1,代码如下

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

int main(int argc, char const *argv[])
{
    int n = 658;
    int mark[n];
    for(int i=0;i<n;++i)
        mark[i]=0;
    int count=0;int k=2;
#pragma omp parallel
    for(int k=2; k*k<n; ++k)
    {
        if(!mark[k])
            for(int j=k;j*k<n;++j)
                mark[j*k]=1;
    }

#pragma omp parallel for reduction(+:count)
    for(int i = 2; i < n; ++i)
    {
        if(mark[i] == 0)
            count++;
    }

    printf("%d",count);
}
```

或者另一种方式：计算$\sqrt{n}$之前的素数(如果$\sqrt{n}$过大，计算$\sqrt{\sqrt{n}}$之前的素数，如果还是很大继续迭代，并利用计算得到的素数执行下一步)，用openmp并行化寻找$\sqrt{n}$(或$\sqrt{\sqrt{n}}$等) 到n(或$\sqrt{n}$等)的素数，每个线程执行一样的代码，通过线程id确定各个线程的数值区间如l/q(l为计算的总区间

的长度，q为线程数)的素数。之后各个线程计算自己区间内的素数个数并相加。

2,当任务数相差过大时，比如一个线程任务队列已满，而另一个线程任务队列为空，或者该程序分配的任务数比较多，但每个任务数的粒度比较小时，窃取一半比较适用,；而当给各个线程分配的任务数少，并且每个线程被分配的任务的粒度较大时，窃取一个任务比较合适。

设计方法:可以预先估计每个任务的粒度大小(可通过程序员设置相应的数值等方法)，让窃取者计算被窃取者的总的任务粒度，根据任务粒度窃取合适的任务。此时如果各个任务的粒度较大，则窃取的任务少；窃取的任务粒度小，则窃取的任务多。