Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort
2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

MCS 572 Lecture 26
Introduction to Supercomputing
Jan Verschelde, 25 October 2024

Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort
2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

a data archival application

Consider a pipeline of four processors:

$$\longrightarrow \boxed{P_0} \longrightarrow \boxed{P_1} \longrightarrow \boxed{P_2} \longrightarrow \boxed{P_3} \longrightarrow$$

The most recent data is stored on P_0.

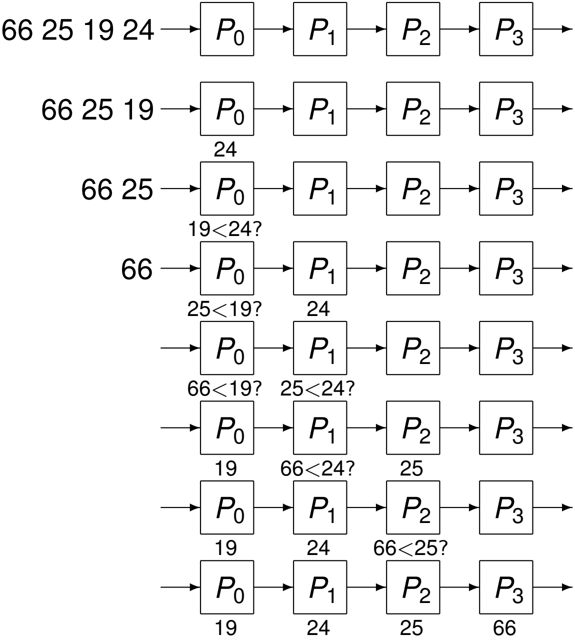
1. When P_0 receives new data, its older data is moved to P_1.
2. When P_1 receives new data, its older data is moved to P_2.
3. When P_2 receives new data, its older data is moved to P_3.
4. When P_3 receives new data, its older data is archived to tape.

This is a type 1 pipeline. Every processor does the same three steps:

(1) receive new data, (2) sort data, (3) send old data.

This leads to a pipelined sorting of numbers.

sorting


66 25 19 24
P_0
P_1
P_2
P_3
66 25 19
24
66 25
19<24?
66
25<19?
24
66<19?
25<24?
19
66<24?
25
19
24
66<25?
19
24
25
66

a parallel version of insertion sort

Sorting p numbers with p processors.

Processor i does $p - i$ steps in the algorithm:

for step 0 to $p - i - 1$ do
    manager receives number;
    worker i receives number from $i - 1$;
    if step = 0 then
        initialize the smaller number;
    else if number $>$ smaller number then
        send number to $i + 1$;
    else
        send smaller number to $i + 1$;
        smaller number $:=$ number;
    end if;
end for.

a pipeline session with MPI

```
$ mpirun -np 4 /tmp/pipe_sort
The 4 numbers to sort :  24 19 25 66
Manager gets 24.
Manager gets 19.
Node 0 sends 24 to 1.
Manager gets 25.
Node 0 sends 25 to 1.
Manager gets 66.
Node 0 sends 66 to 1.
Node 1 receives 24.
Node 1 receives 25.
Node 1 sends 25 to 2.
Node 1 receives 66.
Node 1 sends 66 to 2.
Node 2 receives 25.
Node 2 receives 66.
Node 2 sends 66 to 3.
Node 3 receives 66.
The sorted sequence :  19 24 25 66
```

Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort
2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

the program pipe_sort.c

```
int main ( int argc, char *argv[] )
{
   int i,p,*n,j,g,s;
   MPI_Status status;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&p);
   MPI_Comm_rank(MPI_COMM_WORLD,&i);
   if(i==0) /* manager generates p random numbers */
   {
      n = (int*)calloc(p,sizeof(int));
      srand(time(NULL));
      for(j=0; j<p; j++) n[j] = rand() % 100;
      if(verbose>0)
      {
         printf("The %d numbers to sort : ",p);
         for(j=0; j<p; j++) printf(" %d", n[j]);
         printf("\n"); fflush(stdout);
      }
   }
```

