CS 415P/515 Parallel Programming

Assignment 2 Info

Jingke Li

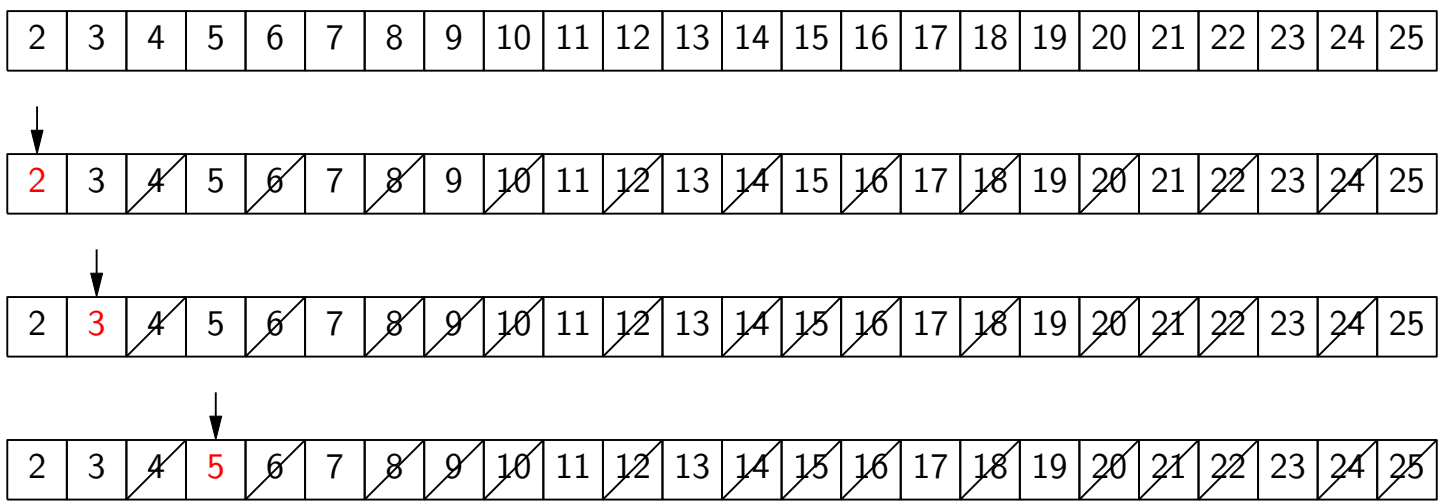
Portland State University

Spring 2022

Prime-Finding Algorithm

Find all primes within a given bound n.

0. Consider all numbers from 2 to n as candidates.
1. Start with the first prime, 2, mark off all its multiples.
2. The next unmarked number is a new prime. The marking process continues for all primes up to $\sqrt{n}$ (these are called sieve primes).
3. At the end, all remaining unmarked numbers are primes.



Sequential Version

prime.cpp

```
int main(int argc, char **argv) {
  int N = 100;
  cout << "Program prime over [2.." << N << "] starting ...\n";

  bool candidate[N+1];
  for (int i = 2; i <= N; i++)
    candidate[i] = true;

  for (int i = 2; i <= sqrt(N); i++)
    if (candidate[i])
      for (int j = i+i; j <= N; j += i)
        candidate[j] = false;

  int totalPrimes = 0;
  for (int i = 2; i <= N; i++)
    if (candidate[i])
      totalPrimes++;
  cout << "Found " << totalPrimes << " primes\n";
}
```

Question: Which loops are parallelizable?

Parallelization Strategies

Consider the main loop-nest in the sequential program:

```
for (int i = 2; i <= sqrt(N); i++)
  if (candidate[i])
    for (int j = i+i; j <= N; j += i)
      candidate[j] = false;
```

- Strategy 1. Parallelizing the inner loop
  - natural parallelism, balanced workload, easy for OpenMP, ... *but*
  - domain keeps changing, not as easy for explicit thread programming
- Strategy 2. Parallelizing the outer loop
  - also easy for OpenMP, seems to work, ... *but*
  - performs useless work, e.g. will assign threads to 4, 6, ...

Strategy 1 Improvement

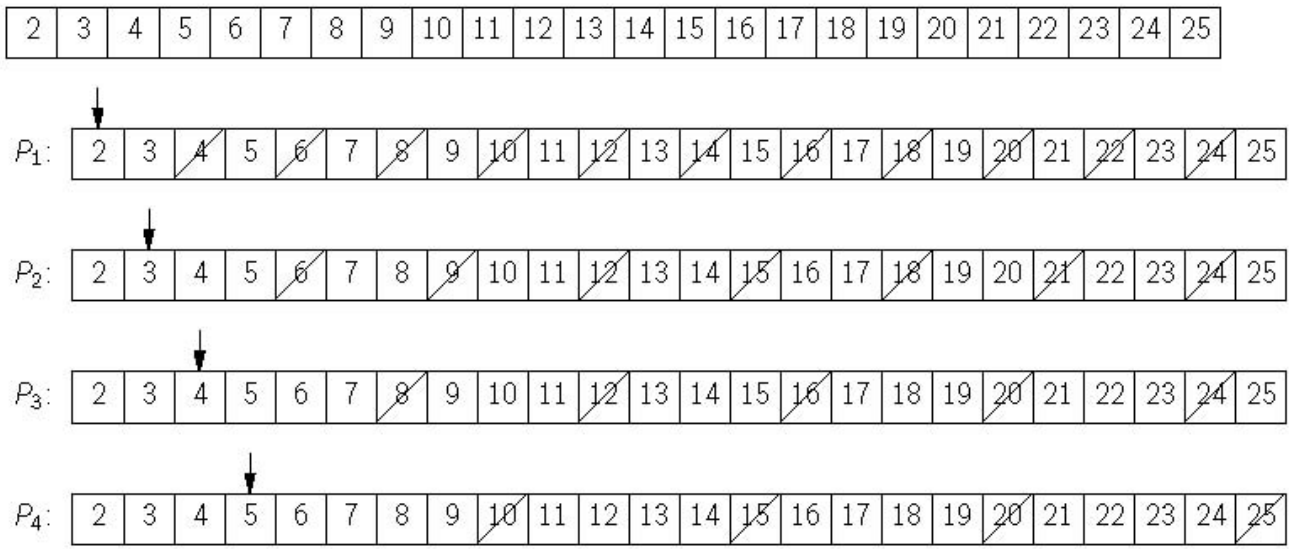
- Have master to perform the task of finding sieves in $[2..\sqrt{N}]$
- Partition a fixed domain $[\sqrt{N}..N]$ for the workers
- All threads can run concurrently

```
// Master: keep finding and broadcasting new sieve primes
  for (int i = 2; i <= sqrt(N); i++)
    if (candidate[i])
      for (int j = i+i; j <= sqrt(N); j += i)
        candidate[j] = false;

// Workers: each works on a fixed section of (sqrt(N)..N]
  for <each sieve prime p>
    for (int j = start; j <= end; j += p)
      candidate[j] = false;
```

A Closer Look at Strategy 2

Assign a thread to each iteration of the outer loop



- All threads work independently; no synchronization is needed
- Threads will work on composite numbers, wasting resource

Question: Is there a race condition?
(Multiple threads may mark the same cell.)

Strategy 2 Improvement

Assign threads to primes only

- Start a single thread first (for sieving prime 2)
- As soon as a new sieve prime is found, a new thread may start