High Performance Computing
Homework 3
Report

Peiyun Hu, 2010011297
Department of Computer Science & Technology

November 2, 2012

1 Environment

1.1 CPU

Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz cpu cores : 4 processors : 8

1.2 Compiler

gcc version 4.6.1 (Ubuntu/Linaro 4.6.1-9ubuntu3)

1.3 Makefile

gcc -fopenmp -w -o $exec_path -O2 $src_path -lm -lrt

1.4 Number of Processes

All the results of parallel programming below is based on 8 processes.

2 The Effect of Parallelism on Different Level of Nested Loop

We will research on the multiplication of matrix m × k and matrix k × n, of which the product is matrix m × n.

2.1 The Place of OpenMP directive

2.1.1 Implement

The place of OpenMP directive is decisive on the performance. For example, if we look into this problem by altering the directive '# pragma omp for', which did not indicate the schedule strategy differently yet, to different places above each nested for-loop. What is more, this order of nested for-loop is i-loop, j-loop, k-loop successively. Specifically, part of the code is as follow:

Listing 1: i-loop Paralleled

```
1 #pragma omp parallel private(i, j, k)
```

```
{
#pragma omp for schedule(static)
    for(i=0; i<a_rows; i++)
        for(j=0; j<b_cols; j++)
            for(k=0; k<a_cols; k++)
                res[i*a_cols+j] += v1[i*a_cols+k] * v2[k*b_cols+j
                    ];
}
```

Listing 2: j-loop Paralleled

```
#pragma omp parallel private(i, j, k)
{
    for(i=0; i<a_rows; i++)
    #pragma omp for schedule(static)
        for(j=0; j<b_cols; j++)
            for(k=0; k<a_cols; k++)
                res[i*a_cols+j] += v1[i*a_cols+k] * v2[k*b_cols+j
                    ];
}
```

Listing 3: k-loop Paralleled

```
#pragma omp parallel private(i, j, k)
{
    for(i=0; i<a_rows; i++)
        for(j=0; j<b_cols; j++)
        #pragma omp for schedule(static)
            for(k=0; k<a_cols; k++){
                double tmp = v1[i*a_cols+k] * v2[k*b_cols+j];
                #pragma omp critical
                {
                    res[i*a_cols+j] += tmp;
                }
            }
}
```

Moreover, in the Listing 3, the res[i*a_ cols+j] could be visited at the same time by different processes. So, before assigning new value to res[i*a_ cols+j], we should ensure that only one process is executing this statement, which is implemented using statement 1.

 # pragma omp critical (1)

2.1.2 Result

The result shown in Table 1 is presented under the condition that the order of loops are i, j, k successively, and what is the strategy of schedule is static. The OpenMP directive is placed before the for-loop, as is shown in Listing 1, 2 and 3.

The Place of Directive	Serial	i-loop	j-loop	k-loop
Time(s)	8.710679	2.510435	2.710924	322.320779

Table 1: Performance with different placement of OpenMP directives in nested loops

As is shown in the Table 1, placing the directive before i-loop and j-loop will almost achieve the same performance, and nearly $\frac{1}{4}$ compared to serial performance. But owing to the competition and wait among threads, the version of k-loop paralleled has a unimaginable poor performance.

In Conclusion, the OpenMP directive should be placed in the first two loop, for example, i-loop or j-loop.

2.2 Loop Order

2.2.1 Implement

We ensure that the first loop is paralleled, and change the loop order. For example, we exchange the order of i-loop with j-loop, or i-loop with k-loop. But, if k is the first loop that is paralleled, the execution should be ensured safe, which means that '#pragma omp critical' should be placed before specific assignment. In detail, part of the implement is presented in Table 2:

2.2.2 Result

The result presented in Table 2 is under the condition of different order of for-loops.

The Order of For-loop	i, j, k	j, k, i	k, j, i
Time(Paralleled)(s)	2.510435	3.407517	265.025561
Time(Without Paralleled)(s)	9.077633	8.509484	20.044858

Table 2: Performance with Different Order of Nested Loops

In conclusion, the order of for-loop should be i, j, k or j, i, k, instead of making k-loop the first nested loop.

3 The Effect of Parallelism For Different Parameters

3.1 Experiment

In this experiment, we iterate 1000 iterations with 4 threads. When iterating the **i th loop**, the delay within the dummy function is **i*100**.

The parameters of OpenMP directive is presented as in Listing 4:

Listing 4: The parameter for Different Schedule

```
1 #pragma omp parallel for schedule(static, 200)
2 #pragma omp parallel for schedule(dynamic)
3 #pragma omp parallel for schedule(guided, 100)
```

3.2 Result

The results is shown below in Table 3.

