基于多核的多线程程序优化研究

施惠丰,袁道华
(四川大学 计算机学院,四川 成都 610065)

摘 要:随着主流芯片厂商的大力推广,多核处理器已经变得越来越普及。以往串行化的程序设计方法在多核环境下已经不能充分利用多核CPU的资源。怎样高效地利用多核处理器的计算性能,已经成为软件开发者面临的新的课题。文中在传统的多线程编程基础上,根据Intel处理器的微架构(Microarchitecture)特点,以及Linux内核提供的CPU绑定技术,通过采用Cache优化和CPU亲和力(CPU affinity)优化,消除了多核环境下局部多线程Cache行竞争和伪共享,减少了线程的调度开销,提高了多线程程序的运行效率。

关键词:多核处理器;多线程编程;Cache优化;CPU亲和力
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2010)06-0070-04

Research on Optimizing Multi-thread Programming Based on Multi-core Processor

SHI Hui-feng, YUAN Dao-hua
(College of Computer, Sichuan University, Chengdu 610065, China)

Abstract: Nowadays multi-core processors are becoming more and more popular. Past serial ways of programming can no longer make full use of multi-core CPU power. How to efficiently use the computation ability of multi-core processors has become a new challenge to software developers. Based on traditional multi-thread programming, according to features of Intel Microarchitecure, and CPU binding technology provided by Linux kernel, Cache optimization and CPU affinity optimization are introduced, aiming to eliminate Cache line competition and reduce the cost of thread schedule. After these optimizations, multi-thread program will be running more efficiently.

Key words: multi-core processor; multi-thread programming; Cache optimization; CPU affinity

0 引 言

由于单内核处理器的功耗和散热问题,芯片厂商已经不能再仅仅凭借提高芯片内部的晶体管集成度来提升处理器性能。Intel和AMD等厂商放弃了提高单内核处理器频率的计划,转而投向研发多核处理器。多核处理器顾名思义,就是在一块芯片上集成两个或者多个处理器核心。多核处理器能提供70%到80%的性能提升,同时又很好地解决了功耗和散热问题。随着Intel和AMD多核产品的投放市场,多核产品在桌面应用环境中也越来越普及。

简单地在计算机CPU上增加多个核并不能增加传统应用程序代码的运行速度[1]。因为传统应用代码没有充分考虑到双核乃至多核的运行情况,导致线程的平均分配时间以及线程之间的沟通时间大大增加,尤其是当线程需要反复访问内存的时候,程序的运行效率将大大降低。

文中根据Intel处理器的微架构(Microarchitecture)特点以及Linux内核提供的处理器绑定技术,介绍了基于Cache行和CPU亲和力的程序优化方法。

1 多线程编程

线程(thread),有时也被称为轻量级进程(light-weight process, LWP),是CPU使用的基本单元,它包括独立的堆栈和CPU寄存器状态,每个线程共享的进程资源,包括打开的文件、信号标识及动态分配的内存等。线程对于进程来说是轻量级的,因为激活一个新的进程必须分配给它独立的地址空间,建立众多的数据表来维护它的代码段、堆栈段和数据段,这是一种开销比较大的多任务工作方式,而运行一个进程中的多个线程,它们彼此之间使用相同的地址空间,共享大部分数据,激活一个线程所花费的空间远远小于激活一个进程所花费的空间,而线程间彼此切换所需的时间

收稿日期:2009-10-12;**修回日期**:2010-01-09
作者简介:施惠丰(1985-),男,上海人,硕士研究生,研究方向为分布式并行处理与网格计算;袁道华,教授,研究方向为分布式并行计算、网格计算、移动计算。

