Homework 2

ICT-LXB

May 3, 2014

Contents

1 There Loops Resolution

1.1 Source Code

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

```
#define DIM 4096
typedef double(* MATRIX)[DIM];

void multiply(MATRIX z, const MATRIX x, const MATRIX y, int dim)
{
    int row, col, idx, tmp;
    for (row = 0; row < dim; row++)
        for (col = 0; col < dim; col++) {
            tmp = 0;
            for (idx = 0; idx < dim; idx++)
                tmp += x[row][idx] * y[idx][col];
            z[row][col] = tmp;
        }
}

void init_matrix(MATRIX m, int dim, int isRandom)
{
    int row, col;
    if (isRandom) {
        srandom(time(NULL));
        for (row = 0; row < dim; row++)
            for (col = 0; col < dim; col++)
                m[row][col] = random() % 263;
    } else
        memset((void *)m, 0, dim * dim * sizeof(m[0][0]));
}

void print_matrix(MATRIX m, int dim)
{
    int row, col;
    for (row = 0; row < dim; row++) {
        for (col = 0; col < dim; col++)
            printf("%f ", m[row][col]);
        printf("\n");
    }
}

int main(void)
{
    MATRIX x = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX y = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX z = (MATRIX)malloc(DIM*DIM*sizeof(double));
    init_matrix(x, DIM, 1);
    init_matrix(y, DIM, 1);
```

```
    init_matrix(z, DIM, 0);
    struct timeval start, end;
    gettimeofday(&start, NULL);
    multiply(z, x, y, DIM);
    gettimeofday(&end, NULL);
    int time = (int)(end.tv_sec-start.tv_sec);
    unsigned long long opts = (unsigned long long)DIM*DIM*(2*DIM-1);
    double speed = opts / (double)(time * 1000000);
    printf("%d matrix multiply\n", DIM);
    printf("time : %ds\tspeed: %.2f MFLOPS\n", time, speed);
    return 0;
}
```

1.2 Results

1.2.1 No Optimization

```
ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix
4096 matrix multiply
time : 1065s    speed: 129.03 MFLOPS
```

1.2.2 -O2 Optimization

```
ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix
4096 matrix multiply
time : 813s speed: 169.03 MFLOPS
```

1.3 Conclusion

After analysing the results above, we can see that the same source code compiled with different optimization options might result in very different performance. The compiler can improve the performance by reordering the instructions, which might cause bubble in the instruction parallel, and so on.

2 Three Loops Resulotion (4097X4097)

2.1 Source Code

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

```
#define DIM 4097
typedef double(* MATRIX)[DIM];

void multiply(MATRIX z, const MATRIX x, const MATRIX y, int dim)
{
    int row, col, idx, tmp;
    for (row = 0; row < dim; row++)
        for (col = 0; col < dim; col++) {
            tmp = 0;
            for (idx = 0; idx < dim; idx++)
                tmp += x[row][idx] * y[idx][col];
            z[row][col] = tmp;
        }
}

void init_matrix(MATRIX m, int dim, int isRandom)
{
    int row, col;
    if (isRandom) {
        srandom(time(NULL));
        for (row = 0; row < dim; row++)
            for (col = 0; col < dim; col++)
                m[row][col] = random() % 263;
    } else
        memset((void *)m, 0, dim * dim * sizeof(m[0][0]));
}

void print_matrix(MATRIX m, int dim)
{
    int row, col;
    for (row = 0; row < dim; row++) {
        for (col = 0; col < dim; col++)
            printf("%f ", m[row][col]);
        printf("\n");
    }
}

int main(void)
{
    MATRIX x = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX y = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX z = (MATRIX)malloc(DIM*DIM*sizeof(double));
    init_matrix(x, DIM, 1);
    init_matrix(y, DIM, 1);
```

```
    init_matrix(z, DIM, 0);
    struct timeval start, end;
    gettimeofday(&start, NULL);
    multiply(z, x, y, DIM);
    gettimeofday(&end, NULL);
    int time = (int)(end.tv_sec-start.tv_sec);
    unsigned long long opts = (unsigned long long)DIM*DIM*(2*DIM-1);
    double speed = opts / (double)(time * 1000000);
    printf("%d matrix multiply\n", DIM);
    printf("time : %ds\tspeed: %.2f MFLOPS\n", time, speed);
    return 0;
}
```

2.2 Results

1.
```
ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix
4097 matrix multiply
time : 801s speed: 171.69 MFLOPS
```

2.
```
ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix
4097 matrix multiply
time : 880s speed: 156.28 MFLOPS
```

3.
```
ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix
4097 matrix multiply
time : 795s speed: 172.98 MFLOPS
```

Average Time: 825s; Average Speed: 166.98 MFLOPS

2.3 Conclusion

After analysing the results above, we can see that the performance migth vary from time to time. Comparing the average time with the time of Experiment One, we can see that the time does not increase too much and the FLOPS decreases a little. The reasom might be the cache miss rates increase a bit.

