多核微机基于OpenMP的并行计算

蔡佳佳,李名世,郑 锋
(厦门大学 计算机科学系,福建 厦门 361005)

摘 要:随着四核微机走向市场和八十核处理器在实验室研制成功,多核正引领软件研发发生基础性变化。开发人员需要在代码中添加线程来利用系统所提供的多个内核,从而提升PC应用软件的功能和性能。文中探讨在多核微机上进行并行计算的实现技术。介绍了共享存储系统并行编程接口OpenMP的模型、指令和库函数,以及Intel C++编译器9.1和Microsoft Visual Studio 2005等对OpenMP的支持;着重探讨了二维离散快速傅里叶变换并行算法的设计、实现与优化技术;展望了高性能并行计算软构件库的开发前景。

关键词:多核计算机;并行计算;多线程;OpenMP
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2007)10-0087-05

OpenMP-Based Parallel Computation on Multi-Core PC

CAI Jia-jia, LI Ming-shi, ZHENG Feng
(Dept. of Computer Science, Xiamen University, Xiamen 361005, China)

Abstract: Software development will receive a foundational innovation by reason of multi-core technology, along with quad-core PC coming into the market and invention of 80-core in the laboratory. Programmers add threads into codes to make full use of the new processor, that improves both function and performance of Internet applications. Originates from parallel programming on multi-core computers. First introduced OpenMP standard which is an application programming interface(API) on parallel programming model of shared-memory, and gave a quick overview about a set of compiler directives and a library of support functions. Using the OpenMP programs requires an OpenMP-compatible compiler and thread-safe libraries, both Intel C++ compiler 9.1 and Microsoft Visual Studio 2005 are perfect choice. Then studied two-dimensional discrete fast Fourier transform(FFT), focused on parallel program design, realization and optimization technology. Finally, thought that high performance parallel computing software component library must be a perfect exploitation field in the further future.

Key words: multi-core computer; parallel computation; multithreading; OpenMP

0 引 言

在现有工艺下,改善CPU性能的传统方法如提升时钟速度和指令吞吐量等在摩尔定律限制下已经难有大的进展。近年来新型芯片性能提升将主要从超线程、多核和缓存三个方面入手,其中最为瞩目的当属多核技术。Intel、AMD等主要的处理器厂商均将提高处理器性能的途径从提高主频转向整合多个处理引擎。多核处理器已成为未来处理器技术的发展方向。Co-Design Automation Inc.的创始人Simon Davidmann在2005年秋天对《EE Times》表示:"所有的芯片都将成为多处理器,我们必须学习如何给它们编程"。多核引领软件研发发生基础性变化,特别对那些面向一般应用、运行在PC和低端服务器上的应用软件更有非同一般的意义。开发人员需要在代码中添加线程来利用系统所提供的多个内核,并将对性能比较敏感的代码分隔在多个内核上,但同时又必须确保代码具有良好的可伸缩性,无论在单核计算机、双核计算机、四核计算机或者更高级别的计算机上,同样的代码都必须能够运行良好。

传统程序基本上是为顺序处理器书写的,大部分程序在多处理器上不能直接获得加速。解决这一问题的途径之一是使用多处理器编译器把顺序程序自动转换为并行程序。多处理器编译器的自动并行化功能能够解决一部分问题,但是依然不能令人满意,比如Intel的编译器要花费数小时进行编译,而程序的加速只是10%~30%[1]。解决这一问题的另一途径是手工重写程序库。长期以来,计算机界积累了大量的库程

收稿日期:2006-12-31
作者简介:蔡佳佳(1983-),女,福建厦门人,硕士研究生,研究方向为高性能并行计算;李名世,副教授,研究方向为网络、高性能计算及多媒体技术。

序,尤其是在科学计算领域,经典算法均已收入库程序。如果把程序库中所有程序用适合并行计算的方法重写,那么用户在写应用程序时就可以直接调用这些并行程序库,从而加速应用程序的运行。

