文章编号:1005-0523(2010)03-0047-11

肿瘤图像特征统计中多线程技术应用研究

甘 岚,郑 鹏,高 捷,王雪虎,于忠平

摘要:分析了肿瘤图像分析诊断医疗软件中影响速度的瓶颈模块,并应用多线程技术对瓶颈模块进行改造。在单核 CPU 和多核 CPU 平台上比较了不同线程数下运行速度、线程效率、统计误差和线程数的关系。最后对上述指标的平衡进行了讨论,在不影响辅助诊断效果的前提下给出适合的多线程改造方案。

关 键 词:图像特征统计;多线程;多处理器;效率;平衡;误差率
中图分类号:TP391.41 **文献标识码**:A

受目前人工智能技术发展的限制,当今各种医疗图像处理软件在诊断准确率上的差异不是很大[1]。同时,医疗诊断是否及时,关系着患者的健康甚至能否被治愈。所以处理时间越短,越具有实用性和市场竞争力。

一般说来,在图像处理和进行特征统计的过程中,需要对位图的像素进行多次遍历操作。这些操作一般通过循环来实现。当位图较大时,循环次数非常多,非常耗时。

本文通过把输入的位图分割成若干小图,每块图用一个线程进行处理,一方面缩小了每个线程处理问题的规模;一方面增加程序的并行性,更加充分地利用多处理器体系结构,提高程序的执行效率,从而达到缩短处理时间的目的。文中还对该方法引起的误差进行了分析,提出折中的方案。

1 制约程序运行速度瓶颈分析

1.1 系统工作流程

进一步分析系统工作原理,在输入位图后,图像处理功能模块对图像进行处理并输出统计信息。图像处理模块的核心是特征统计子模块。该模块首先对输入的数字图像进行灰度化,然后用松弛迭代算法进行阈值分割,最后再进行边界跟踪,统计周长、面积等特征信息。系统工作流程如图 1 所示,可以看出这是一个串行结构。

1.2 代码分析

分析这段子模块的源代码,灰度化、阈值分割、边界跟踪和统计分别通过 CBmpImage 类的成员函数 ColorToGray( )、LosseRepeat(LPSTR lpDIB, int k)和 ForAreaTrackNuclear(int minvalue, int maxvalue, bool remove)实现。这 3 个函数代码中用到遍历数字图像像素的 for 循环共 17 个,而且这些 for 循环中有不少是两重以上的嵌套。在图像像素值较大时,这些代码的运行就非常费时。以实现边界跟踪和统计功能的 ForAreaTrackNuclear 函数为例,关键代码如下:

```
……
for(i=0;i<lHeight;i++){
```

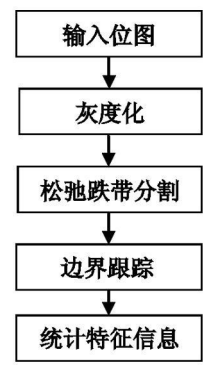

图 1 图像处理工作流程图

收稿日期:2010-04-16

作者简介:甘 岚(1964—),女,硕士,教授,研究方向为图信处理及模式识别、嵌入式系统。

```
for(j=0;j<lWidth;j++){
lpImage=m_pImage+lLineBytes*(lHeight-1-i)+j;
img[i][j] = *lpImage;//转存到二维数组中
img_label[i][j] = 0;//所有点都尚未标识区域
boudary_points[i][j] = false;//尚无边界点
if(img[i][j]==128)//背景换为值是 255 的白点{
*lpImage = 255;
img[i][j] = 255;
}
}
}
.....
```

这段代码包含了两重 for 循环，时间复杂度为 $O(n^2)$。例如，当图像长宽都为 100 像素时，for 循环中的代码执行 10 000 次；当图像长宽都为 200 像素时，循环中的代码执行 40 000 次。可见看出当图像稍大时，计算开销非常大。对于更多层的嵌套，时间复杂度会更大，时间消耗的增长速度将更快。

1.3 小结

从流程图中可以看出，程序是一个串行结构，查看源程序，也可以证实这一点。即 ColorToGray()、LosseRepeat(LPSTR lpDIB，int k)和 ForAreaTrackNuclear(int minvalue，int maxvalue，bool remove)函数后面的函数需要前一程序处理的结果作为输入，根据 Amdahl 定律，这即使程序运行在多核处理器机器上也不能得到很好的加速比。

图像处理模块因为要对像素进行遍历操作含有大量的 for 循环，其中包含了不少嵌套结构。从时间复杂度来看，设 q 是嵌套的层次数，则嵌套的 for 循环的时间复杂度为 $O(n^q)$，使得运行时间增长显著。

综上所述，特征统计子模块由于有大量 for 循环和串行结构，成为影响速度的瓶颈模块。

2 对瓶颈模块的多线程改造

2.1 多线程改造的思路

多线程的优点之一是能充分使用多处理器体系结构，以便每个线程能并行运行在不同的处理器上[2]。根据 Amdahl 定律和 Gustafson 定律[3]，在多核平台上，增加程序的并行性能获得良好的加速比，缩短运行时间。目前多核 CPU 已经越来越普及，双核处理器已成为当下主流中低档 PC 机配置。采用多线程技术可以充分发挥多处理器电脑的硬件优势，提高运行速度。另外，在单 CPU 的情况下，多线程技术的使用可以提高计算机资源的利用率，从而提高程序运行速度。所以本文尝试应用多线程技术来对特征统计子模块进行改造，提高运行的速度，并在单核核多核平台上进行验证。

