Assignment 1

David Warren
University of Alabama
dlwarren2@crimson.ua.edu
CWID: 11703777

I. Introduction

This report reflects the observations and analyses of *Performance Optimization via Register Reuse* of three separate matrix multiplication algorithms: *dgemm0, dgemm1,* and *dgemm2*. To analyze the performance of each algorithm, two square input matrices of 64-bit double floating point random numbers with dimensions of n = 64, 128, 256, 512, 1024, and 2048 were used in each algorithm to generate a product matrix. Each algorithm was run ten times with each n-dimensions to gather an average run time for each dimension (number of runs can be altered upon execution). The difference of all matrix elements for each dimension was calculated for *dgemm0, dgemm1,* and *dgemm2* from their product matrices and compared to verify the correctness of each algorithm's implementations. All three output matrices of each algorithm are also checked with the *diff* Linux command-line utility to detect differences in the files via shell script.

II. Ease of Use

A. CPU and Compiler Information

CPU used was Intel Xeon Gold 4180 CPUs on normal nodes to gather performance metrics. Compiler used was gcc (GCC) 7.3.0. Optimization flag was set to default -O0 least optimized. Reduces compilation time and make debugging produce the expected results.

B. Using the Makefile

- "Make" to compile *dgemm0, dgemm1, dgemm2, dgemm3,* and other matrix files.
- "Make clean" to remove binary and executable files
- "Make submit" to submit dgemm-0.job, dgemm-1.job, and dgemm-2.job to get average execution time over ten runs.
- "Make compareFiles" to compare all output matrices of each dimension.
- "Make compareDifferences JOB0= JOB1= JOB2=" to compare maximum differences between all six dimensions across *dgemm0, dgemm1,* and *dgemm2*. Must provide slurm job numbers as parameters for JOB0, JOB1, and JOB2 for their respective algorithms.

III. Part I Observations

Initial analysis of dgemm0 and dgemm1 is completed given theoretical floating point operations/cycle, penalty cycles for accessing operands in memory, and clock frequency. The calculated performance of dgemm0 and dgemm1 is computed using averaged computation time over x runs and total floating point operations. Performance is measured in GFLOPS (10^9 floating point operations per second).

A. Dgemm0 and dgemm1 Initial Analysis

The equation used to determine computation time will be referred to as equation (1). The symbol ψ represents total number of cycles and the symbol β represents clock frequency. The symbol δ refers to total execution time.

$$\psi/\beta = \delta \tag{1}$$

The equation for total number of cycles will be referred to as equation (2). The symbol γ represents total floating point operations. The symbol ε refers to floating point operations per cycle. The symbol ψ refers to total number of cycles.

$$\gamma\varepsilon = \psi \tag{2}$$

1) dgemm0

Clock frequency is given as 2GHz. The dimension given to us is n = 1000. Analyzing the triple loop of *dgemm0* we see there are two floating point operations per iteration on line 75 of *dgemm0.c*: c[i*n+j] += a[i*n+k] * b[k*n+j]. Therefore, total floating point operations in a triple loop of dimension n is $2n^3$. Operations per cycle ratio is 1 cycle / 4 FLOPS + x(100 cycles / 1 FLOPS) where x is number of operands outside of a register per operation. In line 75, multiplication of a[]*b[] both access operands outside of registers, and c+= a[]*b[] access the operand c outside of a register since a[] and b[] have already been loaded from memory into a register. In total we have 3 accesses to operands outside of a register over 2 FLOPS. To simplify, since each operation access two operands outside of registers the delay would be 300 cycles / 2 FLOPS (or 600 cycles / 4 FLOPS) when x is 2. Thus, our total cycles / FLOPS ratio is 601 cycles / 4 FLOPS. Using equation (2), we get total number of cycles to be $\psi = 2(1000^3)$ FLOPS * 601 cycles / 4 FLOPS = $3.005*10^{11}$ cycles when $n = 1000$. Using equation

(1) when clock frequency is given as 2GHz, $\delta = 3.005*10^{11}$ / 2GHz = 150.25s which is our final execution time of *dgemm0* given the current specifications.

2) dgemm1

Analyzing the triple loop of *dgemm1* we see there are two floating point operations per iteration on line 76 of *dgemm1.c*: r += a[i*n+k] * b[k*n+j]. Specifications are the same as *dgemm0* as well as dimension. This time however, there are only 2 accesses to operands not in registers. The computation a[]*b[] each access an operand not in a register. The computation r += a[]*b[] accesses r which has already been loaded into a register on line 73: register double r = c[i*n+j]. The next operand a[] and b[] have already been loaded into a register. Using equation (2) as above, $\psi = 2(1000^3)$ FLOPS * 401 cycles / 4 FLOPS = $2.005*10^{11}$ cycles when $n = 1000$. Using equation (1) $\delta = 2.005*10^{11}$ / 2GHz = 100.25s which is our final execution time of *dgemm1* given the current specifications.