the function `main` continued

```
   for(j=0; j<p-i; j++)  /* processor i performs p-i steps */
      if(i==0)
      {
         g = n[j];
         if(verbose>0){
            printf("Manager gets %d.\n",n[j]); fflush(stdout);
         }
         Compare_and_Send(i,j,&s,&g);
      }
      else
      {
         MPI_Recv(&g,1,MPI_INT,i-1,tag,MPI_COMM_WORLD,&status);
         if(verbose>0){
            printf("Node %d receives %d.\n",i,g); fflush(stdout);
         }
         Compare_and_Send(i,j,&s,&g);
      }
   MPI_Barrier(MPI_COMM_WORLD); /* to synchronize for printing */
   Collect_Sorted_Sequence(i,p,s,n);
   MPI_Finalize();
   return 0;
}
```

the function `Compare_and_Send`

```
void Compare_and_Send
 ( int myid, int step, int *smaller, int *gotten )
/* Processor "myid" initializes smaller with gotten
 * at step zero, or compares smaller to gotten and
 * sends the larger number through. */
{
   if(step==0)
      *smaller = *gotten;
   else
      if(*gotten > *smaller)
      {
         MPI_Send(gotten,1,MPI_INT,myid+1,tag,MPI_COMM_WORLD);
         if(verbose>0)
         {
            printf("Node %d sends %d to %d.\n",
                   myid,*gotten,myid+1);
            fflush(stdout);
         }
      }
```

Compare_and_Send continued

```
      else
      {
         MPI_Send(smaller,1,MPI_INT,myid+1,tag,
                  MPI_COMM_WORLD);
         if(verbose>0)
         {
            printf("Node %d sends %d to %d.\n",
                   myid,*smaller,myid+1);
            fflush(stdout);
         }
         *smaller = *gotten;
      }
}
```

the function Collect_Sorted_Sequence

```
void Collect_Sorted_Sequence
 ( int myid, int p, int smaller, int *sorted ) {
/* Processor "myid" sends its smaller number to the
 * manager who collects the sorted numbers in the
 * sorted array, which is then printed. */
   MPI_Status status;
   int k;
   if(myid==0) {
      sorted[0] = smaller;
      for(k=1; k<p; k++)
         MPI_Recv(&sorted[k],1,MPI_INT,k,tag,
                  MPI_COMM_WORLD,&status);
      printf("The sorted sequence : ");
      for(k=0; k<p; k++) printf(" %d",sorted[k]);
      printf("\n");
   }
   else
      MPI_Send(&smaller,1,MPI_INT,0,tag,MPI_COMM_WORLD);
}
```

Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort
2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

the sieve of Erathostenes

List all prime numbers between 2 and 21.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

A sieving method proceeds as follows:

1. Remove all multiples of 2.

2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21

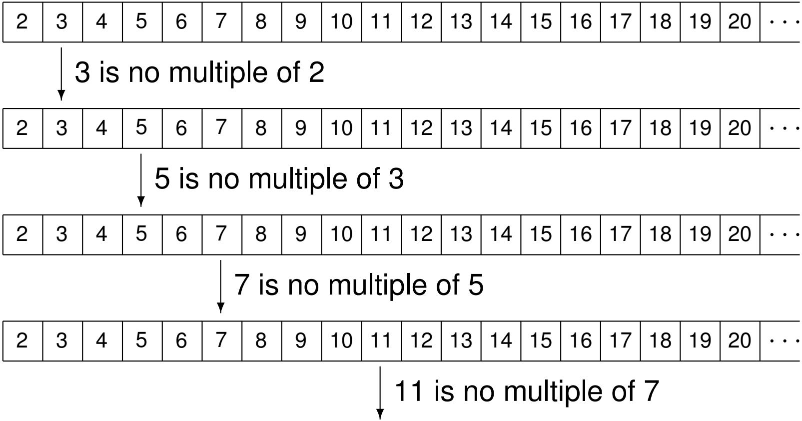
2. Remove all multiples of 3.

2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~

3. The next prime is 5.
   As we have already removed all multiples of 2 and 3
   and $5 \times 5 = 25 > 21$, the sieving stops.

