Lab 3: Maximum Performance

Goals

Predict the maximum performance for your system from the GPU and CPU, measure the best you can achieve with a microbenchmark, and then analyze the actual results compared to your predictions.

Part I: Estimating the impact

1. Read up online about the GPU and CPU in the lab machines so you understand their capabilities, speeds, and design.
    a. Make sure you understand their operating frequency, issue width, SIMD width, number of cores/processors, etc.
2. Predict the maximum performance for the operations listed below for both the CPU and The GPU. Your performance estimates should be in terms of billions per second. (E.g., GFLOPs, GOPS, or GB/s.) This is the “peak” or “theoretical” performance of the hardware.

Predicted CPU Performance Impact

Operation	Max Predicted CPU Performance	Why? (How did you come up with this impact?)
8-bit unsigned add		# cores * frequency * SIMD width * SIMD execution width
Floating point add		
Sequentially load a 32 bit int from an array		
Randomly load a 32 bit int from an array		

Predicted GPU Performance Impact

Operation	Max Predicted GPU Performance	Why? (How did you come up with this impact?)
8-bit unsigned add		
Floating point add		
Sequentially load a 32 bit int from an array		
Randomly load a 32 bit int from an array		
Data transfer from CPU to GPU		

Part 2: Measuring the impact

To measure this impact you will need to write microbenchmarks that isolate these particular performance issues. That is, you want your microbenchmarks to avoid any other bottlenecks than the one they are measuring.

You will have four microbenchmarks for this lab:

1. 8-bit unsigned add: This benchmark will just do as many 8-bit unsigned adds as it can as quickly as possible. You should address (that is, think about, take into account when writing your code, and comment upon in your report) the following:
    a. Loop overhead (unrolling)
    b. Register dependencies
    c. Caching of data
    d. Parallelism
2. Floating point add: This benchmark is strikingly similar to the 8-bit unsigned add benchmark above.
3. Sequentially load an int from an array: This benchmark will load sequential ints (32 bits) from an array as quickly as possible. You should address the following:
    a. Impact of different array sizes (your results should span an appropriately interesting range of array sizes).
    b. Prefetcher/cache-friendly layout
    c. Avoid the compiler eliminating dead code
    d. Loop overhead (unrolling)
    e. Parallelism
    f. Data dependencies
4. Randomly load an int from an array: This benchmark will load randomly chosen ints (32 bits) from an array as quickly as possible. Do not use a random number generator, but instead create the array initially filled with indices so that each load gives you the index into the array of the next load. You should address the following:
    a. Impact of different array sizes (your results should span an appropriately interesting range of array sizes).
    b. Prefetcher/cache-friendly layout
    c. Avoid the compiler eliminating dead code
    d. Loop overhead (unrolling)
    e. Parallelism
    f. Data dependencies
5. Data transfer from CPU to GPU: This benchmark will measure how long it takes to transfer data from the CPU to the GPU. You should address the following:
    a. Impact of different array sizes (your results should span an appropriately interesting range of array sizes).
    b. Ensure that you actually do the transfer and don’t just queue it.

Getting Started

To help you get started we've provided a partial implementation of the 8-bit add microbenchmark for the CPU. Let's walk through it:

1. Download the Microbenchmarks project and make sure it compiles.
2. First version:
    a. Take a look at the first version of the benchmark in the function add_microbenchmark1
        i. The input "scale" just adjusts the number of iterations so faster versions don't go too fast. gettimeofday() measures the time
        ii. Make sure you understand what it is doing and that you believe the output from the printf will be accurate.
        iii. Do you see anything wrong with this benchmark for measuring how fast we can do adds? (We'll fix it in the third version.)
    b. Set the compiler to its lowest level of optimization (-Od) (compile and run in Debug configuration)
        i. Run the benchmark and marvel at how fast the code is.
        ii. Is this what you expected?
    c. Set the compiler to its highest level of optimization (-Ox) (compile and run in Release configuration)
        i. Run the benchmark and marvel at how fast the code is.
        ii. Are all runs the same speed? If not, why not?
        iii. Is this what you expected?
    d. Figure out what is going on with the compiler optimization at its highest level by looking at the generated assembly code.
        i. Put a break point in the function you want to analyze, then right click on it and choose "Go To Disassembly"
        ii. What instructions are in the main loop? Can you find the "add" instruction?
        iii. What optimization did the compiler do that prevented us from measuring what we wanted?
3. Second version:
    a. The second version of the benchmark in the creatively named function add_microbenchmark2 fixes the problem we saw above.
        i. Take a look at this code and figure out why we won't have the problem we saw above with the optimization.
        ii. Run this version and compare the performance with both compiler optimization levels –Od and –Ox.
        iii. Did this fix the problem?
        iv. What is the performance you are getting now, and is it what you expected?
    b. Take a look at the generated assembly code for the second benchmark at optimization level –Ox:
        i. What kind of instructions do you see in the main loop? Can you find the "add" instruction?
        ii. What has the compiler done for you?
4. Third version:
    a. The second version is still doing something very foolish if what we are trying to do is measure how fast we can add. Make sure you identify it before you read further.
    b. Now take a look at add_microbenchmark3. Figure out what the difference in the code is and why this is a better microbenchmark for measuring add performance.
    c. Run the third version with –Ox optimization.
        i. What performance did you get?
        ii. Is this what you expected?