经过分析图像处理模块的代码可以发现处理算法的各步骤间有着严密的顺序关系，后一步的处理依赖上一步的处理结果，这就使得改造时不能采用各步并发执行的方法。进一步发分析代码，可以发现这一模块中耗时较多的原因是在预处理、灰度化、分割和跟踪中需要对位图中每个像素进行扫描和处理，计算量比较大。要想提高速度，可以考虑降低计算量。如果把需要处理的位图面积减小，扫描的量就降低了，计算量也就下降了。如果利用多线程技术并发执行每个小图，最后把结果汇总就可能得到大图的处理结果。经过有关实验验证，这种方法是可行的。

另外，由于多线程执行的并发性(单核平台上并不是严格意义上的并发)，如果多个线程在执行过程中访问了相同的资源，就可能造成一些无法控制的错误[4]。比如某线程在对一变量先进行写操作，然后再读取。当它写入了数据后，分配给它的 CPU 时间片用完，下一线程开始执行，如果正在执行的线程对同一变量进行了写操作，就会覆盖上一线程的写入值，从而使得上一线程下次执行读操作时出现问题。在 VC++中可以通过互斥对象、事件对象和关键代码段等手段避免这种错误，但是会影响执行的速度和效

率,增加额外的系统开销,当线程比较多时反而造成较多的浪费。所以在对单线程程序进行多线程改造时,要尽量避免对同一资源的并发访问[5]。

2.2　MFC 的多线程机制

本系统基于 MFC 开发,可以利用其多线程机制来实现多线程的改造。MFC 中有两类线程,分别称之为工作者线程和用户界面线程。二者的主要区别在于工作者线程没有消息循环,而用户界面线程有自己的消息队列和消息循环。

工作者线程没有消息机制,通常用来执行后台计算和维护任务,如冗长的计算过程,打印机的后台打印等。用户界面线程一般用于处理独立于其他线程执行之外的用户输入,响应用户及系统所产生的事件和消息等。但对于 Win32 的 API 编程而言,这两种线程是没有区别的,它们都只需线程的启动地址即可启动线程来执行任务[6]。

在 MFC 中,一般用全局函数 AfxBeginThread()来创建并初始化一个线程的运行,该函数有两种重载形式,分别用于创建工作者线程和用户界面线程。两种重载函数原型和参数分别说明如下:

(1) CWinThread * AfxBeginThread(AFX-THREADPROC pfnThreadProc,LPVOID pParam,
nPriority=THREAD-PRIORITY-NORMAL,UINT nStackSize=0,DWORD dwCreateFlags=0,LPSECURITY-ATTRIBUTES lpSecurityAttrs=NULL);

pfnThreadProc:指向工作者线程的执行函数的指针,线程函数原型必须声明如下:

UINT ExecutingFunction(LPVOID pParam);

请注意,ExecutingFunction()应返回一个 UINT 类型的值,用以指明该函数结束的原因。一般情况下,返回 0 表明执行成功。

pParam:传递给线程函数的一个 32 位参数,执行函数将用某种方式解释该值。它可以是数值,或是指向一个结构的指针,甚至可以被忽略。

nPriority:线程的优先级。如果为 0,则线程与其父线程具有相同的优先级。

nStackSize:线程为自己分配堆栈的大小,其单位为字节。如果 nStackSize 被设为 0,则线程的堆栈被设置成与父线程堆栈相同大小。

dwCreateFlags:如果为 0,则线程在创建后立刻开始执行。如果为 CREATE-SUSPEND,则线程在创建后立刻被挂起。

lpSecurityAttrs:线程的安全属性指针,一般为 NULL。

(2) CWinThread * AfxBeginThread(CRuntimeClass * pThreadClass,
int nPriority=THREAD-PRIORITY-NORMAL,
UINT nStackSize=0,
DWORD dwCreateFlags=0,
LPSECURITY-ATTRIBUTES lpSecurityAttrs=NULL);

pThreadClass 是指向 CWinThread 的一个导出类的运行时类对象的指针,该导出类定义了被创建的用户界面线程的启动、退出等。

其它参数的意义同形式(1)。

一般情况下,调用 AfxBeginThread()来一次性地创建并启动一个线程,但是也可以通过两步法来创建线程:首先创建 CWinThread 类的一个对象,然后调用该对象的成员函数 CreateThread()来启动该线程。

MFC 应用程序的线程由对象 CWinThread 表示。在多数情况下,程序不需要自己创建 CWinThread 对象。调用 AfxBeginThread 函数时会自动创建一个 CWinThread 对象。本文对原程序的改造就使用了这种创建进程的方法。

2.3　多线程改造方法

2.3.1　对输入图像的分割

根据 2.1 节的思路,先把待处理图像根据处理线程的多少分成和线程数相等的子图,然后对每个子图

分别用一个线程进行处理、统计。

对输入图像的分割(用多少个线程处理就分成多少块)在 CimageView 类中的 OnLButtonUp 消息相应函数中实现,以下是其中的关键代码,实现从原图像中截取一块子图的功能:

```
HBITMAP segment-hBitmapnew;//定义位图句柄
HDC seg,memdcold,memdcnew;//定义 DC 句柄
CDC * pdcdelet =GetDC();//定义 DC 对象
seg = pdcdelet->m-hDC;
memdcold=::CreateCompatibleDC(seg);
memdcnew=::CreateCompatibleDC(seg);
segment-hBitmapnew=CreateCompatibleBitmap(seg,rectWidth,rectHeight);//创建一个空的位图
::SelectObject(memdcold,g-hBitmap);
::SelectObject(memdcnew,segment-hBitmapnew);
BitBlt(memdcnew,0,0,rectWidth,rectHeight,memdcold,part.left,part.top,SRCCOPY);/* 从原图上截取大小为 part.left * part.top 的区域复制到刚才创建的空位图中 */
ReleaseDC(pdcdelet);
DeleteObject(memdcnew);
DeleteObject(memdcold);
::DeleteDC(segp);
g-hBitmap=segment-hBitmapnew;/* 保存截取位图的句柄 */
```

