文章编号:1001 - 9081(2007)04 - 0795 - 03

一种基于二叉胖树模型的并行 FFT 算法

魏文红[1],高大利[2]
(1. 华南理工大学 计算机科学与工程学院,广东 广州 510640;
2. 泉州师范学院 计算机系,福建 泉州 362000)
(hquwwh@ tom. com)

摘 要:二叉胖树网络结构是一种易于实现蝶式计算的网络拓扑结构,基于这一特点,首先构造了一种二叉胖树的逻辑模型,并提出了一种基于该模型的并行快速傅立叶变换算法。该算法使得进程间有良好的负载平衡,相对于串行算法来说,大大降低了时间复杂度。在集群系统和 MPI 环境下,给出了该算法的实现及实验数据分析。

关键词:二叉胖树;蝶式计算;快速傅立叶变换;并行计算
中图分类号: TP393.02 **文献标识码**:A

Parallel fast Fourier transform algorithm based on binary fat tree network

WEI Wen-hong[1], GAO Da-li[2]
(1. *School of Computer Science and Engineering, South China University of Technology, Guangzhou Guangdong* 510640, *China*;
2. *Department of Computer, Quanzhou Normal College, Quanzhou Fujian* 362000, *China*)

Abstract: The binary fat tree is a network topology which is prone to accomplish butterfly computing. According to this feature, a logical model for binary fat tree was constructed at first, and then a parallel Fast Fourier Transform algorithm based on it was developed. In this algorithm, balance of load was achieved in the process, and time complexity was reduced compared with serial algorithm. At last the algorithm was implemented in cluster and MPI, and experimental data was analyzed.

Key words: binary fat tree; butterfly computing; Fast Fourier Transform (FFT); parallel computing

0 引言

计算机技术和通信技术的迅速发展与紧密结合,大大地推动了数字信号处理技术的发展。从 1965 年 Tukey 和 Cooley 发现了离散傅立叶变换(Discrete Fourier Transform, DFT)以来,不断出现了一些改进的和高效的算法。其中,快速傅立叶变换(Fast Fourier Transform, FFT)有特殊的运算结构和数据存取结构,针对其算法特点将计算量从 $O(n^2)$ 下降到 $O(n\log n)$,从而使 DFT 理论在数字信号处理、石油勘探、地震预报、编码理论、图像处理、模式识别等领域得到了广泛应用[1]。

二叉胖树(binary fat tree)[2]由传统的二叉树演变而来的,由于传统的二叉树网络模型在通信时容易造成根部的通信瓶颈问题,所以在二叉胖树中,节点间的通路自叶向根逐渐变宽,它更像一颗真实的树,连向根部的枝叉变得愈来愈粗。如果在二叉胖结构的根节点中多生成些进程,而在其他的分枝节点中,相应地少生成些进程,这样由于根节点的通信带宽比较宽,即使进程多,也不会产生通信拥塞。在各分枝节点中,由于进程数量相对较少,因此,仍然不会产生通信拥塞现象。二叉胖树的网络拓扑结构如图 1 所示,正是由于这一特点,我们研究发现,它的逻辑模型与蝶式计算模型有很大的相似之处,如果在该模型上进行并行 FFT 算法的蝶式计算,则能够表现出显著的优势。实验结果也证明了该假设。

本文介绍了离散傅立叶变换 DFT,引出 FFT 蝶式计算图,然后分析了二叉胖树逻辑模型图与蝶式计算图之间的关系,最后在集群系统与 MPI(消息传递界面)环境下给出了一维 FFT 并行算法的实现及性能分析。

