A TEACHING SCHEMA FOR MULTI-CORE PROGRAMMING

Yun Gao[1] and Xuejie Zhang[1,2]
[1] *High Performance Computing Center, Yunnan University, No.2 North Cuihu Road, Kunming, China*
[2] *School of Information Science and Engineering, Yunnan University, No.2 North Cuihu Road, Kunming, China*

Keywords: Multi-core programming, Teaching schema, Positivity, Instance.

Abstract: As multi-core processors become more and more widespread, multi-core programming attracts attention of the entire society. On this background, the multi-core programming curricula have been opened to the senior undergraduate students in many universities. At present, the teaching schemas about multi-core programming are scarcely mentioned. In this paper, we present a teaching schema on the specific characteristics of multi-core programming curriculum, combining our teaching practice. The schema pays attention to arousing students' enthusiasm and positivity, showing instance demos and improving students' consciousnesses of optimizing multi-core programs. During our practical teaching, the schema has obtained a better teaching effect.

1 INTRODUCTION

With the rapid development of computer hardware technologies and the increase of modern society requirement to the information processing speed, multi-core processor emerges as the times requires. It is named multi-core processor that two or many processing cores are integrated on a processor chip, abbreviated as CMP. But the improved hardware performance can not be transformed into high software execution performance by the traditional programming (Shameem and Jason, 2006) (LIU and LIANG, 2007) (HE and WANG, 2007). The multi-core programming on CMP has aroused general concern of the entire society. On the background, the correlative curricula have already been opened to the senior undergraduate students in many universities. However, which teaching schema should be taken to effectively improve teaching efficiency and impel students to understand the kernel of multi-core programming?

Multi-core programming should be in the programming category. In the recent years, many teaching methods of traditional programming based on one or more single-core processors have been proposed, such as task driving method (REN and LI, 2008), practice teaching method (SUN, 2008), and so on (Christopher Haynes, 2009) (Christoph W. Kessler, 2006). These methods can be used by multi-core teaching for reference. Multi-core programming on CMP is different from traditional programming in nature, so we should take special teaching schema. At present, the teaching schemas about multi-core programming are scarcely mentioned.

In the following sections, by analyzing the specific characteristics of multi-core programming curriculum, we present a teaching schema on its specific characteristics and our teaching practice, aiming at the senior undergrduate students, and then show its practical effects. The schema pays attention to arousing students' enthusiasm and positivity, showing instance demos and improving students' consciousnesses of optimizing multi-core programs.

2 SPECIFIC CHARACTERISTICS

It is useful and necessary to analyze the curricular specific characteristics for selecting teaching schema. The multi-core programming curriculum has mainly several following characteristics distinguished from other programming.

2.1 Stronger Previous Foundations

It is well known that traditional programming in C/C++ or Java hardly needs any curricular basis. But, for designing a reasonable multi-core multi-thread parallel program, it is helpful and necessary to know the hardware frame of CMP, the scheduling mechanism of operating system (OS), the basic

Gao Y. and Zhang X. (2010).
A TEACHING SCHEMA FOR MULTI-CORE PROGRAMMING.
In *Proceedings of the 2nd International Conference on Computer Supported Education*, pages 195-198
DOI: 10.5220/0002857401950198

theory concerning design and analysis of algorithms and to master a programming language. Consequently, there are several necessary previous curricular foundations for multi-core programming curriculum, including Computer Composition Principle, Operating System, Compiler Principle, Design and Analysis of Algorithms, C language, and so on. Multi-core programming curriculum is usually offered to senior undergraduate or postgraduate.

2.2 Non-figurative Theory

The theories of multi-core programming are more. Multi-core program is mainly realized by a multi-thread way on CMP. When realizing multi-core programs, there are four questions or challenges faced by parallel programming designers, and they are multi-thread synchronization, multi-thread communication, load balance and programming scalability, which are also the kernel of multi-core programming. (Shameem and Jason, 2006) Nevertheless, students often mention that it is bald and abstract to understand its theories and that it is especially difficult to apply them during factual programming. By this phenomenon, we must arouse their enthusiasm and positivity in an example style, which will be explained in the next section.

