多线程技术与并行计算*)

Multi-thread and Parallel Computing

张宏莉 田 耕 胡铭曾
(哈尔滨工业大学计算机系 哈尔滨 150001)

Abstract Multithreads play a significant role in improving computer performance. From sotfware aspect, this paper mainly introduces the properties and implementation of thread, content of thread packet and multithread programming, and analyses its contribution for parallel computing in detail.

Keywords Multithread, Parallel computing, Parallel programming

1 引言

自七十年代以来,多线程技术已逐渐渗透到计算机体系结构的方方面面。出现了多线程结构的并行机,如 Tera(1990),Stanford 的 DASH(1988),MIT 的 Alewife(1989),等[1];在操作系统方面,CMU 的 Mach/OS(1986),Solaris、AIX、Linux 等 Unix 操作系统,以及目前流行的 PC 机操作系统 Windows 系列等,都已不同程度地采用了多线程技术[2];在并行编译系统方面,多线程也已为 CC++[3],HPC++[4]等并行编译器采用,成为提高并行程序加速比的重要途径;在并行编程方面,线程包 pThread,语言 Java 中的线程类,运行时系统 pSystem[5],Nexus[6],TPVM[7]等也为用户提供了多线程编程环境。

本文主要从软件方面介绍了线程属性、线程的实现、线程包、线程编程等内容,并定性定量地分析了多线程对计算性能的影响。

2 多线程简介

在这部分,将从以下几方面介绍线程:线程属性、线程的实现、线程包、线程编程以及与进程的比较。

2.1 线程的属性

线程是地址空间中独立的控制流,它与进程的区别是:在一个地址空间中的线程可有多个,而进程只能有一个。传统的单 CPU 进程系统中,进程的属性被多线程系统中的线程和进程分占。

* 进程属性:在多线程系统中,它已退化为一个执行框架,进程的属性有所改动,具有:进程 ID,进程组 ID,用户 ID,用户组 ID;环境属性;工作目录;并仍具有通常的地址空间和系统资源如:文件描述符,信号操作,共享库,进程间通信(如消息队列,管道,信号灯,共享空间)。

* 线程属性:线程是一种调度单元。它只具有一些保证其作为独立控制流的属性,如:程序计数器,栈,调度属性(策略,优先级),挂起和阻塞信号,线程的有关数据(如 errno)。

* 进程与线程的关系:初始线程在创建进程时被自动创建,这保证原来具有单隐含线程的进程与多线程的进程相兼容;它与用户创建的线程不同,为用户所不可见,这是为了以往的单控制流程序与多控制流程序相兼容。初始线程执行用户的主程序 main( )。

2.2 线程的实现

目前有 2 类线程[2]:用户线程和核心线程。通常,用户线程又称线程,核心线程又称轻量级进程(Light Weight Process)。

用户线程是用户使用的处理程序中多个控制流的单元,处理用户线程的 API 由线程库提供。不同进程中的线程不能通信。用户级线程通过与运行栈有关的数据结构集合实现,用户级线程不必与操作

*) 本文受国防科工委九五预研项目资助。张宏莉 博士生,主要研究方向:并行编译技术、面向对象技术。胡铭曾 教授博士生导师,主要研究方向:并行体系结构、并行处理。

系统打交道，只受控于库函数，也就是说内核对用户级线程的存在一无所知。其主要的优点是：线程的创建、撤消开销小；可在进程级操作系统上实现。

核心线程是类似于进程和中断处理器的内核单元，它在进程中运行，由系统调度器调度；可被系统中的其他线程访问，用户不能直接控制它。至于核心级线程，其创建、撤消都要通过系统调用由内核完成，所以开销较大。即是说，用户级线程由库调度，核心级线程由内核调度。二者的共同之处是：运行时的管理开销都比标准的 Unix 进程小得多。IEEE 已制定了多线程的标准，POSIX threads[1,4]，其中已有实现。

2.3 线程包的内容

Pthreads[8]线程包的内容包括：

·创建与终止。线程的创建不同于进程，线程间无父子关系，除初始线程外，所有线程地位相同。创建函数：pthread_creation( )，参数包括：入口过程名及其调用参数。入口过程返回时线程结束，线程终止函数：pthread_cancellation( )。

·同步：用于线程间有效的交互。包括互斥变量(mutex variable)，条件变量(condition variable)，汇聚(join)。

互斥锁可保证因竞争引起的数据一致性问题。如线程 A，B 都对一个共享变量 m(初值为 1)操作：m一，结果应为 3。但生成的汇编语言则表述为：

```
move X,REG
inc REG
move REG,X
```

不难看出，结果又可能为 2。

条件变量保证线程实现阻塞，直到某个事件发生或条件成立。汇聚操作指该线程被阻塞到某个线程结束时条件变量的特例。

·调度：线程库允许程序员控制线程的调度执行，途径有：创建线程时设置调度属性；修改已创建线程的调度属性；创建 mutex 变量；同步操作。调度参数有三：scope(指线程模式)，policy(指调度依据，分时或按优先级)，priority(表征线程的相对重要性)。