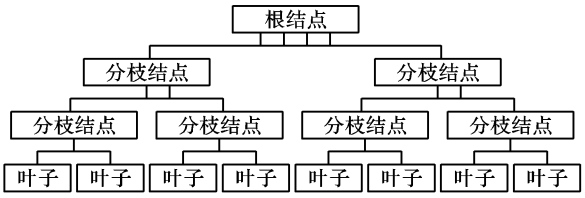

图 1 二叉胖树网络的拓扑结构

1 DFT 变换原理

设 $a(m)$, $m = 0,1,2,\cdots,n-1$ 为时间采样序列,则 $a(m)$ 的傅立叶变换定义为:

$$b_j = \sum_{m=0}^{n-1} \omega^{jm} a_m,\ 0 \leqslant j \leqslant n-1 \tag{1}$$

式中,ω 是单位 n 次元根,即 $\omega = e^{2\pi i/n}$, $i = \sqrt{-1}$。

文献[1]推演了 FFT 的蝶式计算图,如图 2 所示。

根据 FFT 的蝶式计算图可知,可以采用按频率抽取算法(DIF)迭代计算来完成 FFT。则 n 点序列 $a(m)$ 的 FFT 可由两个长度减半的 $n/2$ 点序列迭代计算而得,令 $h = n/2$,则有如下公式:

$$\begin{cases} f(m) = a(m) + a(m+h), \\ g(m) = [a(m) - a(m+h)] * \omega^m, \end{cases} \quad m = 0,1,\cdots,h-1 \tag{2}$$

$$\begin{cases} a(m) = f(m), \\ a(m+h) = g(m), \end{cases} \quad m = 0,1,\cdots,h-1 \tag{3}$$

收稿日期:2006 - 10 - 12;**修订日期**:2006 - 12 - 13

作者简介:魏文红(1977 -),男,江西南昌人,讲师,博士研究生,主要研究方向:网络与并行分布式计算; 高大利(1975 -),女,河南郑州人,讲师,硕士,主要研究方向:网络数据库技术.

式中,当 h 为2的整次幂时,可重复地使 $h = h/2$,将此过程迭代下去,直到 $h = 1$ 为止,最终得到 n 点序列 $a(m)$ 的FFT。另外,从图2中可以看出,如果在图2中标有 $p_0, p_1, \cdots, p_{n/2-1}$ 的位置上放置 $n/2$ 个进程,其规律是:2点FFT放置1个进程,4点FFT放置2个进程,8点FFT放置4个进程,16点FFT放置8个进程,……, n 点FFT放置 $n/2$ 个进程。每个进程在每一次蝶式计算中都做了一次公式(2)与公式(3)的运算及一次进程间的通信工作。如果把一次蝶式计算时的所有进程收缩为一个点,例如,在16点FFT中,$p_0 \sim p_7$ 为合并为一个点,将其放入根节点,$p_0 \sim p_3$ 和 $p_4 \sim p_7$ 分别为合并为两个节点,将它们放入根点的下一层的两个分枝节点中,以此类推,直到叶子节点结束。则得出了如图3所示的基于二叉胖树的逻辑模型的进程结构。

