Modelling and implementation of algorithms in applied mathematics using MPI

Lecture 3: Linear Systems: Simple Iterative Methods and their parallelization, Programming MPI

G. Rapin

Brazil
March 2011

Outline

1 Simple Iterative Methods for linear systems

2 Parallelisation of Iterative Methods

3 Programming MPI

Outline

1 Simple Iterative Methods for linear systems

2 Parallelisation of Iterative Methods

3 Programming MPI

Typical Properties of Linear Systems given by discretization of PDEs

Definitions

- We want to solve a linear system $Ax = b$ where A is non-singular.
- The distance between grid points is h.
- In 3D applications there are $\mathcal{O}(1/h^3)$ grid points.

Properties

- The matrices are sparse with mostly non-regular patterns.
- The linear systems are huge with millions of unknowns.
- The condition numbers of the matrices are high.

Direct Methods

- Direct methods computes the exact solution in a finite number of steps; usually a factorization is computed.
- Gauss-elimination is given by

$$A = LR,$$

  where R is an upper right triangle matrix and L is a lower left triangle matrix with ones on the diagonal.
- Linear system $Ax = LRx = b$ can be solved in two steps: $Ly = b$ and then $Rx = y$.
- Approach is inefficient for sparse systems due to fill-in. The sparsity pattern of A will in general not be preserved.

General Iterative Methods

- We want to solve a linear system $Ax = b$ with $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$.
- Let $B \in \mathbb{R}^{n \times n}$ a non-singular matrix.
- Using $B + (A - B)$ we can rewrite $Ax = b$ as

$$x = B^{-1}(B - A)x + B^{-1}b$$

  This is a fixed-point equation.
- Typically, one uses a simple iteration for the solution of the fixed-point iteration

$$x_{k+1} = B^{-1}(B - A)x_k + B^{-1}b, \quad k \in \mathbb{N}_0$$

  where x_0 is an arbitrary initial guess.

Splitting matrix B

Mathematical Conditions for B

- B^{-1} must exist.
- The sequence $(x_n)_n$ converges.

Algorithmic Conditions for B

- Efficient solution of the system $Bv = g$,
- Fast computation of $(B - A)v$,
- Fast convergence of sequence $(x_n)_n$.

Lipschitz Continuity

Introducing

$$F(x) := B^{-1}(B - A)x + B^{-1}b.$$

we get

$$\begin{aligned} \|F(x) - F(y)\| &= \|B^{-1}(B - A)(x - y)\| \\ &\leq \|B^{-1}(B - A)\| \|x - y\| \\ &= \delta\|x - y\|, \quad x, y \in \mathbb{R}^n \end{aligned}$$

with $\delta := \|B^{-1}(B - A)\|$.

Convergence

Then, the fixed-point theorem of Banach gives

Theorem

Let $\|\cdot\|$ be a vector norm in $\mathbb{R}^n$ and

$$\|C\| := \sup_{x\in\mathbb{R}^n} \frac{\|Cx\|}{\|x\|}, \quad C \in \mathbb{R}^{n\times n}$$

the matrix norm. Assuming $\delta := \|B^{-1}(B-A)\| < 1$, then the sequence $(x_n)_n$ converges for all initial values $x_0 \in \mathbb{R}^n$ to the solution $x \in \mathbb{R}^n$ of $Ax = b$. The error is bounded by

$$\|x_{k+1} - x\| \leq \frac{\delta^k}{1-\delta}\|x_1 - x_0\|, \quad k \in \mathbb{N}_0.$$

Remarks to convergence

- We have to ensure that $\delta < 1$.
- The *spectral radius* $\rho(C)$ of a square matrix C is defined by

  $$\rho(C) := \max_i |\lambda_i|,$$

  where λ_i are the eigenvalues of C.
- It can be proved that $(x_n)_n$ converges if and only if $\rho(C) < 1$.
- An upper bound for the convergence speed is given by δ.

