International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.7 (2014), pp.131-140
http://dx.doi.org/10.14257/ijmue.2014.9.7.12

Research on Multi-Core PC Parallel Computation Based on OpenMP

Lan Xiaowen

School of Information Engineering, Inner Mongolia University of Science and Technology, Baotou, China
lanxiaowenning@163.com

Abstract

Along with quad-core PC coming into the market and invention of 80-core processor in the laboratory, Software development will go through a foundational innovation due to multi-core technology. The technique of parallel programming on multi-core computers is explored in this thesis. First, the OpenMP standard which is an application programming interface (API) on parallel programming model of shared-memory is introduced and an overview of a set of compiler directives and a library of support functions are given. The OpenMP programs requires an OpenMP-compatible compiler and thread-safe libraries, therefore, both Intel C++ compiler 9.1 and Microsoft Visual Studio 2005 are perfect choices. Then, two-dimensional discrete fast Fourier transform (FFT) is studied by focusing on parallel program design, realization and optimization technology. Finally, the thesis predicts that high performance parallel computing software component library must be a perfect exploitation field in the further future.

Keywords: *multi-core computer; parallel computation; multi threading; OpenMP*

1. Introduction

Under the existing technology, it is difficult to innovate the traditional methods to improve CPU performance--such as increasing clock speed and instruction throughput—due to the limits by Moore's law. In recent years, the improvement of new chips performance will be primarily from the ultra-thread, multi-core and cache 3-pronged approaches, the most impressive of which is the multi-core technology. The major processor manufacturers, such as Intel and AMD, are improving processor performance by consolidating multiple processing engines instead of increasing the frequency shift. Multi-core processor has become a future processor technology development direction. Multiple cores lead to the foundational changes of software research and development. It is particularly meaningful to those application soft wares which are general applications oriented and run in the PC and low-end server. Developers need to add threads into Codes to make use of the multiple cores provided by the system and spread the Codes which are sensitive to performance to the cores, but at the same time, make sure that the codes have a good scalability and the same code must be able to run well both in single-core computer, dual-core, 4-core and a high-level computers.

Traditional programs are essentially written for the sequence processor, and most of them cannot directly access speed in multi-processor. One way of addressing that problem is to use multi-processor compiler to automatically convert the sequence processor into a parallel program. The auto-parallelization of multi-processor compiler is able to address some problems, but it is still not satisfactory. For example, it takes Intel Compiler hours to compile and the acceleration rate of programs is only 10% to 30 % [1]. Another solution to the problem is to manually override the library. For a long time, the computer industry has accumulated a lot of library procedures, particularly in scientific computing, classic algorithm has been stored into library procedure. If all the programs in the library are suitable for the override of parallel computing, the user can directly invoke the parallel libraries when writing an application program, thus to accelerate the running of it.

Current selectable multi-threaded development tools are win32 thread library, pThread library and OpenMP. Win32 thread library runs on Win NT and Win 9X platforms and it owns a perfect and complex library which is more improved, but it demands more of programmers; pThread library is the next most commonly used Linux multi-threading support library which can be easily transplanted but is more difficult to use; OpenMP provides parallel computing support to the shared address space parallel computers, with features of simple usage. Currently, Intel strongly recommends multi-threaded development tool OpenMP and provides support for it in the Intel C++ Compiler 9.1 and Microsoft Visual Studio 2005.

2. OpenMP Analysis

OpenMP is an application programming interface designed for parallel programming in the shared memory multiprocessor. It is made up by a small compiler command set, including a set of compilation guidance statements and a supporting function library. OpenMP works with the connection with the standard Fortran, C and C++, providing effective supports for synchronizing shared variables and distributing load reasonably, *etc*. It is simple and develops fast [2, 3].

