CS475: PA1

Sanjay Rajopadhye
With edits by Wim Bohm
Colorado State University

jacobi 1 D

```
t = 0;
while ( t < MAX_ITERATION) {

    for ( i=1 ; i < N-1 ; i++ ) {
        cur[i] = (prev[i-1]+prev[i]+prev[i+1])/3;
    }

   temp = prev;
   prev = cur;
   cur  = temp;
   t++;

}
```

jacobi 2 D

```
t = 0;
while ( t < MAX_ITERATION) {

  for ( i=1 ; i < N-1 ; i++ ) {
      for ( j=1 ; j < N-1 ; j++ ) {
         cur(i,j) = ((prev(i-1,j-1)+prev(i-1,j)+prev(i-1,j+1)
                    +prev(i,j-1)+prev(i,j)+prev(i,j+1)
                    +prev(i+1,j-1)+prev(i+1,j)+prev(i+1,j+1))
                   )/9;
      }
   }
   temp = prev;
   prev = cur;
   cur  = temp;
   t++;
}
```

matrix vector product

```
for ( i=0 ; i < N ; i++ ) {
    c(i) = 0;
    for ( j=0; j < M ; j++ ) {
        c(i) += A(i,j) * b(j);
    }
}
```

CS475: The prime sieve of Erastosthenes in OpenMP

Sanjay Rajopadhye
With edits by Wim Bohm
Colorado State University

Primes problem

- Find all the prime numbers up to a given number *n*
- Sieve of Erastosthenes

  Have an array of prime candidates

  discover a prime, remove all multiples
- Strategy

  Start with a sequential algorithm and systematically parallelize it taking locality into account

Algorithm

Create an array of numbers 2 … n,
    none of which is “marked”
        Invariant: the smallest unmarked number is a prime
k $\leftarrow$ 2 /* k is the “next” prime number */
repeat
    Mark off all multiples of k as non-primes
    Set k to the next unmarked number
        Invariant: which must be a prime
until “done”

Pseudo code

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

Analysis & Improvement

- Where does the program spend its time?

  complexity?
- 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)
  - *So the upper bound of the* `for (i=3; i<=n; i++)` *loop can be tightened to???*

Better loop bounds

```
for (i=0; i<=n; i++) marked[i] = 0;
k = index = 2;
marked[0] = marked[1] = 1;
while (k*k<=n) { // stop at 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:
    //   the next prime
    k = index;
}
```

Why start at k*k?

Can we improve the step?

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

How can we save space?

we don't need the evens

make 2 a special case, save half the space

Efficient sequential code

- Step wise improve the sequential program Sieve 1
  - do the order of magnitude improvements, going to $\sqrt{n}$ only, starting from k*k
  - save space by only storing odds
    - **what about the step now?**
- compare to original
  - Sieve vs. Sieve 1
- measure the running time for large values of *n*
  - find an appropriate range

First easy parallelization

```
for (i=1; i<=n; i++) marked[i] = 0; //
parallelize
k = index = 2;
marked[0] = marked[1] = 1;
while (k*k<=n) {
    for (i=k*k; i<=n; i+= k) marked[i]=1; //
parallelize
    while (marked[++index]) ;
    k = index;
}
```

Analysis

- Does easy parallelization give us good speedup?

  no, **WHY?**

- Bad cache locality!! **WHY?**
- How do we get better cache behavior?

Don’t go all the way to n, but **block** the sieve loop

Blocking the sieve

Preamble: In an array primes[] store primes up to sqrt(n), say there are numprimes of them

Elements of the marked array up to index sqrt(n) have been marked
So we can start **blocking** at that index (call it blockStart):
instead of going all the way to n with one prime at the time we sieve with all primes one block of size BLKSIZE at the time

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

We have changed the order of computation, **is it legal?**

Yes, as long as sieving with prime[j] starts with the proper next multiple of prime[j]

What is the value of start? **The first odd multiple of k >= k*k**

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
```

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
```

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
```

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
```

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
```

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
```

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

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	-

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   -
```

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 -

Interleaved data decomposition

Book also discusses interleaved allocation:

- threads sieve the whole block with interleaved primes
- use thread_num (start) and num_threads (step) to pick the next prime
- BUT

  load imbalance

  locality problems