高性能计算作业1

18340052　何泽

习题1.1

> 为求全局总和例子中的 my_first_i 和 my_last_i 推导一个公式。需要注意的是：在循环中，应该给各个核分配数目大致相同的计算元素。（提示：先考虑 n 能被 p 整除的情况）。

这里假定元素以1开始而非以0开始

- 若n能被p整除，则每个核的计算数量都是$n \div p$

  令a为商，即$a = n \div p$

  则$my_first_i = 1 + ia,\ \ i = 0, 1, \ldots p-1$
  $my\_last\_i = (i+1)a,\ \ \ i = 0, 1, \ldots, p-1$

- 若n不能被p整除，那么可以整除的部分平均分配，剩下的余数按顺序一个核分配一个

  令a为商，b为余数，即$n = p \times a + b$

  - 对于前b个核

$$my\_first\_i = 1 + i(a+1),\ \ i = 0, 1, \ldots b-1$$
$$my\_last\_i = (i+1)(a+1),\ \ \ i = 0, 1, \ldots, b-1$$

- 对于后面剩下的$p-b$个核

$$my\_first\_i = b(a+1) + 1 + ia,\ \ i = 0, 1, \ldots p-b-1$$
$$my\_last\_i = b(a+1) + (i+1)a\ \ \ i = 0, 1, \ldots, p-b-1$$

习题1.6

在下列情况中，推导公式求出 0 号核执行接受与加法操作的次数。

a. 最初的求全局总和的伪代码。

b. 树形结构求全局总和。

c. 制作一张表来比较这两种算法在总核数是 2、4、8、……、1024 时，0 号核执行的接收与加法操作的次数。

- 最初的：n-1次接受，n-1次加法
- 树形结构：$\log_2$p次接受，$\log_2$p次加法
-

总核数	算法1	算法2
2	1	1
4	3	2
8	7	3
16	15	4
32	31	5
64	63	6
128	127	7
256	255	8
512	511	9
1024	1023	10

习题2.2

> 请解释在 CPU 硬件里实现的一个队列，怎么使用可以提高写直达高速缓存（write-through cache）的性能。

要想提高写直达缓存的性能，那么就要减少主存的存取次数，而队列是队尾进队首出，所以其实队列中间的元素对于读写操作并不是需要的，所以为了提高性能，可以只将队首、队尾元素放进缓存，只在有删除、插入元素时再更新队列缓存。

习题2.3

> 回顾之前一个从缓存读取二维数组的示例。请问一个更大矩阵和一个更大的缓存是如何影响两对嵌套循环的性能的？如果 MAX = 8，缓存可以存储 4 个缓存行， 情况又会是怎样的？在第一对嵌套循环中对 A 的读操作，会导致发生多少次失 效？第二对嵌套循环中的失效次数又是多少？

代码如下：

```
double a[MAX][MAX], x[MAX], y[MAX];

for (i = 0; i < MAX; ++i)
    for (j = 0; j < MAX; ++j)
        y[i] += a[i][j] * x[j];

for (j = 0; j < MAX; ++j)
    for (i = 0; i < MAX; ++i)
        y[i] += a[i][j] * x[j];
```

- 对于更大矩阵，对两个循环的效率都会降低，但是因为第一个循环操作的内存地址是相邻的，而第二个不是，所以更大的矩阵对第一个的效率降低更为明显。
- 对于更大缓存，对两个循环的效率都会提高，也是因为第一个循环操作的内存地址是相邻的，而第二个不是，所以对第一个循环效率提升更明显。
- 如果 MAX = 8，缓存可以存储 4 个缓存行
    - 在第一对嵌套循环中对，一行有2次缺失，一共就是$2 \times 8 = 16$ 次失效
    - 在第二对嵌套循环中对，一列有8次缺失，一共就是$8 \times 8 = 64$ 次失效

