CS 420, Parallel Programming, Fall 2019
MP1

Szaday, Justin Shah, Ishan Pankaj

Due Date: September 13th, 2019, 11:59PM CDT

1 Overview

The primary objective of this assignment is to introduce you to the Illinois Campus Cluster, Intel® C/C++ Compiler (ICC), and Intel® VTune™ Amplifier (VTune). It is important to ensure you are comfortable with these tools since we will be using these tools throughout the semester. To this end, you will be asked to implement matrix-matrix multiplication in C, optimize your implementation using loop tiling, and evaluate its performance in VTune.

2 Getting Started

2.1 Logging into the Campus Cluster

The campus cluster can be accessed via SSH using your NetID and its password using the following command:

```
ssh <NetID>@cc-login.campuscluster.illinois.edu
```

For more information, the cluster's user documentation can be found at:
https://campuscluster.illinois.edu/resources/docs/user-guide/

2.1.1 Data Transfer

There are two ways to exchange files between the campus cluster and another machine, Globus Online's GridFTP or SSH-based tools, e.g. SCP or SFTP. We recommend consulting the *Data Transfer* section of the cluster's user documentation for more details:
https://campuscluster.illinois.edu/resources/docs/user-guide/#dt

2.2 Loading the ICC Module

The campus cluster provides users with many common development tools via the *Modules* package, which allows users to dynamically load and unload modules representing each tool and its dependencies. The Intel® C/C++ Compiler (ICC) is available as one of these modules, and can be loaded using:

```
module load intel/.19.0
```

We recommend adding this command to your `$HOME/.bashrc` file; otherwise, you will have to load the ICC module every time you log in.

2.3 Git

You will use GitHub@Illinois to submit your MPs throughout the course; to start, use the repository creator to create your repo:
https://edu.cs.illinois.edu/create-ghe-repo/cs420-fa19/

Next, set up your authorship defaults (replacing the placeholders with your information):

```
git config --global user.name "<FirstName> <LastName>"
git config --global user.email "<IllinoisEmail>"
```

Then, checkout your repository as follows:

```
git clone https://github-dev.cs.illinois.edu/cs420-fa19/<NetId>.git
```

Your repository will be empty to start; to grab the latest version of our assignment and its skeleton code, add the `_release` repository as an extra remote:

```
git remote add release https://github-dev.cs.illinois.edu/cs420-fa19/_release.git
git pull release master
git push origin master
```

3 Tasks

3.1 Part A, Matrix Multiplication

3.1.1 Naive Matrix Multiplication

Your assignment is to implement the standard (naive) matrix multiplication in the following function of the skeleton code we have provided:

```
void matrixMultiply(matrix m1, matrix m2, matrix* result)
```

To get started, refer to `matrix.c` and `matrix.h` inside the `mp1` folder. We have implemented the input and output for you, as explained inside `matrix.h`.

3.1.2 Using Make

We have provided a `Makefile` for you to use to build your program with various options. For example, to build your program without compiler optimizations, use:

```
make matrix_nop
```

Note, the resulting executable will be named `matrix-nop`. To clean your build, use the following command:

```
make clean
```

3.1.3 Running Your Program

Since we have limited compute resources available on the login nodes of the campus cluster, we will need to use a *job script* to run our program with larger matrix sizes on the cluster's compute nodes. A job script is a Bash script with a header provides scheduling directions for the cluster's job scheduler, qsub. A simple job script might look like:

```
#!/bin/bash
#PBS -l walltime=00:20:00
#PBS -l nodes=1:ppn=12
#PBS -N <JobName>
#PBS -q cs

cd $PBS_O_WORKDIR
<RunCommand>
```

Where *JobName* might be replaced with "matmul" and *RunCommand* might be replaced with:

```
./randMatrix 100 100 100 100 | ./matrix-nop
```

This would generate two $100x100$ matrices containing random numbers as input to the non-optimized version of our program. If you named your job script `batch_script.run`, you would submit it to the batch system using:

```
qsub batch_script.run
```

A job identification string will be printed to the screen as confirmation that the job was successfully submitted to the batch system, for example:

```
<JobID>.cc-mgmt1.campuscluster.illinois.edu
```

To check the status of your job, run the following command:

```
qstat <JobID>
```

When your job is complete, its output will be found in a text file in the current working directory (at the time of submission). For more details about running jobs, check the *Running Jobs* section of the campus cluster user guide: https://campuscluster.illinois.edu/resources/docs/user-guide/#jobs

3.1.4 Measuring MFLOPs

The rate of floating point operations per second (FLOPs) is one of the most popular performance metrics in scientific computing. Now that you have implemented and tested your program, we will measure this rate to begin analyzing it. Your assignment is to implement the code to measure MFLOPs a la Listing 1.1 in the textbook, then call the `printAnalytics(long ops, double mFlops)` function with the result. We have provided a program called `randMatrix` to generate huge matrices for this part of the assignment, where its command-line arguments are for the sizes of the matrices to generate. For example, to generate two $100x100$ arrays using `randMatrix` and pipe the results into the matmul program, using the `-oa` flag to output only the analytics, run the following command:

```
./randMatrix 100 100 100 100 | ./matrix-nop -oa
```

Run your program with various matrix sizes and observe how the performance changes. Your program's performance may vary even when the matrix size remains constant. To account for this, you might consider running your program several times and computing the average performance.

What happens if we change interchange the loops, would it impact performance? Try it out and explain the results.

