文章编号：2096-1618(2018)01-0013-05

基于4种并行模式的快速排序算法

张天阳， 陈 华
(中国石油大学(华东)理学院，山东 青岛 266580)

摘要：快速排序作为一种先进的排序算法，以其优异的性能广泛应用于众多领域。然而，传统的串行快速排序算法是单线程模式进行，不能充分利用CPU的多个线程。针对上述问题，基于包括Windows API、OpenMP、MPI和PPL的4种并行模式，设计了一种并行的多线程快速排序算法，对并行计算容易出现的数据竞争问题给出解决方案。设计的并行快速排序算在多线程计算机上进行实验并与单线程算法进行比较，结果表明并行快速排序算法减小了时间开销，达到了比较理想的线性加速效果。

关 键 词：快速排序；并行算法；Windows API；OpenMP；MPI；PPL
中图分类号：TP302.7 **文献标志码**：A
doi：10.16836/j.cnki.jcuit.2018.01.003

0 引言

排序算法在统计学、计算机科学、人工智能、机器学习、高性能计算和大数据等领域具有广泛的应用[1]。从计算机诞生至今，虽然已经研究出几十种排序算法，但是传统的串行排序算法在单核CPU上是顺序执行的，在多核CPU上运行时只能利用其中的一个线程，并不能充分发挥多核CPU的优势，必然运行速度慢、效率低。随着多核技术的发展与普及，越来越多的计算机设备使用多核处理器，高性能并行计算能提高计算机性能并节省存储空间，对串行排序算法作多线程优化以进一步提高排序性能成为越来越多科研人员研究的方向。

陈国良[2]给出一种快速排序并行化的简单思想，在MPI群集系统的基础上使用2^m个处理器完成了对n个输入数据的排序，分析了各种情况下的时间复杂度，指出正常情况下该算法具有更高的常数因子。基于该思想，黄伟等[3]分析快速排序并行化过程并用MPI实现，提出如何选择枢轴是提高快速排序性能的关键。游佐勇等[4]提出基于多核技术的OpenMP并行编程模型的快速排序算法，但是没有考虑同步问题，在某些情况下可能会出现排序错误。刘向娇等[5]给出对常用的一种并行快速排序算法进行改进的思路。王帅等[6]给出另一种简单的并行化的思路。除此之外，石壬息等[7]利用Win32线程函数，给出快速排序算法的多线程实现方法。张火林等[8]在C#线程编程基础上实现基于双核系统的多线程的快速排序算法，并行后的效率有所提升，但要受到核心数的影响。李跃新等[9]在PRAM-CRCW模型上将执行快排序可以看成是构造一棵二叉树，采用中序遍历的方法得到一个有序序列，但在数据较大时，实际应用不具备可行性。

以排序算法中的快速排序算法为例，基于Windows API编程、OpenMP共享内存的应用程序编程接口[10]、MPI消息传递模型[11]和PPL并行模式库4种并行模式，同时考虑数据同步问题，在多核计算机上将串行快速排序算法并行化，并将排序任务有效地分配到不同的线程中去，多线程同时执行。对随机生成的较大规模的无序数据序列进行排序，记录各个并行模式的串并行加速比，分析其优缺点，最终实验结果表明并行后的快速算法效率可以达到线性加速。

1 快速排序的串行算法描述

快速排序算法是由C. A. R. Hoare[12]在1962年提出，是一种依赖于数据比较从而对无序序列进行递归排序的算法。其基本思想是：通过一趟排序将待排序列分割成独立的两部分，其中一部分元素的关键字均比另一部分元素的关键字小，则可分别对这两部分序列继续进行排序，以达到整个序列有序。

设待排序序列为Data[1,2,…,*length*]，首先任意选择一个元素(可选择第i个元素)作为枢轴，然后将所有关键字比它小的元素都安置在它的位置前，将所有关键字比它大的元素都安置在它的位置后。由此以该“枢轴”元素最后所落的位置i作为分界线，将序列Data[1,2,…,*length*]分割成两个子序列Data[1,2,…,$i-1$]和Data[$i+1$,$i+2$,…,*length*]。这个过程称为一趟

收稿日期：2017-06-29
基金项目：国家自然科学基金资助项目(41474100)；山东省自然科学基金项目(ZR2013DM015)

快速排序。

2 并行算法描述与设计

2.1 算法描述

并行快速排序算法在计算的时候,CPU 会将排序任务划分到不同的线程上去,在每个单独的线程上调用串行的快速排序,基于一种简单的思想[2],结合图 1 给出优化并考虑同步问题后的算法思路:

