High Performance Computing, Computing Project 1, Spring 2019
Performance Optimization via Register Reuse

Jonathan Frame
University of Alabama

February 6, 2019

Below are my answers to parts #s 1, 2 and 3 of the first homework assignment for high performance computing. I compiled my c codes using gcc and the o flag, and I ran this code on the Pantarhei head node to get the results presented below.

1 Parts #1 & #2

Assume your computer is able to complete 4 double floating-point operations per cycle when operands are in registers and it takes an additional delay of 100 cycles to access any operands that are not in registers. The clock frequency of your computer is 2 Ghz. How long it will take for your computer to finish the following algorithm dgemm0 and dgemm1 respectively for n= 1000?

For n = 1000 dgemm0 should take:

$$4.005 \times 10^{11}[\text{cycl}] \ = 4[\text{mem ret, k}] \ \times \frac{100[\text{cycl}]}{[\text{mem ret}]} \times n^3 + \frac{1[\text{cycl}]}{4[\text{ops}]} \times 2[\text{ops}] \ \times n^3 \tag{1}$$

In seconds the dgemm0 should take:

$$200.25[\text{seconds}] \ = \frac{4.005 \times 10^{11}[\text{cycl}]}{2[\text{Ghz}]} \tag{2}$$

For n = 1000 dgemm1 should take:

$$2.007 \times 10^{11}[\text{cycl}] \ = 2[\text{mem retrev, k}] \ \times \frac{100[\text{cycl}]}{[\text{mem ret}]} \times n^3 + 2[\text{memory ret, j}] \ \times \frac{100[\text{cycl}]}{[\text{memory ret}]} \times n^2 + \frac{1[\text{cycl}]}{4[\text{ops}]} \times 2[\text{ops}] \ \times n^3 \tag{3}$$

In seconds the dgemm1 should take:

$$100.35[\text{seconds}] \ = \frac{2.007 \times 10^{11}[\text{cycl}]}{2[\text{Ghz}]} \tag{4}$$

How much time is wasted on accessing operands that are not in registers?
dgemm0:

$$200[\text{seconds}] \ = \frac{4[\text{mem ret}] \ \times \frac{100[\text{cycl}]}{[\text{mem ret}]} \times n^3}{2[\text{Ghz}]} \tag{5}$$

dgemm1:

$$100[\text{seconds}] \ = \frac{2[\text{mem ret, k}] \ \times \frac{100[\text{cycl}]}{[\text{mem ret}]} \times n^3 + 2[\text{mem ret, j}] \ \times \frac{100[\text{cycl}]}{[\text{mem ret}]} \times n^2}{2[\text{Ghz}]} \tag{6}$$

Implement the algorithm dgemm0 and dgemm1 and test them on your machine with n= 64, 128, 256, 512, 1024, 2048. Measure the time spend in the triple loop for each algorithm. Calculate the performance (in Gflops) of each algorithm. Performance is often defined as the number of floating-point operations performed per second. A performance of 1 GFLOPS means 1 billion of floating-point operations per second. You must use the system default compiler to compile your program. Your test matrices have to be 64-bit double floating point random numbers. Report the maximum difference of all matrix elements between the two results obtained from the two algorithms. This maximum difference can be used as a way to check the correctness of your implementation.

I used the Pantarhei system defaults on the head node to compile and run my code, which has been uploaded to Blackboard, can be found on Pantarhei here: `/home/sp19hpc_jmframe/hw/1/sp19hpc_jmframe_Homework1`, or in Appendix A. Below is a table showing the resulting seconds and Gflops for each algorithm. Appendix B shows the printout displaying the results of the correctness check of the three algorithms (zero difference between any elements of the matrix multiplication).