也远远小于进程间切换所需要的时间。同时,线程间的通信的方式多,而且简单也更有效率,不需要繁琐的IPC通信。如果对一个交互式应用程序采用多线程,即使其部分线程阻塞或执行冗长的操作,那么该程序仍能继续执行,因此多线程程序具有较高的响应度。然而,在单核环境下,多线程程序并不能充分地发挥出以上所述的这些优势,因为单核环境下的线程是并发执行的,在一个时间片,只有一个线程在执行。多核处理器使得计算机可以真正地在同一时间内运行多个进程或线程,从微观上真正做到并行化,这与以往的并发执行是不同的,但是在实际的编程环境中,仅仅依靠提高程序的并发度并不一定能提高程序运行效率。下面通过一个程序示例来说明这个问题。限于篇幅只贴出代码片段:

```
#define FOO1_MAX   100000000
#define FOO2_MAX   1000000
struct FOO1{
    unsigned long long a;
    unsigned long long b;
}foo1;
struct FOO2{
      int a[FOO2_MAX];
      int b[FOO2_MAX];
}foo2;
int main(void){
  int sum, index;
  for (sum = 0; sum <FOO1_MAX; sum++){
      foo1.a += sum;
      foo1.b += sum;
    }
  sum = 0;
  for (index = 0; index < FOO2_MAX; index++){
      sum += foo2.a[index] + foo2.b[index];
    }
  return 0;
}
```

程序的主要功能是对 foo1.a 和 foo1.b 进行一亿次的累加操作,并进行 foo2.a 和 foo2.b 一百万次的相加操作。

实验平台使用 Intel Xeon 3050 双核处理器,操作系统是 Fedora 10,内核版本 2.6.27,GCC 版本 4.3.2,多线程技术使用基于 POSIX 的 pthread[2] 函数库。先将程序扩展成两个线程:一个线程执行对于 foo1 中数据的操作,另一个线程执行对于 foo2 中数据的操作,然后再将程序扩展成三个线程,因为在 foo1 中,对于 foo1.a 和 foo1.b 计算不存在竞争关系,可以同时进行计算。

图 1 显示了单线程与多线程程序的运行时间比较。由测试数据可得到,双线程程序平均运行时间为 1.15 秒,比单线程的 0.76 秒增加了 51.31%,而三线程的平均运行时间为 1.78 秒,比单线程增加了 134.21%。线程增多以后,程序性能反而下降了。

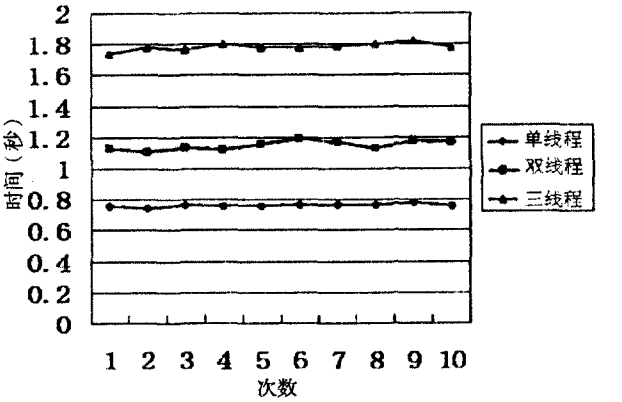

图 1　单线程与多线程时间对比

性能下降的原因在于操作系统创建、调度、销毁线程本身需要额外(相对于执行任务而必需的资源)的开销[3]。而且,过多的线程将导致过度的切换,线程切换带来的性能更是不可估量。系统完成线程切换要经过以下过程[4]:

* 从用户模式切换到内核模式。

* 将 CPU 寄存器的值保存到当前线程的内核对象中。

* 打开一个自旋锁,根据调度策略决定下一个要执行的线程。

* 释放自旋锁,如果要执行的线程不是同一进程中的线程,还需要切换虚拟内存等进程环境。

* 将要执行的线程的内核对象的值写到 CPU 寄存器中。切换到用户模式执行新线程的执行逻辑。

同时线程之间对于局部资源的竞争也会消耗一部分时间。由以上分析可知,在多核编程环境下,仅仅依靠增大程序的线程并发度并不能提高程序运行效率。

2　Cache 优化

2.1　Cache 行竞争

实验平台上的 Intel Xeon 3050 处理器,基于最新的 Intel Core Microarchitecture,在此微架构中 Intel 引入了 Advanced Smart Cache[5] 技术。Advanced Smart Cache 技术是 Intel 专门针对多核环境研发的新的 Cache 技术,旨在可以让多个 CPU 核心共享使用 Level2(L2) Cache,这样 CPU 可以充分利用 L2 Cache 的大小,因为每个核心理论上可以单独使用 L2 Cache 的全部空间。同时对于运行在不同核心上的线程,它们可以通过 L2 Cache 共享数据,这也大大提高了数据的吞

