COMP2006 Computer Organization
Matrix Multiply & Time Measurement

DGEMM

Most floating-point calculations are performed in double precision. Consider the following matrix multiply:

$$C = C + A * B$$

It is commonly called **DGEMM**, for Double precision, General Matrix Multiply. Which calculates the multiplication of matrix **A** and matrix **B** and stores the result in matrix **C**.

The following is a simple version of the **dgemm** function in the *C* program language:

```
void dgemm(int n, double *A, double *B, double *C)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            double cij = C[i * n + j];
            for (int k = 0; k < n; k++)
                cij += A[i * n + k] * B[k * n + j];
            C[i * n + j] = cij;
        }
}
```

We use the following code to create two matrices and use the **dgemm** function to compute the results.

```
#include <stdio.h>

void dgemm(int n, double *A, double *B, double *C){...}

void main()
{
    int i, j, n = 4;
    double A[] = {1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7};
    double B[] = {1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7};
    double C[16] = {0};
    dgemm(n, A, B, C);
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            printf("%.0lf\t", C[i * n + j]);
        printf("\n");
    }
}
```

You can see that we use single dimensional arrays to represent matrices.

Download **dgemm1.c** from our course web page. Then, compile the program and run it.

Compile: ***gcc dgemm1.c -o dgemm1.exe*** *Run:* ***dgemm1.exe***

You will get the results as follows:

```
30      40      50      60
40      54      68      82
50      68      86      104
60      82      104     126
```

Time Measurement

Next, we are going to measure how many seconds the program will take for calculating the result. We use the **clock** function provided by **time.h** to measure the processing time. Therefore, we need to modify the main function as follows:

```c
#include <stdio.h>
#include <sys/time.h>

void dgemm(int n, double *A, double *B, double *C){...}

int main()
{
    int i, j, n = 4;
    double A[] = {1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7};
    double B[] = {1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7};
    double C[16] = {0};

    struct timeval st, et;
    gettimeofday(&st, NULL);

    dgemm(n, A, B, C);

    gettimeofday(&et, NULL);
    float elapsed = ((float)(et.tv_sec - st.tv_sec))
        + (float)(et.tv_usec - st.tv_usec)*0.000001f;

    printf("Calculation time for %d x %d matrix: %0.6f seconds\n", n, n, elapsed);

    return 0;
}
```

Download **dgemm2.c** from our course web page. Then, compile the program and run it.

Compile: ***gcc dgemm2.c -o dgemm2.exe*** *Run:* ***dgemm2.exe***

You will get the results as follows:

```
Calculation time for 4 x 4 matrix: 0.000001 seconds
```

You may get different result. The calculation time is depended on the performance of your computer and the current loading of the processor.

Dynamic Arrays

We then use some larger matrices to test the **dgemm** function. To make our program more flexible, we use dynamic array creation approach instead. We first create a double pointer and use the **malloc** function to allocate memory space with a specific size. Then, use the pointer to point to the memory space.

```
ARRAY = malloc(N * sizeof(data_type));    // create an array with N elements
```

Let's change the program as follows but we keep the **dgemm** function unchanged:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

void dgemm(int n, double *A, double *B, double *C){...}

void initialize(int n, double *X)
{
    int i;
    for (i = 0; i < n; i++)
        X[i] = (double)rand() / (double)RAND_MAX;
}

int main()
{
    int n;
    printf("Enter the matrix width: ");
    scanf("%d", &n);

    double *A = (double *)malloc(n * n * sizeof(double));
    double *B = (double *)malloc(n * n * sizeof(double));
    double *C = (double *)malloc(n * n * sizeof(double));

    srand(1);
    initialize(n * n, A);
    initialize(n * n, B);

    memset(C, 0, n * n * sizeof(double));

    struct timeval st, et;
    gettimeofday(&st, NULL);

    dgemm(n, A, B, C);

    gettimeofday(&et, NULL);

    float elapsed = ((float)(et.tv_sec - st.tv_sec))
        + (float)(et.tv_usec - st.tv_usec)*0.000001f;

    printf("Calculation time for %d x %d matrix: %0.6f seconds\n", n, n, elapsed);

    return 0;
}
```

The **memset** statement in the program code above is used to set all elements to zero.

We also need the **initialize** function to generate some random values for the matrices. Add the following **random** function to your program file too.

Download **dgemm3.c** from our course web page. Then, compile the program and run it.

Compile: ***gcc dgemm3.c -o dgemm3.exe*** *Run:* ***dgemm3.exe***

The following is the sample output of the calculation time for two 512 x 512 matrices:

```
Enter the matrix width: 512

Calculation time for 512 x 512 matrix: 0.603715 seconds
```

You may get different result. The calculation time is depended on the performance of your computer and the current loading of the processor.