The Sieve of Erastothenes

Parallel and Distributed Computing

Department of Computer Science and Engineering (DEI)
Instituto Superior Técnico

October 26, 2011

Outline

- The Sieve of Eratosthenes
- Data decomposition options
- Parallel algorithm development, analysis
- Benchmarking
- Optimizations

The Sieve of Eratosthenes

The Sieve of Eratosthenes

Algorithm for finding all primes up to n, proposed by Greek mathematician Erastothenes (276-194 BC).

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

The Sieve of Eratosthenes

The Sieve of Eratosthenes

Algorithm for finding all primes up to n, proposed by Greek mathematician Erastothenes (276-194 BC).

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

Complexity?

The Sieve of Eratosthenes

The Sieve of Eratosthenes

Algorithm for finding all primes up to n, proposed by Greek mathematician Erastothenes (276-194 BC).

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

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

Example with $n = 60$

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

	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

Example with $n = 60$

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

	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

Example with $n = 60$

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

	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

Example with $n = 60$

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

	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

Example with $n = 60$

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

	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

Example with $n = 60$

1. Create list of unmarked natural numbers $2, 3, \ldots, n$
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

	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

Foster's Methodology on the Sieve of Erastothenes

Partitioning:

Foster's Methodology on the Sieve of Erastothenes

Partitioning:
Make the computation of each element of the list $2, 3, \ldots, n$ a primitive task.

Communication:

Foster's Methodology on the Sieve of Erastothenes

Partitioning:
Make the computation of each element of the list $2, 3, \ldots, n$ a primitive task.

Communication:
Need to send value of k to all tasks.

Agglomeration + Mapping:
What's the best way to partition the list?

Foster's Methodology on the Sieve of Erastothenes

Partitioning:
Make the computation of each element of the list $2, 3, \ldots, n$ a primitive task.

Communication:
Need to send value of k to all tasks.

Agglomeration + Mapping:
What's the best way to partition the list?

- interleaved (cyclic) data decomposition
- block data decomposition

Foster's Methodology on the Sieve of Erastothenes

Partitioning:
Make the computation of each element of the list $2, 3, \ldots, n$ a primitive task.

Communication:
Need to send value of k to all tasks.

Agglomeration + Mapping:
What's the best way to partition the list?

- interleaved (cyclic) data decomposition
  - easy to determine "owner" of each index
  - leads to load imbalance for this problem
- block data decomposition
  - balances loads
  - more complicated to determine owner if n not a multiple of p

Block Decomposition

"The devil is in the details"

What's the best way to distribute n elements over p tasks?

Block Decomposition

"The devil is in the details"

What's the best way to distribute n elements over p tasks?

Some tasks get $\left\lceil \frac{n}{p} \right\rceil$ elements, other get $\left\lfloor \frac{n}{p} \right\rfloor$.

Which task gets which size?

Block Decomposition

"The devil is in the details"

What's the best way to distribute n elements over p tasks?

Some tasks get $\left\lceil \frac{n}{p} \right\rceil$ elements, other get $\left\lfloor \frac{n}{p} \right\rfloor$.

Which task gets which size?

One approach: group larger blocks at lower index tasks.

Let $r = n \mod p$.

> if $r = 0$, assign $\frac{n}{p}$ elements per task.
>
> Otherwise,
>
> > first r tasks get $\left\lceil \frac{n}{p} \right\rceil$
> >
> > remaining tasks get $\left\lfloor \frac{n}{p} \right\rfloor$

Block Decomposition

Range of elements that each task gets?

First element of task i:

Block Decomposition

Range of elements that each task gets?

First element of task i: $\quad i \left\lfloor \frac{n}{p} \right\rfloor + \min(i, r)$

Last element of task i:

Block Decomposition

Range of elements that each task gets?

First element of task i: $i\left\lfloor\frac{n}{p}\right\rfloor + \min(i, r)$

Last element of task i: $(i+1)\left\lfloor\frac{n}{p}\right\rfloor + \min(i+1, r) - 1$

Task owner of element j:

Block Decomposition

Range of elements that each task gets?