OpenMP is an industry standard for portable multiple-thread application development, with very high efficiency in the fine-grained (loop level) and coarse-grained (function level) thread technology. For converting serial program into parallel application, OpenMP instruction is a tool which is powerful and easy to use. It has the potential that can make application program get substantial performance improvement for parallel execution in a symmetric multiprocessor or multi-core system. OpenMP will automatically be loop-threaded and improve application performance on multi processor systems. Users do not have to deal with iterative division, data sharing, thread scheduling and synchronization of the low-level details of [2]. Intel C++ compiler, Visual 9.1 C++ 8 and Microsoft Visual Studio2005 all support OpenMP 2.5. The paper mainly introduces the usage of OpenMP Microsoft in Visual Studio 2005.

2.1. OpenMP Model

Shared storage model is a general centralized multi-processing abstract [1, 2, 4]. Its base is a series of processors which can access the same shared memory. All of the processors can access the memory at the same location, which enables them to interact and synchronize through shared variables. OpenMP adopts standard parallel in the shared storage-fork-join. At the beginning of program execution, only the master thread exists. The main thread performs the serial part, and other parallel parts are performed by other derived threads. When the serial part of the program is re-executed, the threads will be terminated. As shown in Figure 1.

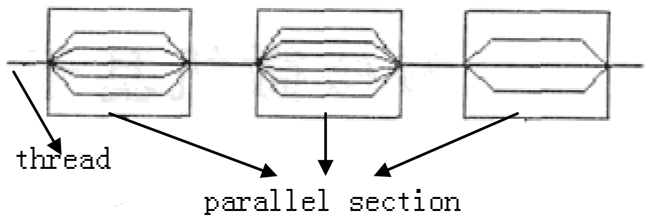

Figure 1. OpenMP Execution Model

2.2. OpenMP Directives

Industry-standard OpenMP is an extension of C language, the purpose which is to support concurrent programming. Writing OpenMP program is similar to write C language [4, 5], but in the writing of the former, compiler directives of OpenMP is added into C program. These compiler directives describe the ways to execute the program in parallel. The C program added with the OpenMP directives can be supported by any OpenMP compiler and executed on the hardware of different platforms. The OpenMP compiler command starts with #pragma, followed by omp, the name and the optional clause, and ends with a new line. Some clauses may appear in different orders, but they need to be defined respectively. Some of the commands will affect the whole structure blocks, and the so-called construct is the compiler command and the subsequent blocks. In C/C++, the OpenMP instruction format is: #pragma omp instruction [clause [clause]......].

OpenMP compilation guidance includes parallel domain structure, shared task structure, shared tasks parallel structure and synchronization structure.[6] The parallel code of domain structure is executed by all the threads, usually using the format: #pragma omp parallel [clause [clause] ......]. Commonly used clauses are first private, if, lastprivate, private, reduction etc. Shared task structure allocates its codes to the members of thread group to execute and it can be divided into parallel for loops, parallel sections and serial implementation. Parallel for loop format is #pragma omp for [clause [[,] clause] ...], generally used before for loop and the loop is assigned to multiple threads to execute in parallel. Sections compilation guidance statement specifies the internal code to be allocated into each thread in the thread group. Different section is executed by different threads,The allocation by the for statement is performed automatically by the system. As long as there is no time gap between cycles, the allocation is very uniform. Dividing the thread by using section is a manual classification of thread parallelism, which ultimately depends on programmers [7]. Single compilation guidance statements are used when internal part of the code is not convenient to parallel implementation. the use of single compiler guidance specifies this part of the code to be executed by only one thread in this thread group, and the format is the #pragma omp single[clause this clause,]]...]. Others threads in the group will wait for the end of the code block except for those using nowait clause.Combined parallel shared tasked divided into the parallel for and parallel section compilation guidance statement. The format of the former is #pragma omp parallel for [clause format...], showing a parallel domain contains an independent for statement; the format of the latter is #pragma omp parallel sections[clause...], showing a parallel region contains a single sections statement. In general, because the use of the parallel command means that more than one thread is required, so it needs to first establish a thread group before using a command. Compiler directive in synchronous structure includes barrier, atomic, master, ordered, thread-private, *etc.*, [1].