Compute in parallel?

making a pipeline



Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort
2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

a pipelined sieve algorithm

One stage in the pipeline:

1. receive a prime,
2. receive a sequence of numbers,
3. remove from the sequence all multiples of the prime,
4. send the filtered list to the next stage.

This pipeline algorithm is of type 2: data are passed to the next stage *before* the completion of the current stage.

The *before* is critical to obtain parallelism.

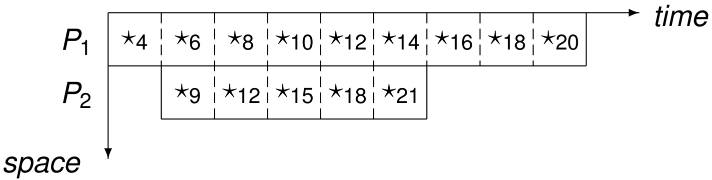
a 2-stage pipeline for all primes ≤ 21

To compute all primes ≤ 21 with the sieve algorithm:

1. P_1 removes all multiples of 2, in nine multiplications;
2. P_2 removes all multiples of 3, in five multiplications.

Although the second stage in the pipeline starts
only after we determined that 3 is not a multiple of 2,
there are fewer multiplications in the second stage.

The space-time diagram with the multiplications is below,
the subscript of each $\star$ is a removed number:



Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort

2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline

3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an *n*-stage pipeline

one sieving processor in a pipeline

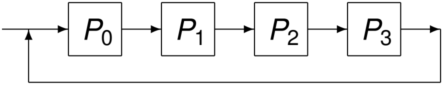
The input of a sieving processor is a tuple $(\mathbf{x}, i)$:

1. $\mathbf{x}$ is a sequence of numbers of fixed size n;
2. i is either $n + 1$ or the index to the next prime,
    - if $i = n + 1$, then all multiples have been removed from $\mathbf{x}$,
    - if $i \neq n + 1$, then x_i is no multiple of any previous number in $\mathbf{x}$.

If $i \neq n + 1$, the sieving processor has two tasks:

- 1. Determine the first index j, $j > i$, so that x_j is no multiple of any previous number in $\mathbf{x}$, or otherwise set $j = n + 1$ if at the end of the sequence.
  2. Send $(\mathbf{x}, j)$ to the next sieving processor in the pipeline.
- Remove all multiples of x_i from $\mathbf{x}$.

running a ring of four sieving processors



P_0	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	12	13	14	15	16	17	18	19	20	$\cdots$

3 is no multiple of 2

P_1	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	12	13	14	15	16	17	18	19	20	$\cdots$

5 is no multiple of 3

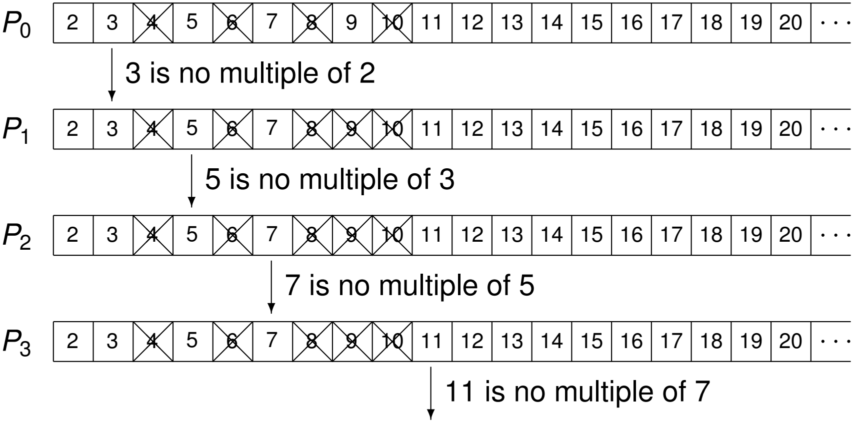
P_2	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	12	13	14	15	16	17	18	19	20	$\cdots$

7 is no multiple of 5

P_3	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	12	13	14	15	16	17	18	19	20	$\cdots$

11 is no multiple of 7

mind the gap ...