First element of task i: $i\left\lfloor\frac{n}{p}\right\rfloor+\min(i,r)$

Last element of task i: $(i+1)\left\lfloor\frac{n}{p}\right\rfloor+\min(i+1,r)-1$

Task owner of element j: $\max\left(\left\lfloor j/\lceil\frac{n}{p}\rceil\right\rfloor,\left\lfloor(j-r)/\lfloor\frac{n}{p}\rfloor\right\rfloor\right)$

Block Decomposition

Grouped approach: group larger blocks at lower index tasks.

First element of task i: $i\left\lfloor \frac{n}{p} \right\rfloor + \min(i, r)$

Last element of task i: $(i+1)\left\lfloor \frac{n}{p} \right\rfloor + \min(i+1, r) - 1$

Task owner of element j: $\max\left(\left\lfloor j/\lceil \frac{n}{p} \rceil \right\rfloor, \left\lfloor (j-r)/\lfloor \frac{n}{p} \rfloor \right\rfloor\right)$

Distributed approach: distribute larger blocks evenly.

Block Decomposition

Grouped approach: group larger blocks at lower index tasks.

First element of task i: $i\left\lfloor\frac{n}{p}\right\rfloor + \min(i, r)$

Last element of task i: $(i+1)\left\lfloor\frac{n}{p}\right\rfloor + \min(i+1, r) - 1$

Task owner of element j: $\max\left(\left\lfloor j/\lceil\frac{n}{p}\rceil\right\rfloor, \left\lfloor(j-r)/\lfloor\frac{n}{p}\rfloor\right\rfloor\right)$

Distributed approach: distribute larger blocks evenly.

First element of task i: $\left\lfloor i\frac{n}{p}\right\rfloor$

Last element of task i:

Block Decomposition

Grouped approach: group larger blocks at lower index tasks.

First element of task i: $i\left\lfloor \frac{n}{p} \right\rfloor + \min(i, r)$

Last element of task i: $(i+1)\left\lfloor \frac{n}{p} \right\rfloor + \min(i+1, r) - 1$

Task owner of element j: $\max\left(\left\lfloor j/\lceil \frac{n}{p} \rceil \right\rfloor, \left\lfloor (j-r)/\lfloor \frac{n}{p} \rfloor \right\rfloor\right)$

Distributed approach: distribute larger blocks evenly.

First element of task i: $\left\lfloor i\frac{n}{p} \right\rfloor$

Last element of task i: $\left\lfloor (i+1)\frac{n}{p} \right\rfloor - 1$

Task owner of element j:

Block Decomposition

Grouped approach: group larger blocks at lower index tasks.

First element of task i: $i\left\lfloor\frac{n}{p}\right\rfloor + \min(i, r)$

Last element of task i: $(i+1)\left\lfloor\frac{n}{p}\right\rfloor + \min(i+1, r) - 1$

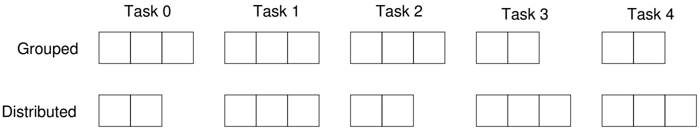
Task owner of element j: $\max\left(\left\lfloor j/\lceil\frac{n}{p}\rceil\right\rfloor, \left\lfloor (j-r)/\lfloor\frac{n}{p}\rfloor\right\rfloor\right)$

Distributed approach: distribute larger blocks evenly.

