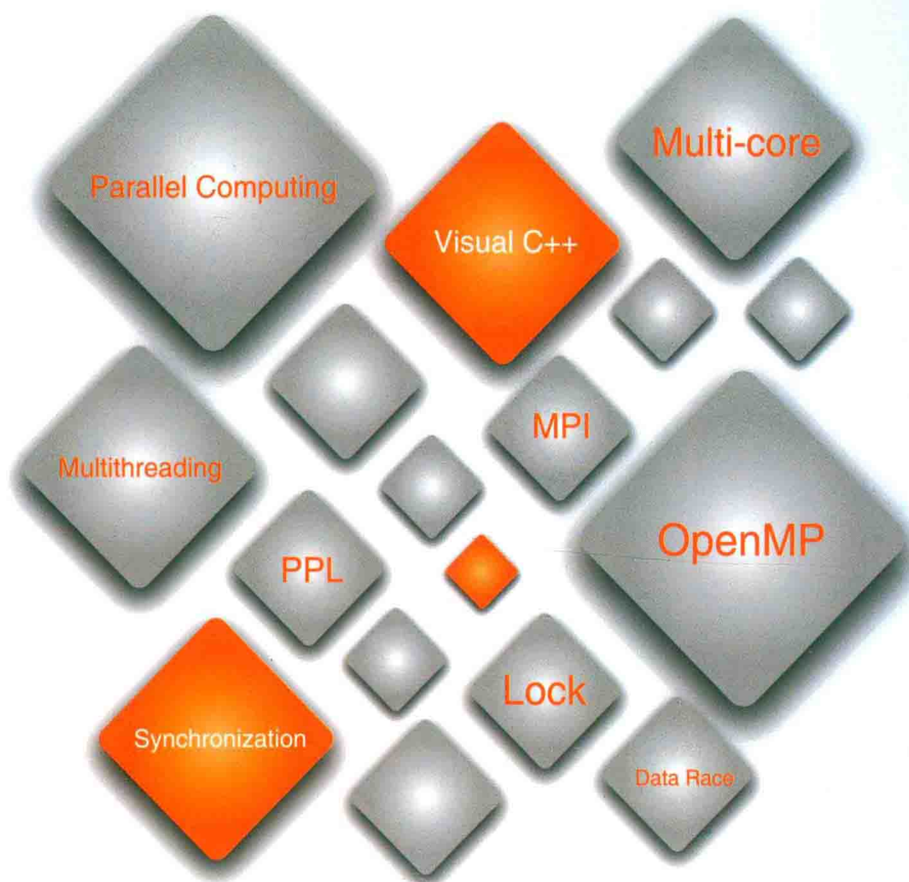
石油高等教育“十三五”规划教材

多核并行计算

Multi-core Parallel Computing

陈 华 编著



中国石油大学出版社
CHINA UNIVERSITY OF PETROLEUM PRESS

石油高等教育"十三五"规划教材

多核并行计算

DUOHE BINGXING JISUAN

陈 华 编著

常州大学图书馆
藏书章

中国石油大学出版社
CHINA UNIVERSITY OF PETROLEUM PRESS

图书在版编目(CIP)数据

多核并行计算 / 陈华编著. —东营:中国石油大学出版社,2017.12

ISBN 978-7-5636-5903-6

Ⅰ. ①多… Ⅱ. ①陈… Ⅲ. ①并行算法 Ⅳ. ①TP301.6

中国版本图书馆 CIP 数据核字(2017)第 310584 号

石油高等教育教材出版基金资助出版

书　　名: 多核并行计算
作　　者: 陈　华

责任编辑: 曹秀丽(电话　0532—86981532)
封面设计: 青岛友一广告传媒有限公司

出 版 者: 中国石油大学出版社
(地址: 山东省青岛市黄岛区长江西路 66 号　邮编: 266580)
网　　址: http://www.uppbook.com.cn
电子邮箱: shiyoujiaoyu@126.com
排 版 者: 青岛天舒常青文化传媒有限公司
印 刷 者: 沂南县汶凤印刷有限公司
发 行 者: 中国石油大学出版社(电话　0532—86981531, 86983437)
开　　本: 185 mm×260 mm
印　　张: 12.25
字　　数: 300 千
版 印 次: 2018 年 3 月第 1 版　2018 年 3 月第 1 次印刷
书　　号: ISBN 978-7-5636-5903-6
定　　价: 30.00 元