3) Time Wasted Accessing Operands Outside Registers

To calculate time wasted, we must find operation time of algorithms accessing all operands from inside registers. We use the same process as the previous two sections. Using equation (2) as above, $\psi = 2(1000^3)$ FLOPS * 1 cycles / 4 FLOPS = $5.0*10^8$ cycles when $n = 1000$. Using equation (1) $\delta = 5.0*10^8$ / 2GHz =0.25s which is our final execution time of both *dgemm0* and *dgemm1* given the current specifications, as both have the same amount of operations and iterations. Thus, wasted time for *dgemm0* is 150s and wasted time for *dgemm1* is 100s.

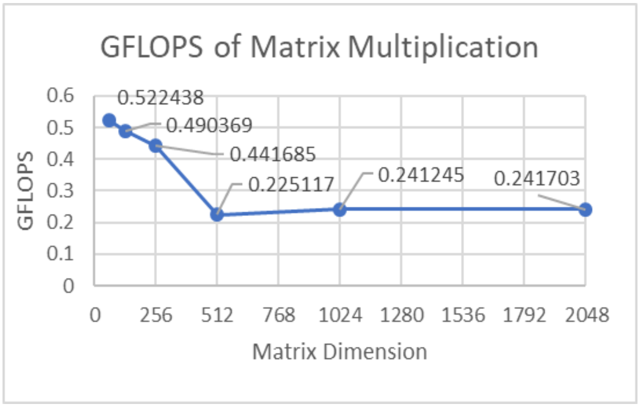
B. *dgemm0 and dgemm1 Calculated Performance*

To calculate actual performance of both algorithms, average execution time was gathered over ten runs of each dimension. This was calculate using the following equation which we will name equation (3). The symbol *a* represents number of operations per iteration which as we know from previous sections is 2 for *dgemm0* and *dgemm1*. The symbol λ represents number of iterations which we know from previous sections is n^3. The symbol δ is average execution time and π is final performance metric GFLOPS.

$$\alpha(10^{-9})\lambda / \delta = \pi \qquad (3)$$

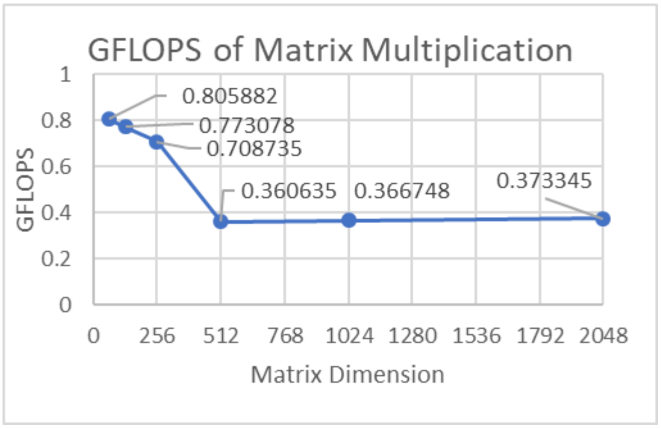
Final performance metric of *dgemm0* is portrayed below in **Figure (1).**

Fig. (1)



Final performance method of *dgemm1* is portrayed below in **Figure (2).**

Fig. (2)



These findings are consistent with initial observations of *dgemm0* and *dgemm1*. Initial analysis gauged *dgemm1* as being 50% more efficient. Calculated performance over ten runs showed *dgemm1* to be ~54.5% more efficient.

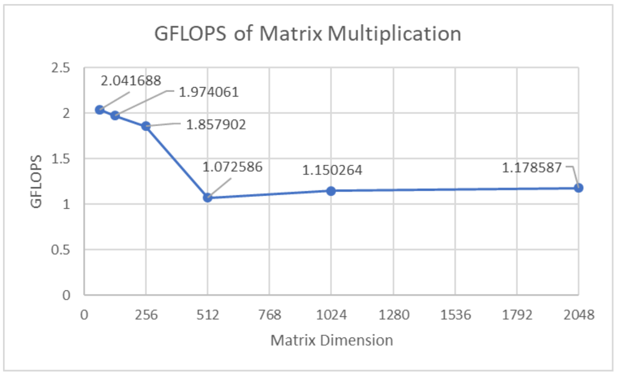
IV. Part II Observations

.

A. *dgemm2 Calculated Performance*

The final performance metric of *dgemm2* is calculated by using equation (3) as before. Final performance metric of *dgemm2* is portrayed below in **Figure (3)**.

Fig. (3)



CPU and compiler information can be found in *II. Ease of Use* section *A) CPU and Compiler Information*.