Schedule	Static	Dynamic	Guided
Time(s)	20.016861	12.549876	16.057561

Table 3: Performance with Different Order of Nested Loops

As is shown in Table 3, we could see that in time consumed using static schedule is more much more than dynamic and guided, while dynamic is less than guided. In this situation, the time invocating **dummy** once is increasing as iterating. Consequently, **if tasks are distributed using static schedule, then the tasks will cost more time when task id increase**. While, **guided schedule will ensure that the scale of task would decrease as iterating**. Moreover, **dynamic schedule could distributed more flexibly**, and consume the least time.

4 Pizza World Scenario

4.1 Experiment

4.1.1 Sample Input

22002 22111 20002 01111 00022 12220 21122 02211 00202 21012 00002 02202 10022 00221 00201 11002 01212 02200 12122 10212 10010 20201 01200 21220

4.1.2 Sample Output

```
Execution time:                          0.001440
Total number of pizza served:            56
Total profit of for the day:             -552
Number of customers came and leave:      13
Rate of customer happiness:              0.993804
18:01 (IIII) (T) (MMOOO OOOOO OOOOO OOOOO) (0,0,0);
18:02 (SSII) (T) (HHMMO OOOOO OOOOO OOOOO) (0,0,0);
18:03 (TTSS) (I) (HHHHO OOOOO OOOOO OOOOO) (0,0,0);
18:04 (IITT) (I) (HHHHO OOOOO OOOOO OOOOO) (2,0,0);
18:05 (IIII) (T) (HHHHM MOOOO OOOOO OOOOO) (4,0,0);
18:06 (SSII) (T) (HHHHH HMMOO OOOOO OOOOO) (4,0,0);
18:07 (TTSS) (T) (HHHHH HHHMM OOOOO OOOOO) (4,0,0);
18:08 (SSTT) (T) (HHHHH HHHHH MOOOO OOOOO) (6,0,0);
18:09 (TTSI) (T) (HHHHH HHHHH HMOOO OOOOO) (8,0,0);
18:10 (SITI) (T) (HHHHH HHHHH HHMOO OOOOO) (10,0,0);
18:11 (TSII) (T) (HHHHH HHHHH HHHMM OOOOO) (11,0,0);
18:12 (STSI) (I) (HHHHH HHHHH HHHHH OOOOO) (12,0,0);
18:13 (TITI) (I) (HHHHH HHHHH HHHHH OOOOO) (13,0,0);
18:14 (IIII) (I) (HHHHH HHHHH HHHHH OOOOO) (15,0,0);
18:15 (IIII) (T) (HHHHH HHHHH HHHHH MMOOO) (13,0,0);
18:16 (SSII) (S) (EEEEH HHHHH HHHHH HHOOO) (11,0,0);
18:17 (TTII) (T) (EEEEH HHHHH HHHHH HHMOO) (11,0,0);
18:18 (SIII) (T) (EEEEH HHHHH HHHHH HHHMO) (13,0,0);
18:19 (TSII) (T) (EEEEH HHHHH HHHHH HHHHM) (11,0,0);
18:20 (STII) (S) (EEEEE EEEHH HHHHH HHHHH) (10,0,0);
18:21 (TIII) (S) (EEEEE EEEEE HHHHH HHHHH) (9,0,0);
18:22 (IIII) (S) (EEEEE EEEEE EHHHH HHHHH) (9,0,0);
18:23 (IIII) (S) (EEEEE EEEEE EEHHH HHHHH) (8,0,0);
18:24 (IIII) (S) (EEEEE EEEEE EEEHH HHHHH) (7,0,0);
18:25 (IIII) (S) (EEEEE EEEEE EEEEE HHHHH) (5,0,0);
18:26 (IIII) (I) (EEEEE EEEEE EEEEE HHHHH) (5,0,0);
18:27 (IIII) (I) (EEEEE EEEEE EEEEE HHHHH) (5,0,0);
18:28 (IIII) (I) (EEEEE EEEEE EEEEE HHHHH) (5,0,0);
