多核环境下边缘提取并行算法研究

张思乾 程 果 陈 荦 熊 伟
(国防科学技术大学电子科学与工程学院 长沙 410073)

摘 要 随着处理器由高主频的单核处理器逐步转向片上多核处理器(CMP),计算机并行处理能力不断提升。通过分析GIS串行算法面临的性能瓶颈,利用CMP的优势,采用线程级并行处理栅格数据。针对边缘提取算法,深入分析和比较了MPI、OpenMP等当前主流的并行编程模式,提出了并行性能估计模型。基于OpenMP编程模型分析线程数、调度方式和分块大小对算法并行性能的影响,实现边缘提取最优并行。实验证明,性能评估模型能够准确预测CMP环境下的并行性能,基于OpenMP实现的边缘提取并行算法能够提高图像边缘提取效率。

关键词 片上多核处理器,OpenMP,边缘提取
中图法分类号 TP311 **文献标识码** A

Research on Parallel Algorithm of Edge Extraction Based on Multi-processor

ZHANG Si-qian CHENG Guo CHEN Luo XIONG Wei
(Department of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract As the development of microprocessors from uniprocessors with high frequency to chip multiprocessors (CMP), the ability of parallel processing of computers is advancing. Through analyzing the performance bottlenecks of the serial algorithm of GIS, raster data was paralleled by multithreads based on the advantage of CMP. Parallel programming models were analyzed and compared to build parallel performance estimating models. Based on OpenMP, the parallel performance could be improved furthest by choosing appropriate parameters. The experiment results show that the parallel performance estimating model can be used to forecast the parallel performance exactly and using OpenMP has an advantage over MPI in CMP environment. The parallel algorithm of the edge extraction based on OpenMP can significantly improve the efficiency of the image edge extraction.

Keywords Chip multi-processor, OpenMP, Edge extraction

1 引言

从1850年Charles Babbage[1]第一次提出并行计算的想法,到如今并行计算机的飞速发展,并行计算技术已经成为解决计算密集型和数据密集型问题的重要手段。计算机的核心——处理器的发展趋势开始从传统的高主频单核处理器转向具有并行计算能力的片上多核处理器[2,3](Chip Multi-Processor,CMP),从依靠高主频、指令级并行层次提高处理器性能转向线程级并行(Thread Level Parallelism,TLP),以多核处理器为主流的新一代处理器体系结构已经占据了处理器发展的主导地位。

新一代处理器技术的典型代表——CMP技术最初出现在20世纪90年代末,2001年IBM推出了第一款多核架构的处理器——双核Power 4,用于IBM的RISC(Reduced Instruction Set Computer)服务器。之后,几乎所有的RISC处理器都实现了从单核向多核的技术革新。AMD和英特尔在2004年公布了各自的双核计划,2005年4月相继推出双核产品,揭开了X86处理器的多核心时代。X86处理器采用多核体系结构标志着处理器多核时代的到来。相比于传统的对称式多处理器(Symmetrical Multi-Processing,SMP),CMP不但从硬件成本上更易被采纳,而且由于芯片内的通信速度大大高于芯片间,其采用共享二级(及二级以上)缓存,使并行效率更高于SMP系统。因此,CMP这种新型多核处理器提供了更高的性价比,为计算需求量更大的算法提供了更好的并行性能。

CMP计算机的普及使得并行技术的实现更为简单,单独一台CMP计算机即可组成具有并行计算能力的硬件环境。越来越多的计算密集型和数据密集型问题在CMP环境下获得更高的并行性能,其中栅格数据[4,5]的并行处理是近年来的研究热点之一。一方面,栅格数据具有数据规模巨大、处理流程复杂的特点,采用并行计算技术来减少大规模栅格数据的处理时间就尤为迫切和必需。另一方面,栅格数据的矩阵

到稿日期:2011-05-29 返修日期:2011-07-02 本文受国家自然科学基金(61070035,60902036,40801160),高等学校博士学科点专项科研基金(20104307110017),国家高技术研究发展计划(863计划)课题(2011AA120306)资助。
张思乾(1987-),女,硕士生,主要研究方向为并行计算、高性能计算、地理计算等,E-mail:zsq19871212@163.com;程 果(1984-),男,博士生,主要研究方向为高性能计算、空间信息处理;陈 荦(1973-),男,博士,教授,主要研究方向为高性能计算、地理空间信息处理、地理信息系统等;熊 伟(1976-),男,博士,讲师,主要研究方向为空间数据库、地理信息系统等。

