Colorado State University

CS 475 Ch 5 (sieve)

Sanjay Rajopadhye
Colorado State University

Eratosthenes: the problem

- Find all the prime numbers up to a given number *n*
- By "filtering out" the multiples of known primes
- Many strategies
  - Start with a sequential algorithm and systematically parallelize it using our known and trusted approach (Foster's method)
  - Think out of the box (a completely different approach)

Colorado State University

The complexity

- How many primes are there?
  - A: See http://primes.utm.edu/howmany.shtml
- What is the (work) complexity of the sieve?
  - $\Theta(n \ln \ln n)$

Colorado State University

Sequential Algorithm

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
47	48	49	50	51	52	53	54	55	56	57	58	59	60	61

Colorado State University

Algorithm (contd)

Create an array of numbers 2 … n, none of which is "marked"

Invariant: the smallest unmarked number is a prime

k ← 2 /* k is the "next" prime number */

repeat

mark off all multiples of k as non-primes

set k to the next unmarked number (which must be a prime)

until "done"

Colorado State University

Pseudo code

```
for (i=1; i<=n; i++) marked[i] = 0;
k = index = 2;
marked[0] = marked[1] = 1;
while (k<=n) {
    for (i=2; i<=n; i++) if (i%k == 0) marked[i]=1;
    while (marked[++index]) ; /* do nothing */
    /* now index has the first unmarked number, so
… */
    k = index;
}
```

Colorado State University

Analysis & Improvement

- Where does the program spend its time?
- How to improve?
  - if $x=a*b$ is a composite number, then at least one of a or b is less than (or equal to) $\sqrt{x}$ (algorithmic improvement)

Colorado State University

Improved code

```
for (i=0; i<=n; i++) marked[i] = 0;
k = index = 2;
marked[0] = marked[1] = 1;
while (k*k<=n) { /* outer loop iterates only until sqrt(n)  */
    for (i=k*k; i<=n; i++) if (i%k == 0) marked[i]=1;
    while (marked[++index]) ; /* do nothing */
    /* now index has the first unmarked number, so … */
    k = index;
}
```

Colorado State University

Sequential performance first (avoid division)

```
for (i=1; i<=n; i++) marked[i] = 0;
k = index = 2;
marked[0] = marked[1] = 1;
while (k*k<=n) { /* outer loop iterates only until sqrt(n)  */
/* for (i=k*k; i<=n; i++) if (i%k == 0) marked[i]=1; */
    for (i=k*k; i<=n; i+= k) marked[i]=1;
    while (marked[++index]) ; /* do nothing */
    /* now index has the first unmarked number, so … */
    k = index;
}
```

Colorado State University

Lessons

- Exercise
  - Write the three programs and measure the running time for large values of n
- Priorities:
  - First improve algorithm (asymptotic running time)
  - Next constant factor gains
- Only then consider parallelization

Colorado State University

HW2 Clarification

- Experiment a bit
  - find a "large enough *n*"
  - running time on one processor ~30 sec
- Then, keep *n* fixed, change the number of threads
  - gather some running time data
  - plot the speedup
  - explain what you see
  - does it jive with your expectations (hypotheses)

Colorado State University

Revisit complexity analysis

- Basic conventions and background:
  - log (base 10), lg (base 2), ln (base *e*)
    - log x is number of digits to represent x
    - lg x is number of bits to represent x
  - #primes no larger than x: x/ln x
  - Sum of reciprocals of integers no larger than x: ln x
  - Sum of reciprocals of primes no larger than x: ln ln x

Colorado State University

Analysis

- Sequential complexity

$$T(n) = O(n \ln \ln n)$$

  - $\log_{10} n$: Number of digits in decimal representation of n
  - $\log_2 n$: Number of bits in the binary representation of n
  - $\log \log_{10} n$: Digits in decimal representation of that
  - $\log \log_2 n$: bits in the binary representation of that

  - Almost linear time complexity

Colorado State University

First parallelization

```
for (i=1; i<=n; i++) marked[i] = 0;
k = index = 2;
marked[0] = marked[1] = 1;
while (k*k<=n) {
   for (i=k*k; i<=n; i+= k) marked[i]=1;
   while (marked[++index]) ; /* do nothing */
   k = index;
}
```

Parallelize these loops

Colorado State University

Further improvement (constant factor)

- We marked off all even numbers in the first iteration
- In all subsequent iterations we mark off all multiples of k-th prime
  - INCLUDING ITS EVEN MULTIPLES
- Why not make the marked array of only n/2 elements:
  - ith element in the array repreent the ith odd integer (2i+1)
- Simple idea, subtle details

Are we done?