前 言

PREFACE

2006年7月23日，英特尔基于酷睿(Core)架构的处理器正式发布，标志着多核时代的真正来临。随后的11月份，英特尔又推出面向服务器、工作站和高端个人电脑的至强(Xeon) 5300和酷睿双核和四核至尊版系列处理器。2016年6月7日，英特尔正式发布了新一代数据中心处理器Xeon E7 v4系列(代号Broadwell-EX)，核心数量最多达到24个(48个线程)，这同时也意味着Broadwell家族的全面普及，英特尔完全进入了14 nm时代。但是，很多程序仍然采用串行程序，为了提高计算效率，有必要进行多核多线程编程。

自2007年起，中国石油大学(华东)理学院开始开设“MPI并行程序设计”课程。随着多核个人计算机的逐渐普及，数学课题组对原有的教学内容不断增改，逐渐形成了“多核并行编程”课程。该课程以Visual Studio为开发平台，以C/C++为开发语言，以OpenMP和MPI等为多核编程工具。本书采用由浅入深、循序渐进的原则展开介绍，以e值计算等案例为主线贯穿全书，注重实践能力的培养，同时还收录了一些经学生整理后发表的课程大作业。本书中的程序代码完整，可以直接运行，为读者理解和使用提供了方便。

全书共分5章。第1章介绍基于Windows API的多核并行程序设计与计算，主要内容包括进程和线程的概念，在Windows平台下Win32多线程API的定义、线程的同步和综合案例；第2章介绍基于OpenMP的多核并行程序设计与计算，主要内容包括OpenMP简介、Visual C++的OpenMP编程步骤、与线程相关的编译指导语句详解、数据处理、同步、循环并行化、OpenMP应用程序设计的考虑因素、程序的性能分析、OpenMP多核并行计算模式和综合案例；第3章介绍基于MPI的多核并行程序设计与计算，主要内容包括MPI简介、MPI程序基础、点对点通信、群集通信、综合案例；第4章介绍基于PPL的多核并行程序设计与计算，主要内容包括概述、C++ Lambda表达式、并行模式库、同步、任务并行和综合案例；第5章介绍综合实验。

在本书的编写过程中，参阅了大量书籍和其他相关资料，得到了石油高等教育教材出版基金和山东省自然科学基金（ZR2013DM015）的资助，同时也得到了中国石油大学（华东）理学院计算数学系师生的支持和帮助，在此表示衷心感谢。