In order for P_3 to determine that 11 is the next prime,

- P_0 must have removed already 8 and 10, and
- P_1 must have removed already 9.

the data vector

The data vector **x** is an array of booleans:

- $x_i = 1$, if x_i is a multiple of some number;
- $x_i = 0$, if x_i is not a multiple.

On a parallel shared memory computer,

- there is only one copy of **x**;
- processors set the values of multiples to 1.

On a parallel distributed memory computer,

- there are as many copies of **x** as there are processors;
- the first processor at the end collects all copies of **x**.

In both implementations, the numbers that need to be removed before passing the sequence to the next step will depend on the number of sieving processors in the pipeline.

Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort
2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

formulas for forward substitution

Expanding the matrix-vector product $L\mathbf{y}$ in $L\mathbf{y} = \mathbf{b}$ leads to

$$\left\{ \begin{array}{lcl} y_1 & = & b_1 \\ \ell_{2,1} y_1 + y_2 & = & b_2 \\ \ell_{3,1} y_1 + \ell_{3,2} y_2 + y_3 & = & b_3 \\ & \vdots & \\ \ell_{n,1} y_1 + \ell_{n,2} y_2 + \ell_{n,3} y_3 + \cdots + \ell_{n,n-1} y_{n-1} + y_n & = & b_n \end{array} \right.$$

and solving for the diagonal elements gives

$$\begin{array}{lcl} y_1 & = & b_1 \\ y_2 & = & b_2 - \ell_{2,1} y_1 \\ y_3 & = & b_3 - \ell_{3,1} y_1 - \ell_{3,2} y_2 \\ & \vdots & \\ y_n & = & b_n - \ell_{n,1} y_1 - \ell_{n,2} y_2 - \cdots - \ell_{n,n-1} y_{n-1} \end{array}$$

formula and algorithm

For $k = 1, 2, \ldots, n$:

$$y_k = b_k - \sum_{i=1}^{k-1} \ell_{k,i} y_i.$$

As an algorithm:

$$
\begin{array}{l}
\text{for } k \text{ from 1 to } n \text{ do} \\
\quad y_k := b_k; \\
\quad \text{for } i \text{ from 1 to } k-1 \text{ do} \\
\quad\quad y_k := y_k - \ell_{k,i} \star y_i.
\end{array}
$$

We count

$$1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2}$$

multiplications and subtractions.

Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort

2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline

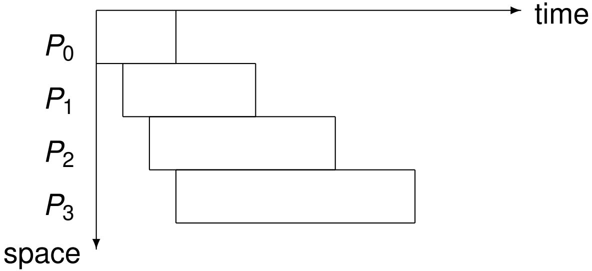
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

a third type of pipeline

Three types of pipelines:

1. Speedup only if multiple instances. Example: instruction pipeline.
2. Speedup already if one instance. Example: pipeline sorting.
3. Worker continues after passing information through.
   Example: solve $L\mathbf{y} = \mathbf{b}$.

Typical for the 3rd type of pipeline is the varying length of each job.



Pipelined Sorting and Sieving

1. Sorting Numbers
   - a parallel version of insertion sort
   - MPI code for a pipeline version of insertion sort

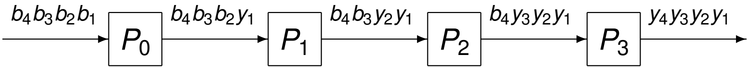
2. Prime Number Generation
   - the sieve of Erathosthenes
   - type 2 pipelining
   - sieving processors in a pipeline

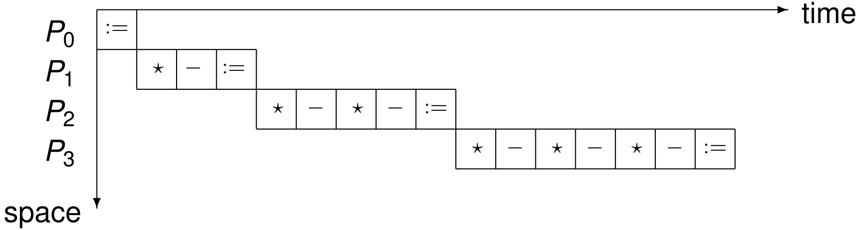
3. Solving Triangular Systems
   - forward substitution formulas
   - a third type of pipeline
   - using an n-stage pipeline

using an n-stage pipeline

We assume that L is available on every processor.