18:29 (IIII) (S) (EEEEE EEEEE EEEEE EEHHH) (3,0,0);
18:30 (IIII) (I) (EEOEE EEEEE EEEEE EEHHH) (3,0,0);
```

```
18:31 (IIII) (T) (EEMEE EEEEE EEEEE EEHHH) (2,0,0);
18:32 (SIII) (S) (EOHEE EEEEE EEEEE EEEEH) (1,0,0);
18:33 (TIII) (T) (EMHEE EEEEE EEEEE EEEEH) (0,0,0);
18:34 (SIII) (S) (EHHOE EEEEE EEEEE EEEEE) (1,0,0);
18:35 (TIII) (T) (EHHME EEEEE EEEEE EEEEE) (1,0,0);
18:36 (SIII) (I) (EHHHE EEEEE EEEEE EEEEE) (2,0,0);
18:37 (TIII) (I) (OHHHE EEEEE EEEEE EEEEE) (2,0,0);
18:38 (IIII) (T) (MHHHE EOEEE EEEEE EEEEE) (3,0,0);
18:39 (SIII) (T) (HHHHE EMEEE EEEEE EEEEE) (3,0,0);
18:40 (TSII) (I) (HHHHE EHEEE EEEEE EEEEE) (3,0,0);
18:41 (ITII) (I) (HHHHE EHEEE EEEEE EEEEE) (4,0,0);
18:42 (IIII) (I) (HHHHE EHEEE EEEOE EEEEE) (5,0,0);
18:43 (IIII) (T) (HHHHE EHOEE EEEME EEEEE) (5,0,0);
18:44 (SIII) (I) (HHHHE EHOEE EEEHE EEEEE) (5,0,0);
18:45 (TIII) (T) (HHHHE EHMEE EEEHE OEEEE) (4,0,0);
18:46 (SIII) (T) (HHHHE EHHEE EEEHE MEEEE) (5,0,0);
18:47 (TSII) (S) (HEEHE EHHEE EOEHE HEEOE) (4,0,0);
18:48 (ITII) (I) (HEEHE EHHEE EOEHE HEEOE) (5,0,0);
18:49 (IIII) (T) (HEEHE OHHOE EMEHE HEEOE) (5,0,0);
18:50 (SIII) (T) (HEEHO MHHME EHEHE HEEOE) (5,0,0);
18:51 (TSSI) (S) (HEEEO HHHHE EHEHE HEEOE) (5,0,0);
18:52 (ITTI) (S) (EEEEO HHHHE EHEHE HEEOO) (5,0,0);
18:53 (IIII) (S) (EEEEO HEHHE EHEHE HEEOO) (6,0,0);
18:54 (IIII) (I) (EEEEO HEHHO EHEHE HEEOO) (6,0,0);
18:55 (IIII) (T) (EEEEM HEHHM EHEHE HEEOO) (6,0,0);
18:56 (SSII) (I) (EEEEH HEHHH EHEHE HEEOO) (6,0,0);
18:57 (TTII) (T) (EEEEH HEHHH EHEHE HEEMM) (5,0,0);
18:58 (SSII) (S) (EEEEH HEHHH EHOEE HEEHH) (7,0,0);
18:59 (TTII) (S) (EEEEH HEEHH EHOEE HEEHH) (6,0,0);
19:00 (IIII) (T) (EEEEH HEEHH EHMEE HEEHH) (7,0,0);
19:01 (SIII) (S) (EEEEH HEEHH EHHEE EEEHH) (7,0,0);
19:02 (TIII) (I) (EEEEH HEEHH OHHEE EEEHH) (7,0,0);
19:03 (IIII) (S) (EEEEH HEEHH OEHEE EEEHH) (7,0,0);
19:04 (IIII) (T) (EEEEH HEEHH MEHEE EEEHH) (5,0,0);
19:05 (SIII) (S) (EEEEH EEEEH HEHEE EEEHH) (5,0,0);
19:06 (TIII) (I) (EEEEH EEEEH HEHEO EEEHH) (5,0,0);
19:07 (IIII) (I) (EEEEH EEEEH HEHEO EEEHH) (6,0,0);
19:08 (IIII) (T) (EEEEH EEEEH HEHEM EEEHH) (6,0,0);
19:09 (SIII) (S) (EEEEE EEEEE HEHEH EOEHH) (4,0,0);
19:10 (TIII) (T) (EEEEE EEEEE HEHEH EMEHH) (4,0,0);
19:11 (SIII) (S) (EEEEE EEEEE HEHEH EHEEE) (3,0,0);
19:12 (TIII) (I) (EEEEE EEEEE HEHEH EHOEE) (3,0,0);
19:13 (IIII) (T) (EEEEE EEEEE HEHEH EHMEE) (4,0,0);
19:14 (SIII) (S) (EOEOE EEEEE HEEOH EHHEE) (3,0,0);
19:15 (TIII) (T) (EMEOE EEEEE HEEOH EHHEE) (3,0,0);
19:16 (SIII) (T) (EHOME EOEEE HEEOH EHHEE) (4,0,0);
19:17 (TSII) (T) (EHMHE EOEEE HEEOH EHHEE) (4,0,0);
19:18 (STII) (S) (EHHHE EOEEE EEEOH EHHEE) (4,0,0);
19:19 (TIII) (I) (EHHHE EOEEE EEEOH EHHEE) (5,0,0);