Table 1: Performance results of matrix multiplication algorithms
performance reported in (seconds, Gflops)

n	dgemm0	dgemm1	dgemm2
64	0, nan	0, nan	0, nan
128	0.02, 12950	0, nan	0.01, 25900
256	0.1, 5150	0.05, 10300	0.03, 17167
512	1.42, 723	0.92, 1116	0.37, 2776
1024	10.7, 192	6.9, 297	2.87, 714
2048	229.2, 18	151, 27	59.9, 68

2 Part #3

I implemented a three block matrix multiplication algorithm using 15 registers. I was not able to compare the results to dgemm0, dgemm1 and dgemm2 at the intervals listed in the homework (i.e., 64 through 2048) because the 3 block calculation can not do consistent matrix sizes as a blocks with multiples of 2. The code for part three has been uploaded to Blackboard, and is shown in Appendix C. The main thing about this algorithm is that it will reuse register storace multiple times in the same calculation. In other words, as the row-column multiplication summation occurs, each multiplication term is calculated separately recycling register storage, then added to the summation. Once these values are added to the summation, there is no use for them in the register memory, so the register space is then given to the next term in the calculation. After changing the to be multiples of 6, and only comparing dgemm3 with dgemm0, the results are provided in the table below.

Table 2: Performance results of matrix multiplication algorithm dgemm3
performance reported in (seconds, Gflops)

n	dgemm0	dgemm3
48	0, nan	0, nan
96	0.01, 19500	0, nan
192	0.04, 9675	0.01, 38799
384	0.36, 2142	0.08, 9638
768	4.66, 330	0.98, 1570
1536	76.4, 40	16.8, 183

A Code for Parts #1 & #2

sp19hpc_jmframe_Homework1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

int main () {

   int l, m, n, i, j, k;

   // LOOP THROUGH TEST VALUES
   for ( l = 8; l <= 2048; l += l ){
      n=l;
      // Number of total elements in the matrices, also the length of the
      // one dimensional vectors to do the matrix multiplications.
      m = n*n;
      printf("The matrix will be %d by %d, with a total of %d elements \n", n,n,m);

      //setting up an array with x rows and y columns
      double *A, *B, *C0, *C1, *C2;
      A = malloc(m * sizeof *A);
      B = malloc(m * sizeof *B);
      C0 = malloc(m * sizeof *C0); // Brute
      C1 = malloc(m * sizeof *C1); // dgemm1
      C2 = malloc(m * sizeof *C2); // dgemm2
      ////////////////////////////////////////////
      // SETTING UP THE MATRICES FOR MULTIPLICATION
      ////////////////////////////////////////////
      for ( i = 0; i < n; i++ ) {
         for ( j = 0; j < n; j++ ){
            A[i * n + j] = (double)rand()/RAND_MAX*2.0-1.0;
            B[i * n + j] = (double)rand()/RAND_MAX*2.0-1.0;
            C0[i * n + j] = 0;
            C1[i * n + j] = 0;
            C2[i * n + j] = 0;
         }
      }

      // Time this program, to use for efficiency tests
      clock_t start_0 = clock();

      ////////////////////////////////////////////////////////////
      // MAIN MATRIX MULTIPLICATION LOOP WITHOUT USING REGISTER MEMORY
      ////////////////////////////////////////////////////////////
      for ( i = 0; i < n; i++ ) {
         for ( j = 0; j < n; j++ ) {
            for ( k = 0; k < n; k++ ) {
               C0[i * n + j] += A[i * n + k] * B[k * n+ j];
            }
         }
      }

      // recording the time of completion for the matrix multiplication
      clock_t end_0 = clock();
      double total_0 = ((double)(end_0 - start_0)) / CLOCKS_PER_SEC;
      printf("dgemm0\n");
      printf("Total matrix multiplication run time WITHOUT Registmemory: %f\n", total_0);
      printf("Performance in GFLOPS: %f\n", (2*n^3)/total_0);
```

```
printf("\n");


// Time this program, to use for efficiency tests
clock_t start_1 = clock();

//////////////////////////////////////////////////////////////////
// MAIN MATRIX MULTIPLICATION LOOP WITH REGISTER MEMORY
//////////////////////////////////////////////////////////////////
for ( i = 0; i < n; i++ ) {
   for ( j = 0; j < n; j++ ) {
     register double r = C1[i * n + j];
     for ( k = 0; k < n; k++ ) {
         r += A[i * n + k] * B[k * n + j];
     }
     C1[ i * n + j] = r;
   }
}

// recording the time of completion for the matrix multiplication
clock_t end_1 = clock();
double total_1 = ((double)(end_1 - start_1)) / CLOCKS_PER_SEC;
printf("dgemm1\n");
printf("Total matrix multiplication run time WITH Register memor%f\n", total_1);
printf("Performance in GFLOPS: %f\n", (2*n^3)/total_1);
printf("\n");

