高性能计算导论 HW4

计 71 张程远 2017011429

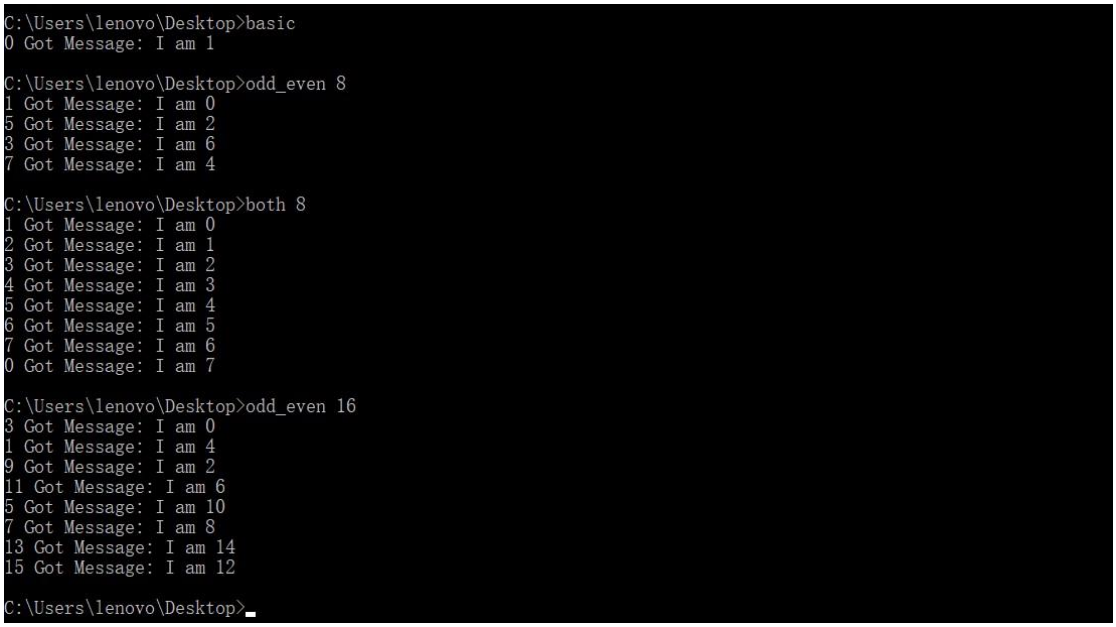
Ex 4.7

代码分 3 部分：basic 为基本实现；odd_even 实现了奇偶线程分别为消费者和生产者的功能；both 实现了“线程既可以当生产者又可以当消费者”的功能，即允许线程 Q 发送一条消息给线程(Q+1)mod t，同时又从(Q-1+t)mod t 接受消息。三个程序的代码基本按照书本 4.1 给的发消息的例程写成，只是添加了要求的 feature，故三个程序主函数基本相同（除了

Basic 指定了线程数为 2），核心代码在于 work 函数中。

主线程初始化为 false 的标志变量为 msg，它代表目前是否有可供接受的信息。若消费者首先进入循环，它看到没有可用的信息，于是解锁后返回，重复上述过程直到生产者产生出信息。Odd_even 在这一基础上，限定了只有奇数标号的线程才可以当消费者。Both 则额外加入了 send 和 recv 的标志变量，表示当前线程做了什么操作。生产者生产信息后，将指定 receiver，只有 receiver 进入消费区域流程才可以继续进行。

三个程序的运行效果如图所示。

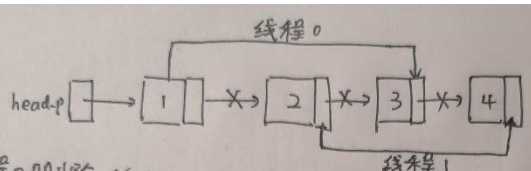


Ex 4.11

如图所示。

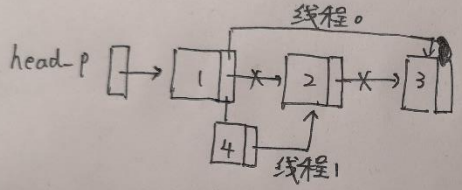
Ex 4.11

a):



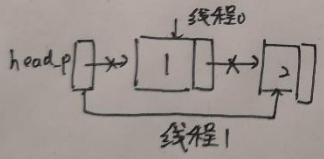
线程0删除 Node 2，而线程1 删除 Node 3，会导致问题.

b):



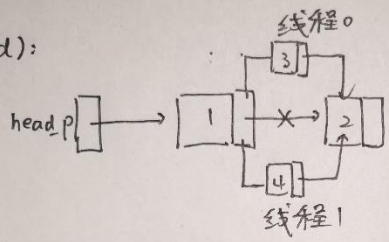
线程0删除Node2，而线程1在1与2间插入 Node 4. 会导致问题.

c):



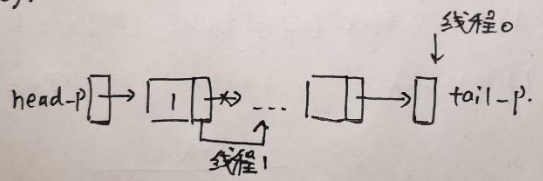
线程0找到3 Node1，在其找到 Node1 并即将返回True时，线程1已经删除3 Node1. 这样相当于线程0给出的答案是毫无意义的.

d):



线程0和1在同一处位置插入两个元素，后插入的结点会让指向先插入结点的指针指向自己，导致先插入结点离开链表.

e):



线程1可能在线程0跑完链表并认为[redacted]元素不在链表中时插入这一元素，导致线程0的查找结果毫无意义.