2.3.2 信息的传递和综合

截取了子图后,要把子图的句柄传递给多线程函数进行图像处理和特征统计。为了避免多线程运行时多个线程访问相同的资源,分别用不同的全局变量记录这些句柄,每个处理子图的线程启动后通过全局变量访问自己要处理的子图像。

为了方便对各个线程得到的统计结果进行综合,同时也避免多线程访问冲突,每个处理子图的线程处理结果记录在不同的全局变量里。当所有线程运行结束(记录子图统计结果的各全局变量都被更新),访问这些记录了统计结果的全局变量,进行综合汇总得到整幅图的统计信息。

处理子图像的线程通过调用 AfxBeginThread 函数创建并启动,图像处理识别的代码添加到线程的处理函数 ThreadFunc(LPVOID lpParam)中(代码略)。

调用 AfxBeginThread 函数启动多线程的部分代码如下:

```
……
pThread1 = AfxBeginThread(ThreadFunc1,this);
pThread2= AfxBeginThread(ThreadFunc2,this);
……
pThread32= AfxBeginThread(ThreadFunc32,this);
….
```

调用的个数和线程数相同,程序中用 switch 语句控制。

2.3.3 运行时间的获取

为了定量分析程序和线程的运行时间,需要一定精度的计时器。在 Windows 平台下,常用的计时器有两种,一种是 timeGetTime 多媒体计时器,它可以提供毫秒级的计时。但这个精度对本实验而言太粗糙了,而且受消息响应的影响,误差较大。另一种是 QueryPerformanceCount 计数器,随系统的不同可以提供微秒级的计数。为了得到较准确的实验数据,本文采用了后者。

QueryPerformanceCount 计数器是通过使用 QueryPerformanceFrequency()和 QueryPerformanceCounter()函数实现的。QueryPerformanceFrequency()函数和 QueryPerformanceCounter()函数的原型如下:

```
BOOL QueryPerformanceFrequency(LARGE_INTEGER * lpFrequency);
```

```
BOOL   QueryPerformanceCounter(LARGE-INTEGER * lpCount);
```

数据类型 ARGE-INTEGER 既可以是一个 8 字节长的整型数，也可以是两个 4 字节长的整型数的联合结构，其具体用法根据编译器是否支持 64 位而定。该类型的定义如下：

```
typedef union -LARGE-INTEGER{
    struct{
        DWORD LowPart;// 4 字节整型数
        LONG   HighPart;// 4 字节整型数
          };
    LONGLONG QuadPart;// 8 字节整型数
    }LARGE-INTEGER;
```

程序中首先调用 QueryPerformanceFrequency()函数获得机器内部定时器的时钟频率，然后在线程创建处(计时的起点)和得到统计结果处(计时的终点)各调用一次 QueryPerformanceCounter()函数，利用两次获得的计数之差除以时钟频率，计算出事件经历的精确时间。定时误差不超过 1 微秒，精度与 CPU 等机器配置有关。

在多核 CPU 平台上运行时，不同 CPU 核的内部计数器不同步。如果程序两次读取这个计数器的时候恰好被轮换到不同的核上，那么用来计时就会有比较大的误差。为解决这个问题，本文采用设定线程亲核性的方法。即用函数 SetThreadAffinityMask 指定某线程只在某些核上运行(由第二个参数设定，每个位代表一个核)。例如，在进行精确计时那个线程里执行 SetThreadAffinityMask(GetCurrentThread(), 0_X00000001)；就能保证该线程只在第一个核上运行，不会因为不同 CPU 核计数器不同步而造成计时误差[7]。

3　实验分析

3.1　实验环境

表 1 给出了测试平台的软、硬件参数。

表 1　实验环境

项目	计算机 1 联想 H3610	计算机 2 戴尔 PowerEdge 2900 MLK 服务器
CPU	Pentium(R) Dual－Core CPU E5200	英特尔 Xeon E5405(4 核×2)
内存	2.0 GB	4.0 GB
操作系统	Microsft Windows XP SP3	Microsoft Windows 2003 Server (SP2)
编译环境	VC＋＋6.0	VC＋＋6.0

3.2　实验设计

3.2.1　实验对象

实验对像是从采集的组织切片位图照片库中随意选取的 4 幅大小不同的位图，如图 2 所示。

由于切片染色不同，颜色稍有区别。

3.2.2　实验指标设计

为了衡量用多线程对程序进行改造后并行运行所获得的性能收益，定义加速比 $q=t_c/t_b$：其中：t_c 表示未进行多线程改造前相应模块运行的时间，t_b 表示多线程该着后并行执行的时间。

为了衡量新增加线程提高运行速度的效率，定义净节省时间 $T'=\frac{T_n-T_{n+1}}{N_{n+1}-N_n}$，其中：$T_n$ 表示模块运行时间，N_n 表示线程个数，n 表示试验的序号。净节省时间 T' 表示线程增加后平均每个新增线程的