2.3 Needing Programming Practices

The theory must be applied finally in the practice, and the theory is improved only by the practice. According to this principle, for understanding the nonfigurative theory of multi-core programming concretely, it is necessary to adopt a practice way. During the actual teaching, explanation in an instance way can promote students' intellect, and programming practice by students themselves can enhance students' operating ability. Ultimately, students can master the kernel of multi-core programming in a high effective style.

2.4 Support of Hardware and Software

Multi-core programming must be developed based on multi-core hardware and corresponding software environment. Therefore, the support of experimental environments is necessary to multi-core programming curriculum. But multi-core computer is not popularized to each senior undergraduate student. So, many universities have built multi-core labs for further promoting the laboratorial conditions of multi-core teaching at present.

3 A TEACHING SCHEMA

In this section, we present a teaching schema of multi-core programming curriculum on its specific characteristics and our teaching practice, aiming at the senior undergraduate students having learned previous curricula. Our expectant teaching purpose is that students can be interested in multi-core programming, master its kernel, and establish multi-core programming foundation for their future work or study. Our schema detailed in the following pays attention to arousing students' enthusiasm and positivity, showing instance demos and improving students' consciousnesses of optimizing multi-core programs.

3.1 Arousing Students' Enthusiasm and Positivity

To whatever curriculum, including multi-core programming, students' study enthusiasm and positivity is closely relative to the final study efficiency. For multi-core programming curriculum, we must firstly make students know its necessity and importance before they study the concrete content. In our schema, it is reached by answering two following questions.

- Is it to study multi-core programming?
- Can the efficiency of application be improved by multi-core programming?

3.1.1 Studying Multi-core Programming is Necessary

Many questions can certainly be preferably solved by single-thread programs. But some questions do exist and must be solved by multi-thread parallel programs. Students can cognize that multi-thread parallel programming is necessary under some circumstances by correlative examples. The following is an example playing two pictures in a mosaic style synchronously in a dialog window.

When realizing it with a single-thread program, there are two visible disadvantages. One is that user messages can not be responded in good time, and the other is that two pictures can not be played synchronously (see Figure 1). That is to say, the requirement can not be reached by single-thread program. If realizing it with a multi-thread program, two foregoing disadvantages can be avoided and the application requirement can be achieved perfectly. Two picture can be played in a mosaic style in left and right synchronously (see Figure 2).

Figure 1: When left played, right must wait until left is finished.

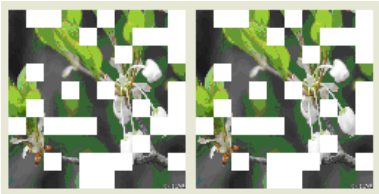

Figure 2: Picture is played synchronously in left and right.

3.1.2 Efficiency of Application can be Improved

Firstly, the executing time of multi-core program is less than that of single-thread for a same multi-core platform and a same question. Secondly, on multi-core platform, genuine parallel can be achieved. So the executable efficiency of application can be improved in a genuine sense. The following is an example calculating the pi value in an integral style (Michael J. Quinn, 2003). The hardware configuration of platform is dual-core CPU, Genuine Intel(R) CPU T2080 1.73GHz.

The single-thread code and its executing result are as follows.

```
num_steps = 1000000;
step = 1.0/(double)num_steps;
start = clock();
for (i=0; i<num_steps; i++) {
    x = (i + 0.5)*step;
    sum += 4.0/(1.0+ x*x);
}
pi = sum*step;
stop = clock();

The pi value is 3.141592653590.
The executing time is 0.109 seconds.
```

The dual-thread code and its executing result are as follows with OpenMP.

```
num_steps = 1000000;
step = 1.0/(double)num_steps;
start = clock();
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for
reduction(+:sum) private(x)
for (i=0; i<num_steps; i++) {
    x = (i + 0.5)*step;
    sum += 4.0/(1.0+ x*x);
}
pi = sum*step;
stop = clock();

The pi value is 3.141592653590.
The executing time is 0.063 seconds.
```

From the results, multi-thread program takes less time and obtains a higher efficiency than single-thread program, and speedup ratio is approximately 1.73, that is to say, the performance of this program is increased by 73%.