目前可选择的多核多线程开发工具有 Win32 线程库、pThread 库以及 OpenMP。Win32 线程库运行于 Win NT 和 Win 9X 平台,拥有完善而复杂的函数库,目前比较成熟,但对编程人员有较高的要求;pThread 库是 Linux 下最常用的多线程支持库,具有方便移植的特点,但使用难度比较大;OpenMP 则针对共享地址空间的并行计算机提供并行计算支持,具有使用简单的特点。目前 Intel 极力推荐多线程开发工具 OpenMP,在 Intel C++ 编译器 9.1、Microsoft Visual Studio 2005 等都增加了对 OpenMP 的支持。

1 OpenMP 剖析

OpenMP 是一个为在共享存储的多处理机上编写并行程序而设计的应用编程接口,由一个小型的编译器命令集组成,包括一套编译制导语句和一个用来支持它的函数库。OpenMP 是通过与标准 Fortran, C 和 C++ 结合进行工作的,对于同步共享变量、合理分配负载等任务,都提供了有效的支持,具有简单通用、开发快速的特点[2,3]。

OpenMP 是可移植多线程应用程序开发的行业标准,在细粒度(循环级别)与粗粒度(函数级别)线程技术上具有很高的效率。对于将串行应用程序转换成并行应用程序,OpenMP 指令是一种容易使用且作用强大的工具,它具有使应用程序因为在对称多处理器或多核系统上并行执行而获得大幅性能提升的潜力。OpenMP 自动将循环线程化,提高多处理器系统上的应用程序性能。用户不必处理迭代划分、数据共享、线程调度及同步等低级别的细节[2]。目前 Intel C++ 编译器 9.1、Visual C++ 8.0 和 Microsoft Visual Studio 2005 均支持 OpenMP2.5。文中主要介绍在 Microsoft Visual Studio 2005 中 OpenMP 的使用。

1.1 OpenMP 模型

共享存储模型是一般的集中式多处理机的抽象[1,2,4]。其底层为一系列处理器,这些处理器访问同一个共享存储器。由于所有处理器可以访问内存中的同一位置,因而它们可以通过共享变量进行交互和同步。OpenMP 采用了共享存储中标准的并行模式 fork-join,当程序开始执行时只有主线程存在,主线程执行程序的串行部分,通过派生出其他的线程来执行其他的并行部分。当重新执行程序的串行部分时,这些线程将终止。如图 1 所示。

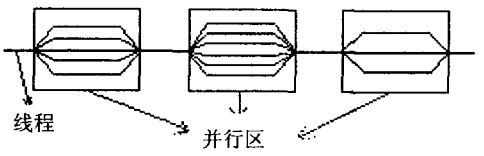

图 1 OpenMP 执行模型

1.2 OpenMP 指令

工业标准 OpenMP 是对 C 语言的一个扩展,目的是支持并行程序设计。书写 OpenMP 程序同书写 C 语言程序相似[4,5],只是在 C 程序中加入了 OpenMP 的编译指示,这些编译指示描述了程序应该以何种方式并行执行。加入了 OpenMP 指示的 C 程序可以由任意支持 OpenMP 的编译器编译,在不同平台的硬件上执行。OpenMP 编译器命令以 # pragma 开始,在其后面是 omp,名字和可选的子句,并用新行结束。某些子句可出现在不同的命令中,但需要对它们加以分别的定义。某些命令将作用于整个结构块,所谓的构造是由编译器命令及跟在其后的结构块所组成。在 C/C++ 中,OpenMP 指令使用格式是 # pragma omp 指令[子句[子句]……]。

