

# Parallel Iterative Methods for Linear Systems

## 1 Jacobi iterations

- derivation of the formulas
- parallel version with butterfly synchronization

## 2 a Parallel Implementation with MPI

- the sequential program
- gather-to-all with `MPI_Allgather`
- the parallel program
- analysis of the computation and communication cost

MCS 572 Lecture 19  
Introduction to Supercomputing  
Jan Verschelde, 5 October 2016

# Parallel Iterative Methods for Linear Systems

## 1 Jacobi iterations

- derivation of the formulas
- parallel version with butterfly synchronization

## 2 a Parallel Implementation with MPI

- the sequential program
- gather-to-all with `MPI_Allgather`
- the parallel program
- analysis of the computation and communication cost

## a fixed point formula

We want to solve  $A\mathbf{x} = \mathbf{b}$  for  $A \in \mathbb{R}^{n \times n}$ ,  $\mathbf{b} \in \mathbb{R}^n$ , for **very large**  $n$ .

Consider  $A = L + D + U$ , where

- $L = [\ell_{i,j}]$ ,  $\ell_{i,j} = a_{i,j}$ ,  $i > j$ ,  $\ell_{i,j} = 0$ ,  $i \leq j$ .  $L$  is lower triangular.
- $D = [d_{i,j}]$ ,  $d_{i,i} = a_{i,i} \neq 0$ ,  $d_{i,j} = 0$ ,  $i \neq j$ .  $D$  is diagonal.
- $U = [u_{i,j}]$ ,  $u_{i,j} = a_{i,j}$ ,  $i < j$ ,  $u_{i,j} = 0$ ,  $i \geq j$ .  $U$  is upper triangular.

Then we rewrite  $A\mathbf{x} = \mathbf{b}$  as

$$\begin{aligned} A\mathbf{x} = \mathbf{b} &\Leftrightarrow (L + D + U)\mathbf{x} = \mathbf{b} \\ &\Leftrightarrow D\mathbf{x} = \mathbf{b} - L\mathbf{x} - U\mathbf{x} \\ &\Leftrightarrow D\mathbf{x} = D\mathbf{x} + \mathbf{b} - L\mathbf{x} - U\mathbf{x} - D\mathbf{x} \\ &\Leftrightarrow D\mathbf{x} = D\mathbf{x} + \mathbf{b} - A\mathbf{x} \\ &\Leftrightarrow \mathbf{x} = \mathbf{x} + D^{-1}(\mathbf{b} - A\mathbf{x}). \end{aligned}$$

The fixed point formula  $\mathbf{x} = \mathbf{x} + D^{-1}(\mathbf{b} - A\mathbf{x})$  is well defined if  $a_{i,i} \neq 0$ .

# the Jacobi iterative method

The fixed point formula  $\mathbf{x} = \mathbf{x} + D^{-1}(\mathbf{b} - A\mathbf{x})$  leads to

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \underbrace{D^{-1}(\mathbf{b} - A\mathbf{x}^{(k)})}_{\Delta\mathbf{x}}, \quad k = 0, 1, \dots$$

Writing the formula as an algorithm:

Input:  $A, \mathbf{b}, \mathbf{x}^{(0)}, \epsilon, N$ .

Output:  $\mathbf{x}^{(k)}$ ,  $k$  is the number of iterations done.

for  $k$  from 1 to  $N$  do

$\Delta\mathbf{x} := D^{-1}(\mathbf{b} - A\mathbf{x}^{(k)})$ ;

$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta\mathbf{x}$ ;

    exit when  $(\|\Delta\mathbf{x}\| \leq \epsilon)$ ;

end for.

# cost and convergence

Counting the number of operations in

```
for  $k$  from 1 to  $N$  do  
   $\Delta \mathbf{x} := D^{-1}(\mathbf{b} - A\mathbf{x}^{(k)});$   
   $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta \mathbf{x};$   
  exit when  $(\|\Delta \mathbf{x}\| \leq \epsilon);$   
end for.
```

we have a cost of  $O(Nn^2)$ ,  $O(n^2)$  for  $A\mathbf{x}^{(k)}$ , if  $A$  is dense.

