先简单讲述一下代码：

根据 culler 流程：

以分配 1001 个任务，一个任务对应一个线程为例，

补充 1001 个任务，两个任务对应一个线程以及

501 个任务一个任务对应一个线程。

首先分解：

```
22 void* f(void *a){
23         A * b = (A*) a;
24         int t = 0;
25         for(int i = 0; i < 1000; i ++){
26                         t = t +((*(b->a))[0][i]) * ((*(b->b))[i][0]);
27                         }
28         cum[*(b->i)] = t;
29         num ++;
30
31         }
32 void* SUM(void * a){
33         while(num != 1000){}
34         for(int i = 0; i  < 1000;  i++){
35                 sum += cum[i];
36                 }
37                 }
```

我将矩阵乘法按数据划分的方法分解成 1*1000 的矩阵乘以 1000*1 的矩阵的 1000 个任务

```
pthread_t thread[1000];
vector<int> ori1(1000,1);
vector<vector<int> > a(1000,ori1);
vector<int> ori2(1,1);
vector<vector<int> > b(1000,ori2);
//subproblems
vector<vector<int> > a1(1,ori1);
for(int i = 0; i < 1000 ;i ++){
        a1[0] = a[i];
        A *aa = new A(a1,b, i);
        //assignment
```

为了突出第三步的线程之间的 orchestration
我增加了求和函数（求出结果矩阵的所有行）来体现线程之间的同步与通信。

然后将为每个任务分配一个线程上：

乘积任务的线程分配 `pthread_create(&thread[i], NULL,f, (void*)aa);`
求和任务的线程分配

```
pthread_t th;
pthread_create(&th, NULL, SUM, NULL);
```

Orchestration

通过共享内存实现通信，当然 sum 函数通信前其他线程需要同步一下：
1000 个乘积线程之间的同步通过 num 变量告知 sum：
乘积线程的求和结果通过 cum 与求和线程进行通信。

```
int num = 0;
```

```
void* f(void *a){
        A * b = (A*) a;
        int t = 0;
        for(int i = 0; i < 1000; i ++){
                        t = t +((*(b->a))[0][i]) * ((*(b->b))[i][0]);
                        }
        cum[*(b->i)] = t;
        num ++;

        }
void* SUM(void * a){
        while(num != 1000){}
```

最后就是映射

映射老师课上说过交由操作系统或者编译器来实现
（总体代码见附件。）

补充：

将两个任务划分给一个线程：

线程由 f1 函数创建，f1 中执行两个 f 任务（乘积函数）

```
void f(A *b){
        int t = 0;
        for(int i = 0; i < 1000; i ++){
                        t = t +((*(b->a))[0][i]) * ((*(b->b))[i][0]);
                        }
        cum[*(b->i)] = t;

        num ++;
        }
void* f1(void *a){
        C * b = (C*) a;
        int t = 0;
        for(int i = 0; i < 500; i ++){
                        f(b -> a);
                        f(b -> b);
                        }


        }
```

创建 501 个任务：

一个乘积任务计算 2*1000 与 1000*1 的矩阵的乘积：

```
void* f(void *a){
         A * b = (A*) a;
         int t = 0,t1 = 0;
         for(int i = 0; i < 1000; i ++){
                         t = t +((*(b->a))[0][i]) * ((*(b->b))[i][0]);
                         }
         for(int i = 0; i < 1000; i ++){
                         t1 = t1 +((*(b->a))[1][i]) * ((*(b->b))[i][0]);
                         }
         cum[2*(*(b->i))] = t;
         cum[2*(*(b->i)) + 1] = t1;
         num ++;
         }
```