OpenMP compilation guidance can optionally contain a clause according to the need. Without other constraints, the clause can be out-of-order and selected arbitrarily [2].

"#pragma omp parallel for the clause ..." is the most frequently used statement in compilation, which can be used together with a firstprivate, if, lastprivate, private, reduction, schedule, *etc*. The firstprivate clause specifies that each thread has its own variable private copies, and the variable is to inherit an initial value from the master thread; lastprivate clause specifies the value of the private variable of the thread to be copied back to the corresponding variable in the main thread after the parallel processing; private clause specifies each thread has its own variable private copy; reduction clause specifies one or more variables are private, and the variable performs the specified operation after the parallel processing; schedule clause specifies how to schedule for iterations of the loop.

2.3. OpenMP Library Functions

OpenMP uses the library function -<omp.h>, including the execution environment of library function, lock function and timing function [2]. Table 1 lists some of the most commonly used functions.

Table 1. Commonly used OpenMP Function

function prototype	Instructions
int omp_get_num_procs;	numbers of physical processors in the multi-processor which return to run the thread
int omp_get_num_threads;	the numbers of active threads which return to current parallel section
int omp_get_thread_num;	returning thread number
void omp_set_num_threads (int num_threads);	set threads in parallel code execution thread number
void omp_init_lock(omp_lock_t *lock);	initiate a simple lock
void omp_set_lock(omp_lock_t *lock);	lock operation
void omp_unset_lock(omp_lock_t *lock);	Unlock operation, paired with matching the omp_set_lock function
void omp_destroy_lock(omp_lock_t *lock);	the matching operation function of omp_init_lock, close a lock
double omp_get_wtick( );	the seconds of returning to successive clock marks
double omp_get_wtime( );	return to a certain time when starting the clock time

3. Parallel Algorithm of FFT

3.1. Two-dimensional Discrete Fast Fourier Transform (FFT)

In digital image processing, two-dimensional Fourier transform (FFT) is often used in spectral analysis. Two-dimensional Fourier transform is derived on the basis of the one-dimensional Fourier transform. That is, two-dimensional Fourier transform operation is broken into horizontal and vertical direction of one-dimensional Fourier transform operation. One-dimensional discrete Fourier transform is:

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-j2\pi ux/M}$$

based on this, an M*N image of two-dimensional discrete Fourier transform

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)}$$

is given. Specific algorithms are as follows [8]:
(1) to get the first address of the data area of the original image and the image height and width; to calculate the image width and height and level, the number of iterations in the; vertical direction in Fourier transform;
(2) to read the values of the data area according to the order and store them into the opening up complex store;
(3) to call one-dimensional Fourier transform function to transform in vertical direction;
(4) to translate transform results to switch transform results in the vertical direction back to time domain store ;
(5) to call one-dimensional Fourier transform function to transform in horizontal direction;
(6) to convert the results into a displayable image, move the coordinate origin to the center of the image so that the image can be displayed throughout the cycle spectrum.

Two dimensional discrete Fast Fourier transform is an important tool for digital signal processing, but large calculating and long-time operation to some extent limit its use. In order to resolve this contradiction, OpenMP is introduced to make full use of dual-core technology so as to achieve a quick calculation.

3.2. FFT Algorithm for Parallel Processing

Experiment platform is Dell Optiplex GX630 series dual-core desktop. Intel Lakeport-G i945G chipset, Intel Pentium d CPU 2.80GHz, 1 MB cache, front side bus 800MHz, HY Dual DDR2 SDRAM memory 1G. are adopted. Operating system is Microsoft Windows XP Professional 5.1.2600 (WinXP Retail). The compiler is Microsoft Visual Studio 2005. On this platform, OpenMP is used to improve two-dimensional discrete Fast Fourier transform and the experimental data is the average data obtained after the program runs more than once.