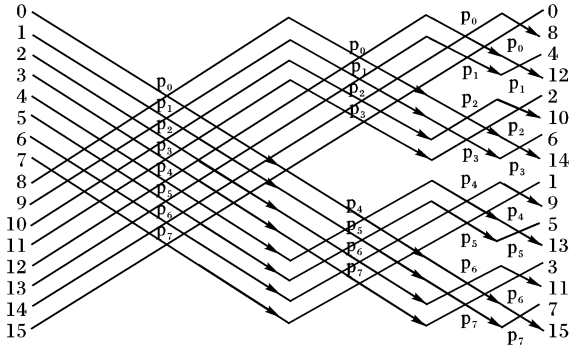

图2 16点FFT的蝶式计算图

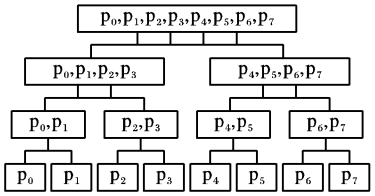

图3 基于二叉胖树模型的进程结构

由图3可以看出,叶子节点只有一个进程,从叶子节点到根节点进程数目越来越多,也即自叶到根愈来愈粗,满足二叉胖树的拓扑结构。

2 基于二叉胖树模型的并行FFT算法

2.1 算法的原理

在二叉胖树模型下,按照图3的逻辑结构产生 p 个进程,则该并行FFT算法的基本思路是:在根节点由 p 个进程共同工作,完成第一次蝶式计算。因为蝶式计算会产生数据分裂,所以运算结果就形成了规模减半的两组数据,进而分别将这两组数据传送给下一层的两个分枝节点上的 $p/2$ 个进程,并完成 $p/2$ 次进程间的通信。然后分别在这两个分枝节点上,又重复根节点的过程,完成第二次蝶式计算,如此反复地执行运算,直到叶子节点为止,最后在叶子节点输出运算结果。以16点FFT为例分析该算法的执行步骤,如下所示:

(1)在根节点,进程 $p_0 \sim p_7$ 并行进行第0级蝶式计算,计算结果分裂为规模减半的两组数据,发送第一组数据给下一层分枝节点的进程 $p_0 \sim p_3$,发送第二组数据给下一层另一分枝节点的进程 $p_4 \sim p_7$。

(2)在第一层节点,第一分枝节点的进程 $p_0 \sim p_3$ 并行进行第1级蝶式计算,计算结果也分裂为规模减半的两组数据,发送第一组数据给下一层分枝节点的进程 p_0、p_1,发送第二组数据给另一个下一层分枝节点的进程 p_2、p_3。与此同时,第二分枝节点的进程 $p_4 \sim p_7$ 同时也并行进行第1级蝶式计算,运算结果同样也分裂为规模减半的两组数据,发送第一组数据给它的下一层分枝节点的进程 p_4、p_5,发送第二组数据给它的下一层另一个分枝节点的进程 p_6、p_7。

(3)在第二层的节点,类似地重复(2)的过程,各进程并行进行第2级蝶式计算,又把运算结果发送给它们各自下一层相应的节点。

(4)在第三层的节点即叶子,由于在这一层的每个节点上都只有一个进程,于是 p_0、p_1、p_2、p_3、p_4、p_5、p_6、p_7 各自同时完成第3级蝶式计算,并保留运算结果。

(5)进程 p_0 收集各进程的运算结果,并负责输出。

从这些执行步骤可以看出,我们所构造的二叉胖树模型的层数为 $\log_2(p) + 1$,p 为所需参与运算的进程数。

2.2 算法的实现

基于前面的理论分析,对该算法进行了实现,实验环境为8台P4 2.4,256M内存,80G硬盘的DELL PC机和100Mb交换机组成的集群系统,每台PC机都安装了Linux操作系统、MPICH2函数库。之所以采用MPI环境,是因为MPI是一个广泛用于编写消息传递并行程序的可移植标准平台。它与成熟度已较高的并行虚拟机(Parallel Virtual Machine,PVM)相比,具有一些重要特点,如MPI能有效地管理消息缓冲区;能实现完全的异步通信,通信与计算能完全同时进行;能在大规模并行机(MPP)和工作站机群上有效运行等。MPI函数库提供了约125个函数,最少时可只用其中六个基本函数就能完成消息传递并行程序编程。为了方便阅读,并没有给出全部的源程序,只是给出了核心算法部分。

设进程数为 p,待进行快速傅立叶变换的信号序列 $X(m)$ 点数为 n,$m = 0, 1, \cdots, n-1$,则 $p = n/2$。

```
begin
  level = log2(p);                          //二叉胖树的层数值
  m = id;          //进程 id 第一次蝶式计算分配到的序列的下标
  for(j = 0; j <= level; j + +)
                    //从二叉胖树的根节点开始,到叶子节点结束
    h = n/2^(j+1);                          //蝶式计算的步长
    f = X[m] + X[m + h];
    g = (X[m] - X[m + h]) * ω^m; //按照公式(3),进行蝶式计算
    if(h! = 1)                              //如果不是叶子节点
      if(id 在下一层节点的左边分枝节点中)
                    //进程 id 先发送新的数据 g 给在下一层节点中
        //将属于右分枝的相应进程,再从该进程接收新的数据 f
        X[m] = f;
        MPI_Send(&g, id + h);
        MPI_Recv(&f, id + h);
        X[m + h/2] = f;
      else          //进程 id 先从在下一层节点中将属于左分枝的
            //相应进程接收新的数据 g,再发送数据 f 给该进程
        X[m + h] = g;
        MPI_Recv(&g, id - h);
        m = m + h/2;
        X[m] = g;
        MPI_Send(&f, id - h);
      endif
    endif
  endfor
  X[m] = f; X[m + h] = g;
                    //蝶式计算后的新的数据 f, g 又放回 X 中
  MPI_Gather(p0);              //进程 p0 收集各进程的运算结果
end
```