First element of task i: $\left\lfloor i\frac{n}{p}\right\rfloor$

Last element of task i: $\left\lfloor (i+1)\frac{n}{p}\right\rfloor - 1$

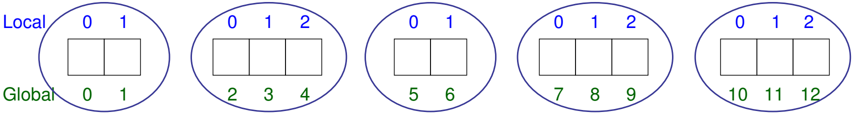
Task owner of element j: $\lfloor (p(j+1)-1)/n \rfloor$



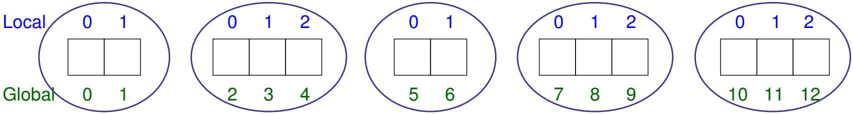
Block Decomposition Macros

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

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

#define BLOCK_SIZE(id,p,n) (BLOCK_HIGH(id,p,n)-BLOCK_LOW(id,p,n)+1)

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

Local vs Global Indexes



Local vs Global Indexes



- 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);
}
```

Parallel Implementation

- use distributed block decomposition

Parallel Implementation

- use distributed block decomposition

- first task responsible for selecting next sieving prime
  - broadcast new sieving prime at end of each iteration
  - first task has first $\lfloor n/p \rfloor$ elements, make sure it includes the largest prime used to sieve, $\sqrt{n}$

Parallel Implementation

- use distributed block decomposition
- first task responsible for selecting next sieving prime
  - broadcast new sieving prime at end of each iteration
  - first task has first $\lfloor n/p \rfloor$ elements, make sure it includes the largest prime used to sieve, $\sqrt{n}$
- to simplify, our program will return the number of primes up to n

Parallel Implementation

1. Create list of unmarked natural numbers $2, 3, \ldots, n$

Parallel Implementation

1. Create list of unmarked natural numbers $2, 3, \ldots, n$

   each process creates its share of the list

2. $k \leftarrow 2$

Parallel Implementation

1. Create list of unmarked natural numbers $2, 3, \ldots, n$

   each process creates its share of the list

2. $k \leftarrow 2$

   all processes perform this

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

Parallel Implementation

1. Create list of unmarked natural numbers $2, 3, \ldots, n$

   each process creates its share of the list

2. $k \leftarrow 2$

   all processes perform this

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

       each process marks its share of the list

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

Parallel Implementation

1. Create list of unmarked natural numbers $2, 3, \ldots, n$

    each process creates its share of the list

2. $k \leftarrow 2$

    all processes perform this

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

        each process marks its share of the list

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

        process 0 only and broadcasts it

    until $k^2 > n$

4. The unmarked numbers are primes

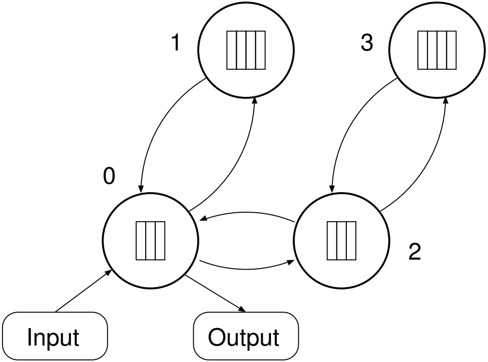
5. Reduction to compute number of primes

Function MPI_Bcast

```
int MPI_Bcast (
   void *buffer,             /* Addr of 1st element */
   int count,                /* # elements to broadcast */
   MPI_Datatype datatype,    /* Type of elements */
   int root,                 /* ID of root process */
   MPI_Comm comm)            /* Communicator */
```

```
MPI_Bcast (&k, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

Task / Channel Model



Analysis of the Parallel Algorithm

Let α be the time to mark a cell.

Sequential execution time?

Analysis of the Parallel Algorithm

Let α be the time to mark a cell.

Sequential execution time: $\alpha n \log \log n$

Computation time of parallel program?

Analysis of the Parallel Algorithm

Let α be the time to mark a cell.

Sequential execution time: $\alpha n \log \log n$

Computation time of parallel program: $\alpha n \log \log n / p$

Number of broadcasts?