(1) serial versions

```
(2)  for(j=0；j< h；j++) // fast Fouriter transform is performed in vertical direction
        QFC(&t[w*j],  &f[w*j],  wp);
  for(j =0；j < h；j++)//convert transform results
        for(i =0；i < w；i++)
                t[j+ h*i] = f[i+ w*j]；
  for(j =0；j < w；j++) // fast Fourier transform is performed in horizontal direction
        QFC(&t[j*h],  &f[j*h],  hp)；
```

The first for cycle completes fast Fourier transform in vertical direction, QFC (&t[w*j],&f[w*j],WP) is a function to achieve fast Fourier transform. &t[w*j]&f[w*j] are the pointers pointing to the hour domain and frequency domain respectively. WP is the width (2 of whole number times party) to perform fast Fourier transform. The second for cycle converts transform results, switching transform results of vertical direction back to the hour domain storage district. And the last for cycle completes fast Fourier transformation in horizontal direction, calling QFC function.

(2) parallel versions

Microsoft Visual Studio 2005 supports OpenMP through a compile switch option. In the "C/C++" language page of "configuration properties" in the dialog box, the Option can be changed to "Yes/(OpenMP)"in order to support OpenMP. In two-dimensional discrete Fast Fourier transform program C serial version, one dimensional discrete Fast Fourier transform performs FFT transform mainly in the way of column (or line). There does not exist datarelated limits for cycle, and every cycle is basically isolated, so the latter cycle does not rely on the former one. That is, for cycle meets parallel implementation requirements, so on double nuclear platform OpenMP instructions can be used to allocate one dimensional discrete Fast Fourier transform into to 2 CPU respectively for implementation. Take a fast Fourier transform in vertical direction for example, parallel computing main codes are as follows:

```
#pragma omp parallel private(i)

{
                                        int            id=omp_get_thread_num();
                                                                          i=id;
                                                                    while(i<W)
                                                                             {
                              QFC(&t[i*h]     ,      &f[i*h]     ,         hp);
                                                                        i=i+2;
                                                                             }
}
```

#pragma omp parallel specifies statement blocks to each CPU to implement parallelyl. For cycle in serial version is allocated to 2 CPU which respectively perform one-dimensional FFT transform. Parallel directive is used to create multiple threads for a piece of code and executed in parallel. Compared with the traditional, this thread function is equivalent to repeat calls for a thread entry function to create a thread and waits for the thread to finish execution. Results transformation uses the # pragma omp parallel for statement to allocate a for cycle to more than one thread. OpenMP automatically creates the optimal number of threads for the machine during the runtime. if a code is run in a single core processor, all codes will be run on a thread; if you run the same code on a dual-core processor, the code will be run on the two threads and the code will automatically adjust to the best on the current computer for maximum acceleration.

```
  #pragma omp parallel private (i)
  {
          int id=omp_get_thread_num();
          i=id;
          while(i<w)
          {
            QFC(&t[i*h],  &f[i*h],  hp);
            i=i+2;
          }
  }
```

3.3. Parallel Algorithms for Performance Optimization

To achieve maximum speedup on the dual-core platform, the key is to keep the load of threads balanced and the workload of each thread running roughly the same. By default, OpenMP splits the code into several equal pieces, and then runs on its own thread block to schedule cycle [7]. Usually the default block is large enough and when all threads run through the block, the loop body ends its running. OpenMP provides several scheduling options which add schedule (static[, chunksize]), schedule(dynamic[, chunksize]) or schedule(guided [, chunksize]) to OpenMP directives to control the way to schedule each thread. Figure 2 is to examine three scheduling policies by taking a parallel version of the two-dimensional fast Fourier transform of a discrete processing of 256 * 256 pixels picture.