OpenMP 编译制导包括并行域结构、共享任务结构、组合的并行共享任务和同步结构四类[6]。并行域结构中并行域代码被所有的线程执行,通常使用 # pragma omp parallel [子句[子句]……]这样的格式,常用的子句有 firstprivate, if, lastprivate, private, reduction 等。共享任务结构将它所包含的代码划分给线程组的各成员来执行,可分为并行 for 循环、并行 sections、串行执行三种情况。并行 for 循环格式为 # pragma omp for [子句[[,]子句]…],一般用于 for 循环前,将循环分配到多个线程中并行执行。sections 编译制导语句指定内部的代码被划分给线程组中的各线程,不同的 section 由不同的线程执行。用 for 语句来分摊是由系统自动进行,只要每次循环间没有时间上的差距,那么分摊是很均匀的,使用 section 来划分线程是一种手工划分线程的方式,最终并行性的好坏依赖于程序员[7]。single 编译制导语句用于内部部分代码不方便并行执行的情况,使用 single 编译制导语句指定这部分代码只有线程组中的一个线程执行,格式为 # pragma omp single[子句[[,]子句]…],线程组中没有执行 single 语句的线程会一直等待代码块的结束,使用 nowait 子句除外。组合的并行共享任务分为 parallel for 和 parallel sections 编译制导语句,前者格式 # pragma omp parallel for[子句…]表明一个并行域包含一个独立的 for 语句,后者格式 # pragma omp parallel sections[子句…]表明一个并行域包含单独的一个 sections 语句。一般而言,因为使用 parallel 命令意味着需要多于一个线程,

所以在使用命令前需要先建立线程组,同步结构中的编译指导语句有 barrier,atomic,master,ordered,threadprivate 等[1]。

OpenMP 编译制导可根据需要包含子句项,在没有其它约束条件下,子句可以无序,也可以任意地选择[2]。#pragma omp parallel for[子句…]是最频繁使用的编译指导语句,可搭配使用的子句有 firstprivate,if,lastprivate,private,reduction,schedule 等。firstprivate 子句指定每个线程都有它自己的变量私有副本,并且变量要被继承主线程中的初值;lastprivate 子句指定将线程中的私有变量的值在并行处理结束后复制回主线程中的对应变量;private 子句指定每个线程都有它自己的变量私有副本;reduction 子句指定一个或多个变量是私有的,并且在并行处理结束后这些变量要执行指定的运算;schedule 子句指定如何调度 for 循环迭代。

1.3 OpenMP 库函数

OpenMP 使用库函数<omp.h>,库函数包括执行环境函数、锁函数和定时三类[2]。表 1 列出一些最常用的函数。

表 1　OpenMP 常用函数

函数原型	使用说明
int omp_get_num_procs;	返回运行本线程的多处理机的物理处理器个数
int omp_get_num_threads;	返回当前并行区域中的活动线程个数
int omp_get_thread_num;	返回线程号
void omp_set_num_threads (int num_threads);	设置并行执行代码时的线程个数
void omp_init_lock(omp_lock_t * lock);	初始化一个简单锁
void omp_set_lock(omp_lock_t * lock);	上锁操作
void omp_unset_lock(omp_lock_t * lock);	解锁操作,要和 omp_set_lock 函数配对使用
void omp_destroy_lock (omp_lock_t * lock);	omp_init_lock 函数的配对操作函数,关闭一个锁
double omp_get_wtick( );	返回连续时钟记号间的秒数
double omp_get_wtime( );	返回自过去某个时间开始所花费的时钟时间

2 FFT 并行算法探讨

2.1 二维离散快速傅里叶变换(FFT)

数字图像处理中进行频谱分析经常使用二维傅里叶变换(FFT),二维傅里叶变换在一维傅里叶变换的基础上推导得出,即将一个二维傅里叶变换的运算分解成水平和垂直方向上的一维傅里叶变换运算。一维离散傅里叶变换公式为:

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

在此基础上的一个尺寸为 $M\times N$ 的图像二维离散傅里叶变换由

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

给出。具体实现算法如下[8]:

(1)获取原图像的数据区首地址、图像的高度和宽度,计算进行傅里叶变换的图像宽度和高度及水平、垂直方向的迭代次数。

(2)按行列顺序依次读取数据区的值,存储到开辟的复数存储区。

(3)调用一维傅里叶变换函数进行垂直方向的变换。

(4)转换变换结果,将垂直方向的变换结果转存回时域存储区。

(5)调用一维傅里叶变换函数进行水平方向的变换。

(6)将计算结果转换成可显示图像,并将坐标原点移至图像中心位置,使图像可以显示整个周期频谱。

二维离散快速傅里叶变换是数字信号处理的重要工具,但是计算量较大运算时间较长在某种程度上限制了使用,为了解决这一矛盾,引入 OpenMP,充分利用双核技术从而达到快速运算目的。

2.2 FFT 算法的并行处理

实验平台为 Dell Optiplex GX630 系列双核台式机。采用 Intel Lakeport - G i945G 芯片组,Intel Pentium D CPU 2.80GHz,高速缓存 1 MB,前端总线 800MHz,内存 HY Dual DDR2 SDRAM 1G。操作系统为 Microsoft Windows XP Professional 5.1.2600 (WinXP Retail),编译器为 Microsoft Visual Studio 2005。在此平台上,用 OpenMP 改进二维离散快速傅里叶变换,实验数据为程序多次运行的平均值。

(1)串行版本。

```
for(j = 0;j < h;j ++)//在垂直方向上进行快速傅里叶变换
    QFC(&t[w * j],&f[w * j],wp);
for(j = 0;j < h;j ++)//转换变换结果
    for(i = 0;i < w;i ++)
    t[j + h * i] = f[i + w * j];
for(j = 0;j < w;j ++)//水平方向进行快速傅里叶变换
    QFC(&t[j * h],&f[j * h],hp);
```

第一个 for 循环完成垂直方向的快速傅里叶变换,QFC(&t[w * j], &f[w * j],wp) 是一个实现快速傅里叶变换的函数, &t[w * j], &f[w * j] 分别是指向时域和频域的指针,wp 是进行快速傅里叶变换

的宽度(2 的整数次方);第二个 for 循环转换变换结果,将垂直方向的变换结果转存回时域存储区;最后一个 for 循环完成水平方向的快速傅里叶变换,同样地调用了 QFC 函数。