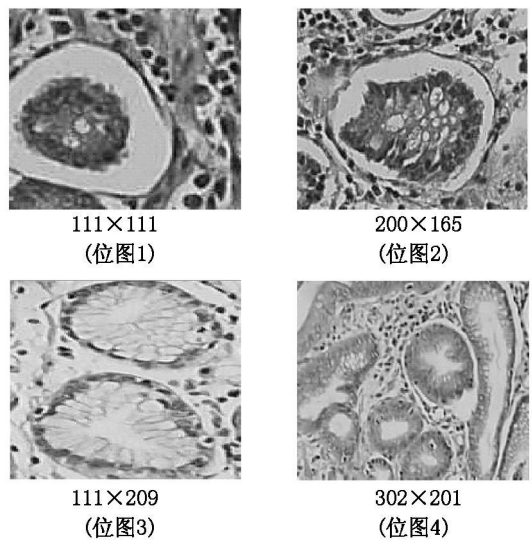

图 2　实验位图图片及其大小(单位：像素)

运行时间减小量，反映了新增线程对缩短整体运行时间的贡献度。$T'>0$ 时，说明线程的增加使运行时间缩短，T' 值越大，缩短的时间越多，贡献度越大，效率越高；$T'=0$ 时，说明线程的增多不能缩短运行时间；$T'<0$ 时，说明线程的增多不但不能提高运行速度，反而使运行时间增加。

为衡量多线程改造对统计结果的影响，定义误差率 $\eta=\frac{p_n-p_0}{p_0}\times 100\%$，其中 p_n 表示某次试验统计的结果，共有 4 个值：周长在阈值内的细胞核区域个数、平均周长(单位像素)、面积在阈值内的细胞核区域个数和平均面积(单位像素)。p_0 表示统计结果的正确结果，考虑到误差，用单线程时的统计结果多次取平均值近似代替，相应也有 4 个值。所以误差率 η 表示多线程改造后统计结果的误差量和正确值的比率。

3.2.3 实验方案和结果处理

对每幅图分别用单线程和多线程处理，多线程的线程个数从 2,4,8,16,32 个增加到 64 个。记录每次试验运行的时间和统计结果。每幅图共进行 7 组实验(每换一次线程数为一组)。程序运行时机器是动态运行的，不同时刻机器的软硬件环境可能会不同，从而使得运行时间有差异。为了保证结果准确，每组试验相同条件重复 5 次取平均值，减小偶然误差对结果的影响。

为了分析程序在多线程改造后在单核和多核 CPU 平台上性能的变化，在四核的 HP DL580G3 企业级服务器上通过 windows 任务管理器的“处理器关系”设置来选择执行进程的 CPU，分别在单核、双核和四核情况下按上述实验方法分别进行实验。采用这种方法可以尽可能使实验的软硬件环境只有 CPU 核数和线程数的变化，从而更好地比较性能与 CPU 数和线程数的关系。

3.3 实验结果和分析

3.3.1 加速比与 CPU 数及线程数的关系

加速比是衡量程序并行运行所获得的性能收益的指标，由 Amdahl 定律和 Gustafson 定律可知，理论上加速比的大小取决于程序中不可并行执行部分所占比例和 CPU 的数量，具体到本文就是取决于用多少线程改造原模块和多核 CPU 的核数。郑锋在文[8]中从理论上论证了减少程序中串行部分所占的比例，增加并行部分的比例比增加处理器核数对加速比的贡献更大。陈勇等人在文[9]中指出多处理器系统中要根据处理器的数量选择合适的线程数才能较好地提升程序运行速度。本文通过实验对以上观点进行验证，并找出加速比和 CPU 数及线程数的关系，为线程数和 CPU 数的选择提供依据。取单线程、单核 CPU 环境下运行时间为 t_c，多线程改造后运行时间为 t_b，由 $q=t_c/t_b$ 计算出各幅位图在不同 CPU 数和线程数下的加速比。发现在相同 CPU 数和线程数时，加速比因为位图大小而有一些差别，为便于说明问题，取 4 个加速比的均值进行比较。实验结果如表 2 所示。

表 2 平均加速比与 CPU 数和线程数的关系

平均加速比	CPU 核数=1	CPU 核数=2	CPU 核数=4	CPU 核数=8
线程数=1	1.000	1.007	1.010	1.011
线程数=2	1.075	2.121	2.134	2.131
线程数=4	1.129	2.205	4.327	4.331
线程数=8	1.174	2.294	4.389	8.345
线程数=16	1.274	2.498	4.547	8.528
线程数=32	1.398	2.726	5.140	9.576
线程数=64	1.561	3.052	5.696	9.918

从表中可以看出以下几点：

(1) 当线程数等于 CPU 核数时，加速比随着线程数或核数的增加近似呈线性增长，即加速比随着线程数或 CPU 核数加倍而加倍。

(2) 当线程数小于 CPU 核数时，加速比随着线程数的增加近似呈线性增长；随着 CPU 数的增加有限、缓慢地增长。

(3) 当线程数大于 CPU 核数时，加速比随着 CPU 数的增加近似呈线性增长；随着线程数的增加有限、

缓慢地增长。

(4) 单独增加 CPU 数或线程数都能提高加速比，但单独增加前者的提高幅度小于后者。

因为不管有多少 CPU，单线程进程只能运行在一个 CPU 上，当 CPU 数大于线程数时，会有 CPU 闲置；而当线程数大于 CPU 数时，会有多个线程竞争一个处理器，从而增加了系统的开销。所以只有在 CPU 数等于线程数时，每个处理器处理一个线程，才能使加速比线性增加(本实验存在一个前提：每个线程的工作量大约均等，优先级相同。如果各 CPU 上负载不平衡，也会影响并行的效果)。此外，现象(4)验证了文[8]中增加并行部分的比例比增加处理器核数对加速比的贡献更大的结论。究其原因，增加线程数从减少程序中串行部分所占的比例和减小处理问题规模两个方面提高程序的运行速度，其中减小处理问题规模方面与实际处理问题和算法有关，与硬件环境无关。增加 CPU 数从增加处理器数量一方面增加了程序的加速比，且要受到运行在其上的线程数的制约。考虑到增加处理器数量的成本高于增加线程的成本，且线程数大于处理器数后对加速比仍有一定的提升能力，在选择处理器和线程的数时，可以让线程数略大于处理器数。