算法中,MPI_Send(&f,p)表示执行该语句的进程发送数据f给进程p,这个函数的实际参数格式为MPI_Send(buf,

count,datatype,dest,tag,comm)[3],在算法实现时 buf 中是被发送的数据 f,datatype 指定的数据类型是 MPI 的复数型 MPI_COMPLEX,dest 指定接收数据 f 的目的进程 p。同理,MPI_Recv(&f,p)表示执行该语句的进程接收进程 p 发送的数据 f,它的实际参数格式为 MPI_Recv(buf,count,datatype,source,tag,comm,status),buf 存放所接收的数据 f,source 指定发送数据 f 的源进程 p。

3　算法分析及实验结果分析

3.1　算法分析

通常在分析并行算法的时间复杂度时既要考虑算法的计算时间,同时也要考虑并行进程之间的通信时间,并行算法总的时间复杂度应为计算时间复杂度加上通信时间复杂度。如果我们用 t 表示算法总的执行时间,$t_{计算}$ 和 $t_{通信}$ 分别表示算法的计算时间和进程间的通信时间,则 $t = t_{计算} + t_{通信}$。设 t 的单位为一个时间单位。

1)计算时间:根据算法,变量 f 和 g 做了 $level$ 次蝶式计算,其中,$level = \log_2(p)$,$p = n/2$。因此,$t_{计算} = O(\log(n))$。

2)通信时间:根据二叉胖树模型,在第一层,只有一个根节点,有 p 个进程,各进程间要进行 $p/2$ 次数据收发的通信,共有 $p/2$ 个通信时间。例如,在 16 点 FFT 中,p_0 要接收 p_4 的数据 f,同时 p_0 又要发送数据 g 给 p_4。p_1 与 p_5 之间、p_2 与 p_6 之间、p_3 与 p_7 之间也是一样。总共有 4 个通信时间。在第二层,有两个分枝节点,在每一分枝节点中有 $p/2$ 个进程,此时两个分枝节点可以同时进行各自节点中各进程间的通信,因此,在这一层中,共有 $p/4$ 个通信时间。根据此分析,在叶子节点中,就只有 1 个通信时间了。综上所述,算法总的通信时间为:

$$t_{通信} = 1 + 2 + \cdots + 2^{level-1} = 2^{level} = O(n)$$

所以,整个算法的执行时间为:$t = t_{计算} + t_{通信} = O(n + \log(n))$。

根据文献[1],串行运行时间复杂度为 $O(n\log n)$,因此得到加速比如下:

$$s = \frac{T_s}{T_p} = \frac{O(n\log n)}{O(n + \log n)} = O(\frac{n\log n}{n\log n}) = O(\log n) \qquad (4)$$

其中 T_s 是串行 FFT 算法的时间复杂度,T_p 是并行 FFT 算法的时间复杂度。根据公式(4)可以计算出该算法具有很好的加速比。

3.2　实验结果分析

在实验中,分别对 2 点、4 点、8 点、16 点、32 点、64 点、128 点、256 点及 512 点的 FFT 进行了测试,并且还与相应的串行算法进行了比较,得出了算法运行时间(单位:10^{-6}s)和问题空间 N 的折线图,如图 4 所示。