Choices for B

Decompose matrix $A = (a_{ij})$ into

$$A = A_d + A_u + A_o, \quad A_d, A_u, A_o \in \mathbb{R}^{n\times n}$$

$A_d = diag(a_{11}, \ldots, a_{nn})$ is a diagonal matrix and

$$A_u := \begin{pmatrix} 0 & & & & \\ a_{21} & 0 & & & \\ a_{31} & a_{32} & 0 & & \\ \vdots & \vdots & \ddots & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 0 \end{pmatrix}, \quad A_o := \begin{pmatrix} 0 & a_{12} & \cdots & \cdots & a_{1n} \\ & 0 & a_{23} & \cdots & a_{2n} \\ & & \ddots & \ddots & \vdots \\ & & & 0 & a_{n-1,n} \\ & & & & 0 \end{pmatrix}$$

Jacobi method

Choose $B = A_d$!

Algorithm 1: **Jacobi method (general)**

```
Choose initial vector $u^0 \in \mathbb{R}^n$
For $k = 1, 2, \ldots$
  For $i = 1, 2, \ldots, n$
    $u_i^k = \frac{1}{a_{ii}} \left( b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} u_i^{k-1} \right)$.
  end i
```

Jacobi method for Finite-Differences

Inserting the matrix of the Finite Differences we get

Algorithm 2: **Jacobi method (Poisson Problem)**

```
Choose initial vector $u^0 \in \mathbb{R}^n$
For $k = 1, 2, \ldots$
  For $j = 1, 2, \ldots, (N-1)$
    For $i = 1, 2, \ldots, (N-1)$
      $u_{ij}^{k} = \frac{1}{4}\left(u_{i,j-1}^{k-1} + u_{i-1,j}^{k-1} + u_{i,j+1}^{k-1} + u_{i+1,j}^{k-1} + h^2 f_{ij}\right)$
    end i
  end j
```

with $h = 1/N$.

Gauss-Seidel method

Choose $B = A_d + A_u$.

Algorithm 3: **Gauss-Seidel method (general)**

```
Choose initial vector $u^0 \in \mathbb{R}^n$
For $k = 1, 2, \ldots$
  For $i = 1, 2, \ldots, n$
    $u_i^k = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} u_i^k - \sum_{j=i+1}^{n} a_{ij} u_i^{k-1} \right).$
  end i
```

Gauss-Seidel method applied to the Poisson Problem

Algorithm 4: **Gauss-Seidel method (Poisson Problem)**

```
Choose initial vector $u^0 \in \mathbb{R}^n$
For $k = 1, 2, \ldots$
  For $j = 1, 2, \ldots, (N-1)$
    For $i = 1, 2, \ldots, (N-1)$
      $u_{ij}^k = \frac{1}{4}\left(u_{i,j-1}^k + u_{i-1,j}^k + u_{i,j+1}^{k-1} + u_{i+1,j}^{k-1} + h^2 f_{ij}\right)$
    end i
  end j
```

M matrices

Definition

We call a matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ a M-matrix if the following conditions are fulfilled.

- $a_{ij} \leq 0$, $i, j = 1, \ldots n$, $i \neq j$.
- $A^{-1} \geq 0$ exists.

Remark

The discretized matrices of the Poisson problem using Finite Differences are M-matrices.

Convergence Criteria

- If a matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ is strongly diagonal dominant, i.e.

$$q_\infty := \max_{i=1,\dots,n} q_i < 1, \quad q_i := \sum_{\substack{k=1\\k\neq i}}^{n} \left|\frac{a_{ik}}{a_{ii}}\right|,$$

  then Gauss-Seidel and Jacobi method converges.
- Let A be a M matrix. Then, Gauss-Seidel and Jacobi method converge. The spectral radius of the Gauss-Seidel method is smaller than the spectral radius of Jacobi method.

Convergence Criteria

Definition

A matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, $n > 1$ is called indecompasable, if there are no permutation matrices P satisfying

$$P^t AP = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix}$$

with quadratic matrices A_{11} and A_{22}.

For indecompasable matrices we can show

Theorem

Let A be indecompasable and weakly diagonal dominant, i.e.

$$q_i \leq 1 \qquad \exists\, j \in \{1, \ldots n\} \mid q_j < 1.$$

Then, Gauss-Seidel and Jacobi method converges.