3 Three Loops Resulotion With Transpose

3.1 Source Code

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

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

#define DIM 4097
typedef double(* MATRIX)[DIM];

void transpose(MATRIX m, int dim)
{
    int row, col, tmp;
    for (row = 0; row < dim; row++)
        for (col = 0; col < dim; col++) {
            if (col == row)
                continue;
            tmp = m[row][col];
            m[row][col] = m[col][row];
            m[col][row] = tmp;
        }
}

void multiply(MATRIX z, const MATRIX x, const MATRIX y, int dim)
{

    int row, col, idx, tmp;
    transpose(y, dim);
    for (row = 0; row < dim; row++)
        for (col = 0; col < dim; col++) {
            tmp = 0;
            for (idx = 0; idx < dim; idx++)
                tmp += x[row][idx] * y[col][idx];
            z[row][col] = tmp;
        }
}

void init_matrix(MATRIX m, int dim, int isRandom)
{
    int row, col;
    if (isRandom) {
        srandom(time(NULL));
        for (row = 0; row < dim; row++)
            for (col = 0; col < dim; col++)
                m[row][col] = random() % 263;
    } else
        memset((void *)m, 0, dim * dim * sizeof(m[0][0]));
}
```

```
void print_matrix(MATRIX m, int dim)
{
    int row, col;
    for (row = 0; row < dim; row++) {
        for (col = 0; col < dim; col++)
            printf("%f ", m[row][col]);
        printf("\n");
    }
}

int main(void)
{
    MATRIX x = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX y = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX z = (MATRIX)malloc(DIM*DIM*sizeof(double));
    init_matrix(x, DIM, 1);
    init_matrix(y, DIM, 1);
    init_matrix(z, DIM, 0);
    struct timeval start, end;
    gettimeofday(&start, NULL);
    multiply(z, x, y, DIM);
    gettimeofday(&end, NULL);
    int time = (int)(end.tv_sec-start.tv_sec);
    unsigned long long opts = (unsigned long long)DIM*DIM*(2*DIM-1);
    double speed = opts / (double)(time * 1000000);
    printf("%d matrix multiply\n", DIM);
    printf("time : %ds\tspeed: %.2f MFLOPS\n", time, speed);
    return 0;
}
```

3.2 Results

1. ```
   ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix_transpose
   4097 matrix multiply
   time : 234s speed: 587.70 MFLOPS
   ```

2. ```
   ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix_transpose
   4097 matrix multiply
   time : 235s speed: 585.20 MFLOPS
   ```

3. ```
   ict-lxb@ictlxb-Zhaoyang-E49:~/Workspace/test$ ./matrix_transpose
   4097 matrix multiply
   time : 234s speed: 587.70 MFLOPS
   ```

Average Time: 234.33s; Average Speed: 586.87 MFLOPS
   ```

3.3 Conclusion

After analysing the results above, we can see that the performance is improved so much. The reasons for that might be that the array in C program language is row-major and that cache miss rate decreases due to the transpose. Temporal locality and spatial locality is the key point in this improvement.

4 Three Loops Resulotion With Partition

4.1 Source Code

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

#define B 512
#define DIM 4096
typedef double(* MATRIX)[DIM];

#define MIN(x, y) ((x) > (y) ? (y) : (x))

void multiply(MATRIX z, const MATRIX x, const MATRIX y, int dim)
{
    int row, col, idx, ri, ci, tmp;
    for (row = 0; row < dim; row+=B)
        for (col = 0; col < dim; col+=B)
            for (ri = 0; ri < dim; ri++)
                for (ci = col; ci < MIN(dim, col+B); ci++) {
                    tmp = 0;
                    for (idx = col; idx < MIN(dim, col+B); idx++)
                        tmp += x[ri][idx] * y[idx][ci];
                    z[ri][ci] += tmp;
                }
}

void init_matrix(MATRIX m, int dim, int isRandom)
{
    int row, col;
    if (isRandom) {
        for (row = 0; row < dim; row++)
            for (col = 0; col < dim; col++)
```

```
                m[row][col] = random() % 263;
    } else
        memset((void *)m, 0, dim * dim * sizeof(m[0][0]));
}

void print_matrix(MATRIX m, int dim)
{
    int row, col;
    for (row = 0; row < dim; row++) {
        for (col = 0; col < dim; col++)
            printf("%f ", m[row][col]);
        printf("\n");
    }
}

int main(void)
{
    srandom(time(NULL));
#if 1
    MATRIX x = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX y = (MATRIX)malloc(DIM*DIM*sizeof(double));
    MATRIX z = (MATRIX)malloc(DIM*DIM*sizeof(double));
    init_matrix(x, DIM, 1);
    init_matrix(y, DIM, 1);
    init_matrix(z, DIM, 0);
#else
    double x[DIM][DIM] = {
        {1, 2},
        {3, 4}
    };
    double y[DIM][DIM] = {
        {4, 3},
        {2, 1}
    };
    double z[DIM][DIM] = {
        {0, },
    };
#endif
    struct timeval start, end;
    gettimeofday(&start, NULL);
    multiply(z, x, y, DIM);
    gettimeofday(&end, NULL);
    int time = (int)(end.tv_sec-start.tv_sec);
    unsigned long long opts = (unsigned long long)DIM*DIM*(2*DIM-1);
```

```
    double speed = opts / (double)(time * 1000000);
    printf("%d matrix multiply\n", DIM);
    printf("time : %ds\tspeed: %.2f MFLOPS\n", time, speed);
#if 0
    print_matrix(x, DIM);
    print_matrix(y, DIM);
    print_matrix(z, DIM);
#endif
    return 0;
}
```

4.2 Results

picture

4.3 Conclusion

After analysing the results above, we can see that the performance is optimal when B is 64.