// Time this program, to use for efficiency tests
clock_t start_2 = clock();

//////////////////////////////////////////////////////////////////
// MAIN MATRIX MULTIPLICATION LOOP WITH AGGRESIVE REGISTER MEMORY
//////////////////////////////////////////////////////////////////
for ( i = 0; i < n; i += 2 ) {
   for ( j = 0; j < n; j += 2 ) {
     register double c0 = C2[i * n + j];
     register double c1 = C2[(i + 1) * n + j];
     register double c2 = C2[i * n + (j + 1)];
     register double c3 = C2[(i + 1) * n + (j + 1)];
     for ( k = 0; k < n; k += 2 ) {
        register double a0 = A[i * n + k];
        register double a1 = A[i * n + (k + 1)];
        register double a2 = A[(i + 1) * n + k];
        register double a3 = A[(i + 1) * n + (k + 1)];
        register double b0 = B[k * n + j];
        register double b1 = B[(k + 1) * n + j];
        register double b2 = B[k * n + (j + 1)];
        register double b3 = B[(k + 1) * n + (j + 1)];
        c0 = a0 * b0 + a1 * b1 + c0;
        c1 = a2 * b0 + a3 * b1 + c1;
        c2 = a0 * b2 + a1 * b3 + c2;
        c3 = a2 * b2 + a3 * b3 + c3;
     }
     C2[i * n + j] = c0;
     C2[(i + 1) * n + j] = c1;
     C2[i * n + (j + 1)] = c2;
     C2[(i + 1) * n + (j + 1)] = c3;
   }
}
```

```
      // recording the time of completion for the matrix multiplication
      clock_t end_2 = clock();
      double total_2 = ((double)(end_2 - start_2)) / CLOCKS_PER_SEC;
      printf("dgemm2\n");
      printf("Total matrix multiplication run time... \n");
      printf(" with AGGRESSIVE register reuse: %f\n", total_2);
      printf("Performance in GFLOPS: %f\n", (2*n^3)/total_2);
      printf("\n");

      // Check that the matrix multiplactions are the same for each
      // NOTE: the matrix multiplications were tested in dgemm1 & dgemm0_1D
      double maxDiff1 = 0;
      double maxDiff2 = 0;
      double diff1;
      double diff2;
      for ( i = 0; i < m; i++ ) {
         diff1 = C1[i] - C0[i];
         diff2 = C2[i] - C0[i];
         if ( abs(diff1) > maxDiff1 ){
            maxDiff1 = diff1;
         }
         if ( abs(diff2) > maxDiff2 ){
            maxDiff2 = diff2;
         }
      }
      printf("The maximum difference between dgemm0 and dgemm1 is: %lf\n",maxDiff1);
      printf("The maximum difference between dgemm0 and dgemm2 is: %lf\n",maxDiff2);
      printf("------------------------------------------------------------\n\n\n");
   }
   return 0;
}
//End of program
```

B Printout for Parts # 1 & Part #2

```
sp19hpc_jmframe@pantarhei 1]$ ./sp19hpc_jmframe_Homework1
The matrix will be 64 by 64, with a total of 4096 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 0.000000
Performance in GFLOPS: inf

dgemm1
Total matrix multiplication run time WITH Register memory: 0.000000
Performance in GFLOPS: inf

dgemm2
Total matrix multiplication run time...
 with AGGRESSIVE register reuse: 0.000000
Performance in GFLOPS: inf

The maximum difference between dgemm0 and dgemm1 is: 0.000000
The maximum difference between dgemm0 and dgemm2 is: 0.000000
------------------------------------------------------------


The matrix will be 128 by 128, with a total of 16384 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 0.020000
Performance in GFLOPS: 12950.000000

dgemm1
Total matrix multiplication run time WITH Register memory: 0.000000
Performance in GFLOPS: inf

dgemm2
Total matrix multiplication run time...
 with AGGRESSIVE register reuse: 0.010000
Performance in GFLOPS: 25900.000000

The maximum difference between dgemm0 and dgemm1 is: 0.000000
The maximum difference between dgemm0 and dgemm2 is: 0.000000
------------------------------------------------------------