19:20 (IIII) (T) (EHHHE EMEEE EEEMH EHHEE) (6,0,0);
19:21 (SSII) (I) (OHHHE EHEEE EOEHH EHHEE) (6,0,0);
19:22 (TTII) (T) (MHHHE EHEEE EOEHH EHHEE) (5,0,0);
19:23 (SIII) (T) (HHHHE EHEEE EMEHH EHHOE) (7,0,0);
19:24 (TSII) (T) (HHHHE EHEEE EHEHA EHHME) (6,0,0);
19:25 (STII) (S) (HHHHE EHEEE EHEHE EEHHE) (7,0,0);
19:26 (TIII) (I) (HHHHE EHEEE EHEHE EEHHE) (8,0,0);
```

```
19:27 (IIII) (S) (HHHHE EHEEE EHEHE EEEHE) (8,0,0);
19:28 (IIII) (I) (HHHHE EHEEE EHEHE EEEHE) (8,0,0);
19:29 (IIII) (S) (HEHHE EHOEE EHEHE EEEHE) (7,0,0);
19:30 (IIII) (S) (HEHEE EHOEE EHEHE EEEHE) (6,0,0);
19:31 (IIII) (T) (HEHEE OHMEE EHEHE EEEHE) (5,0,0);
19:32 (SIII) (T) (HEHEE MHHEE EHEHE EEEHE) (5,0,0);
19:33 (TSII) (S) (HEEEE HHHEE EHEHE EEEHO) (5,0,0);
19:34 (ITII) (T) (HEEEE HHHEE EHEHE EEEHM) (4,0,0);
19:35 (SIII) (S) (HEEEE HEHEE EHEEE EEEHH) (5,0,0);
19:36 (TIII) (S) (EEEEE HEHEE EHEEE EEEHH) (4,0,0);
19:37 (IIII) (S) (EEEEE HEHEE OEEEE OEEHH) (4,0,0);
19:38 (IIII) (T) (EEEEE HEHEE MEEEE MEEHH) (3,0,0);
19:39 (SSII) (S) (EEEEE HEHEO HEEEE HOEEH) (3,0,0);
19:40 (TTII) (T) (EEEEE HEHEM HEEEE HMEEH) (3,0,0);
19:41 (SSII) (I) (EEEEE HEHEH HEEEE HHEEH) (5,0,0);
19:42 (TTII) (I) (EEEEO HEHEH HEEEE HHEEH) (5,0,0);
19:43 (IIII) (I) (EEEEO HEHOH HEEEE HHEEH) (7,0,0);
19:44 (IIII) (T) (EEEEM HEHOH HEEEE HHEEH) (7,0,0);
19:45 (SIII) (S) (EOEEH HEEOH HEOEE HHEEH) (6,0,0);
19:46 (TIII) (S) (EOEEH EEEOH HEOEE HHEEH) (5,0,0);
19:47 (IIII) (I) (EOEEH EEEOH HEOEE HHEEH) (6,0,0);
19:48 (IIII) (S) (EOEEH EEEOH HEOEE HHEEE) (5,0,0);
19:49 (IIII) (I) (EOEEH EEEOH HEOEE HHEEE) (5,0,0);
19:50 (IIII) (I) (EOEEH EEEOH HEOEO HHEEE) (5,0,0);
19:51 (IIII) (I) (EOEEH EEEOH HEOEO HHEEE) (5,0,0);
19:52 (IIII) (S) (EOEEH EEEOH EEOOO EHEEE) (3,0,0);
19:53 (IIII) (I) (EOEEH EEEOH EEOOO EHEOE) (3,0,0);
19:54 (IIII) (S) (EOOEH EEEOE EEOOO EEEOE) (1,0,0);
19:55 (IIII) (I) (EOOOH EEEOE EEOOO EEEOE) (1,0,0);
19:56 (IIII) (I) (EOOOH EEEOE EEOOO EEEOE) (1,0,0);
19:57 (IIII) (I) (EOOOH EEEOE EEOOO EEEOE) (1,0,0);
19:58 (IIII) (S) (EOOOE EOEOE EEOOO EEEOE) (0,0,0);
19:59 (IIII) (I) (EOOOE EOEOE EEOOO EEEOE) (0,0,0);
20:00 (IIII) (I) (EOOOE EOEOE EEOOO EEEOE) (0,0,0);
```

Appendix: Code Package

.1 Directories

./code: prob1 prob2 prob3

./code/prob1: prob1_i_ijk_static.c prob1_j_ijk_static.c prob1_j_jik_static.c prob1_k_ijk_static.c prob1_k_kji_static.c

./code/prob2: prob2_dynamic.c prob2_guided.c prob2_static.c

./code/prob3: pizza_new.cpp

.2 About Problem 1

`prob1_$v_$seq_$schedule.c`: $v is denoted for the identifier of loop which is paralleled; $seq is the order of loop, $schedule contains static, dynamic, guided.