CS475: PA1

wim bohm

cs, csu
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

```c
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

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 \( \ldots \) n, none of which is “marked”

\[ k \leftarrow 2 \] /* k is the “next” prime number */

repeat

Mark off all multiples of k as non-primes
Set k to the next unmarked number

\[ \text{Invariant: which must be a prime} \]

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=3; 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 $x = a \times b$ 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;
}
Sequential Algorithm

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

```c
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<min(blockStart+BLKSIZE, 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 $n=100$, $\text{BLKSIZE} = 30$

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)
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)

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)

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>5</th>
<th>7</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>13</td>
<td>17</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>29</td>
<td>31</td>
<td></td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>41</td>
<td>43</td>
<td>47</td>
<td>49</td>
<td>53</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>59</td>
<td>61</td>
<td>65</td>
</tr>
<tr>
<td>67</td>
<td></td>
<td></td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>73</td>
<td>75</td>
<td>77</td>
<td>79</td>
<td>81</td>
</tr>
<tr>
<td>83</td>
<td>85</td>
<td>87</td>
<td>89</td>
<td>91</td>
</tr>
<tr>
<td>93</td>
<td>95</td>
<td>97</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>
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)

\[
\begin{array}{cccccccccccc}
& & & & & & & & & & & \\
\text{start} = 75 & & & & & & & & & & & \\
\end{array}
\]
9: Sieve block 3 with 7 (start = 77)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 3 5 7 -
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