make a few changes to the code, you could deliver a much better speedup with your parallel processor design. How would you change the code to improve its performance by improving *speedup*? (A simple description of the new code is fine). If your answer was NO in part one, how many processors are required to achieve 5× speedup now? If your answer was YES, approximately what speedup do you expect from your previously proposed machine on the new code? (*Note: we are NOT looking for answers that optimize the program by rolling multiple divisions into one.*)

C. (3 pts) Assume that the following year, Yinzer, Inc. decides to produce a 32-core version of your parallel CPU design. In addition to adding cores, your boss gives you the opportunity to further improve the processor through one of the following three options.

- You may double each processor's cache to 2 MB.
- You may increase memory bandwidth by 50%
- You may add a 4-wide SIMD unit to the core so that each core can perform 4 floating point operations per clock.

If each of these options has the same cost, given the code you produced in part B (and what you learned from assignment 1), which option do you recommend to your boss? Why?

Problem 2: Parallel Histogram (10 pts)

A sequential algorithm for generating a histogram from the values in a large input array `input` is given below. For each element of the input array, the code uses the function `bin_func` to compute a "bin" the element belongs to (`bin_func` always returns an integer between 0 and `NUM_BINS-1`) and increments the count of elements in that bin.

```
int bin_func(float value);      // external function declaration
float input[N];                 // assume input is initialized and N is a very large

int histogram_bins[NUM_BINS];  // assume bins are initialized to 0

for (int i=0; i<N; i++) {
   histogram_bins[bin_func(input[i])]++;
}
```

You are given a massively parallel machine with N processors and asked by a colleague to produce an efficient parallel histogram routine. To help you out, your colleague hands you a library with a highly optimized parallel sort routine.

```
void sort(int count, int* input, int* output);
```

The library also has the ability to execute a bulk launch of N independent invocations of an application-provided function using the following CUDA-like syntax:

```
my_function<<<N>>>(arg1, arg2, arg3...);
```

For example the following code (assuming `current_id` is a built-in id for the current function invocation) would output:

```
void foo(int* x) {
   printf("Instance %d : %d\n", current_id, x[current_id]);
}

int A[] = {10,20,30}
foo<<<3>>>(A);

"Instance 0 : 10"
"Instance 1 : 20"
"Instance 2 : 30"
```

(question continued on next page)

A. (10 pts) Using only `sort`, `bin_func` and bulk launch of any function you wish to create, implement a data-parallel version of histogram generation that makes good use of N processors. You may assume that the variable `current_id` is in scope in any function invocation resulting from a bulk launch and provides the number of the current invocation.

```
// External function declarations.  Your solution may or may not use all
// of these functions.
void  sort(int count, int* input, int* output);
int   bin_func(float value);

// input: array of numbers, assume input is initialized
float input[N];

// output: assume all bins are initialized to 0
int   histogram_bins[NUM_BINS];
```

OPTIONAL Problem 3: Changing CUDA's `__syncthreads` Function

The CUDA language provides the `__syncthreads()` primitive, which functions as a barrier for all CUDA threads in *one CUDA thread block*. CUDA does not provide a version of `__syncthreads()` that acts as a barrier across *all CUDA threads in all thread blocks* (across all CUDA threads instantiated by a single kernel launch). Consider the execution of the simple CUDA code below on the GTX 480 GPU we have in the lab (15 cores, 1536 CUDA threads per core, 48 KB of shared memory per core). Discuss the implications of changing the semantics of `__syncthreads()` to be a barrier across *all CUDA threads from the same kernel launch*. To receive credit, an answer should describe how the work defined by a launch is scheduled on a GPU today, and how this change in semantics impacts (and possibly constrains) scheduling decisions. You should also consider effects on state management, such as the overall footprint required to run the computation.

```
__global__ void myKernel() {

   __shared__ int scratch[1024];

  ... do stuff ...
  __syncthreads();
  ... do more stuff ...
}

dim3 blockDim(32, 32);
dim3 gridDim(100, 100)
myKernel<<<gridDim, blockDim>>>();
```