   d. Take a look at the generated assembly code for the third benchmark at optimization level –Ox.
      i. What instructions do you see in the main loop?
      ii. What has the compiler done for you? (Remember the compiler thinks you want the correct result, not a particular set of instructions.)
5. Fourth version:
   a. In the third version the compiler outsmarted us and came up with a faster way to calculate the "result" of our benchmark. (Since it's a benchmark we don't care about the result and just want the computer to do what we say, but the compiler doesn't know that.)
   b. Take a look at the fourth version and see if this will fix the problem.
   c. Run it and see if we get the expected performance.
   d. Why is the performance here so much lower than in the 2nd benchmark?
6. Fifth version:
   a. Take a look at the fifth version and estimate the performance improvement you expect from the 4th to the 5th.
   b. Run the 5th version and see if you get that performance.
   c. Is this the maximum you expect from the machine?
7. Sixth version:
   a. Compare the code between the 5th and 6th versions. What is the difference?
   b. Do you expect this to make a difference in performance? (Think about the processor's SIMD issue width and the effects of unrolling.)
   c. Run the results and see if it did.
8. Seventh version:
   a. This code is only using one core. Fix it and see what performance you can really get.

You can use this benchmark as a template for the other CPU benchmarks. You can't quite do this level of detail with the GPU benchmarks unless you use CUDA and insert PTX (Nvidia assembly) manually. Don't bother with this for the GPU benchmarks, but do keep the issues you saw above in mind when writing the GPU benchmarks!

You should answer the following questions in your report for the add microbenchmark walkthrough:

1. Why were all benchmark runs not the same speed in 1 and 2?
2. What was the problem with 1 and 2 that we fixed in 3?
3. What optimization did the compiler do for 1 that prevented us from measuring what we wanted?
4. What did the code change in 2 do to prevent the compiler from doing this optimization?
5. What optimization did the compiler do for 3 that prevented us from measuring what we wanted?
6. Why was the performance of 4 so much slower than 2?
7. What was the optimization in 6 and why does it improve performance?

Please make sure you've completed part 1 before you do this part.

Actual Performance Impact

Operation	Predicted Max CPU Performance	Predicted Max GPU Performance	Max Measured CPU Performance	Max Measured GPU Performance
8-bit unsigned add				
Floating point add				
Sequentially load a 32 bit int from an array				
Randomly load a 32 bit int from an array				
Data transfer from CPU to GPU				

Analysis

You have now tried to predict the maximum possible performance you can get from your CPU and GPU and written microbenchmarks to measure it. The point of this lab is to compare the peak theoretical performance to what you can actually achieve with a rather worthless benchmark. (None of these microbenchmarks do anything useful; all they tell you is the best you can actually get on real hardware.)

Note: The most important thing is to show that you understand what is going on. If your estimates were far off the actual performance then you should analyze the real results and use the profiler to figure out what is really happening. (As well as talk to the TA.) Your report should explain what is really going on and what you missed when you did your first estimates. You will not lose any points for having plausible, but incorrect, initial estimates if you plausibly identify what was wrong about them from the actual implementation. (Indeed, that's the whole point!)

You should produce a concise report with the following 4 sections:

- **1. Performance Estimates**: Your estimates of the peak CPU and GPU performance and how you came up with them.
  - **Answers to the questions for the add microbenchmark**
- **2. Performance Achieved**:
  - Your measured performance results.
- **3. Discussion**:
  - A discussion of the measured results and how they differ from your predictions.
  - A discussion of what you learned about the hardware from measuring its performance.
  - A discussion of how you wrote your benchmarks and any issues you encountered and how you solved them.
  - Discuss how confident you are that your microbenchmarks accurately reflect the best obtainable performance.
  - Comment on any unexpected or odd results.
  - Comments on how this compares to your results from Labs 1 and 2 for the color conversion.
- **4. Lab comments**: Any feedback on the lab itself.

The comparative results should be presented in a graphical form to make it easy to see trends and analysis.