Mpi and the Sieve of Eratosthenes

Outline:

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

Sequential algorithm for finding primes

- 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

Representation of 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: $\Theta(n \ln \ln n)$

Identify what can be parallelized

- Domain decomposition what is the domain? Represent the data as an array of integers.
- Divide data into pieces
- Associate computational steps with data
- One primitive task per array element
- The tasks in 3(a) and 3(b) need to be analyzed

First, consider the tasks in 3(a)

Mark all multiples of k between k^2 and n Sequential algorithm:

```
for all j where k^2 \le j \le n do

if j mod k = 0 then

mark j (it is not a prime)

endif

endfor
```

Note that we can parallelize the j loop

And then, consider the tasks in 3(b)

Find smallest unmarked number > k

This step ignores the marked array elements, so the number of tasks has been reduced. Can be accomplished by:

- Perform a reduction to find the smallest unmarked number > k
- Broadcast the result to all processes

Agglomeration of the tasks

- Consolidate tasks each iteration of the sieve algorithm reduces the number of elements to consider.
- Reduce communication cost current value of k needs to be shared with all processes.
- Balance computations among processes as the calculation proceeds, less tasks remain with smaller indices.

How to divide up the data

- Interleaved (cyclic) Tasks are assigned "round robin"
- Easy to determine "owner" of each index
- Leads to load imbalance for this problem
- Block decomposition each process is given a contiguous block of tasks
- Balances loads
- More complicated to determine owner if n not a multiple of p

Load balance problem in interleaved division of data

Consider p = 4, so

p₀ has tasks with values 2, 6, 10, 14, 18, ...

p₁ has tasks with values 3, 7, 11, 15, 19, ...

p₂ has values 4, 8, 12, 16, 20, ...

p₃ has values 5, 9, 13, 17, 21, ...

Processes p_0 and p_2 have no more tasks after the case k = 2.

How does block decomposition work?

- Want to balance workload when n, the number of tasks, is not a multiple of p, the number of processes
- Each process gets either ceil(n/p) or floor(n/p) elements
- Seek simple expressions to identify task and process
- Find low, high indices given a process number
- Find the process given an array index

First approach to block decomposition

- Let $r = n \mod p$
- If r = 0, all blocks have same size and it is straighforward to find which array elements belong to which process
- Else
- First r blocks have size ceil(n/p)
- Remaining p-r blocks have size floor(n/p)

When r != 0

First element controlled by process i:

```
j = i*floor(n/p) + min(i,r)
```

Last element controlled by process i:

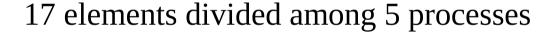
```
j = (i+1)*floor(n/p) + min(i+1,r) - 1
```

Process, q, controlling element j:

```
q = min(floor(j/(floor(n/p)+1),floor(j-r)/floor(n/p)))
```

Some examples using the first approach





17 elements divided among 3 processes

Second approach – scatter larger blocks among smaller blocks

17 elements divided among 7 processes

17 elements divided among 5 processes

17 elements divided among 3 processes

Assigning indices to processes in second approach

First element controlled by process i:

```
j = floor(i*n/p)
```

Last element controlled by process i:

$$j = floor((i+1)*n/p)-1$$

Process controlling element j:

$$q = ceil((p*(j+1)-1)/n)$$

Macros to program the second approach

The method of decomposition affects the implementation

- The largest prime used in the algorithm to remove multiples is \sqrt{n}
- The first process has floor(n/p) elements
- The algorithm finds all possible primes if $p < \sqrt{n}$
- The first process always broadcasts the next sieving prime
- No reduction step is needed

Fast marking of rejected elements

Block decomposition allows same marking as sequential algorithm:

```
mark elements 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
- $2. k \leftarrow 2$

Each process creates its share of list

Each process does this

3. Repeat

Each process marks its share of list

- (a) Mark all multiples of k between k^2 and n
- (b) k ← smallest unmarked number > k

Process 0 only

- (c) Process 0 broadcasts k to rest of processes until $k^2 > n$
- 4. The unmarked numbers are primes
- 5. Reduction to determine number of primes