从图 4 中的折线可以看出,当问题空间比较小时(例如:在我们的实验环境下,当问题空间 N 小于 32 时),并行算法的执行时间大于串行算法的执行时间。这是因为算法的计算时间相对较短,而进程间的通信时间占了主导地位。当问题空间达到一定规模后,情况就有了变化,随着问题空间 N 的增大,并行算法的执行时间越来越小于串行算法的执行时间。

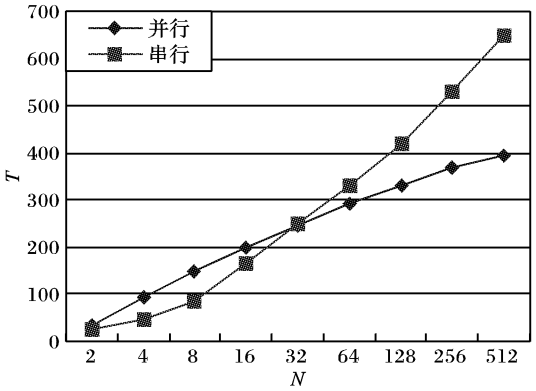

图 4　算法的实验数据

4　结语

二叉胖树模型原本是为了解决树形网络根部的通信瓶颈问题而提出来的,经过我们的研究发现,二叉胖树模型与 FFT 的蝶式计算在结构上有着极其相似的地方,因此,用二叉胖树模型来构建 FFT 蝶式计算的拓扑结构使得算法各进程间通信不会产生拥塞现象,同时又因为每个进程都负责着相等的蝶式计算任务,所以基于这种模型下的 FFT 算法能达到良好的负载平衡。

参考文献:

[1] 陈国良. 并行计算—结构、算法、编程[M]. 北京: 高等教育出版社, 2003.

[2] LEISERSON CE. Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing[J]. IEEE Transactions on Computers, 1985, 34(10): 892 - 901.

[3] QUNINN MJ. Parallel Programming in C with MPI and OpenMP [M]. 北京: 清华大学出版社, 2005.

[4] TAKAHASHI D, BOKU T, SATO M. A blocking algorithm for parallel 1-D FFT on clusters of PCs[A]. Proceedings of the 8th International Euro-Par Conference on Parallel Processing, LNCS 2400[C]. London, UK: Springer-Verlag, 2002. 691 - 700.

[5] TAKAHASHI D. A parallel 1 - D FFT algorithm for the Hitachi SR8000[J]. Parallel Computing, 2003, 29(6): 679 - 690.

[6] KIM Y, KWON O-Y, HAN T-D, *et al.* Design and performance analysis of the practical fat tree network using a butterfly network[J]. Journal of Systems Architecture, 1997, 43(1 - 5): 355 - 364.

(上接第 794 页)

参考文献:

[1] FLOYD S. TCP and explicit congestion notification[J]. ACM Computer Communication Review, 1994, 24(5): 10 - 23.

[2] GIBBENS RJ, KELLY FP. Resource pricing and the evolution of congestion control[J]. Automatica, 1999, 35(12): 1969 - 1985.

[3] JANAKI TM, GUPTE N. Connectivity strategies for enhancement of capacity of a load-bearing network[J]. Phys. Rev. E, 2003, 67(2): 021503.

[4] ARENAS A, DIAZ-GUILERA A, GUIMERA R. Communication in networks with hierarchical branching[J]. Phys. Rev. Lett, 2001, 86(14): 3196 - 3199.

[5] FUKS H, LAWNICZAK AT, VOLKOV S. Packet delay in data network models[J]. ACM Transactions on Modeling and Computer Simulation, 2001, 11(3): 233 - 250.

[6] KLEINROCK L. Queueing Systems, Vol. 1, Theory[M]. New York: Wiley, 1976. 102 - 248.

[7] CHEN H, JIN H, SUN JH. Analysis of large-scale topological properties for Peer-to-Peer networks[A]. Proceedings of International Symposium on Cluster Computing and the Grid (CCGird 2004)[C]. Los Angeles: IEEE Press, 2004. 27 - 34.