(2)并行版本。

Microsoft Visual Studio 2005 中通过一个编译开关选项支持 OpenMP。在并行版本的项目属性对话框"配置属性"中的"C/C++"语言页里,将 OpenMP 支持选项改为"是/(OpenMP)"就能够支持 OpenMP。在二维离散快速傅里叶变换 C 程序串行版本中,一维离散快速傅里叶变换主要以列(或者行)的方式进行 FFT 变换,for 循环中不存在数据相关的限制,每次循环基本独立,后一次循环不依赖于前面的循环,即满足并行执行的要求,因此在双核平台上可以使用 OpenMP 指令把一维离散快速傅里叶变换的操作分派到 2 个 CPU 上分别执行。下面以垂直方向一维快速傅里叶变换为例,并行计算的主要代码如下:

```
#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 指定后面的语句块由每个 CPU 并行执行。串行版本中的 for 循环分派到 2 个 CPU 中,每个 CPU 分别对图像中的奇、偶数列做一维 FFT 变换。parallel 指令用于为一段代码创建多个线程且并行执行的。与传统的创建线程函数相比,相当于为一个线程入口函数重复调用创建线程函数来创建线程并等待线程执行完。转换变换结果代码部分使用 #pragma omp parallel for 语句将一个 for 循环分配到多个线程中执行。OpenMP 具有运行时为当前计算机自动创建最佳线程数的优异特性,如果在单核处理器上运行代码,则将在一个线程上运行所有代码,如果在双核处理器上运行相同代码,就将在两个线程上运行代码,代码将在其当前计算机上自动调整至最佳状态,以尽可能实现最大加速比。

2.3 并行算法的性能优化

在双核平台上实现最大加速比的方法是使各个线程的负荷尽可能保持平衡,每个线程运行的工作量大致相同。默认情况下,OpenMP 通过将代码拆分为几个相等的块,然后在各自的线程上运行块以调度循环[7]。通常默认的块足够大,当所有的线程运行一遍的时候该循环体运行结束。OpenMP 提供了几个调度选项,通过将 schedule (static[, chunksize])、schedule (dynamic[, chunksize]) 或 schedule(guided[, chunksize]) 添加到 OpenMP 指令控制调度各个线程的方式。图 2 以二维快速离散傅里叶变换的并行版本处理一个 256 * 256 像素的图片为例考察三种调度策略。

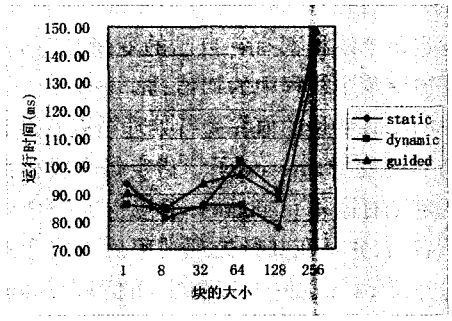

图 2 FFT 并行版本运行时间比较

(1)调度策略。

静态调度静态地请求将块中的工作分发给各个线程,当线程执行完一个块后,以轮询的方式在所有的线程中获取另一个同样大小的块。而动态调度不遵循静态调度的轮询顺序,只要线程需要更多工作,就可以马上获得下一个迭代块。编译器可以对静态部分进行一定的优化,但是对动态部分没有优化功能。引导调度是一种混合策略,每个线程在第一次运行时获得一个较大的块,以后运行中将逐步缩减块的大小,直到缩减至 chunkSize 指定的限制值,块的大小按需要动态计算,这与块大小逐渐减少的动态调度类似。

(2)块的大小。

chunkSize 参数意味着各个调度之间有所区别。在所有情况下,返回到线程调度代码要支出一些额外开销,以此换来较好的灵活性。各次静态运行存在些许差异,较小的块运行的时间略微有点长,因为运行调度代码的次数更多。OpenMP 的默认调度方法总是按照迭代计数除以正在运行的线程数计算块的大小,在本例中,chunkSize 等于 128。比较代码中的各种调度时,始终以评测这种情况为基础。由此可得,大小为 1 的块和大小为 8 的块之间的差数(大约 12 毫秒)即为静态调度开销,或者是设置和启动各轮迭代的开销。如果调度块的大小为 1,则必须设置和启动循环 128 次(它在每个线程上运行一个块)。对于大小为 8 的块,必须设置和启动 16 次。比较可得,在 12 毫秒的时间里额外的开销有 112 次,因此使用此调度方法的开销比运行每个循环的开销仅少了 107 微秒。虽然在处理 256 * 256 像素图片时候差距并不明显,但是大规模处理其他大型图片仍然需要考虑在静态调度中使用略

微大一些的块。动态调度的灵活性最大,但是在调度出错时,造成的性能损失也最大。对于平衡良好的代码,应避免使用非常小的块。本例子中调度块大小为128明显是很好的选择。而对于那些不平衡的代码,正确的块大小应该在运行较少的块和达到更佳的线程平衡之间做一个权衡。引导调度使用较小的块作为限制时运行最佳、灵活性最大。限定为较大的块大小时,运行的时间将很长。引导调度的上限是循环总大小的二分之一,对于平衡良好的代码,应该相当于在一次运行中将循环拆分为相等几部分的静态情况。

3 结　语

OpenMP 是针对共享地址空间并行计算机提供的并行计算库,目前 Microsoft 和 Borland 公司都有相应产品支持 OpenMP2.5。使用 OpenMP 不必写诸如 CreateThread 之类的线程管理代码,编写多线程程序简便高效,而且 OpenMP 提供了丰富的指令,对于同步共享变量、合理分配负载等任务,都提供了有效的支持。不过 OpenMP 也存在着一些不可避免的缺点:第一,OpenMP 主要以预编译指令(#pragma)实现多线程并行,所以在单核机器上编译的程序在多核机器上运行时无法体现多核的优势;第二,OpenMP 对编译器要求比较高,一般要求 Microsoft Visual Studio 2005 或者需要 Intel 编译器。不过长远来讲,OpenMP 的优势是明显的。Intel 技术官曾说"今后的处理器发展是内部优化与集成多核而不是单纯地提升处理器的频率,采用多线程的软件也将会是今后软件的主流。"充分发挥多核处理器的优势使软件性能最优化,给中国带来一个极大的机遇。开发 OpenMP 软构件库需要大量的人力资源,这项工作适合在中国进行。OpenMP 库在未来的经济效益可以同微软的操作系统相比,后者为用户顺利使用 PC 提供工具软件支持,前者为顺利使用多处理器提供库函数支持。世界正在进入多处理器时代,OpenMP 库将成为程序员必不可少的工具。