## Theorem

*The Jacobi method converges for strictly row-wise or column-wise diagonally dominant matrices, i.e.: if*

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}| \quad \text{or} \quad |a_{i,i}| > \sum_{j \neq i} |a_{j,i}|, \quad i = 1, 2, \dots, n.$$

# Parallel Iterative Methods for Linear Systems

## 1 Jacobi iterations

- derivation of the formulas
- parallel version with butterfly synchronization

## 2 a Parallel Implementation with MPI

- the sequential program
- gather-to-all with `MPI_Allgather`
- the parallel program
- analysis of the computation and communication cost

# parallel version of Jacobi iterations

```
for  $k$  from 1 to  $N$  do  
   $\Delta \mathbf{x} := D^{-1}(\mathbf{b} - A\mathbf{x}^{(k)});$   
   $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta \mathbf{x};$   
  exit when  $(\|\Delta \mathbf{x}\| \leq \epsilon);$   
end for.
```

To run the code above with  $p$  processors:

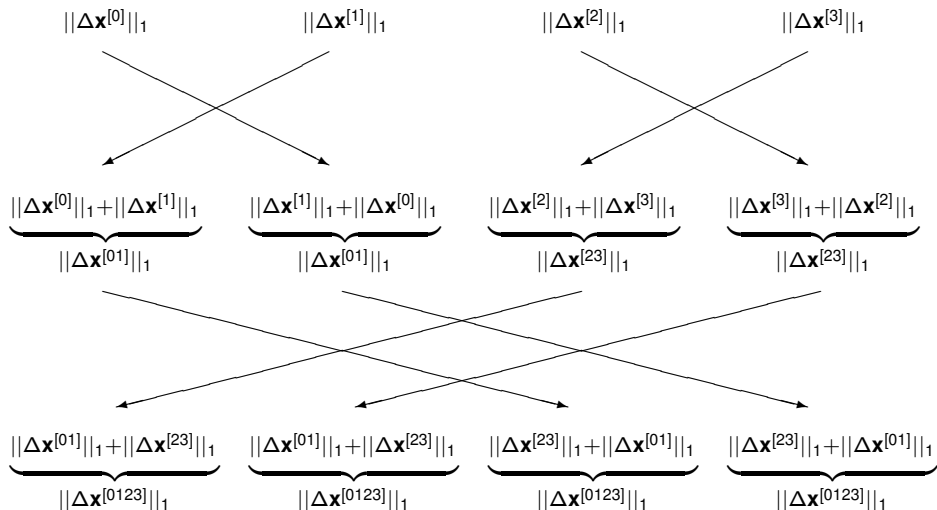
- The  $n$  rows of  $A$  are distributed evenly (e.g.:  $p = 4$ ):

$$D \star \begin{bmatrix} \Delta \mathbf{x}^{[0]} \\ \Delta \mathbf{x}^{[1]} \\ \Delta \mathbf{x}^{[2]} \\ \Delta \mathbf{x}^{[3]} \end{bmatrix} = \begin{bmatrix} \mathbf{b}^{[0]} \\ \mathbf{b}^{[1]} \\ \mathbf{b}^{[2]} \\ \mathbf{b}^{[3]} \end{bmatrix} - \begin{bmatrix} A^{[0,0]} & A^{[0,1]} & A^{[0,2]} & A^{[0,3]} \\ A^{[1,0]} & A^{[1,1]} & A^{[1,2]} & A^{[1,3]} \\ A^{[2,0]} & A^{[2,1]} & A^{[2,2]} & A^{[2,3]} \\ A^{[3,0]} & A^{[3,1]} & A^{[3,2]} & A^{[3,3]} \end{bmatrix} \star \begin{bmatrix} \mathbf{x}^{[0],(k)} \\ \mathbf{x}^{[1],(k)} \\ \mathbf{x}^{[2],(k)} \\ \mathbf{x}^{[3],(k)} \end{bmatrix}$$

- Synchronization is needed for  $(\|\Delta \mathbf{x}\| \leq \epsilon).$

# butterfly synchronization

For  $|| \cdot ||$ , use  $||\Delta \mathbf{x}||_1 = |\Delta x_1| + |\Delta x_2| + \dots + |\Delta x_n|$ .