(1)准备,设序列总长度为 *length*,设置系统调用的线程数 *numthreads*,创建 *no* 个线程,这里 *no* = *numthreads*。

(2)分段排序,将待排序序列 Data[1,2,…,*len*] 平均分割为相同个数的短序列,则短序列的长度为 *len* = *length*/*numthreads*,第 *i* 个序列为 Data[(*i*-1)×*len*+1,(*i*-1)×*len*+2,…,*i*×*len*],其中 *i* = 1,2,…,*numthreads*。每个短序列依次首尾相接,各线程根据对相应的短序列调用串行快速排序算法,得到一个分段有序序列。考虑不同线程之间结束排序任务的顺序不同,为防止数据竞争,应当设置同步,使排序不出错。

(3)合并,判断 *len* 是否等于 *length*,即有序的短序列是否已经合并成原序列长度大小的序列。如果是,则算法停止;如果不是,继续合并第 *i* 个序列和第 *i*+1 个序列,其中 *i* = 1,3,5,*numthreads*-1。由于每合并一次,短序列的个数就减小一半,相应地,调用的线程数也减小一半,每个线程执行合并函数,使相邻的两个短序列合并成一个短序列,此时也应该设置同步,防止数据竞争。合并结束后,原序列已经是一个有序序列,排序算法结束。

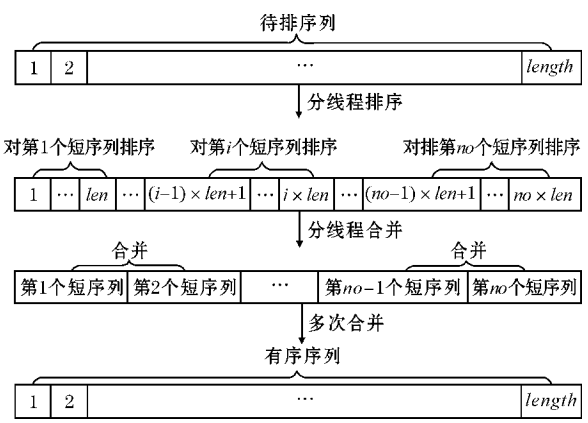

图 1 并行快速排序设计

2.2 基于 Windows API 的并行算法设计

Windows API 是操作系统为内核及应用程序之间提供的接口,以此定义线程函数可实现多核编程。基于此类并行模式的算法思路如下:

(1)创建线程并传递参数。主线程设定并行计算所需的线程个数 numthreads,调用 Windows API 的 CreateProcess 函数创建相同数量的线程,同时将计算排序区间下标所需要的参数结构体传入到创建好的线程中运行。

(2)系统将待排序序列的划分成 numthreads 个相邻的短序列,各线程根据线程号计算对应短序列的起始结束下标,在相应的短序列内调用串行快速排序函数进行小范围的分区排序,使整个序列变为局部有序序列。

(3)利用 CreateProcess 函数创建一半的线程数,每调用一次合并函数合并两个相邻的短序列,就减少一半线程总数,直到所有短序列合并为一个有序序列。

程序在使用多线程时,各个线程同时运行,由于资源分配不均等原因,往往每个线程执行的速率是不同的,这就使线程任务完成的次序有差别。一般情况下,必须等待运行速率最慢的线程运行完毕后,才能在其处理结果的基础上进行下一步操作否则,主线程会在线程处理任务结束前就去访问处理结果,使最慢的线程来不及执行其余语句而随主线程的结束而结束。为避免此类现象发生,添加 WaitForSingleObject 函数,让主线程等待从线程执行完所有语句,再继续执行后面的语句。