尽管书稿几经修改，但由于作者学识有限，书中难免有疏漏与不当之处，恳请各位同仁和读者不吝赐教。

作　者

2017年10月

试读结束：需要全本请在线购买： www.ertongbook.com

目 录

CONTENTS

第 1 章　基于 Windows API 的多核并行程序设计与计算 …… 1

1.1　进　程 …… 1

1.2　线　程 …… 6

1.3　线程的同步 …… 17

1.4　综合案例 …… 33

第 2 章　基于 OpenMP 的多核并行程序设计与计算 …… 38

2.1　OpenMP 简介 …… 38

2.2　Visual C++的 OpenMP 编程步骤 …… 44

2.3　与线程相关的编译指导语句 …… 47

2.4　数据处理 …… 59

2.5　同　步 …… 70

2.6　循环并行化 …… 79

2.7　OpenMP 应用程序设计的考虑因素 …… 87

2.8　程序的性能分析 …… 87

2.9　OpenMP 多核并行计算模式 …… 88

2.10　综合案例 …… 93

第 3 章　基于 MPI 的多核并行程序设计与计算 …… 101

3.1　MPI 简介 …… 101

3.2　MPI 程序基础 …… 104

3.3　点对点通信 …… 109

3.4　群集通信 …… 119

3.5　综合案例 …… 128

第 4 章　基于 PPL 的多核并行程序设计与计算 …… 132

4.1　概　述 …… 132

4.2　C++ Lambda 表达式 …… 134

4.3　并行模式库 …… 138

4.4　同　步 …… 145

4.5　任务并行 …… 153

4.6　综合案例 …… 157

第 5 章　综合实验 …… 166

附　录 …… 169

附录一　基于四种并行计算模式的自然对数底并行计算方法 …… 169

附录二　多线程并行快速求解 e 值的六种方法 …… 176

附录三　基于多核并行的非线性方程蒙特卡罗计算方法 …… 183

参考文献 …… 189

第1章

基于 Windows API 的多核并行程序设计与计算

1.1 进　程

进程的概念是20世纪60年代初首先由麻省理工学院的 MULTICS 系统和 IBM 公司的 CTSS/360 系统引入的。进程(Process)是指具有一定独立功能的程序关于某个数据集合上的一次运行活动,是系统进行资源分配和调度的一个独立单位。程序只是一组指令的有序集合,它本身没有任何运行的含义,只是一个静态实体。而进程则不同,它是程序在某个数据集上的执行,是一个动态实体。进程因创建而产生,因调度而运行,因等待资源或事件而处于等待状态,因完成任务而被撤销,它反映了一个程序在一定的数据集上运行的全部动态过程。图1-1为 Windows 任务管理器中的进程一览表,其中每个进程都表现为一个独立的执行程序。

图1-1　Windows 任务管理器的进程显示

1.1.1 进程的特征

进程具有以下特征。

(1) 动态性:进程的实质是程序在多道程序系统中的一次执行过程,它是动态产生、动态消亡的。

(2) 并发性:任何进程都可以和其他进程一起并发执行。

(3) 独立性:进程是一个能独立运行的基本单位,也是系统分配资源和调度的独立单位。

(4) 异步性:进程间的相互制约使其具有执行的间断性,即进程按各自独立的、不可预知的速度向前推进。

(5) 结构特征:进程由程序、数据和进程控制块三部分组成。

多个不同的进程可以包含相同的程序,而一个程序在不同的数据集里构成不同的进程,能得到不同的结果,但是在执行过程中程序不能发生改变。

1.1.2 进程的状态

进程执行时的间断性决定了进程可能具有多种状态。事实上,运行中的进程可能具有如图 1-2 所示的三种基本状态:就绪态、运行态和阻塞态。

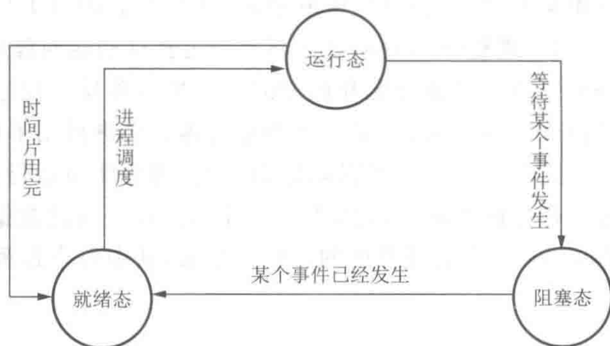


图 1-2 进程状态图

(1) 就绪态(Ready)。

进程已获得除处理器之外的所需资源,只要分配了处理器就可执行。就绪进程可以按多个优先级来划分队列。例如,当一个进程由于时间片用完而进入就绪状态时,排入低优先级队列;当进程由 I/O 操作完成而进入就绪状态时,排入高优先级队列。