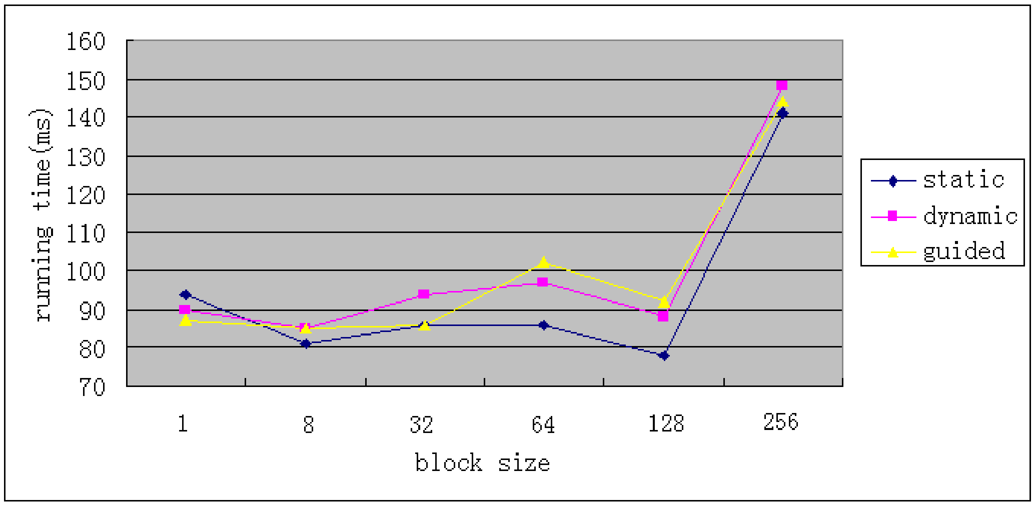

Figure 2. The Contrast between the Running Time of Parallel Version FFT

(1) Scheduling Policy

Static scheduling statically requires the distribution of work in the block to individual thread. While the thread completes the execution of a block, the same size in blocks in all threads is achieved by polling. While dynamic scheduling does not follow the order of static scheduling of the polls, as long as the thread needs more work, you can get the next iteration. Compilers can make a certain amount of optimization to the static part, but not dynamic ones. Guided scheduling is a hybrid strategy, each thread gets a larger block in the first run and the size will be gradually reduced in later running to specified limit value specified by chunkSize. The size of the block needs to be calculated dynamically which is similar to the dynamic scheduling.

(2) Block Size

The chunkSize parameter means that each scheduling differs from another. In all cases, returning to thread scheduling code costs some extra overhead in exchange for better flexibility. Each static running is a little bit different, the running time of smaller block is a little longer because running schedules code more often. The default scheduling method of OpenMP is always to get iteration count divided by the number of running threads. In this case, the chunkSize is 128. The comparison of a variety of scheduling in the code is always based on evaluating the situation. The difference between the block whose size is 1 and the block whose size is 8 is the static scheduling overhead (about 12 ms), or setting and start overhead of the iteration.

If the scheduling block size is 1, you must set up and start the loop 128 times (it runs a block on each thread). If the block size is 8, you must set up and start the loop 16 times. The extra overhead is 112 times in a time of 12 milliseconds, so using this scheduling method can only save 107 microseconds compared with running each loop. The difference is not clear while working on the 256*256 pixel, but slightly larger blocks still need to be used when working on large pictures in static scheduling. Dynamic scheduling is the most flexible, but performance loss is also the greatest when scheduling errors. Very small chunks should be avoided in order to balance the good code. In this example, a block size of 128 is obviously a good choice. But for those unbalanced codes, correct block size should be a trade-off between running fewer blocks and achieving a better balance between threads. Guided scheduling runs best and most flexible when using a smaller block as a limit. When you qualify for a larger block size, running time will be longer. The maximum size of Guided scheduling is one-second of the total circulation. As for balanced codes, it is the static situation in which the cycle is broken into several parts in one running.