Ex 4.12

不是安全的。假设有两个线程 0 和 1 要对链表的同一个元素进行删除，两者都找到了要删除的位置。两个线程现在要释放读锁并添加写锁，假设 0 号线程先添加了写锁，那么 1 号线程的写锁就要等待；当 0 号线程删除完毕释放写锁后，1 号线程掌握的关于该节点的删除信息就完全失效，造成问题。

Ex 5.4

&&——1；||——0；|——0；^——0；

&：$1111\cdots\cdots111_2$

Ex 5.5

a. 1010.0； b. 对于线程 0 而言，结果 local_num0 为 4.0；对于线程 1 而言，结果为 1000.0（因为

1004.0 经过四舍五入被舍掉了）；两者相加，输出结果仍为 1000.0（因为 1004.0 四舍五入，4 被舍掉了）。

Ex 5.8

直接给出 a 数组关于 i 的计算公式以方便并行。a[i]=(i+1)*i/2，并行化的程序如下面所示（以 n 为 100 为例） `void cal(int a[]){`

```
 for(int i=0;i<100;i++){
  a[i]=(i+1)*i/2;
 }
}
int main(){
 int a[100];
#pragma omp parallel
cal(a);
 return 0;
}
```

Ex 5.14

a&b. 一个缓存行刚好能存下；但更多时候需要两行分别存储。

c. 8 种，对应第一个缓存行存储 0，1，…，7 个元素的时候。

d. 不考虑处理器先后的话，应该有 3 种，即按 01/23、02/13、03/12 三种方式进行分组。

e. 线程 0 处理 y0，y1；线程 1 处理 y2，y3；以此类推。如果不出现伪共享的情况，那么处理器分配到的两个线程，它们所拥有的 y 应该在一个缓存行里，并且该缓存行没有

其他的 y 元素。只能选择 01/23 的分配方式，并且把 y0~y3 分配给缓存行 1，y4~y7 分配给缓存行 2。

f. 3*8=24 种

g. 由 e 中分析知道只有一种可能。

PA 5.1 代码见附件。利用 omp 将循环分配给各个线程分别计数，在桶里计数时用 critical 保护数据防止出错。其余部分参考了 3.1 的程序。本地运行效果如下图所示。

```
C:\Users\lenovo\Desktop>g++ -g -Wall -fopenmp -o 5.1 5.1.cpp
5.1.cpp: In function 'int find_bin(int)':
5.1.cpp:40:1: warning: control reaches end of non-void function [-Wreturn-type]
 }


C:\Users\lenovo\Desktop>5.1 4
Please Enter data_count:40
Please Enter min_meas and max_meas:0 100
Please Enter bin_count:8

0.000-12.500:   XXX
12.500-25.000:  XXXXXXX
25.000-37.500:  XXXX
37.500-50.000:  XXX
50.000-62.500:  XXXXXXXX
62.500-75.000:  XXX
75.000-87.500:  XXX
87.500-100.000: XXXXXXXXX

C:\Users\lenovo\Desktop>
```

实际测试在本地计算机进行，结果多线程计算反而耗时较长，猜测是开辟多线程需要花费较多的时间。下面是在数据量较小和较大时的两组测试结果。

数据量\进程数	1	2	4	8
20000	0.001s	0.003s	0.003s	0.005s
20000000	1.075s	1.692s	1.922s	2.584s

PA P_Thread：代码见附件。这里采用的是 lld 存储已经计算出的结果（不然花费时间有点长），为了防止炸范围所以要求斐波那契数列的项数小于 50。Maxn 表示线程最大数目，cal 函数是将产生新线程的函数，所以如果线程数目达到了 maxn 的上限则剩余部分直接在本线程计算，即调用不产生新线程的函数 calc。

使用 OpenMP 实现一个递归问题其实是非常困难的，因为这里的递归会依赖于之前线程计算出的结果，所以我认为这里是不能使用 OpenMP 的。

本机的运行效果如下图。

命令提示符

```
C:\Users\lenovo\Desktop>g++ -g -Wall -lpthread -o pt pt.cpp
pt.cpp: In function 'void* cal(void*)':
pt.cpp:26:9: warning: unused variable 'r1' [-Wunused-variable]
     int r1 = pthread_create(&th1, NULL, cal, &k1);
         ^~

pt.cpp:31:9: warning: unused variable 'r2' [-Wunused-variable]
     int r2 = pthread_create(&th2, NULL, cal, &k2);
         ^~

pt.cpp: In function 'int main()':
pt.cpp:74:10: warning: unknown conversion type character 'l' in format [-Wformat=]
   printf("%lld\n",Data.fib[n]);
          ^~~~~~~

pt.cpp:74:10: warning: too many arguments for format [-Wformat-extra-args]
pt.cpp:74:10: warning: unknown conversion type character 'l' in format [-Wformat=]
pt.cpp:74:10: warning: too many arguments for format [-Wformat-extra-args]
pt.cpp:71:7: warning: unused variable 'r0' [-Wunused-variable]
   int r0 = pthread_create(&th, NULL, cal, &n);
       ^~


C:\Users\lenovo\Desktop>pt
Please enter the num which is less than 100:
5
Please enter max num of threads:
4
5

C:\Users\lenovo\Desktop>pt
Please enter the num which is less than 100:
24
Please enter max num of threads:
6
46368

C:\Users\lenovo\Desktop>pt
Please enter the num which is less than 100:
50
Please enter max num of threads:
3
12586269025
```