因此，从尽可能提高加速比的目的出发，在硬件条件许可的情况下，尽量选择多核处理器，且处理器数量尽可能多。同时线程数大于处理器的数量。

另外对比相同位图在不同 CPU 核数条件下和在不同计算机上得到的程序运行速度、效率和线程数关系的折线图，发现 CPU 核数和硬件条件的变化对上述关系的影响仅限于折线的高低、拐点出现的线程数及拐点前后折线的斜率，不影响定性分析其中的关系。而对于误差率和线程数的关系则没有影响。为于接近一般医院硬件条件和便于论述，下文以在计算机 1 平台上所得到的实验结果为例来分析上述关系。

3.3.2　程序运行速度与线程数的关系

程序的运行速度用执行时间作为衡量指标，图 3 给出了 4 幅位图进行处理所需时间和线程数的关系。图中数据点是在线程为 1,2,4,8,16,32 和 64 时实验所得的数据。

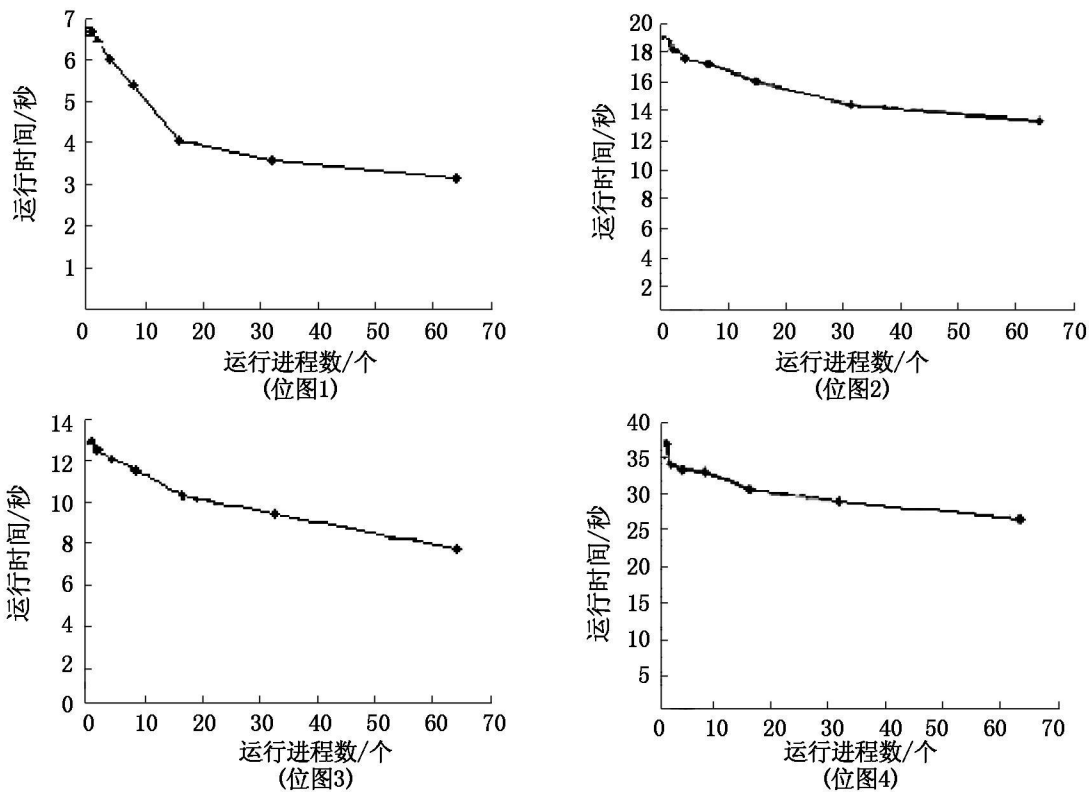

图 3　运行时间折线图

运行时间折线图反映了随着线程增多，处理该位图消耗时间的变化情况。从中可以看出多线程改造后，运行速度得到提高，运行时间随线程增多而减少。但是减小幅度随着线程数的增加越来越小，线程增加对程序速度的提升趋于停顿。这是因为增加线程会增加系统开销，系统开销一般由两部分组成：操作系统实际开销和线程间的活动开销，例如线程调度、同步以及其他形式的线程间通信开销。如果系统开销比较大，它会抵消因增加线程提高程序并行性带来的速度提高。从反映新增加线程提高运行速度的效率的净节省时间公式可以发现净节省时间实质上描述的是图中曲线的变化率。从图 4 净节省时间随着线程数增加趋于零的现象能推测出图 3 运行时间的减少是有极限的，并且根据文[10]中实验结论可以推测到随着线程数进一步增加，系统开销的增大将使运行时间反而增加。