式存储结构更易于并行化。

本文通过分析当前主流并行编程模式在 CMP 环境下的并行性能，选择更优的并行编程模式对栅格图像边缘提取进行并行化实现。在算法并行化的过程中，全面分析线程数、调度方式和分块大小对并行性能的影响，实现 CMP 环境下边缘提取的最优并行算法。

2 并行编程模式分析

任何并行算法最终都要通过并行编程在具体的并行机上实现，不同的并行机的体系结构也需要采用不同的编程模式。目前已有多种并行算法编程模式，主流的并行编程模式依然是共享存储模式和消息传递模式，OpenMP 和 MPI 是其广泛使用的并行应用编程接口(API)。

独立 CMP 计算机拥有共享二级(及二级以上)缓存的多个计算核心，可同时访问的共享内存，线程间不需要专门的消息传递，通过访问共享内存存储的变量完成彼此的消息传递。基于这样的硬件平台，分析 MPI 与 OpenMP 并行编程模型的并行性能。

根据参考文献[6]，提出不同并行编程模式影响并行性能的参数，总结出如下数学计算公式。

基于消息传递的并行编程模式运行时间包括计算时间和通信时间，即

$$T=T_{\text{compute}}+T_{\text{communicate}}$$

其中，

$$T_{\text{communicate}}=T_{\text{init-send}}+T_{\text{message-pass}}+T_{\text{init-receive}}$$

通信时间包括发送初始化时间、接收初始化时间和消息传递时间。

基于共享存储的并行编程模式运行时间包括并行时间和串行时间，即

$$T=T_p+T_s$$

并行时间 T_p 为

$$T_p=t_{fork}+\sum_{i=1}^{n}t_{p_i}+t_{join}+t_{cache}+t_{compile}$$

$$t_{p_i}=\max\{t_{t1},t_{t2},\cdots,t_{tn}\}$$

式中，t_{fork} 为划分子线程的时间，t_{join} 为各子线程合并的时间，t_{cache} 为缓存优化时间，$t_{compile}$ 为编译器优化时间，t_{p_i} 为各子线程计算时间最大值。

串行时间 T_s 为

$$T_s=\sum_{j=1}^{m}t_{t_s}$$

无论是串行还是并行程序，一般来说，最大的开销都是内存读取和通信时间。其中的通信时间占更大比重，特别是对海量的遥感栅格数据进行处理，其数据量大的特点导致通信时间更长。由上述两种并行编程模式的性能计算公式可以看出，针对单机 CMP 环境中各处理器拥有共享缓存的特性，数据可共享的存储于主存储器中，采用 OpenMP 编程模式避免了采用 MPI 编程模式时必不可少的消息传递时间，各线程直接读取共享内存中的数据进行计算处理，大大减少了并行运行时间。因此，基于单机 CMP 环境采用 OpenMP 编程模式比 MPI 编程模式更具有优势。

3 CMP 环境下基于 OpenMP 的边缘提取并行算法研究

3.1 图像边缘提取算法分析

图像锐化技术的重要应用之一是图像边缘提取。采用 Sobel 算子[7]，利用两行或两列加权和的差值，不像普通梯度算子用两个像素的差值。Sobel 算子具有一定的方向性，其有两个优点：引入了平均因素，因而对图像中的随机噪声有一定的平均作用；它是相隔两行或两列的差分，使边缘两侧的元素得到了增强，故边缘显得粗且亮。

Sobel 算子为

$$G[f(i,j)]=\{G_x[f(i,j)]^2+G_y[f(i,j)]^2\}^{1/2}$$

或

$$G[f(i,j)]=|G_x[f(i,j)]|+|G_y[f(i,j)]|$$

其中

X,Y 方向算子为 $G_x=\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}$，$G_y=\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$。

3.2 线程数对并行性能的影响

线程[8]是程序流程中的单一控制流。采用 OpenMP 并行编程模型，子句 num_threads(整数表达式)指定由 parallel 命令创建的线程数目。线程数目对并行算法性能的影响与 CMP 计算机的处理器数目有关。

3.2.1 并行运行时间

并行算法的运行时间[8,9]是最基本的性能度量。

(1)当线程数小于处理器数时，随着线程数的增加，并行运行时间不断减少。因为此时分块数与线程数相同，将划分后的数据块分给各处理器并行处理后，存在处理器未获得数据分块而处于空闲状态，处理器未充分利用的情况。随着线程数的增加，分块数随之增加，参与计算的处理器数增加，数据块所含数据量不断减少，那么各处理器并行处理数据块的时间也相应减少，即并行运行时间减少。

