The Sieve of Eratosthenes

Jorge Barbosa, FEUP

Outline

- Sequential algorithm
- Sources of parallelism
- Data decomposition options
- Parallel algorithm development, analysis
- Benchmarking
- Optimizations

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

Complexity: $\Theta(n \ln \ln n)$

Pseudocode

1. Create list of unmarked natural numbers 2, 3, …, n
2. $k \leftarrow 2$
3. Repeat
   (a) Mark all multiples of k between k^2 and n
   (b) $k \leftarrow$ smallest unmarked number $> k$
   until $k^2 > n$
4. The unmarked numbers are primes

Sources of Parallelism

- Domain decomposition
  - Divide data into pieces
  - Associate computational steps with data
- One primitive task per array element

Making 3(a) Parallel

Mark all multiples of k between k^2 and n

$\Rightarrow$

for all j where $k^2 \leq j \leq n$ do
  if j mod $k = 0$ then
    mark j (it is not a prime)
  endif
endfor

Making 3(b) Parallel

Find smallest unmarked number > k

$\Rightarrow$

Min-reduction (to find smallest unmarked number > k)

Shared variable (to get results from all processes)

Agglomeration Goals

- Consolidate tasks
- Reduce sharing costs
- Balance computations among processes

Data Decomposition Options

- Interleaved (cyclic)
  - Easy to determine "owner" of each index
  - Leads to load imbalance *for this problem*

  (with P=2, one processor would be idle after first step)

- Block
  - Balances loads
  - More complicated to determine owner if *n* not a multiple of *p*

Block Decomposition Options

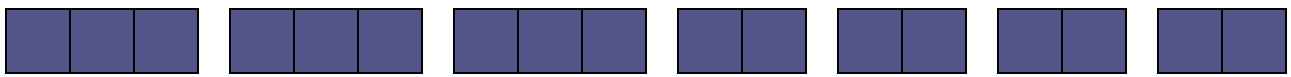
- Want to balance workload when n not a multiple of p
- Each process gets either $\lceil n/p \rceil$ or $\lfloor n/p \rfloor$ elements
- Seek simple expressions
  - Find low, high indices given an owner
  - Find owner given an index

Method #1

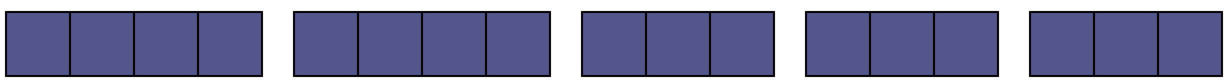
- Let $r = n \bmod p$
- If $r = 0$, all blocks have same size
- Else
  - First r blocks have size $\lceil n/p \rceil$
  - Remaining p-r blocks have size $\lfloor n/p \rfloor$

Examples

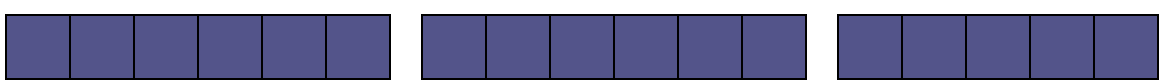
17 elements divided among 7 processes



17 elements divided among 5 processes



17 elements divided among 3 processes



Method #1 Calculations

- First element controlled by process *i*

$$i\lfloor n/p \rfloor + \min(i, r)$$

- Last element controlled by process *i*

$$(i+1)\lfloor n/p \rfloor + \min(i+1, r) - 1$$

- Process controlling element *j*

$$\max(\lfloor j/(\lfloor n/p \rfloor + 1) \rfloor, \lfloor (j-r)/\lfloor n/p \rfloor \rfloor)$$

Method #2

- Scatters larger blocks among processes
- First element controlled by process *i*

$$\lfloor in / p \rfloor$$

- Last element controlled by process *i*

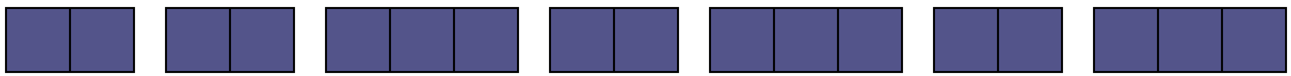
$$\lfloor (i+1)n / p \rfloor - 1$$

- Process controlling element *j*

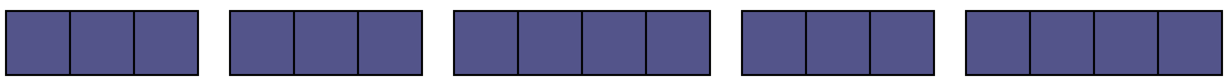
$$\lfloor (p(j+1)-1) / n \rfloor$$

Examples

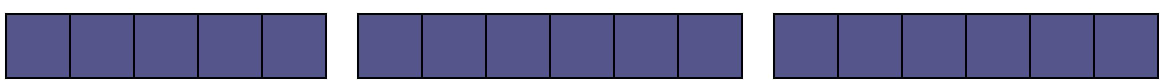
17 elements divided among 7 processes



17 elements divided among 5 processes



17 elements divided among 3 processes



Comparing Methods

Our choice

Operations	Method 1	Method 2
Low index	4	2
High index	6	4
Owner	7	4

Assuming no operations for “floor” function

Block Decomposition Macros

```
#define BLOCK_LOW(i,p,n)  ((i)*(n)/(p))

#define BLOCK_HIGH(i,p,n) \
        (BLOCK_LOW((i)+1,p,n)-1)

#define BLOCK_SIZE(i,p,n) \
        (BLOCK_LOW((i)+1)-BLOCK_LOW(i))

#define BLOCK_OWNER(index,p,n) \
        (((p)*(index)+1)-1)/(n))
```