Analysis of the Parallel Algorithm

Let α be the time to mark a cell.

Sequential execution time: $\alpha n \log \log n$

Computation time of parallel program: $\alpha n \log \log n / p$

Number of broadcasts: $\sqrt{n} / \log \sqrt{n}$

Broadcast time?

Analysis of the Parallel Algorithm

Let α be the time to mark a cell.

Sequential execution time: $\alpha n \log \log n$

Computation time of parallel program: $\alpha n \log \log n/p$

Number of broadcasts: $\sqrt{n}/\log \sqrt{n}$

Broadcast time: $\lambda \lceil \log p \rceil$

Reduction time?

Analysis of the Parallel Algorithm

Let α be the time to mark a cell.

Sequential execution time: $\alpha n \log\log n$

Computation time of parallel program: $\alpha n \log\log n/p$

Number of broadcasts: $\sqrt{n}/\log\sqrt{n}$

Broadcast time: $\lambda\lceil\log p\rceil$

Reduction time: $\lambda\lceil\log p\rceil$

Expected parallel execution time?

Analysis of the Parallel Algorithm

Let α be the time to mark a cell.

Sequential execution time: $\alpha n \log\log n$

Computation time of parallel program: $\alpha n \log\log n / p$

Number of broadcasts: $\sqrt{n}/\log\sqrt{n}$

Broadcast time: $\lambda\lceil\log p\rceil$

Reduction time: $\lambda\lceil\log p\rceil$

Expected parallel execution time:

$$\alpha\frac{n\log\log n}{p}+\lambda\frac{\sqrt{n}\lceil\log p\rceil}{\log\sqrt{n}}+\lambda\lceil\log p\rceil$$

Code (1/4)

```
#include <mpi.h>
#include <math.h>
#include <stdio.h>
#define BLOCK_LOW(id,p,n)  ((i)*(n)/(p))
#define BLOCK_HIGH(id,p,n) (BLOCK_LOW((id)+1,p,n)-1)
#define BLOCK_SIZE(id,p,n) (BLOCK_LOW((id)+1)-BLOCK_LOW(id))

int main (int argc, char *argv[])
{
   ...
   MPI_Init (&argc, &argv);
   MPI_Barrier(MPI_COMM_WORLD);
   elapsed_time = -MPI_Wtime();
   MPI_Comm_rank (MPI_COMM_WORLD, &id);
   MPI_Comm_size (MPI_COMM_WORLD, &p);
   if (argc != 2) {
      if (!id) printf ("Command line: %s <m>\n", argv[0]);
      MPI_Finalize(); exit (1);
   }
```

Code (2/4)

```
n = atoi(argv[1]);
low_value = 2 + BLOCK_LOW(id,p,n-1);
high_value = 2 + BLOCK_HIGH(id,p,n-1);
size = BLOCK_SIZE(id,p,n-1);
proc0_size = (n-1)/p;
if ((2 + proc0_size) < (int) sqrt((double) n)) {
   if (!id) printf ("Too many processes\n");
   MPI_Finalize(); exit (1);
}

marked = (char *) malloc (size);
if (marked == NULL) {
   printf ("Cannot allocate enough memory\n");
   MPI_Finalize(); exit (1);
}

for (i = 0; i < size; i++) marked[i] = 0;
```

Code (3/4)

```
if (!id) index = 0;
prime = 2;
do {
   if (prime * prime > low_value)
      first = prime * prime - low_value;
   else {
      if (!(low_value % prime)) first = 0;
      else first = prime - (low_value % prime);
   }
   for (i = first; i < size; i += prime) marked[i] = 1;
   if (!id) {
      while (marked[++index]);
      prime = index + 2;
   }
   MPI_Bcast (&prime,  1, MPI_INT, 0, MPI_COMM_WORLD);
} while (prime * prime <= n);
```