V: Part III

I was only able to get correct results from *dgemm3* for the *n* = 64 dimension matrices. *Dgemm2* has a running time of 0.000297s for n = 64 and dgemm3 has a running time of 0.000209s for n = 64. I do not have enough data to reliably calculate performance metrics. The code and outputs, however, have been included in the report. Outputs, like the others, can be found in the tar file.

VI: Source Code And Results

Source code is attached after the report. Input and output matrices are available in the tar file along with the raw source code itself in proj1/testFiles and proj1/output/console respectively. You may also find the error logs on proj1/output/error. Sample output from each is included as well in the /proj1/output/console folder.

VII: Appendix

Figure (1): Final performance metric of *dgemm0* is portrayed below.

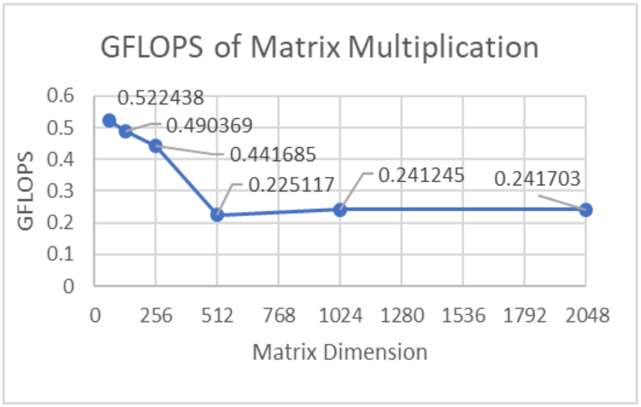


Figure (2): Final performance method of *dgemm1* is portrayed below.

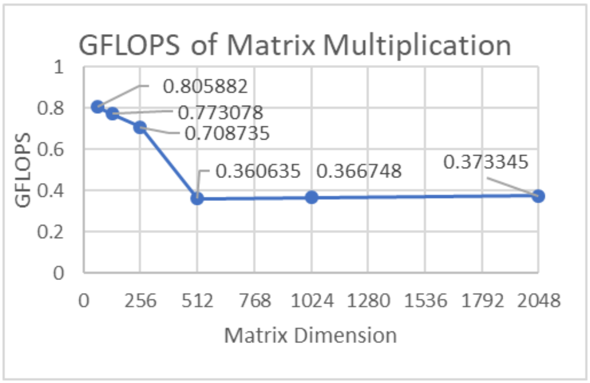
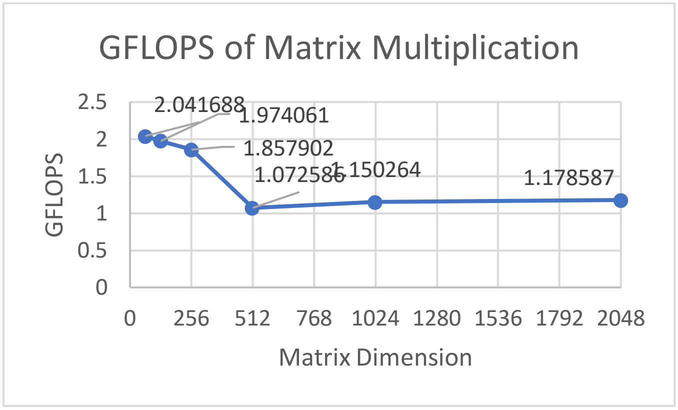


Figure (3): Final performance metric of *dgemm2* is portrayed below.



Assignment 1

dgemm0.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include "matrix.h"

#define BILLION 1000000000L

double dgemm0(int n, MATRIX *x, MATRIX *y, MATRIX *z);

struct timespec start, end;

int main()
{
    int n[6] = {64, 128, 256, 512, 1024, 2048};

    char * inputA[6] = {"./testFiles/inputA-64.dat", "./testFiles/inputA-128.dat", "./testFiles/inputA-2
                            "./testFiles/inputA-512.dat", "./testFiles/inputA-1024.dat", "./testFiles/in

    char * inputB[6] = {"./testFiles/inputB-64.dat", "./testFiles/inputB-128.dat", "./testFiles/inputB-2
                            "./testFiles/inputB-512.dat", "./testFiles/inputB-1024.dat", "./testFiles/in

    char * dgemm0OutputC[6] = {"./output/dgemm0/outputC-64.dat", "./output/dgemm0/outputC-128.dat", "./o
                            "./output/dgemm0/outputC-512.dat", "./output/dgemm0/outputC-1024.dat", "./ou
    int i, j;
    for (i=0; i<6; i++)
    {
        double dgemm0TimeAllRuns = 0.0;
        double numOps = pow(n[i], 3);
        for (j=0; j<1; j++)
        {
            printf("RUN %d: \n", j);
            MATRIX *x = newMATRIXFromFile(inputA[i], n[i]);
            MATRIX *y = newMATRIXFromFile(inputB[i], n[i]);
            MATRIX *z = newEmptyMATRIX(n[i]);