(2)当线程数等于处理器数时，并行运行时间达到最小。按线程数(即处理器数)均匀划分数据，将数据块分配给各处理器。此时没有空闲处理器，充分利用了处理器资源，且数据的均匀划分保证了各处理器计算负载均等，因此并行运行时间达到最小。

(3)当线程数大于处理器数时，以处理器数的整数倍为周期，在线程数比处理器倍数多 1 时，达到该周期内最大并行运行时间；随着线程数的增加，并行运行时间减少。随着周期的增加，分块数不断增加，会逐渐均衡各线程分块的数据量差异，并行运行时间最终保持稳定状态。

3.2.2 加速比和效率

加速比和效率[9]体现了在并行计算机上运行并行算法求解实际问题所能获得的实际效能，它是最传统的并行算法评价标准。

(1)当线程数小于或等于处理器数时，加速比与处理器数成正比，具有线性加速比，则并行效率保持较优状态。

(2)当线程数大于处理器数时，加速比与并行运行时间成反比，最终将保持稳定；参与计算的处理器数固定，则并行效率与加速比成正比。

3.3 调度策略对并行性能的影响

OpenMP 中的 schedule 子句用来将迭代分配给线程，支持 static、dynamic 和 guided 调度[6,8]，其命令的一般形式为 schedule(scheduleing_class[,parameter])。

(1)当分块数小于线程数时，随着分块数的增加，并行运行时间不断减少。因为将划分后的数据块分给各线程并行处理，存在未获得分块数据的线程处于空闲状态的情况。随着

分块数的增加，每数据块的数据量减少，并行处理时间也减少，但又不超过线程数，即每个线程只处理一块数据，因此并行运行时间相应减少。

(2)当分块数大于线程数时，并行运行时间成周期性的振动。假设 $N_{chunks}=N_{threads}*n+k$，$n=1,2,\cdots,k$ 为小于 $N_{threads}$ 的整数。

当 $k=0$ 时，即分块数为线程数的 n 倍，达到该周期内并行运行时间最短。此时，每个线程分得相同数据量的数据，各线程运行时间相等，即并行运行时间。由于 static 调度和 dynamic 调度是均匀分块，各线程分配的数据量叠加后相等，因此，n 的取值不同不会影响并行运行时间；而 guided 调度是非均匀分块，块的大小按指数减小，若 n 的取值不同，各线程分配的数据量叠加后不等，因此，n 的取值会影响并行运行时间，随着 n 的增加，并行运行时间也略有增加。

当 $k=1$ 时，达到该周期内最大并行运行时间，之后随着分块数增加，并行运行时间减少。因为 $k=1$ 时，各线程数据负载差异最大，k 增加数据负载差异减小。随着 n 的增加，分块数不断增加，会逐渐均衡各线程分块的数据量，周期内振动幅度逐渐减小，并行运行时间最终将趋于稳定状态。

4 实验与结果分析

4.1 实验环境

(1)实验硬件环境：

CPU：Intel(R) Core(TM) i5-2300，2.80GHz，物理四核；

内存：DDR3 4GB；

硬盘：7200 转 SATA 500G。

(2)实验软件环境：

操作系统：Linux(fedora 14)；

编译环境：GCC 4.5.1；

IDE：Eclipse Helios。

(3)实验数据：4 种数据集(见表 1)。

表 1 实验数据集

数据集	数据大小	数据规模
数据集 1	520MB	21000 * 13000
数据集 2	154MB	18020 * 9010
数据集 3	47.9MB	10020 * 5010
数据集 4	68.7MB	6001 * 6001

4.2 OpenMP 和 MPI 性能对比实验

采用 MPI 并行算法，设置 4 个进程，其中主进程控制读写和数据分发，将图像分成 3 块，分发给 3 个子进程进行并行计算。OpenMP 并行算法，将图像划分成 3 块分配给 3 个线程进行并行计算。两种并行算法的运行时间与串行算法运行时间比较如图 1 所示。