3.2 Showing Instance Demos

Multi-core programming can be realized on multiform OS and in multiform programming languages. In the limited curriculum hour, it is almost impossible to explain all involved grammars or functions. It is also useless all the same if these grammars or functions can not be used in factual programming. Therefore, in our schema, we only explain some representative grammars or functions to make students master the kernel of multi-core programming. For the grammars or functions not taught, students can completely master them in a self-study way.

As an example, barrier is one of the effective means to realize multi-thread synchronization, and a thread meeting barrier can not continue until all threads in the same parallel area reach the same barrier. (Shameem and Jason, 2006) If this is explained only by language, it is difficult for students to use barrier. On the contrary, instance demos can help students not only to master its usage but also to sense the synchronization mechanism. The following is an example calculating variable *y* and *z* synchronously.

```
#pragma omp parallel shared(x,y,z)
num_threads(2){
    int tid = omp_get_thread_num();
    if (tid == 0) {
        y = fn70(tid);
    }
    else {
        z = fn80(tid);
    }
    #pragma omp barrier
    #pragma omp for
    for ( k=0; k<100; k++) {
        x[k] = y+z+fn10(k)+fn20(k);
    }
}
```

3.3 Improving Students' Consciousness of Optimizing Multi-core Programs

Multi-core program is an effective way to exhibit the multi-core hardware performance. But it must be mentioned that the efficiency of multi-core program on CMP can be worse than serial program if multi-core program is not designed in a better thinking. Therefore, in our schema, we think much of cultivating students' consciousness to optimize realized multi-core programs, a very important habit to a software designer. In our factual teaching, we can compare several different realizations for a same problem.

As an example, for there is no relativity between the data of the first "for" circle block and that of the second "section" code block, one thread can continue to run directly witout waiting. Under the circumstance, we can import a "nowait" clause to remove the hidden barrier (Michael J. Quinn, 2003), and so the waiting time for synchronization is saved.

```
#pragma omp parallel {
    #pragma omp for nowait
    for (int k=0;k<m;k++){
        fn1(k);
        fn2(k);
    }
    #pragma omp sections private(y,z)
    {
        #pragma omp section {
            y = sectionA(x);
            fn7(y);
        }
        #pragma omp section {
            z = sectionB(x);
            fn8(z);
        }
    }
}
```

4 CONCLUSIONS

In this paper, we present a teaching schema of multi-core programming curriculum on its specific characteristics and our teaching practice, aiming at the senior undergraduate students. The schema pays attention to arousing students' enthusiasm and positivity, showing instance demos and improving students' consciousnesses of optimizing multi-core programs. During our practical teaching with the schema, it is proved that students' positivity can be aroused, students' practical ability can be enhanced and students' consciousness to optimize programs can be improved.

As multi-core programming is a newly arisen technique, both its theory and its teaching schema need further to be perfected all the same. In our future work, we will further explore more effective teaching schemas about multi-core programming by our teaching practices.

ACKNOWLEDGEMENTS

The authors thank Intel-Yunnan University Multi-core Technology Lab and High Performance Computing Center of Yunnan University for experimental support. This work was also supported by the Project of Building up Characteristic Disciplines under Grant No. TS11135 from Ministry of Education of China and by Innovation Group Project from Yunnan University of China.

REFERENCES

Shameem Akhter and Jason Roberts, 2006. *Multi-core Programming Increasing Performance through Software Multi-threading*, Intel Press.

Liu Jin-guang and Liang Man-gui, 2007. *The Development and the Software System Architecture of Multi-core Multi-threading Processor*. Microprocessors.

He Jun and Wang Biao, 2007. *Research on Architecture Design of Multi-core Processor*. Computer Engineering.

Ren Jing-ying, Li Ying, 2008. *Application of Duty Actuation Teaching Method in C++ Language Programming*, Combining Curricula with Infomation Technologies.

Sun Geng, 2008. *Project Teaching Method in C++ Programming*, Computer Education.

Michael J. Quinn, 2003. *Parallel Programming in C with MPI and OpenMP*, McGraw Hill Higher Education.

Christopher Haynes, 2009. *Experience with an Analytic Approach to Teaching Programming Languages,* Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education.

Christoph W. Kessler, 2006. *Teaching Parallel Programming Early,* Proceedings of Workshop on Developing Computer Science Education.