3.3.3 多线程的效率与线程数的关系

多线程改造提高运行速度的效率用单个线程平均净节省时间衡量。实验结果如图 4。

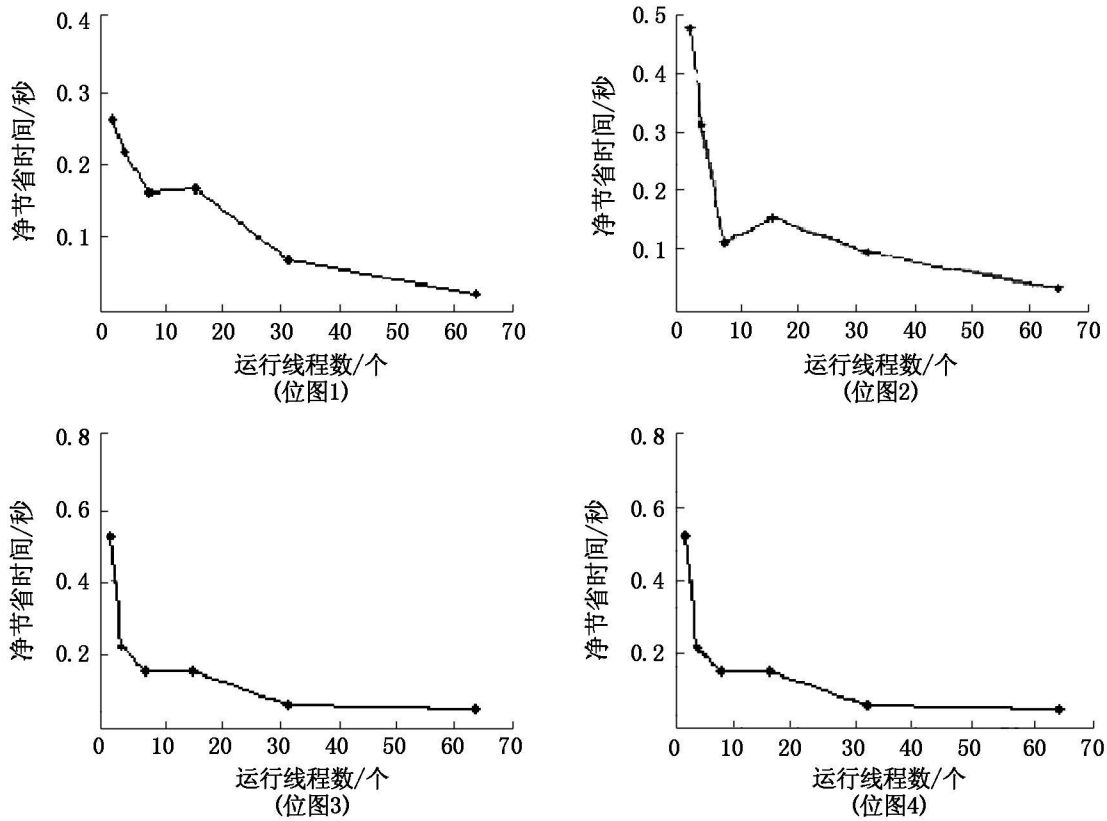

图 4 单个线程平均净节省时间折线图

图中数据点同样是在计算机 1 平台上线程为 1,2,4,8,16,32 和 64 时试验所得的数据。单个线程平均净节省时间折线图反映了随着线程增加，平均新增一个新线程能节省多少时间。从图中可以看出随着线程增加，平均每个线程节省的时间先呈降低趋势，在 10 到 20 线程之间时出现一个短暂回升的趋势，然后又降低，在线程数较大，如大于 30 时，平均每个线程节省的时间小于 0.1 秒，并呈进一步降低趋势，可以推测线程数更大时平均每个线程节省的时间趋于零，甚至为负值。这里的原因还是如上文分析的，线程的增加增加了系统的开销，当开销抵消了因增加线程提高程序并行性带来的节约时间后，新增线程不但不能降低运行时间，反而增加运行时间。折线图中短暂的回升原因仍然是线程增加提高并行性节约运行时间和系统开销随线程增加而增大增加运行时间这对矛盾的变化。当增加少量线程时，如增加 1 到 2 个进程，一方面，增加了程序并行性，提高了运行速度，一方面对操作系统来说负担并不大，系统开销主要是线程调度、同步和通信等开销。所以增加线程后虽然平均每个线程节约时间在降低，但仍然处在一个较高效率

上，如图 4 所示在第一个下降段前期(线程数小于 8)节约时间能维持在 0.1 秒以上。当线程进一步增多，如增加十几个线程，对操作系统的压力仍然不大，但因为线程数目较多，对系统资源如 CPU 时间的利用更加充分，所以此时净节省时间有一个回升的趋势，比之前的拐点值要高出一些。线程进一步增加，如增加到 30 个以上时，对操作系统产生较大的影响，使系统开销增长较快，净节省时间再次下降。

3.3.4　统计结果误差与线程数的关系

多线程时，由于要把待处理图像分成若干块，在子块的边缘可能造成细胞截断等现象，影响周长、面积信息的统计结果。实验中把每幅位图在单线程条件下重复运行 5 次所得的统计结果的平均值作为基准值，和多线程条件下的运行结果比较，得到误差率。4 幅位图的误差率如图 5 所示。

误差率折线图反映了统计的 4 个特征指标随线程增加误差的变化。从图 4 中可以看出一定面积核区域个数和一定周长核区域个数两个指标的误差规律比较明显，随着线程数增多，误差先反向增大，在拐点处开始减小，过零点后继续正向增大。平均面积和平均周长虽然折线不如另两个指标规律明显，但仍然呈一定的规律：随着线程增加，误差率先正向增大，到达拐点后开始减小，并有向负方向增大的趋势。从总体上看，前两个指标的误差率变化幅度比较平缓，且在线程数小于 8 时，都在 30%范围之内，线程数在 20 到 30 之间能够找到两者都处于误差正负 30%的范围内的点。后两个指标变化比较陡，误差先以比较快的速度正向增大，然后较缓地反方向减小，在线程数小于 8 时，误差表现的比较规律，都是正向误差，一般不会超过 100%，线程大于 8 时，规律不明显。