(2) 运行态(Running)。

进程占用处理器资源,其数目小于等于处理器的数目。在没有其他进程可以执行时(如所有进程都在阻塞状态),通常会自动执行系统的空闲进程。

(3) 阻塞态(Blocked)。

当进程等待某种条件(如 I/O 操作或进程同步)时,在这些条件满足之前即使把处理机分配给该进程,也无法继续执行。

1.1.3 进程的创建

在 Windows 操作系统中,一般采用 API(Application Programming Interface,应用编程

接口)函数 CreateProcess 建立一个新进程和它的主线程。该新进程执行指定的可执行文件,并且独立运行于调用进程。

CreateProcess 的函数原型为:

```
BOOL CreateProcess(
LPCTSTR lpApplicationName,
LPTSTR lpCommandLine,
LPSECURITY_ATTRIBUTES lpProcessAttributes,
LPSECURITY_ATTRIBUTES lpThreadAttributes,
BOOL bInheritHandles,
DWORD dwCreationFlags,
LPVOID lpEnvironment,
LPCTSTR lpCurrentDirectory,
LPSTARTUPINFO lpStartupInfo,
LPPROCESS_INFORMATION lpProcessInformation
);
```

其中,各参数含义如下。

(1) 参数 lpApplicationName:指定要执行的应用程序的名字。该名字可以是全路径名。如果该参数为 NULL,则程序名必须是 lpCommandLine 指向的字符串的第一个标识符。该参数通常置为 NULL,而将程序名和参数放在 lpCommandLine 指定的字符串中。

(2) 参数 lpCommandLine:一个以 NULL 结尾的字符串的指针。它指向命令行参数。参数 lpApplicationName 和 lpCommandLine 不允许同时空,否则系统找不到新进程所对应的可执行程序的文件名。

(3) 参数 lpProcessAttributes 和 lpThreadAttributes:它们指向 SECURITY_ATTRIBUTES 结构,分别用来确定待创建的进程和待创建进程的主线程的安全属性。如果使用默认安全属性,则该值为 NULL。

(4) 参数 bInheritHandles:用来确定新建的进程能否继承产生它的进程的句柄。若它的值为 TRUE,则这个进程和线程所建立的句柄都可以被这个进程所建立的新进程所继承,即继承的句柄和原来的句柄有相同的值和存取权限。