            dgemm0TimeAllRuns += dgemm0(n[i], x, y, z);

            if (j==0) // we only need to write once. wastes time to overwrite the file many times
            {
                outputExistingMATRIXToFile(z, dgemm0OutputC[i]);
                printf("Max difference is %f\n", findMaxDifference(x, y));
            }

            freeMATRIX(x);
            freeMATRIX(y);
            freeMATRIX(z);
        }

        printf("\n");
        double dgemm0GFLOPS = 2.0 * 1.0e-9 * numOps / (dgemm0TimeAllRuns / j);
        printf("numOps: %f\n", numOps);
        printf("TOTAL TIME FOR dgemm0: %f\n", dgemm0TimeAllRuns);
        printf("AVERAGE GLOPS OF dgemm0 (RUNS: %d, n: %d) = %f\n", j, n[i], dgemm0GFLOPS);
        printf("\n\n");
    }

    return 0;
}

/*dgemm0: simple ijk version triple loop algorithm*/
double dgemm0(int n, MATRIX *x, MATRIX *y, MATRIX *z)
{
    double *a = x->data;
    double *b = y->data;
    double *c = z->data;
    int i,j,k;
    double time_spent = 0.0;
    clock_gettime(CLOCK_MONOTONIC, &start);
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            for (k=0; k<n; k++)
            {
                c[i*n+j] += a[i*n+k] * b[k*n+j];
            }
        }
    }
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_spent = ((end.tv_sec - start.tv_sec)*BILLION + end.tv_nsec - start.tv_nsec) * 1e-9;
    printf("dgemm0 (dim=%d):: Total time in seconds is: %f\n", n, time_spent);
    return time_spent;
}
```

PDF document made with CodePrint using Prism

Assignment 1

dgemm2.c

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include "matrix.h"

#define BILLION 1000000000L

double dgemm2(int n, MATRIX *x, MATRIX *y, MATRIX *z);

struct timespec start, end;

int main()
{
    int n[6] = {64, 128, 256, 512, 1024, 2048};

    char * inputA[6] = {"./testFiles/inputA-64.dat", "./testFiles/inputA-128.dat", "./testFiles/inputA-2
                            "./testFiles/inputA-512.dat", "./testFiles/inputA-1024.dat", "./testFiles/in

    char * inputB[6] = {"./testFiles/inputB-64.dat", "./testFiles/inputB-128.dat", "./testFiles/inputB-2
                            "./testFiles/inputB-512.dat", "./testFiles/inputB-1024.dat", "./testFiles/in

    char * dgemm2OutputC[6] = {"./output/dgemm2/outputC-64.dat", "./output/dgemm2/outputC-128.dat", "./o
                            "./output/dgemm2/outputC-512.dat", "./output/dgemm2/outputC-1024.dat", "./ou
    int i, j;
    for (i=0; i<6; i++)
    {
        double dgemm2TimeAllRuns = 0.0;
        double numOps = pow(n[i], 3);
        for (j=0; j<10; j++)
        {
            printf("RUN %d: \n", j);
            MATRIX *x = newMATRIXFromFile(inputA[i], n[i]);
            MATRIX *y = newMATRIXFromFile(inputB[i], n[i]);
            MATRIX *z = newEmptyMATRIX(n[i]);

            dgemm2TimeAllRuns += dgemm2(n[i], x, y, z);

            if (j==0) // we only need to write once. wastes time to overwrite the file many times
            {
                outputExistingMATRIXToFile(z, dgemm2OutputC[i]);
                printf("Max difference is %f\n", findMaxDifference(x, y));
            }

            freeMATRIX(x);
            freeMATRIX(y);
            freeMATRIX(z);
        }
        printf("\n");
        double dgemm2GFLOPS = 2 * 1.0e-9 *numOps / (dgemm2TimeAllRuns / j);
        printf("numOps: %f\n", numOps);
        printf("TOTAL TIME FOR dgemm2: %f\n", dgemm2TimeAllRuns);
        printf("AVERAGE GLOPS OF dgemm2 (RUNS: %d, n: %d) = %f\n", j, n[i], dgemm2GFLOPS);
        printf("\n\n");
    }

    return 0;
}

/* Multiply n x n matrices x and y  */
double dgemm2(int n, MATRIX *x, MATRIX *y, MATRIX *z)
{
    double *a = x->data;
    double *b = y->data;
    double *c2 = z->data;
    double time_spent = 0.0;
    clock_gettime(CLOCK_MONOTONIC, &start);
    int i, j, k;
    for (i = 0; i < n; i+=2)
    {
        for (j = 0; j < n; j+=2)
        {
            register int t = i*n+j; register int tt = t+n;
            register double c00 = c2[t]; register double c01 = c2[t+1];  register double c10 = c2[tt]; r

            for (k = 0; k < n; k+=2)
            {
                register int ta = i*n+k; register int tta = ta+n; register int tb = k*n+j; register int
                //only need 4 registers if we switch out after multiplying quadrants
                register double a00 = a[ta]; register double a10 = a[tta]; register double b00 = b[tb];


                c00 += a00*b00 ; c01 += a00*b01 ; c10 += a10*b00 ; c11 += a10*b01 ;
                a00 = a[ta+1]; a10 = a[tta+1]; b00 = b[ttb]; b01 = b[ttb+1]; // switch out a/b 00 and a1
                c00 += a00*b00 ; c01 += a00*b01 ; c10 += a10*b00 ; c11 += a10*b01 ;
            }

            c2[t] = c00;
            c2[t+1] = c01;
            c2[tt] = c10;
            c2[tt+1] = c11;
        }
    }
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_spent = ((end.tv_sec - start.tv_sec)*BILLION + end.tv_nsec - start.tv_nsec) * 1e-9;
    printf("dgemm2 (dim=%d) :: Total time in seconds is: %f\n", n, time_spent);
    return time_spent;
}
```