算法流程图如图 2 所示。

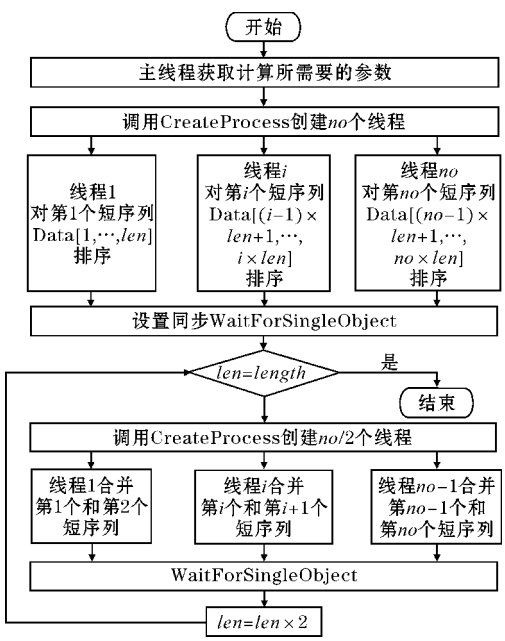

图 2 基于 Windows API 的算法流程图

2.3 基于 OpenMP 的多核并行算法设计

OpenMP 是一个共享储存并行系统上的应用编程

接口,由编译制导、运行库例程和环境变量 3 大部分组成。OpenMP 具有简单、移植性好和可扩展等优点,同时提供了 Fortan 和 C/C++等的应用编程接口,已经应用于多种平台[2]。算法的设计如下:

(1)调用 omp_set_num_threads 函数设置线程数 numThreads,利用 parallel 指令将代码复制到 *no* 个线程上执行。

(2)将排序任务划分到不同的线程中去,每个子线程调用 omp_get_num_threads 函数获取当前线程号。主线程调用 omp_get_num_threads 函数获取当前并行的线程个数,各线程计算对应序列的起始结束下标,从而将数据序列平均切割成几个短序列,对每个序列调用串行快速排序函数进行排序。这样,在所有子线程排序结束后,整个序列变成一个局部有序序列。

(3)利用 parallel for 指令将循环所需工作量按一定的方式分配到 *no*/2 个线程上执行,进行两两合并,最后得到一个完整的有序数据序列。

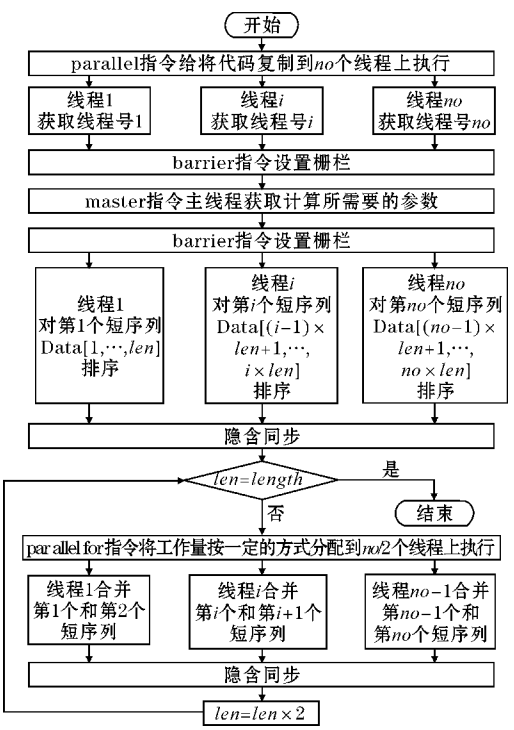

图 3　基于 OpenMP 的算法流程图

OpenMP 是用于共享内存并行系统的多线程程序设计,通过共享内存空间上的存储单元进行线程之间的数据传递,从而完成线程之间的通信。若多个线程对同一存储单元进行写操作,就会产生数据竞争。为避免数据竞争的发生,通常使用 private 子句将一个或多个变量声明为线程的私有变量,其他线程无法访问;用 shared 子句将一个或多个变量声明为共享量,所有线程都可以访问。另外,算法中,有些代码必须在其他代码执行完毕后才能执行,因此引入事件同步机制避免上述问题的发生。barrier 指令(又称栅栏)可以使线程在遇到栅栏时必须等待,直到并行域内所有线程都到达了同一栅栏,才能继续执行,从而避免了数据竞争问题。

算法流程图如图 3 所示。

2.4　基于 MPI 的多核并行算法设计

MPI(message passing interface)是一种基于消息传递的并行编程技术,消息传递接口是一种编程接口标准,而不是一种具体的编程语言。执行 MPI 程序时,各进程之间主要进行消息传递工作,将数据放入缓冲区发送再接收[2]。

在设计 MPI 并行模式算法时需要注意,与其他并行模式不同,MPI 模式是全局并行,需要将串行的部分放入根进程,而并行的部分则放在根进程之外,其基本设计思路如下:

(1)初始化 MPI 环境和通信域,获得当前进程号和进程数。由根进程生成数组,返回全局并行模式。

(2)根进程根据调用的进程数计算每个短序列的长度,开辟同等大小的内存空间。利用 MPI_Scatter 函数将原序列划分后的短序列依次撒播到不同的进程上。每个进程分别生成同等数量的小数组,对其进行排序。利用 MPI_Gather 函数重新收集排好的数组,至此,整个数组变为局部有序序列。