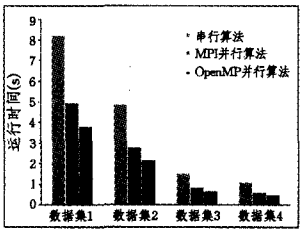

图 1 OpenMP、MPI 并行算法性能比较

实验结果分析：

由图 1 可见，数据集规模大小对并行算法性能无影响，实验结果与第 2 节的理论分析相符。

两种并行算法对串行算法都有很好的优化，其运行时间大大减少。同时 OpenMP 并行编程模型比 MPI 并行编程模型并行运行时间更短，有更优的并行性能。

采用 MPI 编程模型不仅需要在开始时将读取的数据分发给各子进程，在计算过程中还需要将计算的中间结果在各进程间进行消息传递，这种分发数据和接收数据的时间大大降低了并行效率和性能。

采用 OpenMP 编程模型就避免了这些时间消耗，其读取的数据和计算的中间结果都可存储于共享内存中，各线程均可直接读取调用。增加的 fork 和 join 时间损耗很短，且通过编译器和缓存的优化可进一步缩短运行时间，提高计算性能。

4.3 OpenMP 中线程数对并行性能的影响

对边缘提取算法进行 OpenMP 并行化，采用 4 组不同规模的数据集进行实验。对每组数据集，改变线程数(从 2 个线程到 12 个线程)，测试线程数对并行性能的影响，实验结果如图 2、图 3 所示。

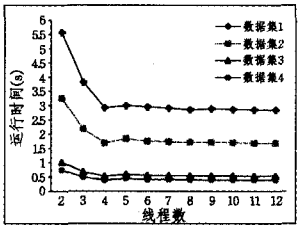

图 2 线程数对并行运行时间的影响

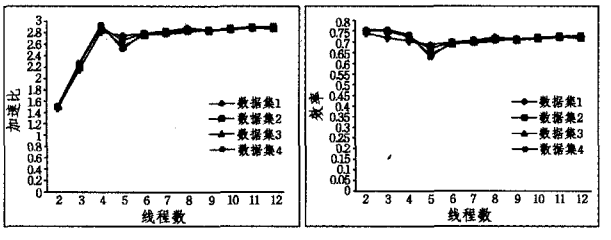

图 3 线程数对加速比和效率的影响

4.3.1 并行运行时间

实验结果分析：

由图 2 可见，数据集规模大小对实验结果无影响，实验所用的是物理四核的硬件环境，实验结果与 3.2 节的理论分析相符。

(1)当划分小于 4 个线程时，有空闲的处理器未被利用，随着线程数的增加，更多地利用处理器资源，并行运行时间减少，性能不断提升。

(2)当划分为 4 个线程，即与处理器数相等时，每个处理器处理一个线程数据，所有处理器充分利用，且每个线程均等计算，运行时间达到最小值，并行性能达到最优。

(3)在划分为 5、6、7 个线程时，运行时间增加，因为划分的线程无法被处理器均分，各线程数据负载不同。5 个线程时，负载最不均衡，所以运行时间最大，与理论分析相符。

(4)随着线程数的不断增加，运行时间最终趋于稳定状态。

由于实验所用数据规模不够大，因此在第二个周期基本就保持了稳定状态。若采用更大数据，则应有随着周期的增加，波动逐渐减小的趋势。

4.3.2 加速比和效率

实验结果分析：

由图3可见，数据集规模大小对实验结果无影响，与3.2节的理论分析相符。

在达到4个线程之前，即当线程数小于处理器数时，加速比与处理器数成正比，具有线性加速比，并行效率保持较优状态，最高并行效率可达到75%以上。

4.4 OpenMP中调度策略对并行性能影响

采用表1中的数据集1，设定4个线程，分别对static调度、dynamic调度和guided调度进行实验。对每种调度方式，改变其分块大小(分块数从2块到24块)，测试调度策略对并行性能的影响，实验结果如图4所示。

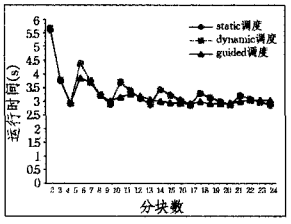

图4 调度策略对并行性能的影响

实验结果分析：

由图4可见，3种调度方式的实验结果与3.3节理论分析结果相符合。

分块数以线程数的整数倍为周期，即

分块数=线程数$*n+k$，$n=1,2,3\cdots$，$k=0,1,2,3$

(1) 当$k=0$时，即分块数为线程数的整数倍，运行时间最短；

(2) 当$k=1$时，划分给各线程的块最不均匀，运行时间为该周期内最长，随着k的增加，可逐渐均化各线程的负担；

(3)随着n的增加，波动的起伏越来越小，最终趋于稳定。

由于图像边缘提取实验中，各线程做相同的计算，因此对于相等数据量处理的时间也相同。dynamic调度与static调度都是将迭代空间分割成大小相等的块，则这两种调度下的各线程获得的数据量相同，所以两种调度的运行时间线完全重合。同时因为各线程进行相同计算，在分块数为线程数的整数倍时，各线程处理的数据不同，但处理的数据量相等，因此运行时间也相等。