PDF document made with CodePrint using Prism

Assignment 1

dgemm2.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include "matrix.h"

#define BILLION 1000000000L

double dgemm2(int n, MATRIX *x, MATRIX *y, MATRIX *z);

struct timespec start, end;

int main()
{
    int n[6] = {64, 128, 256, 512, 1024, 2048};

    char * inputA[6] = {"./testFiles/inputA-64.dat", "./testFiles/inputA-128.dat", "./testFiles/inputA-2
                            "./testFiles/inputA-512.dat", "./testFiles/inputA-1024.dat", "./testFiles/in

    char * inputB[6] = {"./testFiles/inputB-64.dat", "./testFiles/inputB-128.dat", "./testFiles/inputB-2
                            "./testFiles/inputB-512.dat", "./testFiles/inputB-1024.dat", "./testFiles/in

    char * dgemm2OutputC[6] = {"./output/dgemm2/outputC-64.dat", "./output/dgemm2/outputC-128.dat", "./o
                            "./output/dgemm2/outputC-512.dat", "./output/dgemm2/outputC-1024.dat", "./ou
    int i, j;
    for (i=0; i<6; i++)
    {
        double dgemm2TimeAllRuns = 0.0;
        double numOps = pow(n[i], 3);
        for (j=0; j<1; j++)
        {
            printf("RUN %d: \n", j);
            MATRIX *x = newMATRIXFromFile(inputA[i], n[i]);
            MATRIX *y = newMATRIXFromFile(inputB[i], n[i]);
            MATRIX *z = newEmptyMATRIX(n[i]);

            dgemm2TimeAllRuns += dgemm2(n[i], x, y, z);

            if (j==0) // we only need to write once. wastes time to overwrite the file many times
            {
                outputExistingMATRIXToFile(z, dgemm2OutputC[i]);
                printf("Max difference is %f\n", findMaxDifference(x, y));
            }

            freeMATRIX(x);
            freeMATRIX(y);
            freeMATRIX(z);
        }
        printf("\n");
        double dgemm2GFLOPS = 16.0 * 1.0e-9 *numOps / (dgemm2TimeAllRuns / j);
        printf("numOps: %f\n", numOps);
        printf("TOTAL TIME FOR dgemm2: %f\n", dgemm2TimeAllRuns);
        printf("AVERAGE GLOPS OF dgemm2 (RUNS: %d, n: %d) = %f\n", j, n[i], dgemm2GFLOPS);
        printf("\n\n");
    }

    return 0;
}

/* Multiply n x n matrices x and y  */
double dgemm2(int n, MATRIX *x, MATRIX *y, MATRIX *z)
{
    double *a = x->data;
    double *b = y->data;
    double *c2 = z->data;
    double time_spent = 0.0;
    clock_gettime(CLOCK_MONOTONIC, &start);
    int i, j, k;
    for (i = 0; i < n; i+=2)
    {
        for (j = 0; j < n; j+=2)
        {
            register int t = i*n+j; register int tt = t+n;
            register double c00 = c2[t]; register double c01 = c2[t+1];  register double c10 = c2[tt]; r

            for (k = 0; k < n; k+=2)
            {
                register int ta = i*n+k; register int tta = ta+n; register int tb = k*n+j; register int
                //only need 4 registers if we switch out after multiplying quadrants
                register double a00 = a[ta]; register double a10 = a[tta]; register double b00 = b[tb];