吐率。

在 Cache 存储系统中，把 Cache 和主存储系统都分成大小相同的块，块的大小通常以在主存储器的一个存储周期中能访问到的数据长度为限，一般是 8 到 32 个字[6]。在上面示例程序中，fool.a 和 fool.b 在 L2 Cache 中处于同一个 Cache 块(或者称为 Cache 行)。因此，当处理 fool.a 和处理 fool.b 的线程要写入相应的 Cache 行时，就会竞争该 Cache 行。因此，计算 fool.a 和 fool.b 的两个线程实际上是串行执行的。

2.2 Cache 行伪共享

虽然每个核心共享 L2 Cache，但是在其内部，每个核心独享一个 Level 1(L1) Cache。在 Intel Xeon 3050 中每个核分别有 32kB 指令 L1 Cache 和 32kB 数据 L1 Cache。这两个独立的数据 L1 Cache 在需要读取同一 Cache 行时，会共享该 Cache 行，如果在其中一个 L1 Cache 中，该 Cache 行被写入，而在另一个 L1 Cache 中该 Cache 行被读取，那么即使读写的地址不相交，也需要在这两个 L1 Cache 之间移动数据，这就是所谓的 Cache 伪共享，导致核心之间来回传递这个 Cache 行[7]。

2.3 Cache 优化方案

在三线程程序中在 fool 的两个成员 a 和 b 之间插入冗余数据，使其位于不同的 Cache 行中，以此来消除 Cache 行竞争和伪共享，修改后的代码片断如下：

```
struct FOO1{
    unsigned long long a;
    char ch[32]; /*根据实际情况，插入不同长度的数据*/
unsigned long long b;
}fool;
```

由如图 2 所示的优化结果可知，在使用了 32 字节的 Cache 优化方案后，程序的平均运行时间减少了 75.14%。32 字节和 64 字节的优化结果相当，这是因为一般 Cache 块的大小为 8 到 32 个字，所以 32 字节的 Cache 优化已经足够。Cache 优化是以空间换时间，所以在进行 Cache 优化时，也要在时间和空间两者之间进行平衡，过度增大 Cache 会造成内存资源的浪费。

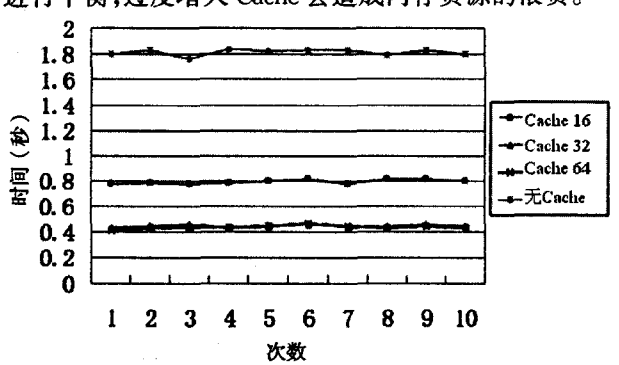

图 2 是否使用 Cache 优化对比

3 CPU 亲和力优化

3.1 CPU 亲和力简介

CPU 亲和力指的是程序开发者可以将一个或者多个线程绑定到一个特定的处理器上执行的技术[8]。在多核单处理器环境下，就是将某个线程绑定到某个 CPU 核心上执行。Linux 从 kernel 2.5 开始提供了具有 CPU 亲和力功能的系统调用[9]。

