華中科技大學

研究生课程报告

多线程求素数

学　　　　号	M202470678
姓　　　　名	梅硕
专　　　　业	机械工程
课程指导老师	周健
院（系、所）	机械科学与工程学院

2024 年 12 月 24 日

目 录

1 问题描述

我们公司面临一个任务：需要查找 1 到 10^9 之间的所有素数。计划使用 8 个并发线程的来加速这一过程。通过设计一个高效的程序，既能确保计算准确，又能尽量缩短计算时间，降低机器使用的成本。（由于 10^8 范围内，使用筛法求素数效率极高，使用线程带来的常数提升无法弥补使用线程的开销，因此考察 10^9）

强调一下所求范围为 10^9，比原题高一个数量级

2 解决方案

高效求取素数一个有效的算法是埃拉托斯特尼筛法（Sieve of Eratosthenes）。该算法通过遍历数字并标记每个素数的倍数来高效地筛选出非素数。埃拉托斯特尼筛法的时间复杂度为$\mathcal{O}(nloglog(n))$，尽管该方法的时间复杂度并不是求素数的方法中最优的，通过筛至平方根和使用 C++ 类 bitset 的优化方式，在给定的求素数范围内可以得到比欧拉筛法更好的效果[1]，同时该方法更易于并行计算。此外，还可以使用只筛奇数的方法对程序进一步优化，下面对使用的方法做进一步的描述。

2.1 只筛奇数优化

给出埃式筛伪代码之前，首先介绍一下本程序中进行只筛奇数的优化的具体方法。我们使用如下的映射，即 i 映射到 $2i+1$，程序中使用下标表示原像，下面给出在该映射下乘法和平方数的计算规则。记 $\circ$ 为该体系下的乘号。

$$i \Rightarrow 2i+1 \tag{2.1}$$

$$i(\Rightarrow 2i+1) \circ j(\Rightarrow 2j+1) = 2ij+i+j(\Rightarrow 4ij+2i+2j+1) \tag{2.2}$$

$$i(\Rightarrow 2i+1) \circ i(\Rightarrow 2i+1) = 2i^2+2i(\Rightarrow 4i^2+4i+1) \tag{2.3}$$

然而如果仅仅做这样的映射，实际上计算性能会有所降低，因为尽管避免了偶数项的计算，奇数项的乘法计算却从一次乘法变为了两次乘法和两次加法，我们使用 C++ 位移运算符代替 2 的乘法，可以将计算效率提高。即最终的乘法计算公

式为：

$$i \circ j = (ij << 1) + i + j \tag{2.4}$$

$$i \circ i = ((i+1)i << 1) \tag{2.5}$$

这样可以将奇数项的计算变为一次乘法两次加法和一次位移，对运算效率有一定的帮助。程序中均使用该方法来进行下标的计算，但后续的算法描述中为了简洁，均不对该映射做任何说明。

2.2 常数优化

在 10^9 数据范围内使用只筛奇数筛至平方根的埃式筛相当于时间复杂度为$\mathcal{O}(n)$，并存在一个常数倍数约为 1.134，多线程对于线性时间夫再度算法的优化也只能是常数项的提升，为了达到更为严苛的测试环境，依据[1] 中提出的，对埃式筛使用 bitset 类进行常数优化，以验证多线程方法的有效性，即线程调度的开销不会导致负优化的产生。

2.3 单线程埃氏筛

埃式筛的算法基于素数的定义，将为素数倍数的数标记为合数，剩下未标记的即为素数。算法相当简单，为方便后续说明，给出伪代码如下。

算法 2.1. 埃式筛伪代码

Algorithm 2.1 Eratosthenes Sieve

```
Input: UpBound N
Output: Array PrimeFlag[N]
Initialize PrimeFlag[N] as all true
PrimeFlag[0] = false, PrimeFlag[1] = false
for i = 2, ..., √N do
    if PrimeFlag[i] then
        for j = i, ..., N do
            if i · j > N then break
            end if
            PrimeFlag[ i · j ] = true
        end for
    end if
end for
return result
```

2.4 多线程埃氏筛

接下来考虑如何对该算法进行并行计算。该算法主要的操作集中在第二个循环内部，需要对 i 的倍数在待求区间内进行标记。一个显然的并行化策略是将待标记区间按线程数 t 均匀分为 t 段，每段由不同线程进行计算，从而加快运算效率。该方法在对于 i 的倍数进行标记时的局部范围内各个线程互不干扰高度并行。然而对于不同的 i，线程间需要进行同步，以避免多个线程对同一个标记数组的相同区域进行操作。每个线程将自己的区域标记完成后，等待其他线程的的操作完成，才能继续外部对于 i 的访问循环。使用基本的同步屏障 Barrier 方式可以完成这一任务。