·其他功能：线程可维护私有数据；线程可控制栈的大小和地址。

·线程库 API。

2.4 多线程编程示例

下面是一个简单的多线程程序，并发地输出“hello”，“world”，并通过 join 语句保证其先后顺序。

```
#include<pthread.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<assert.h>
/*hello_thread-print the "hello" part.*/
void *hello_thread(void *arg)
{ printf("hello");
  return(0);}
/*world_thread-print the "world\n" part.*/
void *world_thread(void *arg)
{ int n;
  pthread_t tid=(pthread_t)arg;
  if(n=pthread_join(tid,NULL)){
    fprintf(stderr,"pthread_join:%s\n",strerror(n));
    exit(1);
  }
  printf("world");
  pthread_exit(0);
  return(0);
}
main (int argc,char *argv[]){
    int n;
    pthread_attr_t attr;
    pthread_t htid,wtid;
    assert(sizeof(pthread_t)<=sizeof(void *));
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr,PTHREAD_
        CREATE_UNDETACHED);
    if (n=pthread_create(&htid,&attr,hello_thread,
       NULL)){
       fprintf(stderr,"pthread_create:%s\n",strerror
       (n));
       exit(1);
    }
        pthread_attr.destroy(&attr);
    if(n=pthread_create(&wtid,NULL,world_thread,
       (void *)htid)){
       fprintf(stderr,"pthread_create:%s\n",strerror
       (n));
       exit(1);
    }
    printf("\n");
    return(0);
}
```

2.5 与进程的比较

线程与进程的区别在于多线程是共享同一地址空间的多个控制流，线程优于进程主要表现在：1)线程的创建、撤销时间，以及切换时间比进程短，至少相差一个数量级，如在运行 Solaris 操作系统的 Sun 工作站上，线程和进程的创建时间分别为：0.00014s，0.0042s；2)同一上下文中的线程通过共享全局数据实现通信，免去了进程间通过消息通信带来的额外开销。这两点对并行计算的性能有相当的提高。

3 多线程与并行计算

3.1 定性分析

多线程已被广泛应用于实现动态、异步、并发程序，尤其适合开发细粒度并行。在并行计算中采用多线程主要是出于功能和性能的考虑。

从功能角度看，基于线程的模型有两点优势：

1 基于小粒度的数据分解可以被无效率损失地实现。这对通信间计算量较小的问题如树搜索算法、整数计算、数据库查询系统格外重要。2 这种模式与 Client-Server 计算很相近，服务可通过使用线程登记机制输出、用类似 RPC 的方式调用；这种办法对非数值计算应用程序很有用，尤其那些数据库和事务处理领域。

性能方面，多线程能屏蔽计算-通信延迟。在传统的进程中，通信-计算-通信循环导致了空闲周期，但用多线程则可以高效率地使用 CPU，即一个线程阻塞了，又转而执行另一个线程，且这种开销比进程少得多。Sundraw 等人的初步实验表明，采用多线程可提高通信性能35%[7]。

3.2 定量分析

鉴于以上的定性分析，我们在由运行操作系统 AIX 4.1的工作站 RS6000组成的网络上做了对照实验：一台节点机 master 向另一台节点机 slave 发 n 次消息，slave 分别1)在 n 个进程中；2)在一个进程中创建 n 个线程响应各消息，针对任务计算粒度分别为10^2，10^3，10^4，10^5，10^6的情况，分别测得程序执行时间，如表3-1所示。

表3-1 不同执行方式的程序执行时间 （单位：秒）

执行方式 / 计算粒度	多进程并行	多线程并行
100次整数乘法	0.405821	0.075107
10^3次整数乘法	0.448177	0.083401
10^4次整数乘法	0.461189	0.087426
10^5次整数乘法	0.586142	0.204409
10^6次整数乘法	1.486253	0.585102

结果表明，采用 n(=10)个进程的并发执行时间是采用 n 个线程的3～6倍，进而定量地证明：多线程对提高并发计算的性能意义重大。

参考文献

1 (美)Hwang 著. 高等计算机系统结构：并行性，可扩展性，可编程性. 王鼎兴，等译. 北京：清华大学出版社，1995

2 Tanenbaum A S. Distibuted Operating Systems. PRENTICE HALL. 1996

3 Chandy K M, Kesselman C. CC++: a declarative concurrent object oriented programming notation. In Research Directions in Object Oriented Programming, MIT Press. 1993

4 Gannon D, et al. HPC++: A Draft White Paper. In preparation. 1995

5 Lopes L M B, Silva F M A. Thread-and Process-based Implementations of the pSystem Parallel Programming Environment. Software Practice and Experience, 1997, 27(3): 329～351

6 Foster I, et al. Nexus: Runtime Support for Task Parallel Programming Languages. In: Intl. Workshop on Parallel Processing-1994. India, 1994

7 Draft Version, TPVM: A Threads-based Interface and Subsystem for PVM. 1994. Available at: http: // uvcs. cs. Virgnia. edu/~ aifij/tpvm. html

8 Muller F. A Pthreads Library Interface. [Tech. Rep.]: Department of Computer Science, Florida State University, July 1993

（上接第44页）

5 Chandrasekaran B. Design Problem Solving: A task analysis. AI Magazine, 1990, 11: 59～71

6 Chandrasekaran B. Functional representation and causal processes, in Advances in Computers, Vol. 38, M. C. Yovits, Ed.: Academic Press, 1994. 73～143

7 Lemke A C, Fischer G. A Cooperative Problem Solving System for User Interface Design. AAAI-90, 1990. 479～484

8 Fischer G, et al. Supporting Indirect Collaborative Design with Integrated Knowledge-Based Design Environment. Human-Computer Interaction, 1992, 7(3): 281～314

9 Mostow J. Design by Derivational Analogy: Issues in the Automated Replay of Design Plans. Artificial Intelligence, 1989