CPU 亲和力分为软亲和力(soft CPU affinity)和硬亲和力(hard CPU affinity)。软亲和力指的是内核中的线程调度器会尝试着将某个线程绑定到特定的 CPU 核心上运行，当然这只是尝试，如果这个 CPU 核一直处于忙碌状态，那么调度器会将此线程调度到其它 CPU 核上执行。硬亲和力则具有强制性，具有硬亲和力的线程，只能在绑定的 CPU 核心上运行。

如表 1 所示，传统的线程调度机制会产生“乒乓效应”。

表 2 显示的是，在具体有良好 CPU 亲和力的内核调度机制下线程的运行情况。

表 1 乒乓效应(The Ping-Pong Effect)

	Time 1	Time 2	Time 3	Time 4
Process A	CPU 0	CPU 1	CPU 0	CPU 1

表 2 具有良好 CPU 亲和力的进程

	Time 1	Time 2	Time 3	Time 4
Process A	CPU 0	CPU 0	CPU 0	CPU 0

多线程环境下使用 CPU 亲和力，可以有效提高线程 Cache 的命中率。因为在同一时间，某个数据只能保存在某个处理器核心的 Cache 缓存中，否则系统将不能确定到底哪个处理器核心的 Cache 里面保存的是最新的数据。如果多个处理器的 Cache 中有相同的内存数据缓存，而其中一个处理器对数据做了修改，那么其它处理器 Cache 中的相关数据就会变成无效数据[10]。如果一个进程(或者是线程)在不同的 CPU 核心之间被来回调度执行，就会造成 Cache 命中率的下降。而 CPU 亲和力由于使用了 CPU 绑定技术，可以很好地提高 Cache 的命中率。同样，如果多个进程或者线程要访问相同或相邻的内存数据，将它们绑定到同一个 CPU 核心上，也可以提高 Cache 的命中率。

3.2 CPU 亲和力优化方案

在 CPU 亲和力优化方案中，使用 setsched()函数，分别将计算 fool.a，fool.b 和 foo2 的三个线程绑定到 CPU0，CPU1，CPU0 (CPU0 表示第一个核心，CPU1 表示第二个核心)，运行结果如图 3 所示。

在三线程情况下，不使用 CPU 亲和力的平均运行时间是 1.79 秒，而使用 CPU 亲和力的程序平均运行

时间是1.37秒,使用CPU亲和力后,程序执行时间明显减少。

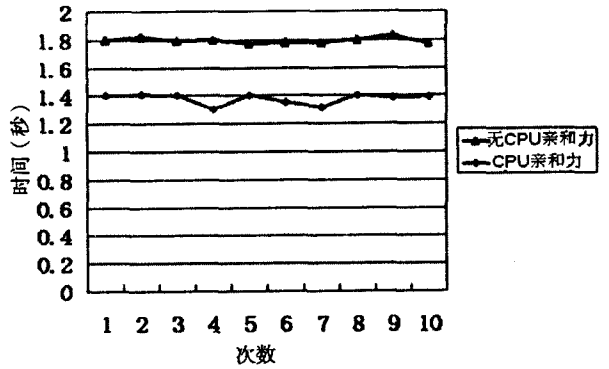

图3　是否使用CPU亲和力的时间比较

在上例中,计算fool.a和fool.b的两个线程没有绑定到同一个CPU核心,而fool.a和fool.b由于在同一结构体中,所以在内存中是相邻的数据,现在将计算fool.a和fool.b的线程绑定到同一个核心上。

如图4所示,将fool.a和fool.b绑定到同一核心后,程序平均运行时间由原来的1.37秒减少到了0.89秒。这是因为将访问相同数据或相邻数据的一组线程绑定到一个CPU核心上,可以提高一组线程的Cache命中率,从而提高程序执行效率。