算法 2.2. 多线程埃式筛伪代码

Algorithm 2.2 Eratosthenes Sieve

Input: UpBound N Threadid id Threadnum n
Output: Array PrimeFlag[N]

Initialize PrimeFlag[N] as all true
PrimeFlag[0] = false, PrimeFlag[1] = false
for $i = 2, ..., \sqrt{N}$ **do**
  if PrimeFlag[i] **then**
    $jmax = \lfloor N/i \rfloor + 1$
    $jmin = i$
    $piece = (jmax - jmin)/n$
    $startj = id \cdot piece$
    $endj = (id + 1) \cdot piece$
    for $j = startj, ..., endj$ **do**
      if $i \cdot j > N$ **then break**
      end if
      PrimeFlag[$i \cdot j$] = true
    end for
  end if
  Barrier wait for all threads finish
end for
return result

需要说明的是其中一些细节方面的处理，如整除问题和保证数组访问不越界的限制被略去了，如需了解可以参考源代码。

3　实验结果及分析

3.1　实验结果

实验环境：**Windows11 wsl2 Ubuntu20.04**

对于单线程的优化已经达到相当高的水平，在这样的条件下，多线程仍能表现出比单线程更优秀的性能表现。

表 3-1　单线程与多线程耗时

	单线程	多线程
用时	$2215ms$	$1000ms$

表 3-1 数据表明，使用 8 个线程计算能够进一步提高筛法计算的效率，优化后的算法运行时间从 2215ms 降至 1000ms，优化后的算法运行时间从 2215ms 降至 1000ms，实现了约 55% 的时间节省和约 2.2 的加速比，能够更为有效地利用 cpu 计算能力和内存 IO 性能。当然，需要说明的是，实验中测试结果是有一定浮动的，但基本保持在该值附近，因此选取上述取值。

此外，给出每条线程标记素数部分的用时作为线程负载均衡情况的参考。

表 3-2　线程素数标记用时

线程 id	0	1	2	3
用时	$882110us$	$882110us$	$882100us$	$882075us$
线程 id	4	5	6	7
用时	$882012us$	$881931us$	$882104us$	$882146us$

依据表 3-2 中的数据，可知线程负载基本是均衡的，但是实际上这个结果是不可信的。因为线程间存在同步操作，而计时又在同步操作之外，时间的近似相等是程序默认保证的结果。

3.2　实验结果分析

尽管上述的提高是较为令人满意的，但是不容忽视的问题：是使用 8 个线程并发的情况下，加速比仅仅为 2.2。下面提出产生该问题的几点可能原因。

3.2.1 线程同步的影响

在本解决方案中，有一个较大的性能瓶颈是对于每个新的素数，进行筛法之前，线程间必须进行同步，该同步发生的次数等于待求区间中的素数个数，由素数分布定理 $\pi(x) \approx \frac{x}{ln(x)}$，其中 $\pi(x)$ 为素数计数函数。因此在相当于存在一个接近$C \cdot \mathcal{O}(n)$ 的时间复杂度附加项，C 为一个小于 1 的常数。此外，更为糟糕的是，这样的同步等待导致线程的进度被最慢的线程所限制，因此相当依赖于操作系统本身的线程调度策略，当操作系统的线程调度越均衡时，程序的并行性越高。

3.3 素数标记外的时间开销

表 3-2 尽管不能够反应线程的负载是否均衡，但结合该表与表 3-1 的结果，却向我们指出了另一个信息：素数标记之外的时间开销是存在的且不应该被忽略。二者之差，即素数标记之外的时间开销占到了整个程序时间开销的约 12%，其中包含的部分有：bitset 类的初始化（非并行），素数的求和统计（并行），线程的派生（非并行），文件的读写（非并行）。可以预测的是，bitset 类的初始化在进一步增大时应当将之设置为并行的，否则其可能会称为程序效率的瓶颈所在。

4 调试记录

4.1 虚假唤醒问题

该问题发生在 Windows11 下使用 clang19 msvc 版本的调试中，由于在筛法最后进行的部分，对应于一个数值需要标记的部分不足线程数，此时选择的方法是只用线程 0 标记，然而在调试中出现了部分线程执行完成，但程序无法退出的问题。反复阅读相关文档后推测是线程到达屏障的间隔太小，发生了条件变量虚假唤醒的情况。在不进行标记操作的线程中插入微量的延迟，问题解决。

参考文献

[1] OIWIKI. bitset: 与埃氏筛结合 [EB/OL]. .
https://oi-wiki.org/lang/csl/bitset/#%E4%B8%8E%E5%9F%83%E6%B0%8F%E7%AD%9B%E7%BB%93%E5%90%88.