- Does the easy parallelization give good speedup? Why?
- Cache misses cost hundreds of cycles
- How to exploit locality?
- Change the order of execution to improve locality

Improving locality (blocking)

- Consider a large sub-array of marked array of size BLKSIZE.
- Instead of marking all the multiples of a k from k^2 to n, just mark off those multiples that are in the current block
- Then increment k
- Move to the next block
  - Only when the multiples of all the primes in the current block are marked off

Blocking

- Preamble: In an array primes[] store primes up to sqrt(n), say there are numprimes of them
- Elements of marked, up to index sqrt(n) already marked during preamble
- So start blocking at there (call it blockStart) but
  - instead of going all the way to n one prime at a time
  - with all primes one block of size BLKSIZE at the time

Blocked Sieve n=100, BLKSIZE=30

```
 1  3  5  7  9
11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
41 43 45 47 49 51 53 55 57 59 61 63 65 67 69
71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Colorado State University

1: Pre compute primes in block < sqrt(n)

```
 -  3  5  7  -
11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
41 43 45 47 49 51 53 55 57 59 61 63 65 67 69
71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Colorado State University

2: Sieve block 1 with 3 (start = 15)

```
–  3  5  7  –
11 13  – 17 19  – 23 25  – 29 31  – 35 37  –
41 43 45 47 49 51 53 55 57 59 61 63 65 67 69
71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Colorado State University

3: Sieve block 1 with 5 (start = 25)

```
–  3  5  7  –
11 13  – 17 19  – 23  –  – 29 31  –  – 37  –
41 43 45 47 49 51 53 55 57 59 61 63 65 67 69
71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Colorado State University

4: Sieve block 2 with 3 (start = 45)

```
–  3  5  7  –
11 13  – 17 19  – 23  –  – 29 31  –  – 37  –
41 43  – 47 49  – 53 55  – 59 61  – 65 67  –
71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Colorado State University

5: Sieve block 2 with 5 (start = 45)

```
–  3  5  7  –
11 13  – 17 19  – 23  –  – 29 31  –  – 37  –
41 43  – 47 49  – 53  –  – 59 61  –  – 67  –
71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Colorado State University

6: Sieve block 2 with 7 (start = 49)

```
 -  3  5  7  -
11 13  - 17 19  - 23  -  - 29 31  -  - 37  -
41 43  - 47  -  - 53  -  - 59 61  -  - 67  -
71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Colorado State University

7: Sieve block 3 with 3 (start = 75)

```
 -  3  5  7  -
11 13  - 17 19  - 23  -  - 29 31  -  - 37  -
41 43  - 47  -  - 53  -  - 59 61  -  - 67  -
71 73  - 77 79  - 83 85  - 89 91  - 95 97  -
```

Colorado State University

8: Sieve block 3 with 5 (start = 75)

```
-  3  5  7  -
11 13  - 17 19  - 23  -  - 29 31  -  - 37  -
41 43  - 47  -  - 53  -  - 59 61  -  - 67  -
71 73  - 77 79  - 83  -  - 89 91  -  - 97  -
```

Colorado State University

9: Sieve block 3 with 7 (start = 77)

```
-  3  5  7  -
11 13  - 17 19  - 23  -  - 29 31  -  - 37  -
41 43  - 47  -  - 53  -  - 59 61  -  - 67  -
71 73  -  - 79  - 83  -  - 89  -  -  - 97  -
```

Colorado State University

Code after preamble

```
for (j=0; j<=numprimes; j++)
  for (i=primes[j]*primes[j]; i<=n; i+= primes[j])
    marked[i]=1;
```

Rewrite inner loop:

```
for (j=0; j<=numprimes; j++)
  for (ii=start; ii<min(start+BKSIZE, n); ii+=BKSIZE){
    for (i=FMIB(ii,j); i<=min(start+BKSIZE, n); i+= primes[j])
      marked[i]=1;
```

What is FMIB[ii,j]?

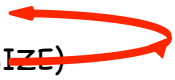
First Multiple of primes[j] In ii^{th} Block

What is start?

But nothing has changed

So interchange the loops

```
for (j=0; j<=numprimes; j++)
  for (ii=start; ii<min(start+BKSIZE, n); ii+=BKSIZE)
    for (i=FMIB(ii,j); i<=min(start+BKSIZE, n); i+= primes[j])
      marked[i]=1;
```



```
for (ii=start; ii<min(start+BKSIZE, n); ii+=BKSIZE)
  for (j=0; j<=numprimes; j++)
    for (i=FMIB(ii,j); i<=min(start+BKSIZE, n); i+= primes[j])
      marked[i]=1;
```

But is this legal?

- This is the key issue (HW2)
- We must ensure that the modified code does exactly what the original program did