                c00 += a00*b00 ; c01 += a00*b01 ; c10 += a10*b00 ; c11 += a10*b01 ;
                a00 = a[ta+1]; a10 = a[tta+1]; b00 = b[ttb]; b01 = b[ttb+1]; // switch out a/b 00 and a1
                c00 += a00*b00 ; c01 += a00*b01 ; c10 += a10*b00 ; c11 += a10*b01 ;
            }

            c2[t] = c00;
            c2[t+1] = c01;
            c2[tt] = c10;
            c2[tt+1] = c11;
        }
    }
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_spent = ((end.tv_sec - start.tv_sec)*BILLION + end.tv_nsec - start.tv_nsec) * 1e-9;
    printf("dgemm2 (dim=%d) :: Total time in seconds is: %f\n", n, time_spent);
    return time_spent;
}
```

PDF document made with CodePrint using Prism

Assignment 1

dgemm3.c

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include "matrix.h"

#define BILLION 1000000000L

double dgemm3(int n, MATRIX *x, MATRIX *y, MATRIX *z);

struct timespec start, end;


//WARNING: THIS HAS BUGS. I WAS NOT ABLE TO FULLY COMPLETE dgemm3. n=64 matrix is the only one that work
int main()
{
    int n[6] = {64, 128, 256, 512, 1024, 2048};

    char * inputA[6] = {"./testFiles/inputA-64.dat", "./testFiles/inputA-128.dat", "./testFiles/inputA-2
                            "./testFiles/inputA-512.dat", "./testFiles/inputA-1024.dat", "./testFiles/in

    char * inputB[6] = {"./testFiles/inputB-64.dat", "./testFiles/inputB-128.dat", "./testFiles/inputB-2
                            "./testFiles/inputB-512.dat", "./testFiles/inputB-1024.dat", "./testFiles/in

    char * dgemm3OutputC[6] = {"./output/dgemm3/outputC-64.dat", "./output/dgemm3/outputC-128.dat", "./o
                            "./output/dgemm3/outputC-512.dat", "./output/dgemm3/outputC-1024.dat", "./ou
    int i, j;
    for (i=0; i<1; i++)
    {
        double dgemm3TimeAllRuns = 0.0;
        double numOps = pow(n[i], 3);
        for (j=0; j<1; j++)
        {
            printf("RUN %d: \n", j);
            MATRIX *x = newMATRIXFromFile(inputA[i], n[i]);
            MATRIX *y = newMATRIXFromFile(inputB[i], n[i]);
            MATRIX *z = newEmptyMATRIX(n[i]);

            dgemm3TimeAllRuns += dgemm3(n[i], x, y, z);

            if (j==0) // we only need to write once. wastes time to overwrite the file many times
            {
                outputExistingMATRIXToFile(z, dgemm3OutputC[i]);
            }

            //freeMATRIX(x);
            //freeMATRIX(y);
            //freeMATRIX(z);
        }
        printf("\n");
        //double dgemm3GFLOPS = 36.0 * 1.0e-9 *numOps / (dgemm3TimeAllRuns / j);
        //printf("numOps: %f\n", numOps);
        //printf("TOTAL TIME FOR dgemm2: %f\n", dgemm3TimeAllRuns);
        //printf("AVERAGE GLOPS OF dgemm2 (RUNS: %d, n: %d) = %f\n", j, n[i], dgemm3GFLOPS);
        //printf("\n\n");

    }

    return 0;
}

/* Multiply n x n matrices x and y  */
double dgemm3(int n, MATRIX *x, MATRIX *y, MATRIX *z)
{
    double *a = x->data;
    double *b = y->data;
    double *c2 = z->data;
    double time_spent = 0.0;
    clock_gettime(CLOCK_MONOTONIC, &start);
    int i, j, k;
    for (i = 0; i < n; i+=3)
    {
        for (j = 0; j < n; j+=3)
        {
            register int t = i*n+j; register int tt = t+n; register int ttt = t+n+n;
            register double c00 = c2[t]; register double c01 = c2[t+1];  register double c02 = c2[t+2];
            register double c10 = c2[tt]; register double c11 = c2[tt+1]; register double c12 = c2[tt+2]
            register double c20 = c2[ttt]; register double c21 = c2[ttt+1]; register double c22 = c2[ttt

            for (k = 0; k < n; k+=3)
            {
                register int ta = i*n+k; register int tta = ta+n; register int ttta = ta+n+n;
                register int tb = k*n+j; register int ttb = tb+n;
                register double a00 = a[ta]; register double a10 = a[tta]; register double a20 = a[ttta]
                register double b00 = b[tb]; register double b01 = b[tb+1]; register double b02 = b[tb+2

                c00 += a00*b00 ; c01 += a00*b01 ; c02 += a00*b02;
                c10 += a10*b00 ; c11 += a10*b01 ; c12 += a10*b02;
                c20 += a20*b00 ; c21 += a20*b01 ; c22 += a20*b02;

                a00 = a[ta+1]; a10 = a[tta+1]; a20 = a[ttta+2];
                b00 = b[ttb];  b01 = b[ttb+1]; b02 = b[ttb+2];

                c00 += a00*b00 ; c01 += a00*b01 ; c02 += a00*b02;
                c10 += a10*b00 ; c11 += a10*b01 ; c12 += a10*b02;
                c20 += a20*b00 ; c21 += a20*b01 ; c22 += a20*b02;

            }

            c2[t] = c00;
            c2[t+1] = c01;
            c2[t+2] = c02;
            c2[tt] = c10;
            c2[tt+1] = c11;
            c2[tt+2] = c12;
            c2[ttt] = c20;
            c2[ttt+1] = c21;
            c2[ttt+2] = c22;
        }
    }
    clock_gettime(CLOCK_MONOTONIC, &end);
    time_spent = ((end.tv_sec - start.tv_sec)*BILLION + end.tv_nsec - start.tv_nsec) * 1e-9;
    printf("dgemm3 (dim=%d) :: Total time in seconds is: %f\n", n, time_spent);
    return time_spent;
}