习题2.16

1. 假定一个串行程序的运行时间为$T_{串行} = n^2$，运行时间的单位为毫秒。并行程序的运行时间为$T_{并行} = n^2/p + \log_2(p)$。对于 n 和 p 的不同值，请写出一个程序并找出这个程序的加速比和效率。在 $n = 10$、20、40、…、320 和 $p = 1$、2、4、…、128 等不同情况下运行该程序。当 p 增加、n 保持恒定时， 加速比和效率的情况分别如何？当 p 保持恒定而 n 增加呢？

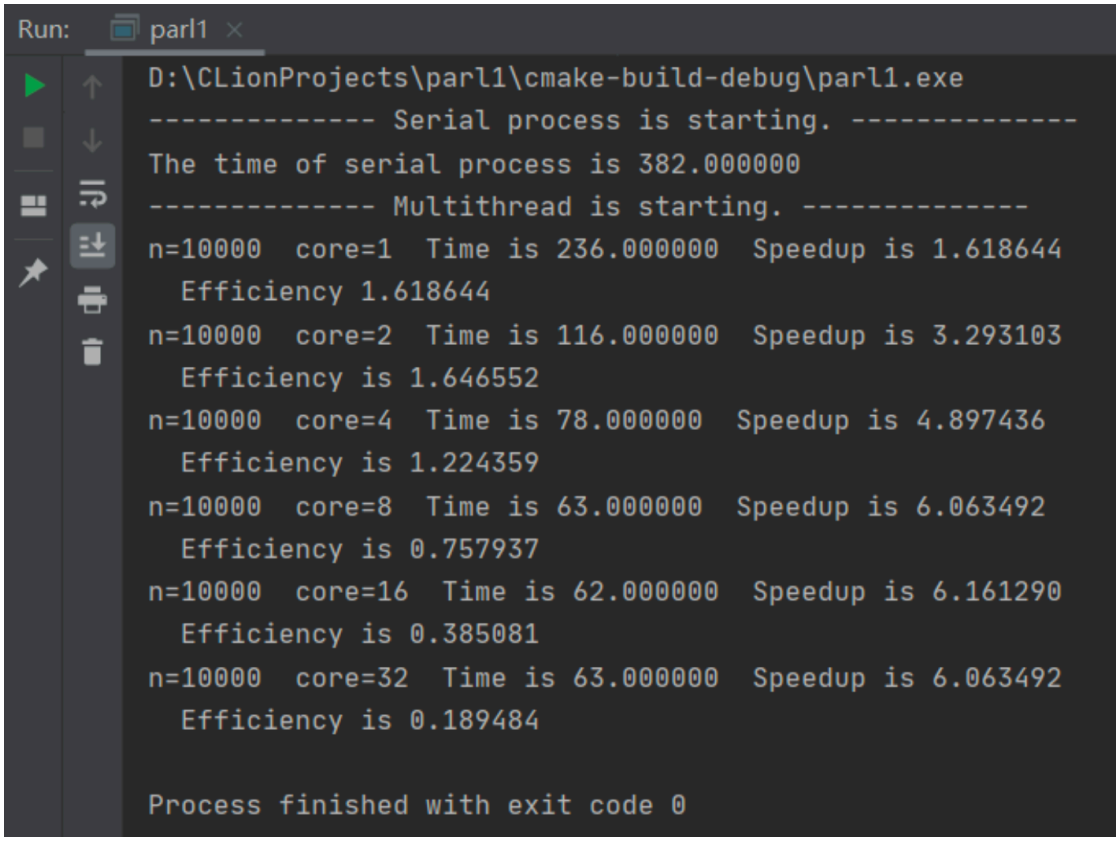
2. 假设 $T_{并行} = T_{串行}/p + T_{开销}$，我们固定 p 的大小，并增加问题的规模。
    - 请解释如果 $T_{开销}$ 比 $T_{串行}$增长得慢，随着问题规模的增加，并行效率也将增加。
    - 请解释如果$T_{开销}$ 比$T_{串行}$增长得快，随着问题规模的增加，并行效率将降低。

- 1.我跑的程序很简单，两个二维矩阵相加,这个程序串行运行时间为$T_{串行}=n^2$

```
void add(int p)
{
   for(int i=p;i<p+N/core;i++){
      for (int y = 0; y < p+N/core; y++) {
         c[i][y]=a[i][y]+b[i][y];
      }
   }
}
```