性能分析：

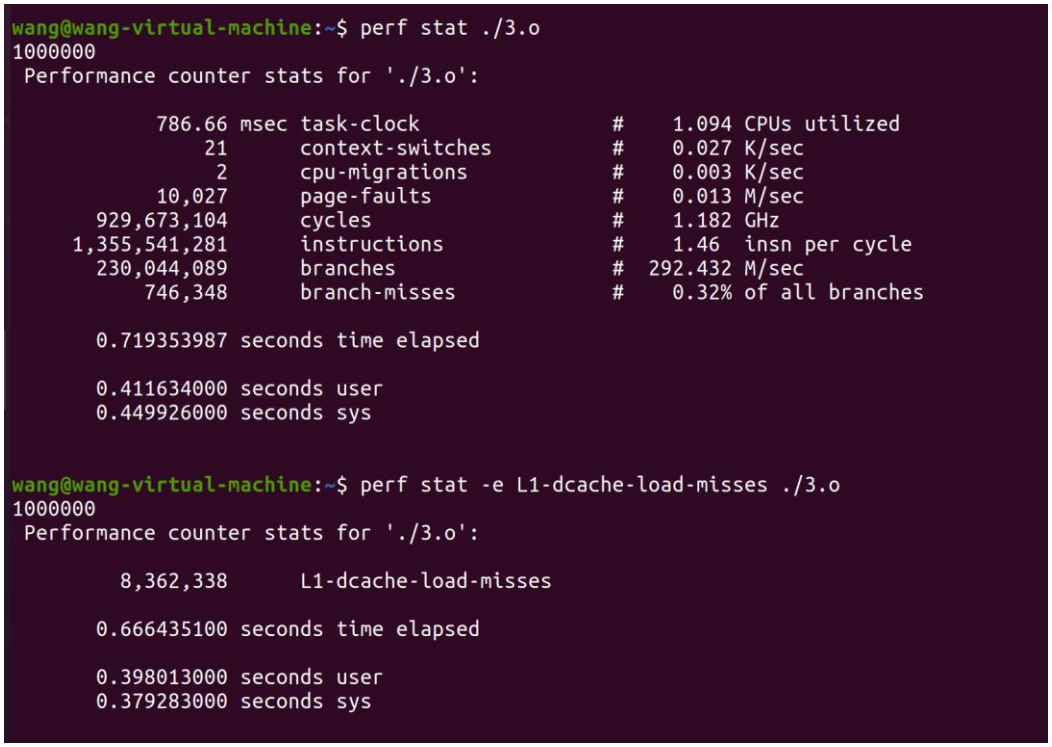
1001 个任务 1001 个线程：

```
wang@wang-virtual-machine:~$ perf stat ./1.o
1000000
 Performance counter stats for './1.o':

          1,343.92 msec task-clock                #    1.065 CPUs utilized
                60      context-switches          #    0.045 K/sec
                 1      cpu-migrations            #    0.001 K/sec
            17,921      page-faults               #    0.013 M/sec
     1,807,408,964      cycles                    #    1.345 GHz
     2,612,902,430      instructions              #    1.45  insn per cycle
       447,781,802      branches                  #  333.191 M/sec
         1,705,463      branch-misses             #    0.38% of all branches

       1.262353745 seconds time elapsed

       0.629564000 seconds user
       0.842505000 seconds sys


wang@wang-virtual-machine:~$ perf stat -e L1-dcache-load-misses ./1.o
1000000
 Performance counter stats for './1.o':

        15,958,826      L1-dcache-load-misses

       1.481078831 seconds time elapsed

       0.871681000 seconds user
       0.804628000 seconds sys
```

1001 个任务 501 个线程:

```
wang@wang-virtual-machine:~$ perf stat ./2.o
1000000
 Performance counter stats for './2.o':

         14,954.95 msec task-clock                #    1.973 CPUs utilized
             4,148      context-switches          #    0.277 K/sec
                69      cpu-migrations            #    0.005 K/sec
            16,892      page-faults               #    0.001 M/sec
    21,409,036,420      cycles                    #    1.432 GHz
    43,624,729,608      instructions              #    2.04  insn per cycle
     5,484,069,670      branches                  #  366.706 M/sec
         2,761,394      branch-misses             #    0.05% of all branches

       7.580793030 seconds time elapsed

      14.702625000 seconds user
       0.267103000 seconds sys


wang@wang-virtual-machine:~$ perf stat -e L1-dcache-load-misses ./2.o
1000000
 Performance counter stats for './2.o':

       493,183,919      L1-dcache-load-misses

       7.693232589 seconds time elapsed

      14.699963000 seconds user
       0.325871000 seconds sys
```

501 个任务 501 个线程:



可以看出在此案例中相同的任务数量，但线程数不同时，影响程序性能的主要因素是任务分配给线程时额外增加的指令数量，CPU 的迁移，以及 cache 的缺失：从图中我们可以观察到 1001 个任务 501 个线程代码所编译出的指令数量是 1001 个任务 1001 个线程的 20 倍，cache 缺失的数量达到 40 倍。可能是由于我在一个线程中调用了两个函数，由此导致了分支的增加，引起代码量以及 cache 缺失率的增加。

而相同线程数量，但任务数量不同时，由于在此问题中，数组比较大，数据空间局部性以及时间局部性较强，所以任务处理数据的量增加时可以减少 cache 缺失。