The matrix will be 256 by 256, with a total of 65536 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 0.100000
Performance in GFLOPS: 5150.000000

dgemm1
Total matrix multiplication run time WITH Register memory: 0.050000
Performance in GFLOPS: 10300.000000

dgemm2
Total matrix multiplication run time...
 with AGGRESSIVE register reuse: 0.030000
Performance in GFLOPS: 17166.666667

The maximum difference between dgemm0 and dgemm1 is: 0.000000
The maximum difference between dgemm0 and dgemm2 is: 0.000000
------------------------------------------------------------
```

```
The matrix will be 512 by 512, with a total of 262144 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 1.420000
Performance in GFLOPS: 723.239437

dgemm1
Total matrix multiplication run time WITH Register memory: 0.920000
Performance in GFLOPS: 1116.304348

dgemm2
Total matrix multiplication run time...
 with AGGRESSIVE register reuse: 0.370000
Performance in GFLOPS: 2775.675676

The maximum difference between dgemm0 and dgemm1 is: 0.000000
The maximum difference between dgemm0 and dgemm2 is: 0.000000
------------------------------------------------------------


The matrix will be 1024 by 1024, with a total of 1048576 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 10.690000
Performance in GFLOPS: 191.861553

dgemm1
Total matrix multiplication run time WITH Register memory: 6.900000
Performance in GFLOPS: 297.246377

dgemm2
Total matrix multiplication run time...
 with AGGRESSIVE register reuse: 2.870000
Performance in GFLOPS: 714.634146

The maximum difference between dgemm0 and dgemm1 is: 0.000000
The maximum difference between dgemm0 and dgemm2 is: 0.000000
------------------------------------------------------------


The matrix will be 2048 by 2048, with a total of 4194304 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 229.190000
Performance in GFLOPS: 17.884724

dgemm1
Total matrix multiplication run time WITH Register memory: 151.030000
Performance in GFLOPS: 27.140303

dgemm2
Total matrix multiplication run time...
 with AGGRESSIVE register reuse: 59.910000
Performance in GFLOPS: 68.419296

The maximum difference between dgemm0 and dgemm1 is: 0.000000
The maximum difference between dgemm0 and dgemm2 is: 0.000000
------------------------------------------------------------
```

C Code for Part #3

```
//////////////////////////////////////////////////////////////////
// MAIN MATRIX MULTIPLICATION LOOP WITH OPTIMIZED REGISTER MEMORY
//////////////////////////////////////////////////////////////////
for ( i = 0; i < n; i += 3 )
   for ( j = 0; j < n; j += 3 ) {
      register double c00 = C3[i * n + j];
      register double c10 = C3[(i + 1) * n + j];
      register double c20 = C3[(i + 2) * n + j];
      register double c01 = C3[i * n + (j + 1)];
      register double c11 = C3[(i + 1) * n + (j + 1)];
      register double c21 = C3[(i + 2) * n + (j + 1)];
      register double c02 = C3[i * n + (j + 2)];
      register double c12 = C3[(i + 1) * n + (j + 2)];
      register double c22 = C3[(i + 2) * n + (j + 2)];
      for ( k = 0; k < n; k += 3 ) {
         register double a00 = A[i * n + k];
         register double a10 = A[(i + 1) * n + k];
         register double a20 = A[(i + 2) * n + k];
         register double b00 = B[k * n + j];
         register double b01 = B[k * n + (j + 1)];
         register double b02 = B[k * n + (j + 2)];
         c00 = c00 + a00 * b00;
         c10 = c10 + a10 * b00;
         c20 = c20 + a20 * b00;
         c01 = c01 + a00 * b01;
         c11 = c11 + a10 * b01;
         c21 = c21 + a20 * b01;
         c02 = c02 + a00 * b02;
         c12 = c12 + a10 * b02;
         c22 = c22 + a20 * b02;
         // These variable keep the same name, but no longer represent thesevalues.
         a00 = A[i * n + (k + 1)];
         a10 = A[(i + 1) * n + (k + 1)];
         a20 = A[(i + 2) * n + (k + 1)];
         b00 = B[(k + 1) * n + j];
         b01 = B[(k + 1) * n + (j + 1)];
         b02 = B[(k + 1) * n + (j + 2)];
         // Now add the values to the C matrix
         c00 = c00 + a00 * b00;
         c10 = c10 + a10 * b00;
         c20 = c20 + a20 * b00;
         c01 = c01 + a00 * b01;
         c11 = c11 + a10 * b01;
         c21 = c21 + a20 * b01;
         c02 = c02 + a00 * b02;
         c12 = c12 + a10 * b02;
         c22 = c22 + a20 * b02;
         // Again, these values do not correspond with the variable names.
         a00 = A[i * n + (k + 2)];
         a10 = A[(i + 1) * n + (k + 2)];
         a20 = A[(i + 2) * n + (k + 2)];
         b00 = B[(k + 2) * n + j];
         b01 = B[(k + 2) * n + (j + 1)];
         b02 = B[(k + 2) * n + (j + 2)];
         // Make the final sum into the resulting C matrix
         c00 = c00 + a00 * b00;
         c10 = c10 + a10 * b00;
         c20 = c20 + a20 * b00;
```