采用guided调度类，在计算的进行过程中减少块的大小，从而更充分地均化各线程的数据处理量，因此在分块数不等于线程数时，可比static和dynamic调度达到更优的并行性能。

结束语 CMP技术实现了一块CPU芯片上可集成多个计算核心，具有共享的二级(及二级以上)缓存，使得单独一台CMP计算机即可构成最简单的高性能计算环境，无论从性能上还是成本上都体现出了很强劲的优势。OpenMP并行编程模型是基于共享内存架构的线程级并行，通过对共享内存的读写完成线程间通信，更适用于CMP计算机这种新型硬件环境。OpenMP编写多线程程序简便高效，而且提供了多种调度策略，对同步共享变量、合理分配负载等任务都提供了有效的支持。理论分析和实验结果均表明，基于OpenMP的并行算法，通过对线程数、调度方式和分块大小的选择，可以最大限度地提升并行性能，获得更高的加速比和并行效率。本文的研究基于边缘提取算法，但其中并行优化方面的理论研究成果可广泛应用于栅格数据并行处理领域，对通用栅格数据并行处理研究具有很好的推动作用。

参考文献

[1] Kuck, David J. High Performance Computing: Challenges for Future Systems[M]. New York, NY: Oxford University Press, 1996

[2] Hennessy J L, Patterson D A. Computer Architecture (4th edtion)[M]. Morgan Kaufman, 2007

[3] Olukotun K, Hammond L, Laudon J. Chip Mutiprocessor Architecture: Techniques to Improve Throughput and Latency[M]. Morgen & Claypool Publishers, 2007

[4] Guan Qingfeng, Clarke, Keith C. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model[J]. International Journal of Geographical Information Science, 2010, 24(5): 695-722

[5] Bernhardsen T. 地理信息系统导论(第三版)[M]. 王泌，李浩川，译. 北京：机械工业出版社，2006

[6] Liao Chunhua. A Compile-time OpenMP Cost Model[D]. US: University of Houston, 2007

[7] 孙即祥. 图像处理[M]. 北京：科学出版社，2004

[8] Grama M, et al. 并行计算导论(第二版)[M]. 张武，毛国勇，等译. 北京：机械工业出版社，2005

[9] 孙世新，卢光辉，等. 并行算法及其应用[M]. 北京：机械工业出版社，2005

(上接第260页)

[23] Wang S, Zhu W. Matroid theory in covering-based rough sets [J]. To appear in International Journal of Granular Computing, Rough Sets and Intelligent Systems, 2011

[24] 胡军，王国胤，张清华. 一种覆盖粗糙模糊集模型[J]. 软件学报，2010(05): 968-977

[25] Gau W, Buehrer D J. Vague sets[J]. IEEE Transactions on Systems, Man and Cybernetics, 1993, 23(2): 610-614

[26] 闫德勤，迟忠先. 粗糙集与Vague集[J]. 计算机科学，2004(08): 133-135

[27] 徐久成，张倩倩. 覆盖粗糙Vague集的不确定性度量研究[J]. 计算机科学，2010(10): 225-227

[28] Bonikowski Z, Bryniarski E, Wybraniec-Skardowska U. Extensions and intensions in the rough set theory[J]. Information sciences, 1998, 107(1-4): 149-167

[29] Zhu W, Wang F Y. Reduction and Axiomization of Covering Generalized Rough Sets[J]. Information Sciences, 2003, 152: 217-230

[30] Zhu W. Relationship between generalized rough sets based on binary relation and covering[J]. Information Sciences, 2009, 179(3): 210-225

[31] Zakowski W. Approximations in the Space(U, Π)[J]. Demonstratio Mathematica, 1983(16): 761-769

[32] Zhu W, Wang F. A new type of covering rough set[C]. London, United kingdom: Institute of Electrical and Electronics Engineers Inc. 2006