(5) 参数 dwCreationFlags:该参数决定新进程产生的方式,它可以用逻辑或(|)的方式把下列值结合起来。

① CREATE_NEW_CONSOLE:为新进程建立一个新的控制台窗口。

② DETACHED_PROCESS:在默认情况下,新进程使用的是父进程的控制台窗口。

③ CREATE_NEW_PROCESS_GROUP:这个新进程将是一个新进程组的根,进程组包括该进程的所有子进程。

④ CREATE_SUSPENDED:新进程的主线程被创建在挂起状态,直到 Resume_Thread 函数被调用后才运行。

⑤ DEBUG_PROCESS:如果设置该标志,则调用该进程。新进程准备接受调试,系统把进程被调试时所发生的调试事件通知给父进程。

另外,还有其他的值,可以参阅 MSDN 来学习。

(6) 参数 lpEnvironment:指向一个用于新进程的环境块。如果该参数为 NULL,则新进程继承调用进程的环境。

(7) 参数 lpCurrentDirectory:指向新进程的当前驱动器和目录的字符串。如果该参数为 NULL,则使用调用进程的当前驱动器和目录。

(8) 参数 lpStartupInfo:指向一个 STARTUPINFO 结构,用于说明如何显示新进程的主窗口。

(9) 参数 lpProcessInformation:指向一个 PROCESS_INFORMATION 结构,用于接收有关新进程的标识信息。

函数 CreateProcess( )调用成功,返回值为 TRUE,否则为 FALSE。

1.1.4 从进程

【例 1-1】 创建一个从进程,使其打开计算器程序,并通过获取计算器进程句柄关闭该进程。

```
#include <windows.h>
#include <stdio.h>
int main(int argc,char * argv[])
{
    //创建进程
    //进程路径
    char commandLine[]="calc.exe";
    STARTUPINFO startInfo={ sizeof(startInfo) };
    PROCESS_INFORMATION procInfo;
    startInfo.dwFlags=STARTF_USESHOWWINDOW;//指定 wShowWindow 成员有效
    startInfo.wShowWindow=TRUE;/*若此成员设为 TRUE,则显示新建进程的主窗口;若设为
    FALSE,则不显示*/
    BOOL bRet=::CreateProcess (
        NULL,//不在此指定可执行文件的文件名
        commandLine,//命令行参数
        NULL,//默认进程安全性
        NULL,//默认线程安全性
        FALSE,//指定当前进程内的句柄不可以被子进程继承
        CREATE_NEW_CONSOLE,//为新进程创建一个新的控制台窗口
        NULL,//使用本进程的环境变量
        NULL,//使用本进程的驱动器和目录
        &startInfo,
        &procInfo);
    //关闭进程
    //进程句柄
    if(bRet)
    {
        //既然不使用这两个句柄,就最好立刻将它们关闭
        ::CloseHandle (procInfo.hThread);
```


试读结束：需要全本请在线购买： www.ertongbook.com

```
        ::CloseHandle (procInfo.hProcess);
        printf(" 新进程的进程 ID 号:%d \\n",procInfo.dwProcessId);
        printf(" 新进程的主线程 ID 号:%d \\n",procInfo.dwThreadId);
    }
    Sleep(2000);
    HANDLE processHandle;
    //进程 ID
    ULONG processID;
    //进程窗口句柄
    HWND theWindow;
    //FindWindow("窗口类名","窗口标题")
    theWindow=::FindWindow(NULL,"计算器");
    //获取窗口进程 ID
    ::GetWindowThreadProcessId(theWindow,&processID);
    //获取进程句柄
    processHandle=::OpenProcess(PROCESS_TERMINATE,FALSE,processID);
    //关闭进程
    ::TerminateProcess(processHandle,4);
    return 0;
}
```

【例 1-2】创建一个从进程，使其打开搜狐网站。

```
#include <windows.h>
#include <stdio.h>
int main(int argc,char* argv[])
{
    //创建进程
    PROCESS_INFORMATION procInfo;
    //进程初始化信息(必备参数)
    STARTUPINFO startInfo;
    ZeroMemory(&startInfo,sizeof(startInfo));
    startInfo.cb=sizeof(startInfo);
    startInfo.dwFlags=STARTF_USESHOWWINDOW;
    startInfo.wShowWindow=SW_SHOW;
    //可选参数,进程安全属性
    SECURITY_ATTRIBUTES processAtt,threadAtt;
    processAtt.nLength=sizeof(processAtt);
    processAtt.lpSecurityDescriptor=NULL;
    processAtt.bInheritHandle=true;
    threadAtt.nLength=sizeof(threadAtt);
    threadAtt.lpSecurityDescriptor=NULL;
    threadAtt.bInheritHandle=true;
    //带命令行参数
```

```
    ::CreateProcess("C:\\\\Program Files\\\\Internet Explorer\\\\IEXPLORE.EXE",
                    "IEXPLORE.EXE http://www.sohu.com" ,&processAtt,&threadAtt,FALSE,
                    CREATE_NEW_CONSOLE,NULL,NULL,&startInfo ,&procInfo);
    return 0;
}
```

1.2 线　程

线程(Thread)是进程的一个实体,是 CPU 调度和分派的基本单位。线程不能独立执行,必须依存在进程中,由进程提供多个线程来执行控制。从内核角度讲,线程是活动体对象,而进程只是一组静态的对象集,进程必须至少拥有一个活动线程才能维持运转。图 1-3 表明一个进程最多可以调用 8 个线程。

图 1-3　进程状态图

线程和进程的关系是:线程是属于进程的,线程运行在进程空间内,同一进程所产生的线程共享同一内存空间,当进程退出时该进程所产生的线程都会被强制退出并清除。线程可与属于同一进程的其他线程共享进程所拥有的全部资源,但是其本身基本上不拥有系统资源,只拥有在运行中必不可少的信息(如程序计数器、一组寄存器和栈)。

在操作系统中引入线程的主要好处是:

(1) 在进程内创建、终止线程比创建、终止进程要快。

(2) 同一进程内线程间的切换比进程间的切换要快,尤其是用户级线程间的切换。

另外,线程的引入还有以下几个原因:

(1) 并发程序的并发执行,在多处理环境下更为有效。一个并发程序可以建立一个进程,而这个并发程序中的若干并发程序段可以分别建立若干线程,这些线程可以在不同的处理机上并发执行。

(2) 每个进程都具有独立的地址空间,该进程内的所有线程共享该地址空间,这样就可以解决父子进程模型中子进程必须复制父进程地址空间的问题。

(3) 线程对解决客户/服务器模型非常有效。

1.2.1 线程的生命周期

1) 线程的标识

通常用一个整数来标识一个线程,可用 GetCurrentThreadId 获得当前线程 ID。

2) 线程的创建

创建线程可采用 Windows 的 API 函数 CreateThread。该函数的原型为:

```
HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes,
DWORD dwStackSize,
LPTHREAD_START_ROUTINE lpStartAddress,
LPVOID lpParameter,
DWORD dwCreationFlags,
LPDWORD lpThreadId);
```

其中,各参数含义如下。

(1) 参数 lpThreadAttributes:指向一个 SECURITY_ATTRIBUTES 结构,用于指定线程的安全属性。如果使用默认安全属性,则置为 NULL。

(2) 参数 dwStackSize:指定线程用于分配堆栈的大小(以字节为单位)。如果为 0,则堆栈大小默认为与该进程主线程的堆栈大小相同。

(3) 参数 lpStartAddress:指向新线程执行代码的开始地址,不能默认为 NULL,通常为包含线程代码的线程函数名,该函数的原型为:

```
DWORD WINAPI ThreadProc(LPVOID lpParam);
```

从线程在调用该线程函数时可以以 void 指针形式传递参数,执行结束后可以返回一个状态信息,这是从线程与主线程之间的一种通信机制。

(4) 参数 lpParameter:指定传递给线程函数的 32 位参数值,该参数可以是一个数值,也可以是一个数据结构指针。用于主线程向从线程传递参数,其数据类型与从线程形参的类型一致。

(5) 参数 dwCreationFlags:线程创建标志。如果为 CREATE_SUSPENDED,则该线程创建在挂起状态,直至调用 ResumeThread 函数后才运行;如果为 0,则创建后立即运行。

(6) 参数 lpThreadId:指向一个 32 位变量,是用于接收该从线程的唯一标识符,即创建从线程 ID。如果不需要使用从线程 ID,该变量可以设置为 NULL。

如果该函数调用成功,则返回新线程的句柄,否则返回 NULL。

另外,也可以使用 C 运行库函数_beginthread,其函数定义为:

```
uintptr_t _beginthread(void( * start_address)(void * ),unsigned stack_size,void * arglist);
```

返回值:如果成功,函数将会返回一个新线程的句柄,用户可以像这样声明一个句柄变量存储返回值:HANDLE hStdOut = _beginthread(ThreadFun,0,NULL);如果失败,则_beginthread 将返回-1。

参数 start_address:新线程的起始地址,指向新线程调用的线程函数的起始地址。一般

线程函数的声明形式为：

```
void ThreadFun(void * pParam);
```

参数 stack_size：新线程的堆栈大小，可以为 0。

参数 arglist：传递给线程的参数列表，无参数时为 NULL。

3）线程的终止

执行完毕或者调用了 ExitThread 函数，只有在异常情况下才使用 TerminateThread 来终止线程。主线程退出会导致整个进程终止。线程退出函数的声明形式为：

```
VOID ExitThread(DWORD dwExitCode);
```

该函数用于线程终结自身的执行，主要在线程的执行函数中被调用。其中参数 dwExitCode 用来设置线程的退出码。

一般情况下，线程运行结束之后，线程函数正常返回，但应用程序可以调用 TerminateThread 强行终止某一线程的执行。该线程终止函数的声明形式为：

```
BOOL TerminateThread(HANDLE hThread,DWORD dwExitCode);
```