# communication and computation stages

The communication stages:

- At the start, every node must have  $\mathbf{x}^{(0)}$ ,  $\epsilon$ ,  $N$ ,
  - ▶ a number of rows of the matrix  $A$ ; and
  - ▶ the corresponding part of the right hand side vector  $\mathbf{b}$ .
- After each update  $n/p$  elements of  $\mathbf{x}^{(k+1)}$  must be scattered.
- The butterfly synchronization takes  $\log_2(p)$  steps.

The scattering of  $\mathbf{x}^{(k+1)}$  can coincide with the butterfly synchronization.

The computation effort:  $O(n^2/p)$  in each stage.

# Parallel Iterative Methods for Linear Systems

## 1 Jacobi iterations

- derivation of the formulas
- parallel version with butterfly synchronization

## 2 a Parallel Implementation with MPI

- the sequential program
- gather-to-all with `MPI_Allgather`
- the parallel program
- analysis of the computation and communication cost

# the test system

For the dimension  $n$ , we consider the diagonally dominant system:

$$\begin{bmatrix} n+1 & 1 & \cdots & 1 \\ 1 & n+1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & n+1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 2n \\ 2n \\ \vdots \\ 2n \end{bmatrix}.$$

The exact solution is  $\mathbf{x}$ : for  $i = 1, 2, \dots, n$ ,  $x_i = 1$ .

We start the Jacobi iteration method at  $\mathbf{x}^{(0)} = \mathbf{0}$ .

Parameters:  $\epsilon = 10^{-4}$  and  $N = 2n^2$ .

# running the program

```
$ time /tmp/jacobi 1000
  0 : 1.998e+03
  1 : 1.994e+03
...
8405 : 1.000e-04
8406 : 9.982e-05
computed 8407 iterations
error : 4.986e-05

real      0m42.411s
user      0m42.377s
sys       0m0.028s
```

# C code to run Jacobi iterations

```
void run_jacobi_method
( int n, double **A, double *b,
  double epsilon, int maxit,
  int *numit, double *x );
/*
 * Runs the Jacobi method for  $Ax = b$ .
 *
 * ON ENTRY :
 *   n           the dimension of the system;
 *   A           an n-by-n matrix  $A[i][i] \neq 0$ ;
 *   b           an n-dimensional vector;
 *   epsilon     accuracy requirement;
 *   maxit       maximal number of iterations;
 *   x           start vector for the iteration.
 *
 * ON RETURN :
 *   numit       number of iterations used;
 *   x           approximate solution to  $Ax = b$ . */
```

# local variables

```
void run_jacobi_method
( int n, double **A, double *b,
  double epsilon, int maxit,
  int *numit, double *x )
{
    double *dx,*y;
    dx = (double*) calloc(n,sizeof(double));
    y = (double*) calloc(n,sizeof(double));
    int i,j,k;

    for(k=0; k<maxit; k++) { ... } /* main loop */

    *numit = k+1;
    free(dx); free(y);
}
```

# the main loop in C

```
for(k=0; k<maxit; k++)
{
    double sum = 0.0;
    for(i=0; i<n; i++)
    {
        dx[i] = b[i];
        for(j=0; j<n; j++)
            dx[i] -= A[i][j]*x[j];
        dx[i] /= A[i][i];
        y[i] += dx[i];
        sum += ( (dx[i] >= 0.0) ? dx[i] : -dx[i]);
    }
    for(i=0; i<n; i++) x[i] = y[i];
    printf("%3d : %.3e\n",k,sum);
    if(sum <= epsilon) break;
}
```

# Parallel Iterative Methods for Linear Systems

## 1 Jacobi iterations

- derivation of the formulas
- parallel version with butterfly synchronization

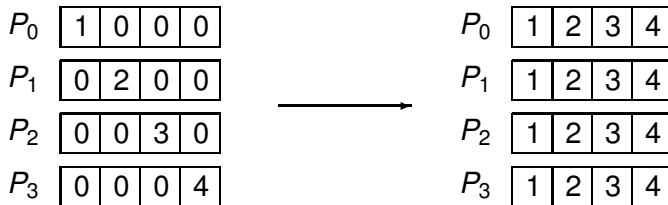
## 2 a Parallel Implementation with MPI

- the sequential program
- **gather-to-all with `MPI_Allgather`**
- the parallel program
- analysis of the computation and communication cost