参考文献:

[1] 陈国良.并行算法实践[M].北京:高等教育出版社,2004.

[2] Quinn M J. Parallel programming in C with MPI and OpenMP [M]. 北京.清华大学出版社,2005.

[3] 赖建新,胡长军,赵宇迪,等. OpenMP 任务调度开销及负载均衡分析[J]. 计算机工程,2006,18:(sup)58-60.

[4] Grama, Ananth. Introduction to parallel computing[M]. 北京:机械工业出版社,2003.

[5] Foster I, Designing and building parallel programs[M]. 北京:机械工业出版社,2002.

[6] Andrew. Multithreading parallel and distributed programming [M]. 北京:高等教育出版社,2002.

[7] Malyshkin V. Parallel computing technologies[C]//8th international conference, PaCT 2005. Krasnoyarsk, Russia, 2005. Berlin; New York: Springer, 2005.

[8] 杨淑莹. VC++图像处理程序设计[M]. 北京:清华大学出版社,2003.

[9] Dongarra J. Parallel computing programming[M]. 北京:电子工业出版社,2005.

[10] Wilkinson B, Allen M. Techniques and applications using networked workstations and parallel computers[M]. 北京:机械工业出版社,2005.

(上接第 83 页)

川大学学报:工程科学版,2005,37(3):118-122.

[14] Govindan R, Tangmunarunkit H. Heuristics for internet map discovery[C]//Proceedings IEEE INFOCOM. [s. l.]: [s. n.], 2000:1371-1380.

[15] 姜　誉,胡铭曾,方滨兴,等. 一个 Internet 路由器级拓扑自动发现系统[J]. 通信学报,2002,23(2):54-62.

[16] 杨家海,任宪坤,王沛瑜. 网络管理原理与实现[M]. 北京:清华大学出版社,2000.

[17] Peterson L L, Davie B S. Computer networks: A Systems Approach[M]. 3rd Edition. USA: Elsevier Science, 2003.

[18] RekhterY. An architecture for IP address allocation with CIDR [S]. RFC1518, 1993.

(上接第 86 页)

erogeneous, Distributed Biological Data Sources[C/OL]// Proceedings of the IJCAI2001 Workshop on Knowledge Discovery from Heterogeneous, Distributed, Autonomous, Dynamic Data and Knowledge Sources. 2001. http://www.cs.iastate.edu/honavar/Papers/ijcaiworkshoppaper.pdf.

[13] Vargas-Vera M. Knowledge Extraction Using an Ontology-Based Annotation Tool [C]//Workshop on Knowledge Markup & Semantic Annotation. [s. l.]: ACM Press, 2001:5-12.

[14] Maedche A, Neumann G, Staab S. Bootstrapping an Ontology-based Information Extraction System[C]//Intelligent Exploration of the Web, Studies in Fuzziness and Soft Computing. Heidelberg: Physica-Verlag GmbH, 2003:345-359.

[15] Yeh C L, Su Y C. Web Information Extraction for the Creation of Metadata in Semantic Web[C]//Proceedings of ROCLING. Tainan, Taiwan: [s. n.], 2005.