参数 hThread：将被终结的线程的句柄。

参数 dwExitCode：用于指定线程的退出码。

使用 TerminateThread( )终止某个线程的执行是不安全的，可能会引起系统不稳定；虽然该函数立即终止线程的执行，但并不释放线程所占用的资源。因此，一般不建议使用该函数。

4）线程的状态

不同的平台对线程的状态定义不同，大致可以定义为运行、挂起、睡眠、阻塞、就绪、终止这六种，如图 1-4 所示。

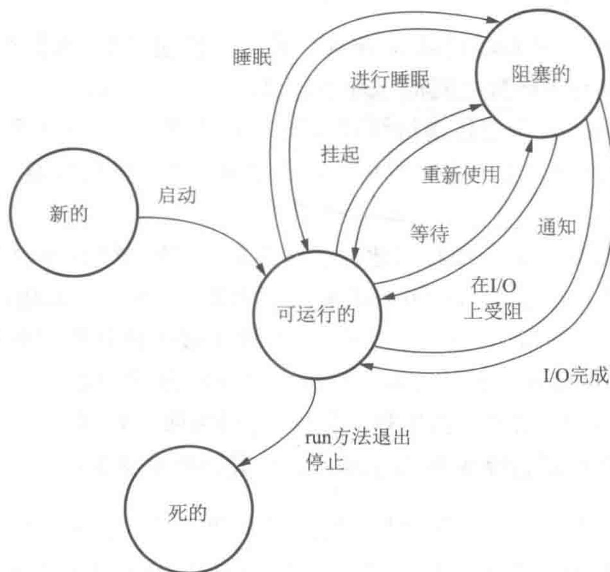


图 1-4　线程状态图

运行：是指线程获得了 CPU 的控制权正在执行计算。

挂起：一般是指被挂起，因为同一时刻需要“同步”运行的线程不止它一个，所以基于时

间片轮转的原则，它在独占了一段时间的CPU后被挂起，线程环境被压栈。

睡眠：一般是指主动挂起，这种情况在Windows平台不存在。

阻塞：与挂起和睡眠类似，都是失去CPU的控制权，但与挂起更相像，也是被挂起的。不同之处在于，被挂起的线程没有额外的表示，而被阻塞的线程会被记录下来，当等待的因素就绪后，线程会转为就绪状态。例如，在线程中调用一些类似WAITFORSINGLEOBJECT的系统服务函数，会引起线程控制权的一次裁决，从而挂起本线程，造成本线程的阻塞。挂起、睡眠、阻塞看起来差不多，但本质上还是有区别的。

就绪：指线程已获得运行的系统资源，等待系统调度。

终止：指线程执行完毕，结束生命周期。

常见的函数有以下几种：

- DWORD SuspendThread(HANDLE hThread);

该函数用于挂起指定的线程。如果函数执行成功，则线程的执行被终止。

- DWORD ResumeThread(HANDLE hThread);

该函数用于结束线程的挂起状态，执行线程。

- DWORD WaitForSingleObject(HANDLE hHandle,DWORD dwMilliseconds);

其中hHandle为要监视的对象（一般为同步对象，也可以是线程）的句柄；dwMilliseconds为hHandle对象所设置的超时值，单位为ms。

- DWORD WaitForMultipleObjects(DWORD nCount,const HANDLE * lpHandles,
  BOOL bWaitAll,DWORD dwMilliseconds);

其中参数nCount为句柄的数量，最大值为MAXIMUM_WAIT_OBJECTS(64)。lpHandles为句柄数组的指针，HANDLE的类型可以为(Event,Mutex,Process,Thread,Semaphore)数组。bWaitAll为等待的BOOL类型，如果为TRUE，则等待所有信号量有效再往下执行；如果为FALSE，则其中有一个信号量有效时就向下执行。dwMilliseconds同上。当在某一线程中调用该函数时，线程暂时挂起，系统监视hHandle所指向的对象的状态。如果在挂起的时间dwMilliseconds内线程所等待的对象就变为有信号状态，则该函数立即返回；如果超出时间已经到达dwMilliseconds，但hHandle所指向的对象还没有变成有信号状态，函数照样返回。参数dwMilliseconds有两个具有特殊意义的值：0和INFINITE。若为0，则该函数立即返回；若为INFINITE，则线程一直被挂起，直到hHandle所指向的对象变为有信号状态为止。