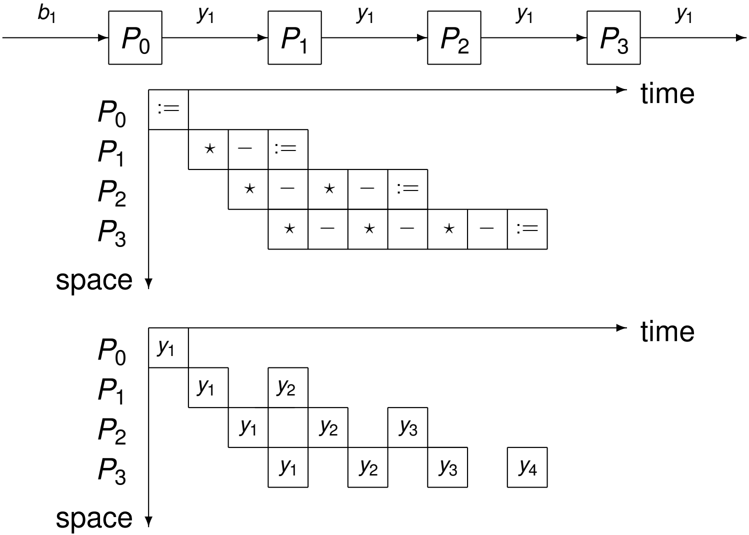
$$
\text{for } n = 4 = p: \quad
\begin{aligned}
y_1 &:= b_1 \\
y_2 &:= b_2 - \ell_{2,1} \star y_1 \\
y_3 &:= b_3 - \ell_{3,1} \star y_1 - \ell_{3,2} \star y_2 \\
y_4 &:= b_4 - \ell_{4,1} \star y_1 - \ell_{4,2} \star y_2 - \ell_{4,3} \star y_3
\end{aligned}
$$



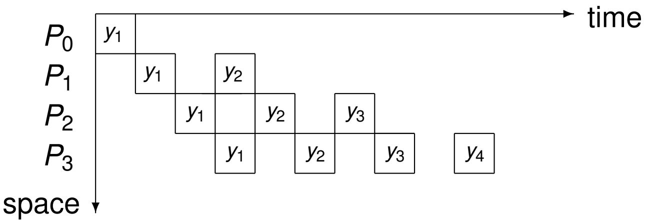


type 3 pipelining

Make y_1 available in the next pipeline cycle:



counting the steps



We count the steps for $p = 4$ or in general, for $p = n$:

1. The latency takes 4 steps for y_1 to be at P_4,
   or in general: n steps for y_1 to be at P_n.
2. It takes then 6 additional steps for y_4 to be computed by P_4,
   or in general: $2n - 2$ additional steps for y_n to be computed by P_n.

So it takes $n + 2n - 2 = 3n - 2$ steps to solve
an n-dimensional triangular system by an n-stage pipeline.

Summary + Exercises

We ended chapter 5 in the book of Wilkinson and Allen.

Exercises:

1. Write the pipelined sorting algorithm with OpenMP or Julia. Demonstrate the correctness of your implementation with some good examples.
2. Use message passing to implement the pipelined sieve algorithm. Relate the number of processors in the network to the number of multiples which must be computed before sending off the sequence to the next processor.
3. Implement the pipelined sieve algorithm with OpenMP and Julia. Can the constraint on the number of computed multiples be formulated with dependencies?

one last exercise

4. Consider the upper triangular system $U\mathbf{x} = \mathbf{y}$, with $U = [u_{i,j}]$, $u_{i,j} = 0$ if $i > j$. Derive the formulas and general algorithm to compute the components of the solution $\mathbf{x}$.
   For $n = 4$, draw the third type of pipeline.