(3)利用 MPI_Scatter 函数将原数组的各个小区间依次撒播到不同的进程上,与排序时的撒播不同,由于合并使数组个数减小,因此每次分配不同长度的新数组。另外,每次合并也不需要全部进程参与,只需要撒播给部分进程即可。合并完成后,利用 MPI_Gather 函数重新收集合并好的数组,至此,整个数组变为有序序列。

(4)结束 MPI 系统。

同步功能主要协调各个进程之间的进度和步伐,利用 MPI_Barrier 函数使调用进程阻塞,直到通信内所有进程都调用它,保证组内所有的进程都已经执行完调用之前的所有操作,然后开始该调用后的操作。

算法流程图如图 4 所示。

2.5　基于 PPL 的多核并行算法设计

并行模式库(parallel patterns library,PPL)为执行细粒度并行操作提供通用的内容和算法。并发运行时定义的许多类型和算法都采用 lambda 表达式作为执行工作的历程。PPL 中的 parallel_for 函数按照最佳方式对任务进行分区,无需指定并行的线程数,对于给出快速排序算法,可以给定划分短序列的个数,CPU 会自动对任务进行分配。

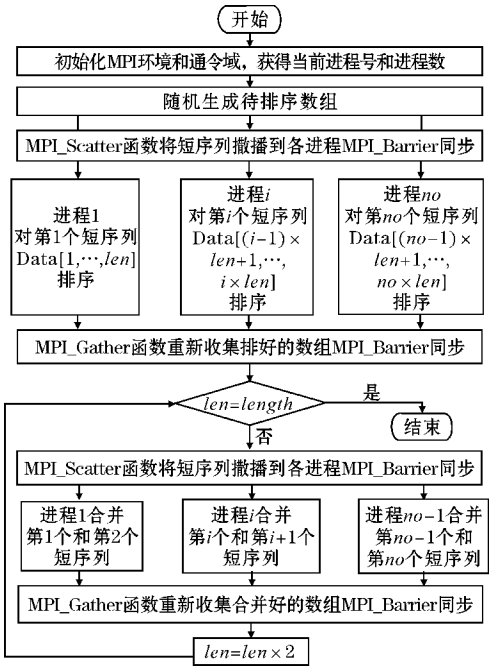

图 4 基于 MPI 的算法流程图

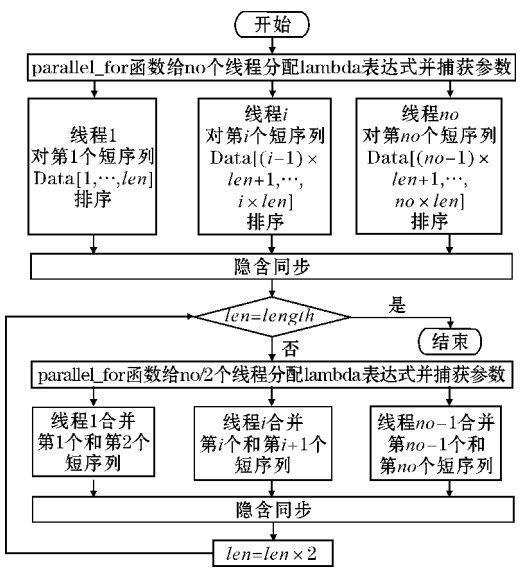

图 5 基于 PPL 的算法流程图

parallel_for 函数隐含同步，能有效解决数据竞争问题。

算法流程图如图 5 所示。

3 实验和结论

3.1 实验环境

实验环境为 Dell Xps13 9360 系列双核四线程笔记本电脑，采用 Intel Core i5-7200 CPU，内存 8G。操作系统 Microsoft Windows 10 Professional，编译器为 Microsoft Visual Studio 2010，编程语言为 C 语言。

3.2 实验规模与数据设置

实验将待排序列的规模依次设定为 100 万、500 万和 2000 万，每次调用 2、4 和 8 个线程执行，PPL 模式自动获取线程数，没有执行 8 线程计算。

实验数据均采用 C 语言 rand 函数生成随机大小的整数，消除因生成的数据序列有序而产生的实验误差。

3.3 实验结果与分析

对于串并行化快速排序算法，用单线程和多线程方式分别对不同数量的随机数据序列进行排序，并对排序所各自花费的平均时间开销进行对比，进而判断不同算法性能的优劣。实验分别运行 5 次，排序的并行加速比图如图 6 所示。