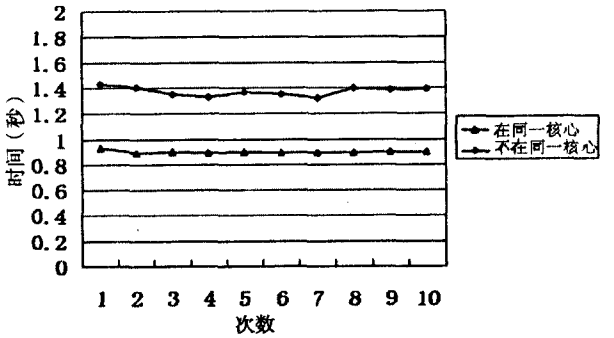

图4　fool.a和fool.b绑定到不同核心的比较

4　结束语

文中针对多核编程环境的特点,在使用传统的多线程并行编程技术的基础上,介绍了基于Cache优化和CPU亲和力的多线程程序优化思路,在实际实验中,单线程原始程序平均耗时0.76秒,最快的三线程优化方案为0.44秒,效率提高了约42%。基于Cache和CPU亲和力的优化方法具有一定的通用性,在软件开发过程中,可以根据实际情况采用这两种优化方法,从而有效地提高软件的运行效率。

参考文献:

[1] Reinders J. Programming For Parallelism[EB/OL]. 2007. http://www.cajcd.edu.cn/pub/wml.txt/980810-2.html.

[2] Stevens W R, Rago S A. Advanced Programming in the UNIX Environment[M]. 北京:人民邮电出版社,2006.

[3] 杨　静,李　炜,万峰松,等. Linux2.6内核进程调度分析与改进[J]. 计算机技术与发展,2009,19(7):105-107.

[4] 王　晶,樊晓桠,张盛兵,等. 多核多线程结构线程调度策略研究[J]. 计算机科学,2007,34(9):256-258.

[5] Doweck J. Inside Intel Core Microarchitecture and Smart Memory Access[EB/OL]. 2006. http://download.intel.com/technology/architecture/s-ma.pdf.

[6] 李晓明,臧斌宇,郑纬民,等. 多核程序设计[M]. 北京:北京大学出版社,2007.

[7] 金国华,陈福接. 简单访问模式下假共享Cache行抖动的消除[J]. 计算机学报,1994(6):435-445.

[8] Love R. CPUaffinity[EB/OL]. 2003. http://www.linuxjournal.c-om/article/6799.

[9] Bovet D P, Cesati M. Understanding the Linux Kernel[M]. 北京:中国电力出版社,2007.

[10] 杨　磊,石　磊,张铁军,等. 多核系统中共享cache的动态划分[J]. 微电子学与计算机,2009,26(5):56-59.

(上接第69页)

(2-3):197-210.

[3] 谢　榕. 数据挖掘与商业智能系统[J]. 计算机系统应用,1999(8):9-10.

[4] 王晓宇,熊　方,凌　波,等. 一种基于相似度的主题提取和发现算法[J]. 软件学报,2003,14(9):1578-1585.

[5] 李晓明,朱家稷,阎宏飞. 互联网上主题信息的一种收集与处理模型及其应用[J]. 计算机研究与发展,2003,40(12):1667-1671.

[6] Cai D, Yu S, Wen J R, et al. Block-based Link Analysis [C]//in 27th Annual International ACM SIGIR Conference on Information Retrival. Sheffield, South Yorkshire, UK: [s. n.], 2004.

[7] 宋　杰,王大玲,鲍玉斌,等. 基于页面Block的Web档案采集和存储[J]. 软件学报,2008,19(2):275:275-290.

[8] Cai D, Yu S, Wen J R, et al. VIPS: a version-based page segmentation algorithm[R]. US: Microsoft, 2003.

[9] Cai D, Yu S, Wen J R, et al. Block-based Web Search[C]// in 27th Annual International ACM SIGIR Conference on Information Retrival. Sheffield, South Yorkshire, UK: [s. n.], 2004.

[10] 张　敏,高剑锋,马少平. 基于链接描述文本及其上下文的Web信息检索[J]. 计算机研究与发展,2004,41(1):221-226.

[11] 宋聚平,王永成,尹中航,等. 面向主题的网页搜索系统[J]. 上海交通大学学报,2003,37(3):401-403.