一种基于多线程技术的并行FFT算法

胡 灏，陈 勇
（天津工业大学 计算机技术与自动化学院，天津 300160）

摘 要：在传统的串行FFT算法基础上提出了一种基于多线程技术的并行FFT算法. 实验数据表明：该算法在一定程度上能提高程序的执行效率，特别是当增大计算负载时，该算法的执行效率与传统的并行算法的比值（加速比）将趋近于处理器的个数.

关键词：多线程；并行FFT算法；加速比
中图分类号：TP312 **文献标识码**：A **文章编号**：1671-024X(2007)01-0058-03

Parallel FFT algorithm based on multi-threading technology

HU Hao, CHEN Yong
(School of Computer Technology and Automation, Tianjin Polytechnic University, Tianjin 300160, China)

Abstract: On the basis of traditional serial FFT algorithms, a parallel FFT algorithm based on multi-threading is put forward. The experiment data indicate that this new algorithm can improve the executive efficiency of program to a certain extent, compared with the traditional serial one, the speedup ratio of the efficiency of new algorithm will tend to the number of processors when adding to the working load.

Key words: multi-threading; parallel FFT algorithm; speedup ratio

随着并行计算机的发展，涌现出了许多并行的FFT算法. 其中大部分算法是基于某种具体的并行机结构的，即某种算法是某种计算机系统结构所专用，算法不具备通用性. 因此程序设计者在设计并行算法时必须考虑具体的硬件结构. 多线程技术的出现为人们在设计并行算法时不用考虑具体的硬件结构提供了可能，因为线程由OS（操作系统）来调度，OS会自动将处理器的时间片分配给各个线程. 本文提出了一种基于多线程技术的并行算法，使得程序设计者在设计并行程序时，只需考虑线程之间的并行性以及线程间的通信和同步等，而除此以外的工作，都交由OS去管理. 这样一来，并行程序的设计工作得到了简化，而且改进后的并行程序的执行速度较之于原来的串行程序有一定的提高.

1 采用多线程技术的并行算法的特点

与传统的并行算法相比，采用多线程技术的并行算法的一个特点是算法设计者无需考虑计算机具体的硬件结构，如网络拓补结构、存储器层次结构等；另外，多线程技术本身就包含了一定的并行性，使用多线程技术来进行并行程序设计也就显得比较合理了.

线程是一种特殊的对象，是操作系统执行多任务的一部分，它允许应用程序的一部分独立于其他对象而单独运行，因此也就脱离了应用程序常规的执行顺序. 可以将进程分解为多个线程，以共享分配给它的时间片. 单个处理器一次只能执行一个线程，进程中的每个线程都被分配了一部分执行时间，以完成它的工作. 但是，如果计算机使用了多个处理器，那么，在同一时刻将会有多个线程被分配执行时间（此工作将由操作系统来完成），进程的各个独立的对象（线程）将并发地执行[1]. 如果进程没有被分解，即使有多个处理器，那么进程一次也只能获得一个执行时间. 因此，在多处理器平台上，使用多线程的应用程序将获得更高的性能. 但是，由于各个线程是相互独立的对象，而且线程一旦开创后便很难得到控制，因此，在设计多线程算法时，应设法找出其算法中的并行性，实际上就是将算法的执行序列进行分解，将每个序列独立出来，并抽象为

收稿日期：2006-05-16 **作者简介**：胡 灏（1980—），男，硕士研究生；陈 勇（1959—），男，副教授，导师，E-mail：chenyong@tjpu.edu.cn

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

一个方法,而这些方法则在各个独立的线程内得到执行. 为了使各个线程中的数据保持正确性,线程之间可能需要进行通信或同步. 与进程类似,线程也有诸多状态,如运行、终止、休眠等. 在设计程序时,可以通过设置线程的状态和属性,从而达到线程间的通信和同步. 与传统并行算法中的通信相比,线程间的通信不需要硬件的支持,与 CPU 之间的网络拓补结构无关,而是通过操作系统的内部机制来完成.