3.1.5 Compiler Optimization

Next, we will build our program with a full suite of compiler optimization flags and measure how much it improves performance:

```
make matrix
./randMatrix 100 100 100 100 | ./matrix -oa
```

Compile the program with these optimizations for all subsequent runs.

What are two compiler optimizations that could have improved our program's performance? List and briefly explain them in your report.

3.1.6 Tiled Matrix Multiplication

It is possible to partition a matrix into sub-matrices, called blocks, and perform matrix-matrix multiplication on these blocks to improve performance; this process is, interchangeably, called blocking or tiling. For example, if you have the input matrices of sizes $4x4$ and a blocks size of 2, they will be treated as blocks of 2x2 and then multiplied:

```
a1  a2  a3  a4          b1  b2  b3  b4
a5  a6  a7  a8    X     b5  b6  b7  b8
a9  a10 a11 a12         b9  b10 b11 b12
a13 a14 a15 a16         b13 b14 b15 b16
~
A1 A2   X    B1 B2
A3 A4        B3 B4

where A1 =  a1 a2 and  B1 = b1 b2 and ...
            a5 a6           b5 b6

=   (A1*B1 + A2*B3) (A1*B2 + A2*B4)
    ...
```

Your assignment is to implement the tiled/blocked version of the matrix multiplication inside the following function:

```
void matrixTiledMultiply(matrix m1, matrix m2, matrix *result, int tileSize)
```

To avoid incurring unnecessary additional overhead, do not store the blocks in separate arrays. To run the tiled version with NxN tiles, where N is an integer, provide the `-t N` flag to the compiled binary as follows:

```
./matrix -t N
```

Try running the tiled version with a few different tile sizes and compare it to the naive version, keeping the matrix sizes constant. Note, you might not notice a difference in the performance between the two versions if your input matrices are too small. Use the best tile size you find for subsequent parts of this assignment.

Why is the tiled version of the program faster? For which matrix size did you notice a difference?

How does changing the tile size affect performance? Which tile size was the best?

3.2 Part B, Introduction to VTune

3.2.1 Overview

We will be exploring many of VTune's features throughout this course, using the data we collect to gain insights into our programs' performance. To start, we will be collecting data about matmul's memory accesses, particularly the number of misses in the Last Level Cache (LLC).

3.2.2 Collecting Data with VTune

We will use VTune's command-line interface (CLI) to collect data about our programs. While the graphical interface is available on the campus cluster, please limit your use of it because there are a limited number of licenses available. Moreover, it is more convenient to use the CLI in your job scripts. To do so, copy your job script header from Part A, and add the following lines (where `RunCommand` is the executable you wish to run and any options for it, such as `-t N`):

```
module load intel/.19.0
amplxe-cl -collect memory-access <RunCommand>
```

The first line ensures that the Intel module is loaded, since it is not loaded by default. The second invokes the CLI of VTune, called `amplxe-cl`, and instructs it to run your program and collect information about its memory accesses.

3.2.3 Generating a Report

When the collection is complete, VTune saves the data it has collected as an analysis result in the default (something in the current working directory that starts with `r`) or the specified (via the `-result-dir` flag) result directory. These analysis results need to be formatted as a report to be human-readable, this can be done via the following command:

```
amplxe-cl -report summary -format csv -report-output report.csv \
    -r <Results>
```

This instructs VTune to output a summary of the results in the file `report.csv`.

3.2.4 Reading the Report

The generated CSV report file can be opened in any text editor or spreadsheet program (e.g. Excel). It will contain a series of metrics and their associated values (e.g. LLC Miss Count), as well as some information about the machine the reported data was collected on. We will explore more details about these reports in a later assignment.

3.2.5 Your Task

Use VTune to collect memory access data for both the original and tiled versions of your matrix-matrix multiplication program. Generate reports for the results, find the LLC Miss Count in the reports, and record these quantities in your submission. Use large enough matrix sizes such that there is a measurable difference.

3.3 Part C, Putting It All Together

Plot *MFLOPs vs. Matrix Size* for the three variants that we have studied:

1. Naive implementation without any compiler optimization
2. Naive implementation with compiler optimization
3. Tiled implementation with compiler optimization

You may use any plotting tool (e.g. MATLAB or Excel) to generate this plot. Include the plot in your report. Run each variant for matrix sizes: 100, 300, 500, 1000, and 2000.

You might consider writing a job script that can run multiple matrix sizes in one job. We can do this by writing a `for` loop in our job script; for example, to run the program with the required matrix sizes:

```
allSizes=(100 300 500 1000 2000)
for n in ${allSizes[@]}; do
  echo "running with n=$n"
  ./randMatrix $n $n $n $n | ./matrix -oa
done
```

4 Submission Guidelines

4.1 Expectations For This Assignment

The graded portions of this assignment are your matrix-matrix multiplication implementation and a PDF report named `m1-report.pdf`, containing the answers to the questions throughout the report, the LLC Miss Counts from Part B, and the plot from Part C.

4.1.1 Points Breakdown

- 4×5 pts. — Short-Answer Questions
- 20 pts. — Matrix-Matrix Multiplication Implementation
- 30 pts. — Tiled Matrix-Matrix Multiplication Implementation

- 5 pts. — FLOPs Calculation Implementation
- 10 pts. — LLC Counts
- 15 pts. — *MFLOPs vs. N (Matrix Size)* Plot(s)

4.2 Pushing Your Submission

Follow the git commands to commit and push your changes to the remote repo. Ensure that your files are visible on the GitHub web interface, your work will not be graded otherwise. Only the most recent commit before the deadline will be graded.