CMU 15-418/618: Parallel Computer Architecture and Programming
Practice Exercise 2

Problem 1: Miscellaneous Practice (5 pts)

A. (2 pts) Imagine you are asked to implement ISPC, and your system must run a program that launches 1000 ISPC tasks. Give one reason why it is very likely more efficient to use a fixed-size pool of worker threads rather than create a pthread per task. Also specify how many pthreads you'd use in your worker pool when running on a quad-core, hyper-threaded Intel processor. Why?

B. (2 pts) CUDA programmers are told variables stored in on-chip shared memory can be accessed with high-bandwidth and low latency (similar to how cached variables can be accessed efficiently on a CPU). Use an *abstraction vs. implementation* argument to describe why Prof. Kayvon says "CUDA shared memory is not a cache!"

C. (1 pt) Assume you want to efficiently run a program with very high **temporal locality**. If you could only choose one, would you add a data cache to your processor or add a significant amount of hardware multi-threading? Why?

Problem 2: Parallel Histogram (8 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 (yes, one per input element) 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)

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 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];
```

Problem 3: Running a CUDA Program on a GPU (12 pts)

The 15-418/618 TAs decide to dip their toes into the GPU design design business and spend their summer creating a new GPU, which, given their poor marketing sense, they call a PKPU (Prof. Kayvon Processing Unit). The processor runs CUDA programs exactly the same manner as the NVIDIA GPUs discussed in class, but it has the following characteristics:

- The processor has eight cores running at 1 GHz.
- Each core provides execution contexts for up to 128 CUDA threads. (Like the GPUs discussed in class, once a CUDA thread is assigned to an execution context, the processor runs the thread to completion before assigning a new CUDA thread to the context.)
- The cores execute threads in an implicit SIMD fashion running 16 consecutively numbered CUDA threads together using the same instruction stream (the PKPU implements 16-wide "warps").
- The cores will fetch/decode one single-precision arithmetic instruction (add, multiply, etc.) per clock. Keep in mind this instruction is executed on an entire warp in that clock.
- All CUDA thread blocks on a single core cannot exceed 16 KB of shared memory storage.

A. (2 pts) When running at peak utilization. What is the processor's **maximum throughput** of single-precision **math operations**? (In your answer, please consider one multiply of two single-precision numbers as one "operation".)

Consider a CUDA kernel launch that executes the following CUDA kernel on the processor. In this program each CUDA thread computes one element of the results array `Y` using 1000 elements from the input array `X` as input. Assume the program is compiled using a thread-block size of 128 threads, and that enough thread blocks are created so there is exactly one thread per output array element.

```
__global__ void foo(float* X, float* Y) {

   // get array index from CUDA block/thread id
   int idx = blockIdx.x * blockDim.x + threadIdx.x;
   int input_idx = 1000 * idx;

   float result = 0.f;

   for (int i=0; i<1000; i++) {                  // 0 cycles (ignore arithmetic here)

      float val = X[input_idx+i];                // memory load (ignore arithmetic here)

      if (((int)val / 1000) % 2 == 0)            // 2 arithmetic cycles
         result += doA(val);                     // 7 arithmetic cycles
      else
         result += doB(val);                     // 7 arithmetic cycles
   }

   Y[idx] = result;                              // memory store

}
```

B. (2 pts) The TAs hook the PKPU up to a memory system that provides **32 GB/sec of bandwidth** and has a **memory request latency of 100 cycles**. They run the code on a 128,000-element input array initialized as `X[] = {1.f, 2.f, 3.f, 4.f, 5.f, 6.f, ...}`. *The output array `Y[]` is only 128 elements*. The TA's observe that the program *does not* realize peak performance on the PKPU. Oguz is devastated. What is the primary reason for the low performance? (Please assume the input and output arrays are resident in GPU memory at the time of the kernel launch. Transfer between host and GPU memory is not relevant in this problem.)

C. (2 pts) Kevin steps in and says, "hey, let me take a look", and runs the same program on an output arrays of size 128×1024 elements and 128×1024×1024 elements. Does he observe significantly different *processing throughput* from the PKPU on the two workloads? Why or why not? (Please assume both the small and large workloads fit comfortably in GPU memory.)

D. (2 pts) Greg looks at the timing of Kevin's experiments on the largest dataset and says "Aw shucks Kevin, this program is not achieving peak utilization of the processor's execution units." What is the problem limiting performance? What percentage of peak PKPU throughput does Greg observe? Remember:

- The processor runs a instruction on an entire warp worth of CUDA threads in a single cycle.
- The memory system provides 32 GB/sec of bandwidth with a request latency of 100 cycles. (When a warp issues a load instruction the data for all threads will return in 100 cycles.)

E. (2 pts) After some intense discussion, the TAs hook a new memory system up to the PKPU that provides **64 GB/sec of bandwidth** (still with a request latency of 100 cycles). Josh also **rewrites the program interleave elements of the input array so that they are stored in the following order:**

```
    0...  999
 2000... 2999
 4000... 4999
     ...
30000...30999
 1000... 1999
 3000... 3999
 5000... 5999
     ...
31000...31999
     ...
32000...32999
34000...34999
     ...
```

Oguz, Greg, Kevin, and Josh congratulate themselves on a job well done and head out to Rose Tea Cafe to celebrate. While at dinner, they get a call from Karima who says, "Why are you out celebrating? I'm here by myself trying to figure out why Josh's program still doesn't get peak performance." What is the problem that is limiting performance, and what percentage of peak is Karima observing?

F. (2 pts) When the rest of the TAs return to the office with their Bubble Teas, Karima is nowhere to be found. They find a note saying "Problem solved! I made a small tweak to the design of the PKPU and now Josh's modified program runs at peak performance now. Heading home to watch Game of Thrones..." What was Karima's tweak? (Note: we are looking for a change to the capabilities of the PKPU core. Leave the memory system is unchanged.)