Code (4/4)

```
   count = 0;
   for (i = 0; i < size; i++)
      if (!marked[i]) count++;

   MPI_Reduce (&count, &global_count, 1, MPI_INT, MPI_SUM,
               0, MPI_COMM_WORLD);

   elapsed_time += MPI_Wtime();
   if (!id) {
      printf ("%d primes are less than or equal to %d\n",
               global_count, n);
      printf ("Total elapsed time: %10.6f\n", elapsed_time);
   }

   MPI_Finalize ();

   return 0;
}
```

Benchmarking

Expected parallel execution time:

$$\alpha \frac{n \log \log n}{p} + \lambda \frac{\sqrt{n} \lceil \log p \rceil}{\log \sqrt{n}} + \lambda \lceil \log p \rceil$$

Benchmarking

Expected parallel execution time:

$$\alpha \frac{n \log \log n}{p} + \lambda \frac{\sqrt{n} \lceil \log p \rceil}{\log \sqrt{n}} + \lambda \lceil \log p \rceil$$

Experimental estimation of α: with $n = 10^8$, runtime$= 24,9s$

$$\alpha = \frac{24,9}{10^8 \log 10^8} = 85,47ns$$

Benchmarking

Expected parallel execution time:

$$\alpha \frac{n \log \log n}{p} + \lambda \frac{\sqrt{n} \lceil \log p \rceil}{\log \sqrt{n}} + \lambda \lceil \log p \rceil$$

Experimental estimation of α: with $n = 10^8$, runtime$= 24,9s$

$$\alpha = \frac{24,9}{10^8 \log 10^8} = 85,47ns$$

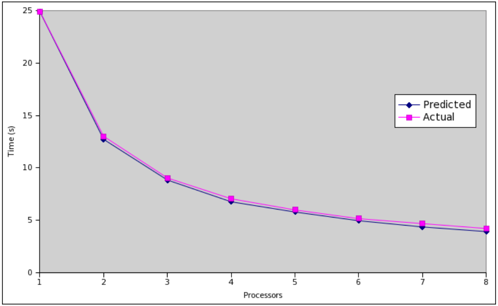
Experimental estimation of λ: sequence of broadcasts using $p = 2, 3, \ldots, 8$ processors.

$$\lambda = 250\mu s$$

Using $n = 10^8$, execution times of the algorithm were measured and compared against the above formula, for $p = 2, 3, \ldots, 8$ processors.

Experimental Results

p	Predicted	Actual
1	24,9	24,9
2	12,7	13,0
3	8,8	9,0
4	6,8	7,1
5	5,8	6,0
6	5,0	5,2
7	4,4	4,7
8	3,9	4,2



Improvements to the Program - I

Every other mark is on an even number:

Improvements to the Program - I

Every other mark is on an even number:
$\Rightarrow$ delete them!

New expected parallel execution time:

Improvements to the Program - I

Every other mark is on an even number:
$\Rightarrow$ delete them!

New expected parallel execution time:

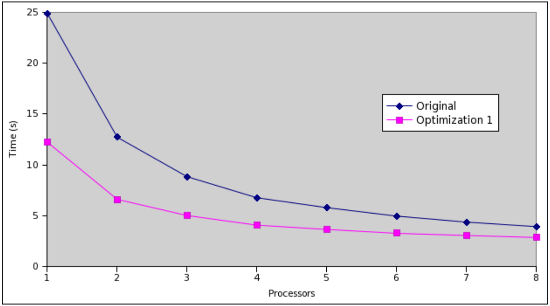
$$\alpha \frac{n \log \log n}{2p} + \lambda \frac{\sqrt{n}\,\lceil \log p \rceil}{\log \sqrt{n}} + \lambda \lceil \log p \rceil$$

Every other mark is on an even number:
$\Rightarrow$ delete them!