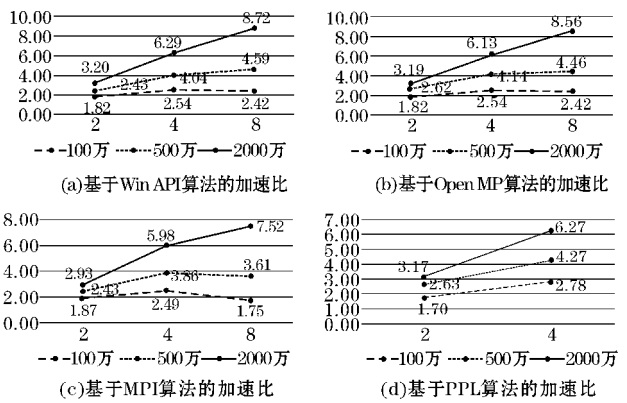

图 6 4 种算法的并行加速比图

从图 6 可得如下结论：

(1)同等规模下，随线程数的增加，并行加速比增加，加速效果大致成线性。当调用线程数大于系统调用最优线程数时，MPI 算法的加速比反而下降，主要原因是通信开销过大。

(2)同种算法下，调用相同的核数，待排序列的规模越大，加速比越大。

(3)4 种方法由于思路一致，同等规模下调用相同的线程数，加速比大致相等。

4 结束语

随着 CPU 多核、众核时代的到来，未来计算机科学将朝着高效、快速的方向发展。设计的基于 4 种并行模式的快速排序算法，将任务均衡地划分到不同的线程中，实现多线程方式并行运行程序，从而提高了处理器的利用率，节省时间开销。

相比于现有算法，文中设计算法的一个显著特点是在并行计算中格外注意数据相关性问题，合理设置

同步,有效防止数据竞争造成的排序错误,确保了排序结果的正确性。

参考文献:

[1] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to Algorithms[M]. MIT Press, 2001.

[2] 陈国良. 并行算法实践[M]. 北京:高等教育出版社,2004.

[3] 黄伟,赖国明. 基于群集系统的快速排序并行化方法[J]. 现代计算机(专业版),2005,(5):89-91.

[4] 游佐勇,罗省贤. 多核计算环境下快速排序并行算法的实现[J]. 电脑与电信,2011,(1):60-62.

[5] 刘向娇,赵学武. 改进的并行快速排序[J]. 计算机与数字工程,2014,42(5):782-784.

[6] 王帅,喻歆,何嘉. 基于 MPI 和 OpenMP 的排序算法并行优化研究[J]. 成都信息工程大学学报,2016,31(3):277-284.

[7] 石壬息,张锦雄,王钧,等. 快速排序异步并行算法的多线程实现[J]. 广西科学院学报,2005,(s1):54-55,65.

[8] 张火林,李国庆,张江维. 基于双核系统的快速排序效率分析[J]. 电脑知识与技术,2008,3(8):705-707.

[9] 李跃新,周四维. 并行计算中快速排序算法的改进[J]. 湖北第二师范学院学报,2011,8(8):1-3.

[10] 周伟明. 多核计算与程序设计[M]. 武汉:华中科技大学出版社,2009.

[11] 周恩强,赵军锁,杨学军. MPI 及 MPI 的高效实现[J]. 计算机工程与科学,1999,21(5):47-51.

[12] 严蔚敏,吴伟民. 数据结构(C 语言版)[M]. 北京:清华大学出版社,2010.

Quick Sort Algorithm based on Four Parallel Patterns

ZHANG Tian-yang,　CHEN Hua

(College of Science of China University of Petroleum (East China), Qingdao 266580, China)

Abstract: As an advanced sorting algorithm, quick sort is widely applied in many fields with its excellent performance. However, the traditional serial quick sort algorithm works on the single thread mode, and it cannot make full advantage of multi-threads of CPU. As to the problem mentioned above, a kind of multi-threads parallel quick sort algorithm is designed and the solution to the data race which is likely arising in parallel computing is given, based on four parallel patterns including Windows API, OpenMP, MPI and PPL. The designed parallel quick sort algorithm is evaluated in a multi-threads computer compared with the single thread algorithm. The results indicate that the parallel quick sort algorithm reduces the time overhead and achieves an ideal linear acceleration effect.

Keywords: quick sort; parallel algorithm; Windows API; OpenMP; MPI; PPL