```

PDF document made with CodePrint using Prism

Assignment 1

matrix.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <time.h>

typedef struct matrix
{
  int dim;
  double *data;
} MATRIX;

MATRIX* newEmptyMATRIX(int dim)
{
    MATRIX *m = malloc(sizeof(MATRIX));
    assert(sizeof(m) > 0);
    m->dim = dim;
    m->data = (double*)malloc(sizeof(double) * m->dim * m->dim);
    assert(sizeof(m->data) > 0);  //make sure we don't have empty data
    return m;
}

MATRIX* newMATRIXFromFile(char *file, int dim)
{
  MATRIX *m = malloc(sizeof(MATRIX));
  FILE *fp = fopen(file, "r");
  assert(fp != NULL);
  m->dim = dim;
  //allocate matrix
  m->data = (double*)calloc(sizeof(double), m->dim * m->dim);
  assert(sizeof(m->data) > 0);  //make sure we don't have empty data

  //populate data array
  int i;
  for (i=0; i< dim * dim; i++)
  {
    fscanf(fp, "%lf", &m->data[i]);
  }
  fclose(fp);
  return m;
}

void generateNewMATRIXFile(char *file, int dim)
{
    srand(time(NULL));
    FILE *fp = fopen(file, "w");
    assert(fp != NULL);
    int i;
    for (i = 0; i< dim * dim; i++)
    {
        fprintf(fp, "%f ", (double)rand());
    }
    fclose(fp);
}

void populateMATRIX(MATRIX *m)
{
    srand(time(NULL));
    int i;
    for (i=0; i<m->dim * m->dim; i++)
    {
        m->data[i] = (double)rand();
    }

}
void outputExistingMATRIXToFile(MATRIX *m, char *file)
{
    FILE *fp = fopen(file, "w");
    assert(fp != NULL);
    int i;
    for (i=0; i<m->dim * m->dim; i++)
    {
        fprintf(fp, "%f ", m->data[i]);
    }
    fclose(fp);
}

double findMaxDifference(MATRIX *a, MATRIX *b)
{
  double maxDifference = 0.0;
  double temp = 0.0;
  int i;
  for (i=0; i<a->dim * a->dim; i++)
  {
    temp = fabs(a->data[i] - b->data[i]);
    if (temp > maxDifference)
    {
      maxDifference = temp;
    }
  }
  return maxDifference;
}

void freeMATRIX(MATRIX *m)
{
  free(m->data);
  free(m);
}

```

PDF document made with CodePrint using Prism

Assignment 1

matrix.h

```c
#ifndef matrix_h
#define matrix_h

typedef struct matrix
{
  int dim;
  double *data;
} MATRIX;

MATRIX* newEmptyMATRIX(int dim);
MATRIX* newMATRIXFromFile (char *file, int dim);
void generateNewMATRIXFile(char *file, int dim);
void populateMATRIX(MATRIX *m);
void outputExistingMATRIXToFile(MATRIX *m, char *file);
double findMaxDifference(MATRIX *a, MATRIX *b);
void freeMATRIX(MATRIX *m);

#endif
```

PDF document made with CodePrint using Prism

Assignment 1

makefile

```
# Makefile package and compute
# --- macros
OOPTS= -Wall -Wextra
CFLAGS= -lm -O0
OBJ= matrix.o
JOB0= 0
JOB1= 0
JOB2= 0