# gather-to-all

Gathering the four elements of a vector to four processors:



## the MPI\_Allgather

The syntax of the gather-to-all command is

```
MPI_Allgather(sendbuf, sendcount, sendtype,  
              recvbuf, recvcount, recvtype, comm)
```

where the parameters are

sendbuf	starting address of send buffer
sendcount	number of elements in send buffer
sendtype	data type of send buffer elements
<hr/>	
recvbuf	address of receive buffer
recvcount	number of elements received from any process
recvtype	data type of receive buffer elements
<hr/>	
comm	communicator

## running use\_allgather

```
$ mpirun -np 4 /tmp/use_allgather
data at node 0 : 1 0 0 0
data at node 1 : 0 2 0 0
data at node 2 : 0 0 3 0
data at node 3 : 0 0 0 4
data at node 3 : 1 2 3 4
data at node 0 : 1 2 3 4
data at node 1 : 1 2 3 4
data at node 2 : 1 2 3 4
$
```

## the code use\_allgather.c

```
int i,j,p;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&i);
MPI_Comm_size(MPI_COMM_WORLD,&p);
{
    int data[p];
    for(j=0; j<p; j++) data[j] = 0;
    data[i] = i + 1;
    printf("data at node %d :",i);
    for(j=0; j<p; j++) printf(" %d",data[j]);
    printf("\n");
    MPI_Allgather(&data[i],1,MPI_INT,
                  data,1,MPI_INT,MPI_COMM_WORLD);
    printf("data at node %d :",i);
    for(j=0; j<p; j++) printf(" %d",data[j]);
    printf("\n");
}
```

# Parallel Iterative Methods for Linear Systems

## 1 Jacobi iterations

- derivation of the formulas
- parallel version with butterfly synchronization

## 2 a Parallel Implementation with MPI

- the sequential program
- gather-to-all with `MPI_Allgather`
- **the parallel program**
- analysis of the computation and communication cost

## running jacobi\_mpi

```
$ time mpirun -np 10 /tmp/jacobi_mpi 1000
```

```
...
```

```
8405 : 1.000e-04
```

```
8406 : 9.982e-05
```

```
computed 8407 iterations
```

```
error : 4.986e-05
```

```
real      0m5.617s
```

```
user      0m45.711s
```

```
sys       0m0.883s
```

Recall the run with the sequential program:

```
real      0m42.411s
```

```
user      0m42.377s
```

```
sys       0m0.028s
```

Speedup:  $42.411 / 5.617 = 7.550$ .

## the parallel `run_jacobi_method`

```
void run_jacobi_method
( int id, int p,
  int n, double **A, double *b,
  double epsilon, int maxit,
  int *numit, double *x )
{
    double *dx,*y;
    dx = (double*) calloc(n,sizeof(double));
    y = (double*) calloc(n,sizeof(double));
    int i,j,k;
    double sum[p];
    double total;
    int dnp = n/p;
    int istart = id*dnp;
    int istop = istart + dnp;
```

## the main loop in `jacobi_mpi.c`

```
for(k=0; k<maxit; k++)
{
    sum[id] = 0.0;
    for(i=istart; i<istop; i++)
    {
        dx[i] = b[i];
        for(j=0; j<n; j++)
            dx[i] -= A[i][j]*x[j];
        dx[i] /= A[i][i];
        y[i] += dx[i];
        sum[id] += ( (dx[i] >= 0.0) ? dx[i] : -dx[i]);
    }
    for(i=istart; i<istop; i++) x[i] = y[i];
}
```



# the all-to-all communication

```
MPI_Allgather(&x[istart], dnp, MPI_DOUBLE, x, dnp,  
              MPI_DOUBLE, MPI_COMM_WORLD);  
MPI_Allgather(&sum[id], 1, MPI_DOUBLE, sum, 1,  
              MPI_DOUBLE, MPI_COMM_WORLD);  
total = 0.0;  
for(i=0; i<p; i++) total += sum[i];  
if(id == 0) printf("%3d : %.3e\n", k, total);  
if(total <= epsilon) break;  
}  
*numit = k+1;  
free(dx);  
}
```

