Lab 1 Report

Andres Calderon

acald013@ucr.edu

October 3, 2016

1 Questions

1.1 dgemm0 and dgemm1

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

In the case of `dgemm0`, it takes 2 floating-point operation (an addition and a product) and 3 access to memory (for each matrix). So, it will take 2 floating-point operation each of them taking a quarter of cycle and 3 memory accesses each of them taking 100 cycles per iteration. As the total number of iteration is n^3, we have:

$$(2 \times \frac{1}{4} + 3 \times 100) \times 1000^3$$

$$(\frac{1}{2} + 300) \times 10^9$$

As the frequency of your computer is 2 Ghz, we have:

$$\frac{(\frac{1}{2} + 300) \times 10^9}{2 \times 10^9} = 150.25$$

From here, 150s are spent on access memory and just 0.25s on floating-point operations.

In the case of `dgemm1`, the computation is similar but it just take 2 access to memory per iteration.

$$(2 \times \frac{1}{4} + 2 \times 100) \times 1000^3$$

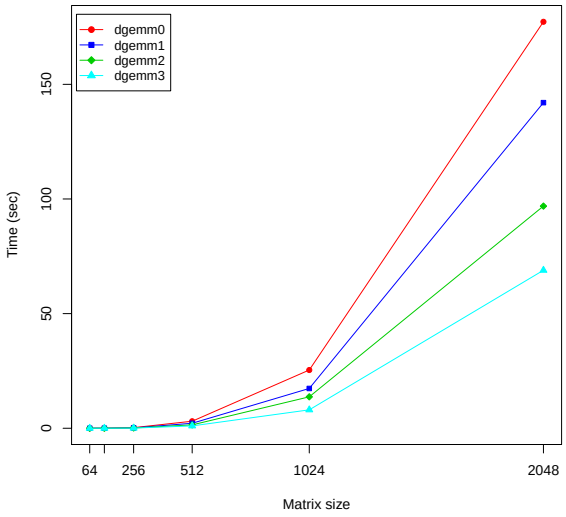

Figure 1: Matrix multiplication performance.

$$(\frac{1}{2} + 200) \times 10^9$$

As the frequency of your computer is 2 Ghz, we have:

$$\frac{(\frac{1}{2} + 200) \times 10^9}{2 \times 10^9} = 100.25$$

So, although it spent the same amount of time for floating-point operations, it just spent 100s accessing memory.

- Implement the algorithm dgemm0 and dgemm1 and test them on TARDIS 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.

Appendix

Source code

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

// Declaring functions...
void dgemm0(double *a, double *b, double *c, int n);
void dgemm1(double *a, double *b, double *c, int n);
void dgemm2(double *a, double *b, double *c, int n);
void dgemm3(double *a, double *b, double *c, int n);
double verification(double *a, double *b, double *c0, double *c1, int n);

int main(int argc, char* argv[]){
  // Reading N from command line...
  int n = atoi(argv[1]);
  uint64_t t0;
  float t;
  struct timespec begin, end;
  double *a, *b;
  double *c0, *c1, *c2, *c3;
  double diff;
  int i;

  // Random seed...
  srand(time(NULL));
  // Creating matrices A and B...
  a = (double *) calloc(sizeof(double), n * n);
  for(i = 0; i < n * n; i++){
    a[i] = rand() / 1000000.0;
  }
  b = (double *) calloc(sizeof(double), n * n);
  for(i = 0; i < n * n; i++){
    b[i] = rand() / 1000000.0;
  }
  // Allocating memory for matrices C's...
  c0 = (double *) calloc(sizeof(double), n * n);
  c1 = (double *) calloc(sizeof(double), n * n);
  c2 = (double *) calloc(sizeof(double), n * n);
  c3 = (double *) calloc(sizeof(double), n * n);

  // Running dgemm0...
  clock_gettime(CLOCK_MONOTONIC, &begin);
  dgemm0(a, b, c0, n);
  clock_gettime(CLOCK_MONOTONIC, &end);
  t0 = 1000000000L * (end.tv_sec - begin.tv_sec) + end.tv_nsec - begin.tv_nsec;
  t = t0 / 1000000000.0;
  printf("%f\t", t);

  // Running dgemm1...
  clock_gettime(CLOCK_MONOTONIC, &begin);
  dgemm1(a, b, c1, n);
```