```
            c01 = c01 + a00 * b01;
            c11 = c11 + a10 * b01;
            c21 = c21 + a20 * b01;
            c02 = c02 + a00 * b02;
            c12 = c12 + a10 * b02;
            c22 = c22 + a20 * b02;
        }
        // Not move the data from the register to slow memory
        C3[i * n + j] = c00;
        C3[(i + 1) * n + j] = c10;
        C3[(i + 2) * n + j] = c20;
        C3[i * n + (j + 1)] = c01;
        C3[(i + 1) * n + (j + 1)] = c11;
        C3[(i + 2) * n + (j + 1)] = c21;
        C3[i * n + (j + 2)] = c02;
        C3[(i + 1) * n + (j + 2)] = c12;
        C3[(i + 2) * n + (j + 2)] = c22;
}
```

D Printout for Part # 3

```
sp19hpc_jmframe@pantarhei 1]$ ./sp19hpc_jmframe_Homework1_part3
The matrix will be 48 by 48, with a total of 2304 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 0.000000
Performance in GFLOPS: inf

dgemm3
Total matrix multiplication run time...
 with MAXIMIZED register reuse: 0.000000
Performance in GFLOPS: inf

The maximum difference between dgemm0 and dgemm3 is: 0.000000
------------------------------------------------------------


The matrix will be 96 by 96, with a total of 9216 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 0.010000
Performance in GFLOPS: 19500.000000

dgemm3
Total matrix multiplication run time...
 with MAXIMIZED register reuse: 0.000000
Performance in GFLOPS: inf

The maximum difference between dgemm0 and dgemm3 is: 0.000000
------------------------------------------------------------


The matrix will be 192 by 192, with a total of 36864 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 0.040000
Performance in GFLOPS: 9675.000000

dgemm3
Total matrix multiplication run time...
 with MAXIMIZED register reuse: 0.010000
Performance in GFLOPS: 38700.000000

The maximum difference between dgemm0 and dgemm3 is: 0.000000
------------------------------------------------------------


The matrix will be 384 by 384, with a total of 147456 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 0.360000
Performance in GFLOPS: 2141.666667

dgemm3
Total matrix multiplication run time...
 with MAXIMIZED register reuse: 0.080000
Performance in GFLOPS: 9637.500000

The maximum difference between dgemm0 and dgemm3 is: 0.000000
------------------------------------------------------------


The matrix will be 768 by 768, with a total of 589824 elements
```

```
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 4.660000
Performance in GFLOPS: 330.257511

dgemm3
Total matrix multiplication run time...
 with MAXIMIZED register reuse: 0.980000
Performance in GFLOPS: 1570.408163

The maximum difference between dgemm0 and dgemm3 is: 0.000000
------------------------------------------------------------


The matrix will be 1536 by 1536, with a total of 2359296 elements
dgemm0
Total matrix multiplication run time WITHOUT Register memory: 76.400000
Performance in GFLOPS: 40.248691

dgemm3
Total matrix multiplication run time...
 with MAXIMIZED register reuse: 16.790000
Performance in GFLOPS: 183.144729

The maximum difference between dgemm0 and dgemm3 is: 0.000000
------------------------------------------------------------
```