为了利用误差率的规律性减小误差，对模块进行多线程改造时线程数不超过 8，相应的 CPU 的数量也限制在 8 以内。

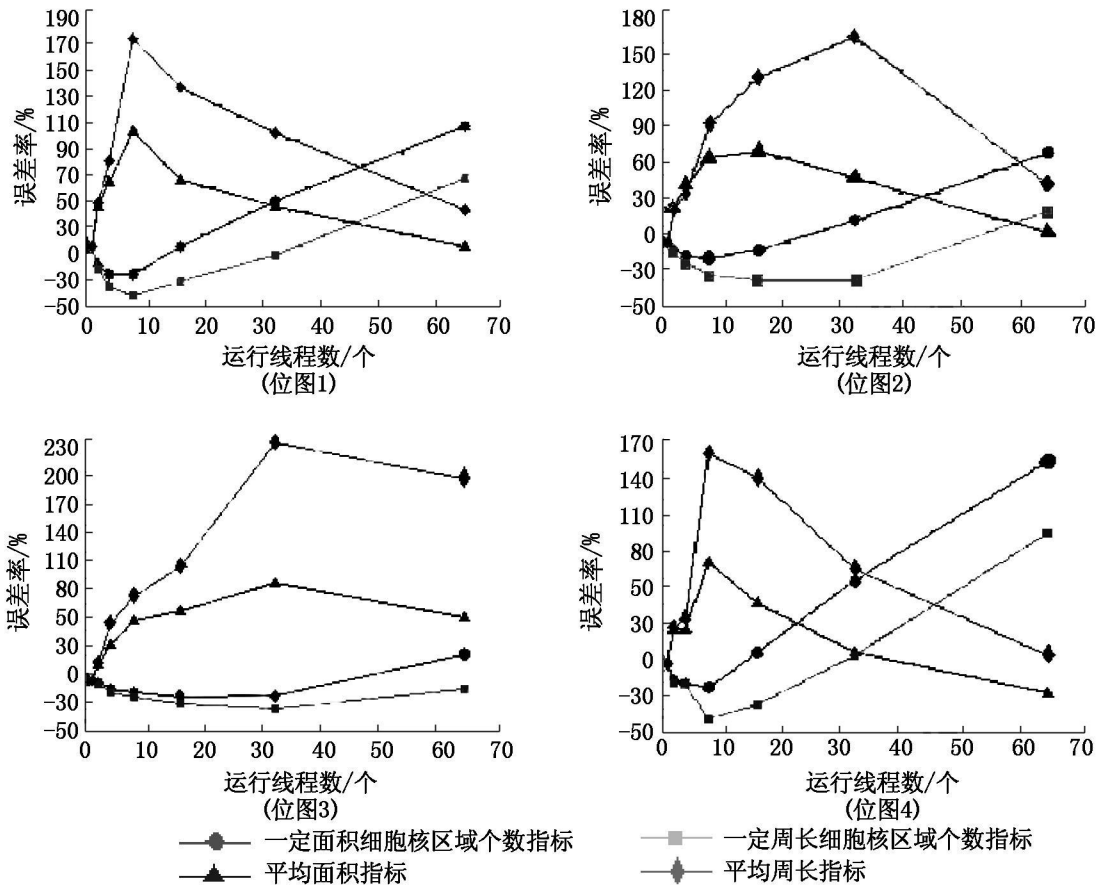

图 5　误差率折线图

3.4 误差分析

造成统计结果误差的主要原因是在对原图像进行分割时，子图像的边缘可能会有细胞被截断，从而影响了统计结果。由于图像的细胞腺体细胞是聚集的，有一定的粒度，当分块大小合适时，即选用合适的线程数，可以使大多数细胞不被截断，还可以从分割带来的误差中带来一定的补偿，减小统计结果误差。据医院方面试用的反馈信息，误差率不超过 30%时，对医疗辅助诊断影响不大。下面综合考虑速度、效率和误差指标，找到比较合适的线程数，使多线程改造造成的误差在可接受范围内。

根据 3.3 节对实验结果的分析，在实验条件下，线程数小于 20 时，多线程对速度的提升比较明显，且效率也比较高，增加线程带来的系统开销与收益相比较小。误差率的 4 个指标在线程数小于 8 时都比较有规律，可以引入一定的补偿(即统计结果减去或加上补偿量)，补偿量可以通过某种以统计值为自变量的经验函数求得。例如，设 x 为统计值，选用简单的线性函数 $y=\pm\alpha x$ 得到补偿量，其中 y 为补偿量；$\alpha\in(0,1)$为经验值，可通过试凑法得到；正负号的取值看指标的误差特点而定，是正向误差的(即结果比准确值大)取负号，反之取正号。则补偿后的统计值 x'可以由公式 $x'=x\pm\alpha x$ 得到。

在本实验条件下，按上述方法进行试凑，发现对一定周长核区域个数、平均周长、一定面积核区域个数和平均面积指标分别用补偿函数 $y=0.1x$、$y=-0.4x$、$y=0.2x$ 和 $y=-0.51x$ 进行补偿，可以使得四个指标的误差率在线程数小于 8 时全部控制在±30%之内，特别是一定周长核区域个数和一定面积核区域个数两个指标补偿后的误差率都在±20%之内。

