PORTLAND STATE UNIVERSITY
CS201: COMPUTER SYSTEMS PROGRAMMING (SECTION 5)
FALL 2016

HOMEWORK 4
**DUE: June 1st, 3:00 PM**

## 1. Goals and Overview

1) Develop practical knowledge of the factors that impact program performance
2) Use loop unrolling optimization techniques to optimize the iterative Matrix-Matrix Multiplication Algorithm
3) Analyze program execution measurements to further guide optimization decisions

## 2. Development Setup

For this programming assignment you must work individually. You will work in the CS Linux Lab (linuxlab.cs.pdx.edu) located in FAB 88-09 and 88-10. If you don't already have an account, go to http://www.cat.pdx.edu/students.html for instructions.

For this Homework you must use the C language and Make. Please refrain from using other languages or any language extensions or 3rd party libraries. We will use the GNU C Compiler (gcc) and Make already installed in the development machines in the lab. To obtain correct analysis results y**ou must compile your code with GCC and the following switches: -O0 -ansi -pedantic -Wall -fgcse**

Grading will be done in this setup so please make sure that your code works under these conditions.

## 3. Introduction

In this Homework we will optimize the iterative Matrix-Matric Multiplication algorithm for square matrices. The iterative Matrix-Matrix Multiplication algorithm is based on the definition of Matrix-Matrix multiplication. The algorithm is as follows:

```
Require A, B matrices of size n x n
  For i= 0 to n-1 do
    For j = 0 to n-1 do
      R[i, j] = 0
      For k =0 to n-1 do
        R[i, j] = R[i, j] + A[i, k] x B[k, j]
      End For
    End For
  End For
Ensure R= A x B, matrix of size n x n
```

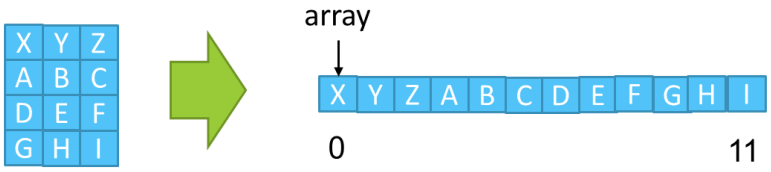
## 4. Problem Description

You are provided with a program that **takes a filename as command line parameter**. Then it opens the specified file and reads 2 matrices from this file. Finally it calls function ByteMult() to compute the Matrix-Matrix multiplication of 2 square matrices and prints out the resulting matrix and the execution time elapsed. Your job is to implement 2 optimized versions of function ByteMult(). You will implement these 2 versions by modify the provided implementation of ByteMult(). Your code must link to the provided main() function in hw4.c in a similar way as we did in Homework 3.

You have also been provided with 2 test cases: matrices1.txt and matrices2.txt but you should implement additional test cases to further test your code. Each test file specifies the size of the matrices in the first line, the first matrix to multiply on the second line and the second matrix to multiply on the third line.

### 4.1 Matrix Representation

Each element in the matrix will be represented as an unsigned integer. We will use the contiguous buffer method for multidimensional arrays described in Lecture 2, Slide 7. The provided implementation computes the corresponding offset within the buffer to access a specific row and column in the 2-Dimentsional Matrix. The contiguous representation is specifically designed to work in conjunction with the loop unrolling optimization that we will implement in the rest of the homework.




### 4.2 Loop Unrolling

As mentioned in class, branches can severely impact performance in pipelined architectures. Loop unrolling is a technique commonly used by optimizing compilers to improve cache access performance and reduce the impact of branches.

As part of this homework, you must create various separate versions of the code, in a similar fashion to Homework 3, each with different loop unrolling depths.

We will unroll **the most inner loop of the multiplication algorithm. You must unroll the loop 4 and 8 times and submit the resulting code as mult4.c and mult8.c**

### 4.3 Performance Analysis

You will measure the execution time of each of the versions of the code with the provided test case (matrices1.txt). Be sure to replace the **FAST_TRIALS** variable in hw4.c with the much longer value in **TRIALS** to ensure that your code is executed a large number of times. If you do not change the number of iterations in the execution loop the computation will be too fast and your collected data will not provide the correct picture.

You must provide a line graph in which the X axis you plot the number of times the code is unrolled (1, 4 and 8) and in the Y axis you plot the elapsed execution time for matrices1.txt. If you use Excel the

correct graph type to use is "**XY Scatter Plot**". Make sure you label the axis of your graph and include the time units for the Y axis in your label! Your graph should be submitted as a JPEG, BMP, PNG or PDF document.

Also you must analyze the data and answer the following questions:

1) What version of the code provided the best performance? (non-optimized, unrolled 4 times, or unrolled 8 times)
2) Did loop unrolling provided any benefit?
3) Why do you think that loop unrolling was or was not beneficial in this case?

### 4.4 Makefile

As part of your solution you should augment the provided Makefile so that it automatically compiles all three variants of the code: Executable non-optimized using the provided code (hw4), Executable with Loop Unrolled 4 times (hw44) and Executable with Loop Unrolled 8 times (hw48). Your Makefile must also provide a cleanup entry (make clean) to delete all the files generated by the compilation process (i.e. object files, executables, libraries, etc.)

As in Homework 3, your 3 versions of the code will link to resulting object (hw4.o) of compiling the unmodified provided hw4.c file.

## 5. Hand-In

For submission, you should provide only source code (*.c, *.h) and a Makefile script as outlined in Section 4.3. More specifically you must provide the original unchanged hw4.c file, the non-optimized multiplication algorithm in mult.c, multiplication algorithm with the loop unrolled 4 times in mult4.c, multiplication algorithm with the loop unrolled 8 times in mul8.c, and the Makefile.

Please pack your files into a TAR file with the following filename structure CS201_name_HW4.tar.

We will use the D2L system for submission (https://d2l.pdx.edu/). Please remember that there is no late policy.

## 6. Rubric

Grading will be done according to the following Rubric with partial credit given for features partially implemented.

| Feature | Possible Points |
|---|---|
| Code with most inner loop unrolled 4 times | 35 |
| Code with most inner loop unrolled 8 times | 35 |
| Makefile that compiles all 3 executables (hw4, hw48 and hw44). | 10 |
| Line graph showing the execution time of hw4, hw44 and hw48 | 10 |
| Code does not have any segmentation faults and follows good software engineering principles | 10 |
| ***Total*** | ***100*** |