2　基于多线程的并行 FFT 算法

基于多线程的并行 FFT 算法的思想是:若计算机有 $m(m=2^l)$ 个处理器,则在程序中开创 m 个线程来计算. 其中每个线程分担的负载都一样,均为 n/m 个点,算法描述如下:

输入:$a=(a_0,a_1,\cdots,a_{n-1})$
输出:$b=(b_0,b_1,\cdots,b_{n-1})$
Begin
(1) for $k=0$ to $n-1$ do
　　$c_k=a_k$
　endfor
(2) for $g=0$ to $m-1$ par-do
　　/* 开创 m 个线程进行计算 */
(2.1) for $h=\log n-1$ to 0 do
(2.2) $p=2^h$
(2.3) $q=n/p$
(2.4) $z=w^p$
(2.5) for $k=gn/2m$ to $(g+1)n/2m-1$ do
　　(ⅰ) $c_{d(k)}=c_{d(k)}+c_{d(k)+p}z^{d(k) \bmod q}$
　　/* 其中 $d(k)$ 满足:
　　$d(k) \bmod p=d(k) \bmod 2p$
　　$d(k)<d(k+1), k=0,1,\cdots,n/2-1$ */
　　(ⅱ) $c_{d(k)}=c_{d(k)}-c_{d(k)+p}z^{d(k) \bmod q}$
　　/* $c_{d(k)}$ 不用(ⅰ)计算的值 */
　　endfor
　　endfor
endfor
(2.6) for $k=gn/m$ to $(g+1)n/m-1$ do
　$b_k=c_{r(k)}$
　　/* $r(k)$ 为 k 的位反 */
　endfor
End

算法第(2)步开创了 m 个线程,每个线程平均分担了计算任务. 由于有 m 个处理器,操作系统每次都会分配 m 个时间片,每个线程都将获得时间片而不必阻塞以等待操作系统重新分配时间片. 这样一来,算法就是在并行地执行. 从(2.5)步开始,每个线程在执行完每一次循环以后,线程之间需要进行同步. 下面以 $n=4, m=2$ 为例进行具体说明. 给定 $\boldsymbol{a}=(a_0, a_1, a_2, a_3)$,计算 $\boldsymbol{b}=(b_0, b_1, b_2, b_3)$. 算法执行到第(2)步,将开创 2 个线程来进行计算,此时 $h=1, p=2, q=2$,满足 $d(k) \bmod p=d(k) \bmod 2p$ 的 $d(k)$ 有 0 和 1,那么第一个线程将负责 c_0 和 c_2 的计算,第二个线程将负责 c_1 和 c_3 的计算. 每一个线程计算完以后,将与其他线程进行通信,判断其他线程是否计算完毕;若其他线程计算未完成,则该线程将等待其他线程,以此实现线程间的同步. 这样做的目的是确保每个线程在进行(2.5)步的每次循环之前,$\boldsymbol{c}$ 中的每个值都是有效的. 当 $h=0$ 时,$p=1, q=4$,满足 $d(k) \bmod p=d(k) \bmod 2p$ 的 $d(k)$ 有 0 和 2,那么第一个线程将负责 c_0 和 c_1 的计算,第二个线程负责 c_2 和 c_3 的计算. 第(2.6)步执行的结果为:$b_0=c_0, b_3=c_3, b_1=c_2, b_2=c_1$. 由于每个线程的计算负载为 n/m 个点,故上述算法的复杂度为:$O((n\log n)/m)$. 线程间的通信和同步主要发生在第(2.5)步每次循环之后,故同步所需要的时间为:$O(\log n)$. 当 n 充分大时,显然计算时间占主导地位. 计算蝶型图如图 1 所示[4].

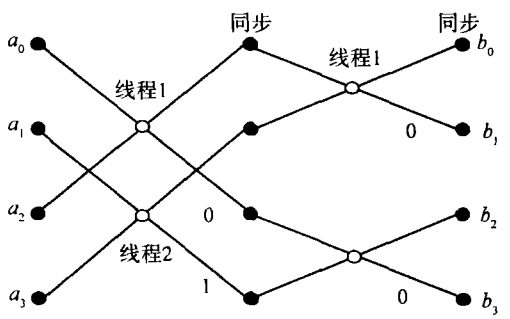

图 1　计算蝶形图
Fig. 1　Computing flow diagram

该算法与传统并行 FFT 算法的一个区别,就是在算法的(2.5)步,使用了一个矢量 d 分配将要计算的点的序号. 矢量 d 中的值将由主线程计算得出,并将其所对应的点分配到各个计算线程. 这样一来,就实现了主线程对分线程的计算负载的动态分配,在必要的时候,可以通过这种方法对某个分线程增加或减少计算负载,而且还可以尽量减少线程间的通信所带来的一些开销。该算法的编程模型具有共享变量模型的特点,因为主线程和分线程之间实际上是通过读写全局变量 d 来进行通信的;另一方面,它又具有消息传递模型的特点,因为分线程之间是通过发送和接受同步信号或中断信号来进行通信的。它具有两种模型的优

