General Submission Guidelines

Files. C or C++ code files, and a PDF report file should be submitted on Gradescope.

Submission Code Structure. Make a different sub-directory for each question with the name question#. Do not create dependencies between two question sub-directories, instead copy the code that you want to re-use. Every question sub-directory should be independently buildable. A simple flat structure is preferred. For example,

```
hw1
|--- question1
     +--- main.cpp
+--- question2
     |--- main.cpp
     +--- worker.cpp
```

Development and Testing. You can develop and test your code on your local machines with gcc. However, before submitting your homework, make sure to execute all of your programs on mc18.cs.purdue.edu.

Grading. mc18 machine will be used for testing your code with GNU compiler. Output your results to stdout using `printf()` or `cout`. If your code needs a different build process other than the one we will use as default, you will have to include a Makefile in the question sub-directory with the default target being .

Extensions. No late submissions will be accepted, unless students have prior extensions from the instructors.

Homework Specific Guidelines

Build Command. By default, we will compile all of your C/ C++ files in each question sub-directory with the command `g++ *.cpp -o ./run`. All of your source files in the question's sub-directory will be built into an executable that outputs to `stdout`. If you do not want to include a faulty source file for grading, remove it from the sub-directory; otherwise it will fail the default build command.

If you do not want to us to use the default build command, include a Makefile in each question's sub-directory. However, it is recommended to use the simplest flat-file structure for your code as shown in the general submission guidelines and thus focus on experiments and results, not software engineering. You will still need to create sub-directories for questions, but the make command is yours now.

Execution. If your program uses command line arguments or some sort of input, you need to explain how to use your program in your report PDF. It is preferable if you avoid command line arguments, or automate the inputs to these arguments in a Makefile target or shell script included with your submission (e.g. you can include a shell script that runs the program with different command line arguments).

Output. We will check your code's output numbers and the numbers mentioned in your report PDF for mc18. Make sure your code compiles, runs, and does not throw errors like segfaults on the test environment.

Clocks. Use `gettimeofday()` for your timing routine.

Question 1. *16.66%.* Estimate the bandwidth of your computer by accessing a large array, for example, using a code of the form:

```
for (i = 0; i < length; i++)
   sum+ = a[i];
```

Vary length from a small number (e.g. 1000) to a larger number (say 10000000). Record the time and fit a line to this data. The slope is the bandwidth.

Report the memory bandwidth of your local machine and mc18 machine.

Question 2. *16.67%.* Estimate the cache size(s) for your computer using strided access using a loop of the following form:

```
for (i = 0; i  < skip * length; i += skip)
    sum += a[i];
```

Make sure the array a used in the loop is initialized outside the timed loop and is of the right size.

In this experiment, for each length (e.g., 10000000), vary skip to be 1, 2, 3, 4, 5, 6, 7, 8, ... Note the time for each of the executions. You should see increases at different values of skips. Use these to reason about the cache line size of the machine. Note that your actual plot may not have clear jumps because of mixed cache line sizes (primary, secondary, tertiary).

Report the cache line size of your local machine and mc18 machine.

Question 3. *25%.* Estimate the impact of loop unrolling. Write an array sum loop (refer question 1), and unroll the loop. You will need to do this carefully, since you don't want to create data dependencies in the unrolling process.

Increase the unrolling from 1, 2, 3, 4, 5, 6. Time each of these unrolled loops and reason about your observations for your local machine and mc18 machine.

Question 4. *16.67%.* Assess the impact of loop interchanging. Write a two-loop matrix-vector multiplication program (multiplying a dense matrix with a vector) of the form:

```
for (i = 0; i < n; i++)
    result[i] = 0;

for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        result[i] += a[i][j]*b[j];
```

Interchange the nested loop, repeat the experiment for values of n = 1000, 10000, and 100000.

Report on the observed runtimes and reason about your observations for your local machine and mc18 machine.

Question 5. *25%.* Assess the impact of data reuse in dense matrix multiplication. Start from a three-loop matrix multiply. Increase matrix sizes as 100, 500, 1000, 5000. Plot the computation rate (not the runtime) as a function of matrix sizes.

The goal is to maximize this computation rate. To achieve this, write general matrix multiplication in terms of multiplications of smaller matrices (whose sizes are tuned to maximize computation rate). Repeat experiment with matrix sizes 100, 500, 1000, and 5000 with this improved (tiled) matrix multiplication.

Plot both charts for your local machine and mc18 machine, include them in your report PDF and reason about your observations.