4. Conclusion

OpenMP is a parallel computing library for computers of shared address and shared space. currently Microsoft and Borland have corresponding products to support OpenMP2.5 Programmers do not need to thread management code like CreateThread, etc.OpenMP provides a rich set of instructions, providing effective support for synchronization, shared variables, reasonable distribution of load. But OpenMP also has some unavoidable disadvantages: first, OpenMP implements multithreaded parallel mainly by precompiling directives (#pragma), so programs compiled on a single-core machine cannot reflect the advantage of multiple cores when running on multi-core machines;second, OpenMP requirements for compiler are relatively high, generally Microsoft visual studio 2005 or intel compiler. But in the long run, the advantage of OpenMP is obvious. An Intel technology officer once said "future processor development is the internal optimization and integration of multiple cores rather than to simply increase the processor frequency, multi-threaded software also will be the mainstream of software in the future." Taking advantage of multi-core processors for software performance optimization brings a great opportunity to China. Developing OpenMP software component libraries needs a lot of human resources, which can be easily solved in China.The future economic benefits of OpenMP library can be compared with Microsoft's operating system. The latter provides tools for users to use PC software support successfully while the former provides library functions for using multiple processors smoothly. The world is entering the age of multiple processors in which OpenMP library will become an indispensable tool for programmers.

Acknowledgments

This research was supported by Natural Science Foundation of General Projects of Inner Mongolia Autonomous Region, China (2013MS0915) and Innovation Foundation Projects of Inner Mongolia University of Science and Technology (2012NCL049).

References

[1] C. Guoliang, “Parallel Algorithms Practice [M]”, Beijing: Higher Education Press, (**2004**).
[2] M. J. Quinn, “Parallel programming in C with MPI and OpenMP [M]”, Beijing: Tsinghua University Press，2005.
[3] L. Jianxin，H. Changju and Z. Yudi, “OpenMP Task Scheduling Overhead and Load-balancing Analysis [J]”, Computer Engineering，vol. 8:(Sup) (**2006**)，pp. 58-60.
[4] A. Grama, “Introduction to parallel computing [M]”, Beijing: Mechanical Industry Press, (**2003**).
[5] I. Foster, “Designing and building parallel programs [M]”, Beijing: Mechanical Industry Press, (**2002**).
[6] Andrew, “Multithreading parallel and distributed programming [M]”, Beijing: Higher Education Press, (**2002**).
[7] V. Malyshkin, “Parallel computing technologies [C]”, 8th international conference, PaCT 2005, Krasnoyarsk, Russia, 2005, Berlin, New York, Springer, (**2005**).
[8] Y. Shuying, “VC++ Design of Image Processing [M]”, Beijing, Tsinghua University Press, (**2003**).
[9] J. Dongarra, “Parallel computing programming [M]”, Beijing: Electronic Industry Press, (**2005**).

[10] B. Wilkinson and M. Allen, "Techniques and applications using networked workstations and parallel computers [M]", Beijing: Mechanical Industry Press, (**2005**).
[11] Multisite co-allocation scheduling algorithms for parallel jobs in computing grid environments, Science in China Series F-Information Sciences, (**2006**), vol. 49, no. 6, pp. 906-926, DOI: 10.1007/s11432-006-2034-2
[12] W. Zhang and H. He, "Fault Tolerance for Conjugate Gradient Solver Based on FT-MPI", Studies in Informatics and Control, ISSN 1220-1766, (**2013**), vol. 22, no. 1, pp. 51-0, http://sic.ici.ro/sic2013_1/art06.php.

Author

Lan Xiaowen, he received the BS in computer science from Inner Mongolia University, China, in 2001 and received the MS in computer science from Chengdu University of Technology, China, in 2008. Currently, his research interests include embeded system and scheduling techniques and parallel algorithms for clusters, and also multi-core processors and numerical simulation.