New expected parallel execution time:

$$\alpha \frac{n \log \log n}{2p} + \lambda \frac{\sqrt{n} \lceil \log p \rceil}{\log \sqrt{n}} + \lambda \lceil \log p \rceil$$

p	Original	Optimized
1	24,9	12,2
2	13,0	6,6
3	9,0	5,0
4	7,1	4,1
5	6,9	3,7
6	5,2	3,3
7	4,7	3,1
8	4,2	2,9



Improvements to the Program - II

Minimize communication:

Improvements to the Program - II

Minimize communication:

duplicate sieve computation on all tasks to avoid broadcast

$\Rightarrow$ replace $\lambda\lceil\log p\rceil\sqrt{n}/\log\sqrt{n}$ by $\alpha\sqrt{n}\log\log\sqrt{n}$

New expected parallel execution time:

$$\alpha\left(\frac{n\log\log n}{2p}+\sqrt{n}\log\log\sqrt{n}\right)+\lambda\lceil\log p\rceil$$

Improvements to the Program - II

Minimize communication:

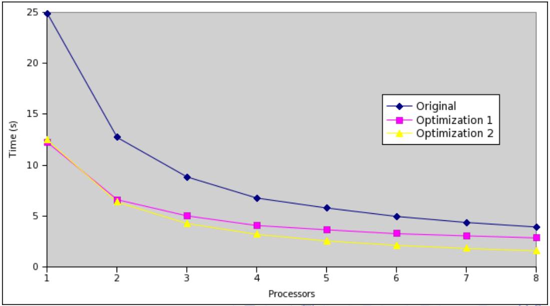
duplicate sieve computation on all tasks to avoid broadcast

$\Rightarrow$ replace $\lambda\lceil\log p\rceil\sqrt{n}/\log\sqrt{n}$ by $\alpha\sqrt{n}\log\log\sqrt{n}$

New expected parallel execution time:

$$\alpha\left(\frac{n\log\log n}{2p}+\sqrt{n}\log\log\sqrt{n}\right)+\lambda\lceil\log p\rceil$$

p	Original	Optimized
1	24,9	12,5
2	13,0	6,4
3	9,0	4,3
4	7,1	3,2
5	6,0	2,6
6	5,2	2,1
7	4,7	1,8
8	4,2	1,6



Improvements to the Program - III

Memory access pattern?

Improvements to the Program - III

Memory access pattern?

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

Improvements to the Program - III

Memory access pattern?

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

19	21	23	25
35	37	39	41
43	45	47	49
51	53	55	57
59	61	63	65
75	77	79	81

Improvements to the Program - III

Memory access pattern?

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

19	21	23	25
35	37	39	41
43	45	47	49
51	53	55	57
59	61	63	65
75	77	79	81

43	45	47	49
59	61	63	65
75	77	79	81

Improvements to the Program - III

Solution?

Improvements to the Program - III

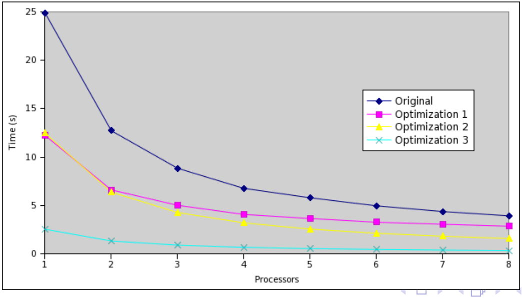
Solution: reorganize loops

$\Rightarrow$ mark each element of the array for all sieves between 3 and $\sqrt{n}$

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

Improvements to the Program - III

p	Original	Opt.I	Opt.II	Opt.III
1	24,9	12,2	12,5	2,5
2	13,0	6,6	6,4	1,3
3	9,0	5,0	4,3	0,9
4	7,1	4,1	3,2	0,7
5	6,0	3,7	2,6	0,5
6	5,2	3,3	2,1	0,5
7	4,7	3,1	1,8	0,4
8	4,2	2,9	1,6	0,3



Review

- The Sieve of Eratosthenes
- Data decomposition options
  - interleaved
  - block
    - grouped
    - distributed
- Parallel algorithm development, analysis
- Benchmarking
- Optimizations

Next Class

- A shortest path algorithm