4 结束语

综合上面的论述，考虑提高加速比和线程效率，控制误差范围等因素，在医院现有设备条件下，得到一个折中的多线程改造方案：选用双核或四核处理器的计算机作为使用平台，用 4～8 个线程对特征统计子模块进行多线程改造来提高程序的运行速度，同时采用补偿函数对统计结果进行补偿，使统计结果的误差在可接受范围内。

根据所提出的改造方案对系统进行了多线程改进，经过医院的试用，较好解决了系统运行过慢的问题，辅助诊断的效果也没有受到影响。

另外，对 3.4 节的所举的补偿函数进行测试，发现虽然能把补偿后的误差率控制在一定得范围内，但是难以把误差率进一步缩小。例如平均面积指标的误差率在 α 从小到大的取值过程中，误差率曲线在线程数小于 8 的部分会在 20%附近波动，无法使所有点的误差率都下降到 20%以内。这说明补偿函数的选取过于简单，对于要达到更高精度的情况，需要寻找更复杂的补偿函数。

参考文献：

[1] 蒋先刚. 显微图像处理系统的软件设计[J]. 华东交通大学学报，2001，18(1)：1-4.

[2] 李实，刘乃琦，郭建东. 多核架构下的多线程负载平衡[J]. 计算机应用，2008，28(12)：139-140.

[3] SHAMEEM A J R. Multi-core programming[M]. Beijing：PublishingHouse of ElectroniesIndustry，2006.

[4] 刘邦桂，李正凡. 用 Java 实现流式 Socket 通信[J]. 华东交通大学学报，2007，24(5)：110-112.

[5] 刘权胜，杨洪斌，吴悦. 同时多线程技术[J]. 计算机工程与设计，2008，29(4)：963-967.

[6] 张智丰，张亚荣，包丽梅. 在 VC++中利用 API 实现多线程编程实例[J]. 内蒙古民族大学学报(自然科学版)，2008，23(4)：382-385.

[7] NAMIR C S. Secrets of the Visual C++masters[M]. Beijing：Tsinghua University Press，1994.

[8] 郑锋. 图像分析多核并行计算类库的构建与优化[D]. 厦门：厦门大学，2008.

[9] 陈勇，陈国良，李春生，何家华. SMP 机群混合编程模型研究[J]. 小型微型计算机系统，2004，25(10)：1 763-1 767.

[10] 郭辉. 多线程的效率[J]. AJR. 计算机应用，2008，28(12)：141-143.

A Research on Multi-threading Technology and Its Application in Statistics of Tumor Image Feature

Gan Lan, Zheng Peng, Gaojie, Wang Xuehu, Yu Zhongping

(School of Information Engineering, East China Jiaotong University, Nanchang 330013, China)

Abstract: The paper analyzes the bottleneck module of the cancer diagnostic medical image analysis software, which influences the running speed, and reforms the bottleneck module by employing the multi-threading technology. Then it compares the running speed, thread efficiency and statistical errors run on different threads with different number and their relationship among the numbers of threads. Finally, the proportion of the various indicators is discussed, and a reform program for multi-threading technology is given without affecting diagnosis results.

Key words: image feature statistics; multi-threading; multi-processor; efficiency; proportion; error rate

(责任编辑　王建华)

(上接第 46 页)

参考文献:

[1] 郑明新, 马国正, 赵升, 等. 彭湖高速公路路基施工沉降观测与变形规律研究阶段报告[R]. 南昌: 华东交通大学, 2009.
[2] 蒋金平, 郑明新. 合六高速公路膨胀土工程特性试验研究[J]. 华东交通大学学报, 2008, 25(1): 7-11.
[3] 郑明新. 滑坡防治工程效果的后评价方法研究[M]. 南京: 河海大学出版社, 2007.
[4] 向科, 罗凤. 分层铺设土工栅格高填方路堤的稳定性验算[J]. 铁道勘察, 2005, 31(2): 47-49.
[5] 陈建峰, 石振明, 孙红. 加筋路堤稳定性综合分析方法[J]. 岩土力学, 2004, 25(2): 433-436.
[6] 陈祖煜. 土质边坡稳定分析—原理、方法、程序[M]. 北京: 中国水利水电出版社, 2003.
[7] 赵春生, 李安玉. 高速公路高填路堤软土地基处理及高填路堤施工技术[J]. 四川水利发电, 2009, 28(6): 14-17.

Stability Sensitive Analysis of High Embankment along Penghu Highway

Wu Zhiqing[1], Xie Lihui[2,3], Zuo Wei[2], Deng Tianqi[2]

(1. Jiangxi Ganyue Expressway Corporation, Nanchang, 330000 China; 2. School of Civil Engineering and Architecture, East China Jiaotong University, Nanchang, 330013 China; 3. China Railway No. 2 Engineering Group Co. Ltd., Chengdu, 610000, China)

Abstract: The physical nature of the indoor soil, compact test and direct shear test are carried out by means of on-the-spot investigation adopting high embankment along Penghu highway as the researching object. Based on c, φ values achieved in different degree of compaction, limit equilibrium method is used to analyze stability of high embankment under different cohesion c, friction angle φ, bulk density γ, slope height H and the slope ratio m. The sensitive relationship between various strength parameters and slope stability is also discussed. The preliminary result shows that effect of the internal friction angle φ on the slope stability is more sensitive than slope ratio m.

Key words: Penghu highway; high embankment; strength parameters; stability sensitive analysis

(责任编辑　刘棉玲)