这个的时间复杂度就是n^2 ， 我的并行策略就是p个线程每个计算n/p行

  - 对于n保持恒定时，为了准确我把数据量增大了一些，到10000，结果如下：



虽然线程最多只跑了32个，但已经能看出结果，n恒定p增加，加速比先增加后降低，效率降低

  - 对于p保持恒定时，结果显示，n增加，加速比和效率都增加

- 2.

$$Efficienty = \frac{T_{串行}}{T_{并行} \times p} = \frac{T_{串行}}{\left(\frac{T_{串行}}{p} + T_{开销}\right) \times p} = \frac{T_{串行}}{T_{串行} + T_{开销} \times p} = \frac{1}{1 + \frac{T_{开销}}{T_{串行}} \times p}$$

于是显而易见

- $T_{开销}$ 比 $T_{串行}$增长得慢，$\frac{T_{开销}}{T_{串行}}$ 减小，效率增加
- $T_{开销}$ 比 $T_{串行}$增长得快，$\frac{T_{开销}}{T_{串行}}$ 增大，效率降低

习题2.17

如果一个并行程序所获得的加速比可以超过 p（进程或线程的个数），则我们有时称该并行程序拥有超线性加速比（superlinear speedup）。然而，许多作者并不讲能够克服“资源限制”的程序视为是拥有超线性加速比。例如，当一个 程序运行在一个单处理器系统上时，它必须使用二级存储，当它运行在一个大 的分布式内存系统上时，它可以将所有数据都放置在主存上。请给出另外一个 例子，说明程序是如何克服资源限制，并获得大于 p 的加速比的。

在并行计算中，有时不同处理器的高速缓存是集合使用的，有时集合的缓存便足以提供计算所需的存储量，算法执行时便不必使用速度较慢的内存，那么在这种情况下存储器读写时间便能大幅降低，这便对实际计算产生了额外的加速效果，便有可能获得大于p的加速比。

习题2.19

假定$T_{串行} = n$，$T_{并行} = n/p + \log_2(p)$，时间单位为毫秒。如果以倍率 k 增加 p，那么为了保持效率值得恒定，需要如何增加 n？请给出公式。如果我们将进程数 从 8 加倍到 16，则 n 的增加又是多少？该并行程序是可扩展的吗？

- $$Efficienty = \frac{T_{串行}}{T_{并行} \times p} = \frac{n}{\left(\frac{n}{p} + \log_2 p\right) \times p} = \frac{n}{n + p \times \log_2 p} = \frac{1}{1 + \frac{p \times \log_2 p}{n}}$$

令n要变为t倍，那么要让效率不变，有等式：

$$\frac{p \times \log_2 p}{n} = \frac{kp \times \log_2 kp}{tn}$$
$$log_2 p = \frac{k \times (\log_2 k + \log_2 p)}{t}$$
$$t = \frac{k \times (\log_2 k + \log_2 p)}{log_2 p}$$

所以t增加的倍率为$\frac{k \times (\log_2 k + \log_2 p)}{log_2 p}$

- 如果将进程数 从 8 加倍到 16，即$k = 2$，代入，$t = \frac{2 \times (\log_2 2 + \log_2 8)}{\log_2 8} = \frac{2 \times (1+3)}{3} = \frac{8}{3}$，即增加为$\frac{8}{3}n$
- 是可扩展的，因为并行程序可扩展的定义 一个并行程序，如果问题的规模与进程/线程数都以一定的倍率增加，而效率保持一个常数值，那么该并行程序就是可扩展的，那么有了以上的结论后，显而易见是可扩展的。

习题2.20

> 一个可以获得线性加速比的程序是强可扩展的吗？请解释。

是，因为当线性加速比时，$T_{并行} = T_{串行} \div p$，即$Efficienty = \frac{T_{串行}}{T_{并行} \times p} = \frac{T_{串行}}{T_{串行} \div p \times p} = 1$，可以看到，效率恒为1，那么是强可扩展的。