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

势,这样的话,可以提高编写多线程程序的灵活性,不妨把这种模型叫做共享变量基础上的消息传递模型.这一模型如图2所示.

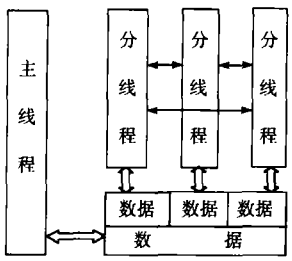

图2 算法模型图

Fig.2 Algorithm model figure

3 实验结果及分析

下面是在HP服务器上进行实验所得到的实验数据.该服务器共有4个主频为3.0 GHz的处理器,操作系统采用的是Windows Server,编程语言为C++.NET.分别用多线程程序和单线程程序在其上运行,共得出两组数据,如表1所示.

表1 程序执行时间

Tab.1 Program's excuting time s

类别	$n=2^{10}$	$n=2^{15}$	$n=2^{20}$
单线程	0.008 935	0.400 054	18.315 8
多线程	0.097 36	0.617 325	7.105 27
类别	$n=2^{21}$	$n=2^{22}$	$n=2^{23}$
单线程	38.329 1	78.458 2	159.016 4
多线程	11.582 3	20.614 6	40.764 1

从上述两组数据可以看出:当n值比较小时,多线程程序的执行效率反而不如单线程程序,这是因为系统开创线程需要一定的开销,当计算量比较小时,这种开销在整个程序开销中所占的比重就比较大,而单线程程序没有此类开销,故单线程程序的执行时间要比多线程程序短;而当n值比较大时,多线程程序的执行效率比单线程程序要高,当n的值逐渐增大时,二者的效率比趋近于4:1.实际上,系统在开创线程的时候,是按顺序开创的,即按串行的方式进行,因而,开创线程的开销实际上是程序中串行分量的开销.线程数固定后,此开销基本上是固定的.令W为问题规模,W_S为程序中的串行分量,W_P为程序中的并行分量,f为程序中串行分量的比例,m为处理器个数,根据Gustafson加速定律[2],加速比为:

$$S=\frac{W_S+mW_P}{W_S+m\cdot W_P/m}=\frac{W_S+mW_P}{W_S+W_P} \tag{1}$$

归一化后可得:

$$S=f+m(1-f)=m+f(1-m)=m-f(m-1) \tag{2}$$

由于W_S基本上是固定的,因而当$W\to\infty$时,$f\to 0$,从而(2)式的右端将趋近于m.也就是说,当问题规模逐渐增大时,加速比将趋近于处理器个数.但前提是线程数固定,一但线程数固定了,那么程序中用来开创线程和用来通信的额外开销W_0也就固定了,问题规模增大时,增大的实际上是程序中的并行部分.

4 结束语

采用上述并行算法编制的程序与处理器的个数几乎成线形关系,并且与具体的硬件平台无关,这样大大提高了程序的执行效率.随着多处理器计算机的越来越普及(AMD公司在2005年率先推出了双核处理器,紧接着Intel公司也开发了自己的双核处理器),由于多线程并行程序的执行效率比较高并且编码工作得到了很大的简化,因此,采用多线程技术来设计并行算法将会越来越受欢迎.

参考文献:

[1] TITUS Tobin. C# Threading Handbook[M]. NewYork: Wrox Inc,2003.

[2] 陈国良.并行计算[M].北京:高等教育出版社,2003.

[3] 郑纬民,汤志忠.计算机系统结构[M].北京:清华大学出版社,2000.

[4] 张傲华,张正鸿.高性能FFT处理器设计[J].电子对抗技术,2005,(4):44-47.

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net