# Parallel Iterative Methods for Linear Systems

## 1 Jacobi iterations

- derivation of the formulas
- parallel version with butterfly synchronization

## 2 a Parallel Implementation with MPI

- the sequential program
- gather-to-all with `MPI_Allgather`
- the parallel program
- analysis of the computation and communication cost

## analysis

Computing  $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + D^{-1}(\mathbf{b} - A\mathbf{x}^{(k)})$  with  $p$  processors costs

$$t_{\text{comp}} = \frac{n(2n+3)}{p}.$$

We count  $2n+3$  operations because of

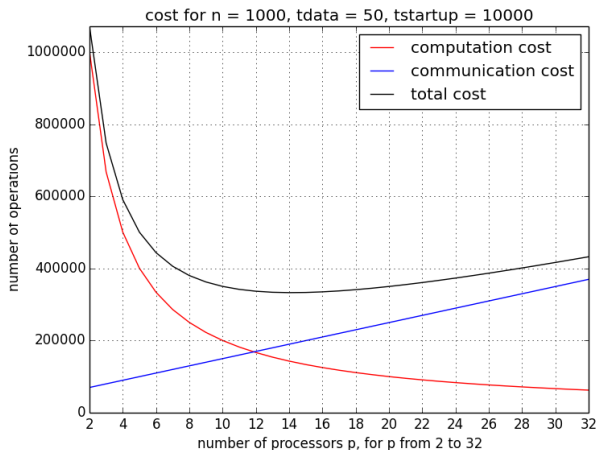
- one  $-$  and one  $\star$  when running over the columns of  $A$ ; and
- one  $/$ , one  $+$  for the update and one  $+$  for the  $\|\cdot\|_1$ .

The communication cost is

$$t_{\text{comm}} = p \left( t_{\text{startup}} + \frac{n}{p} t_{\text{data}} \right).$$

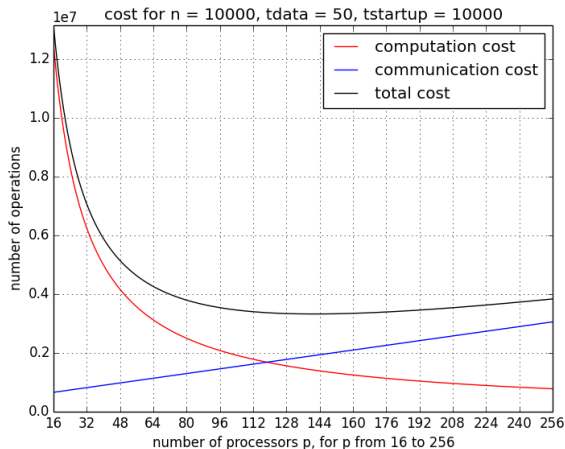
In the examples, the time unit is the cost of one arithmetical operation. Then the costs  $t_{\text{startup}}$  and  $t_{\text{data}}$  are multiples of this unit.

# finding the $p$ with the minimum total cost



The **computation**, **communication**, and total cost for  $p$  from 2 to 32, for 1 iteration,  $n = 1,000$ ,  $t_{\text{startup}} = 10,000$ , and  $t_{\text{data}} = 50$ .

# investigating the scalability



The **computation**, **communication**, and total cost for  $p$  from 16 to 256, for 1 iteration,  $n = 10,000$ ,  $t_{\text{startup}} = 10,000$ , and  $t_{\text{data}} = 50$ .

# Summary + Exercises

We covered §6.3.1 in the book of Wilkinson and Allen.

*Because of its slow convergence, the Jacobi method is seldomly used.*

## Exercises:

- 1 Use OpenMP to write a parallel version of the Jacobi method. Do you observe a better speedup than with MPI?
- 2 The power method to compute the largest eigenvalue of a matrix  $A$  uses the formulas  $\mathbf{y} := A\mathbf{x}^{(k)}$ ;  $\mathbf{x}^{(k+1)} := \mathbf{y}/\|\mathbf{y}\|$ . Describe a parallel implementation of the power method.
- 3 Consider the formula for the total cost of the Jacobi method for an  $n$ -dimensional linear system with  $p$  processors. Derive an analytic expression for the optimal value of  $p$ . What does this expression tell about the scalability?