```
53    clock_gettime(CLOCK_MONOTONIC, &end);
54    t0 = 1000000000L * (end.tv_sec - begin.tv_sec) + end.tv_nsec - begin.tv_nsec;
55    t = t0 / 1000000000.0;
56    printf("%f\t", t);
57
58    // Running dgemm2...
59    clock_gettime(CLOCK_MONOTONIC, &begin);
60    dgemm2(a, b, c2, n);
61    clock_gettime(CLOCK_MONOTONIC, &end);
62    t0 = 1000000000L * (end.tv_sec - begin.tv_sec) + end.tv_nsec - begin.tv_nsec;
63    t = t0 / 1000000000.0;
64    printf("%f\t", t);
65
66    // Running dgemm3...
67    clock_gettime(CLOCK_MONOTONIC, &begin);
68    dgemm3(a, b, c3, n);
69    clock_gettime(CLOCK_MONOTONIC, &end);
70    t0 = 1000000000L * (end.tv_sec - begin.tv_sec) + end.tv_nsec - begin.tv_nsec;
71    t = t0 / 1000000000.0;
72    printf("%f\t", t);
73
74    // Running verifications...
75    diff = verification(a, b, c0, c1, n);
76    printf("%f\t", diff);
77    diff = verification(a, b, c0, c2, n);
78    printf("%f\t", diff);
79    diff = verification(a, b, c0, c3, n);
80    printf("%f\n", diff);
81
82    return 0;
83  }
84
85  void dgemm0(double *a, double *b, double *c, int n){
86    int i,j,k;
87    for(i = 0; i < n; i++){
88      for(j = 0; j < n; j++){
89        for(k = 0; k < n; k++){
90          c[i*n+j] += a[i*n+k] * b[k*n+j];
91        }
92      }
93    }
94  }
95
96  void dgemm1(double *a, double *b, double *c, int n){
97    int i,j,k;
98    for(i = 0; i < n; i++){
99      for(j = 0; j < n; j++){
100       register double r = c[i*n+j];
101       for(k = 0; k < n; k++){
102         r += a[i*n+k] * b[k*n+j];
103       }
104       c[i*n+j] = r;
105     }
106   }
107 }
108
```

```
109 void dgemm2(double *a, double *b, double *c, int n) {
110   int i, j, k;
111   for (i = 0; i < n; i+=2){
112     for (j = 0; j < n; j+=2){
113       for (k = 0; k < n; k+=2){
114         c[i*n+j] = a[i*n+k] * b[k*n+j] + a[i*n+(k+1)] * b[(k+1)*n+j] + c[i*n+j];
115         c[(i+1)*n+j] = a[(i+1)*n+k] * b[k*n+j] + a[(i+1)*n+(k+1)] * b[(k+1)*n+j] + c[(i+1)*n+j];
116         c[i*n+(j+1)] = a[i*n+k] * b[k*n+(j+1)] + a[i*n+(k+1)] * b[(k+1)*n+(j+1)] + c[i*n+(j+1)];
117         c[(i+1)*n+(j+1)] = a[(i+1)*n+k] * b[k*n+(j+1)] + a[(i+1)*n+(k+1)] * b[(k+1)*n+(j+1)] + c[(i+1)*n+(j+1)];
118       }
119     }
120   }
121 }
122
123 void dgemm3(double *a, double *b, double *c, int n) {
124   int i, j, k;
125   for (i = 0; i < n; i+=2){
126     for (j = 0; j < n; j+=2){
127       register double cc0 = c[i*n+j];
128       register double cc1 = c[(i+1)*n+j];
129       register double cc2 = c[i*n+(j+1)];
130       register double cc3 = c[(i+1)*n+(j+1)];
131       for (k = 0; k < n; k+=2){
132         register double aa0 = a[i*n+k];
133         register double aa1 = a[i*n+(k+1)];
134         register double aa2 = a[(i+1)*n+k];
135         register double aa3 = a[(i+1)*n+(k+1)];
136         register double bb0 = b[k*n+j];
137         register double bb1 = b[(k+1)*n+j];
138         register double bb2 = b[k*n+(j+1)];
139         register double bb3 = b[(k+1)*n+(j+1)];
140         cc0 += aa0 * bb0 + aa1 * bb1;
141         cc1 += aa2 * bb0 + aa3 * bb1;
142         cc2 += aa0 * bb2 + aa1 * bb3;
143         cc3 += aa2 * bb2 + aa3 * bb3;
144       }
145       c[i*n+j] = cc0;
146       c[(i+1)*n+j] = cc1;
147       c[i*n+(j+1)] = cc2;
148       c[(i+1)*n+(j+1)] = cc3;
149     }
150   }
151 }
152
153 double verification(double *a, double *b, double *c0, double *c1, int n){
154   double diff, maxA, maxB;
155   int i;
156
157   diff = fabs(c1[0] - c0[0]);
158   maxA = fabs(a[0]);
159   maxB = fabs(b[0]);
160   for(i = 0; i < n * n; i++){
161     if(fabs(c1[i] - c0[i]) > diff)
162       diff = fabs(c1[i] - c0[i]);
163     if(fabs(a[i]) > maxA)
164       maxA = fabs(a[i]);
```

```
165       if(fabs(b[i]) > maxB)
166         maxB = fabs(b[i]);
167     }
168     return diff / (maxA * maxB);
169   }
```