Looping over Elements

- Sequential program

```
for (i = 0; i < n; i++) {
      …
}
```

- Parallel program

```
size = BLOCK_SIZE (id,p,n);
for (i = 0; i < size; i++) {
   gi = i + BLOCK_LOW(id,p,n);
}
```

Decomposition Affects Implementation

- Largest prime used to sieve is $\sqrt{n}$
- First process has $\lfloor n/p \rfloor$ elements
- It has all sieving primes if $p < \sqrt{n}$
- First process always broadcasts next sieving prime
- No reduction step needed

Fast Marking

Find j the *first* multiple of k on the block:
j, $j + k$, $j + 2k$, $j + 3k$, …

instead of

for all j in block
    if j mod $k = 0$ then mark j (it is not a prime)

Parallel Algorithm Development

1. Create list of unmarked natural numbers 2, 3, …, n

Each process creates its share of list

2. $k \leftarrow 2$

Each process does this

3. Repeat

Each process marks its share of the list

(a) Mark all multiples of k between k^2 and n

(b) $k \leftarrow$ smallest unmarked number $> k$

Process 0 only

(c) Process 0 shares k with the rest of processes

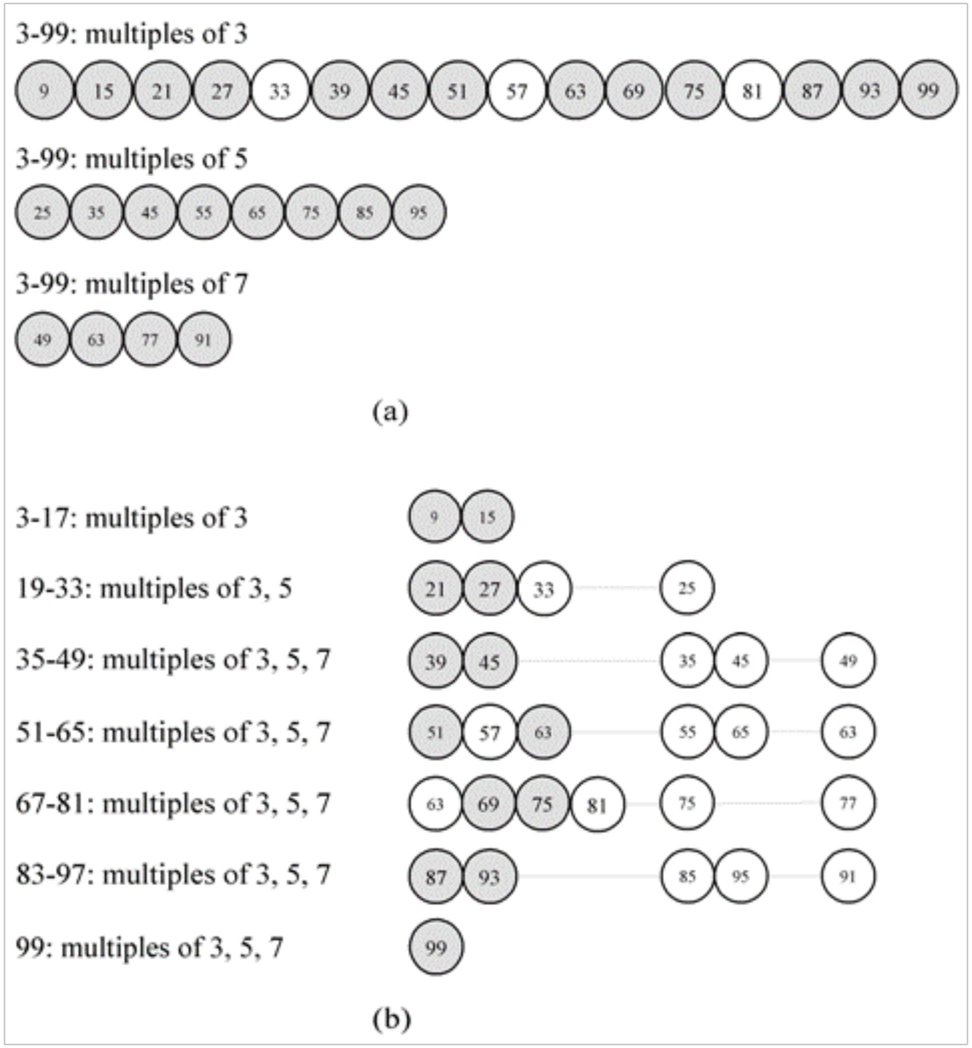
until $k^2 > n$

4. The unmarked numbers are primes

Improvements

- Delete even integers
  - Cuts number of computations in half
  - Frees storage for larger values of *n*
- Each process finds own sieving primes
  - Replicating computation of primes to $\sqrt{n}$
  - Eliminates broadcast step
- Reorganize loops
  - Increases cache hit rate

Reorganize Loops



Lower

Cache hit rate

Higher

Lab work

- Develop a shared memory parallelization of the Sieve of Eratosthenes
- Suggestion:
  - Parallel design by domain decomposition
  - Select block distribution
- Consider optimizations to maximize single-processor (core) performance