线程的等待是采用将自己阻塞而等待其他线程执行到当前位置或指向的对象已进入激活状态，以保证线程之间的执行顺序。

1.2.2 多核多线程并行计算的性能分析

为了更方便地描述多核多线程并行计算的性能，一般采用指标加速比来度量。加速比S定义如下：

$$S = \frac{T_s}{T_p} \tag{1-1}$$

其中，T_s表示单处理器上最优串行化算法计算的时间，T_p表示使用p个CPU处理器并行计算的时间。

当加速比 $S<1$ 时，表示并行计算的时间比串行计算还长，并行计算效率反而降低；当 $S<p$ 时，表示次线性加速；当 $S\approx p$ 时，表示线性加速；当 $S>p$ 时，表示超线性加速。加速比通常都小于 CPU 核数，只有极少数并行算法可以获得超线性加速比。例如，并行搜索工作量少于串行搜索工作量等算法，也有可能是由高速缓存产生的额外加速所导致的。因此，在并行算法设计中，加速比一般要向 CPU 核数靠近，如果加速比为线性加速，那么程序性能就会变好。

对于某些串行程序，并不是所有部分都可以用并行程序替代，有一部分必须要串行执行。令 W_s 为程序中的串行部分，W_p 为程序中的并行部分，则有：

$$W=W_s+W_p$$

根据 Amdahl 定律，在计算规模一定的情况下，加速比定义为：

$$S=\frac{W}{W_s+W_p/p} \tag{1-2}$$

令串行程序比例为 $f=\frac{W_s}{W}$，则式(1-2)为：

$$S=\frac{1}{f+(1-f)/p} \tag{1-3}$$

这是一个递增的函数，随着 CPU 核数 p 的增加，S 值也增加，即这个函数有上限，当 $p\to\infty$ 时，有：

$$S=\frac{1}{f}$$

图 1-5 为串行程序比例分别是 0.0，0.05，0.10，0.15，0.20 时的加速比随 CPU 核数 p 变化(这里核数取 2，4，8，16，24，48)的情形。从图中可以看出，在一定的情况下，加速比随核数的增加而增加，但是在 16 核之前增速较快，之后增速较为缓慢；当 CPU 核数一定时，加速比随着串行程序占比的减少而增加。因此，要想提高并行计算效率，需要综合考虑 CPU 核数和串行程序比例两个因素。

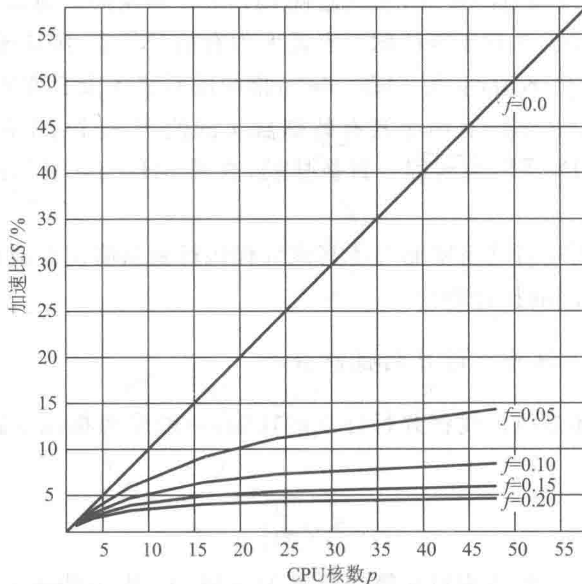


图 1-5 加速比随 CPU 核数 p 变化的情形

试读结束：需要全本请在线购买： www.ertongbook.com