# --- targets
all: dgemm0 dgemm1 dgemm2 dgemm3

dgemm0: $(OBJ) dgemm0.o
    gcc $(OOPTS) $(OBJ) dgemm0.o -o dgemm0 $(CFLAGS)

dgemm1: $(OBJ) dgemm1.o
    gcc $(OOPTS) $(OBJ) dgemm1.o -o dgemm1 $(CFLAGS)

dgemm2: $(OBJ) dgemm2.o
    gcc $(OOPTS) $(OBJ) dgemm2.o -o dgemm2 $(CFLAGS)

dgemm3: $(OBJ) dgemm3.o
    gcc $(OOPTS) $(OBJ) dgemm3.o -o dgemm3 $(CFLAGS)

dgemm0.o: dgemm0.c matrix.h
    gcc $(OOPTS) -c dgemm0.c $(CFLAGS)

dgemm1.0: dgemm1.c matrix.h
    gcc $(OOPTS) -c dgemm1.c $(CFLAGS)

dgemm2.o: dgemm2.c matrix.h
    gcc $(OOPTS) -c dgemm2.c $(CFLAGS)

dgemm3.o: dgemm3.c matrix.h
    gcc $(OOPTS) -c dgemm3.c $(CFLAGS)

matrix.o: matrix.c matrix.h
    gcc $(OOPTS) -c matrix.c $(CFLAGS)

# --- remove binary and executable files
clean:
    rm -f $(OBJ) dgemm0.o dgemm1.o dgemm2.o dgemm3.o dgemm0 dgemm1 dgemm2 dgemm3

# --- submits dgemm-0.job dgemm-1.job dgemm-2.job. Each dimension has 10 iterations to average times.
submit:
    sbatch dgemm-0.job
    sbatch dgemm-1.job
    sbatch dgemm-2.job
    sbatch dgemm-3.job

# --- compare max differences of matrices. Must specify
compareDifferences:
```

```
compareDifferences:
    grep "Max difference is" ./output/console/dgemm0/dgemm0.$(JOB0).out > temp1
    grep "Max difference is" ./output/console/dgemm1/dgemm1.$(JOB1).out > temp2
    grep "Max difference is" ./output/console/dgemm2/dgemm2.$(JOB2).out > temp3
    diff temp1 temp2
    diff temp2 temp3
    rm -f temp1 temp2 temp3
# --- compare all output files.
compareFiles:
    diff ./output/dgemm0/outputC-64.dat ./output/dgemm1/outputC-64.dat
    diff ./output/dgemm0/outputC-128.dat ./output/dgemm1/outputC-128.dat
    diff ./output/dgemm0/outputC-256.dat ./output/dgemm1/outputC-256.dat
    diff ./output/dgemm0/outputC-512.dat ./output/dgemm1/outputC-512.dat
    diff ./output/dgemm0/outputC-1024.dat ./output/dgemm1/outputC-1024.dat
    diff ./output/dgemm0/outputC-2048.dat ./output/dgemm1/outputC-2048.dat
    diff ./output/dgemm2/outputC-64.dat ./output/dgemm1/outputC-64.dat
    diff ./output/dgemm2/outputC-128.dat ./output/dgemm1/outputC-128.dat
    diff ./output/dgemm2/outputC-256.dat ./output/dgemm1/outputC-256.dat
    diff ./output/dgemm2/outputC-512.dat ./output/dgemm1/outputC-512.dat
    diff ./output/dgemm2/outputC-1024.dat ./output/dgemm1/outputC-1024.dat
    diff ./output/dgemm2/outputC-2048.dat ./output/dgemm1/outputC-2048.dat
```

PDF document made with CodePrint using Prism

Assignment 1

dgemm-0.job

```
#!/bin/bash -l

#SBATCH -q defq
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 00:45:00
#SBATCH -J dgemm0
#SBATCH -o ./output/console/dgemm0/dgemm0.%j.out
#SBATCH -e ./output/error/dgemm0/dgemm0.%j.error

./dgemm0
```

PDF document made with CodePrint using Prism

Assignment 1

dgemm-1.job

```
#!/bin/bash -l

#SBATCH -q defq
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 00:45:00
#SBATCH -J dgemm1
#SBATCH -o ./output/console/dgemm1/dgemm1.%j.out
#SBATCH -e ./output/error/dgemm1/dgemm1.%j.error

./dgemm1
```

PDF document made with CodePrint using Prism

Assignment 1

dgemm-2.job

```
#!/bin/bash -l

#SBATCH -q defq
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 00:45:00
#SBATCH -J dgemm2
#SBATCH -o ./output/console/dgemm2/dgemm2.%j.out
#SBATCH -e ./output/error/dgemm2/dgemm2.%j.error

./dgemm2
```

PDF document made with CodePrint using Prism

Assignment 1

dgemm-3.job

```bash
#!/bin/bash -l

#SBATCH -q defq
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 00:15:00
#SBATCH -J dgemm3
#SBATCH -o ./output/console/dgemm3/dgemm3.%j.out
#SBATCH -e ./output/error/dgemm3/dgemm3.%j.error

./dgemm3
```

PDF document made with CodePrint using Prism