Eigenvalues of the Poisson Problem

- The matrix of the discretisation of the Poisson problem is

$$\begin{aligned} A &:= \frac{1}{h^2} tridiag(-I_{N-1}, T, -I_{N-1}) \in \mathbb{R}^{(N-1)^2 \times (N-1)^2}, \\ T &:= tridiag(-1, 4, -1) \in \mathbb{R}^{(N-1)\times(N-1)}. \end{aligned}$$

- The iteration matrix $J := A_d^{-1}(A_u + A_o)$ of Jacobi method is given by

$$J := \frac{1}{4} tridiag(I_{N-1}, B, I_{N-1}) \quad \text{with} \quad B = tridiag(1, 0, 1).$$

- The eigenvalues of J are

$$\mu^{(k,l)} := \frac{1}{2}\left(\cos\left(\frac{k\pi}{N}\right) + \cos\left(\frac{l\pi}{N}\right)\right), \quad k, l \in \{1, \dots, N-1\}.$$

Spectral Radius of the Poisson Problem

- Using the taylor expansion of $\cos(\cdot)$ at 0 yields

$$\cos(x) = 1 - \frac{1}{2}x^2 + \mathcal{O}(x^4)$$

- Therefore we get for the spectral radius of J

$$\rho(J) = \max_{k,l} |\mu^{(k,l)}| = \cos(\frac{\pi}{N}) = \cos(\pi h) = 1 - \frac{1}{2}\pi^2 h^2 + \mathcal{O}(h^4).$$

- Thus, the convergence depends on the grid distance h.
  For larger problems it converges slower.

SOR method

- The Gauss-Seidel method was given by

$$\tilde{u}_i^k = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_i^k - \sum_{j=i+1}^{n} a_{ij} u_i^{k-1} \right).$$

- Now we use some relaxation. For $\omega \in \mathbb{R}$ we get the SOR method.

$$\begin{aligned} u_i^k &= (1-\omega) u_i^{k-1} + \omega \tilde{u}_i^k \\ &= (1-\omega) u_i^{k-1} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_i^k - \sum_{j=i+1}^{n} a_{ij} u_i^{k-1} \right). \end{aligned}$$

- Using matrix notation we obtain

$$(A_d + \omega A_u) u^k = \left((1-\omega) A_d - \omega A_o \right) u^{k-1} + \omega b, \quad \omega \in \mathbb{R}.$$

Convergence Speed

Assume $\Lambda := \rho(A_d^{-1}(A_u + A_o)) < 1$. Then the spectral radius of

$$T(\omega) = (A_d + \omega A_u)^{-1}[(1-\omega)A_d - \omega A_o]$$

will be minimized by

$$\omega_0 = \frac{2}{1+\sqrt{1-\Lambda^2}} > 1$$

with

$$\rho(T(\omega_0)) = \frac{1-\sqrt{1-\Lambda^2}}{1+\sqrt{1-\Lambda^2}}.$$

- For the Poisson problem we have $\Lambda = 1 - \mathcal{O}(h^2)$: Therefore we obtain

$$\rho(T(\omega_0)) = 1 - \mathcal{O}(h).$$

Remarks

- The standard Gauss-Seidel method is given by $\omega = 1$.
- The method is called *successive over-relaxation* method, since faster convergence is achieved for $\omega > 1$.
- Let us assume that A is symmetric and positive definite, i.e. $x^T Ax > 0$ for all $x \in \mathbb{R}^n \setminus \{0\}$. Then, one can prove that the SOR method converges for $\omega \in (0, 2)$.

Jacobi Relaxation (JOR)

- Relaxation can be used for Jacobi method, too.
- It is defined by

$$u^k = (I - \omega A_d^{-1} A) u^{k-1} + \omega A_d^{-1} b.$$

- If the spectrum of $J := -A_d^{-1}(A_u + A_o)$ is real, the optimal relaxation parameter is

$$\omega_0 = \frac{2}{2 - \lambda_{min} - \lambda_{max}}.$$

λ_{min} is the minimal eigenvalue of J and λ_{max} is the maximal eigenvalue.

Outline

1 Simple Iterative Methods for linear systems

2 Parallelisation of Iterative Methods

3 Programming MPI

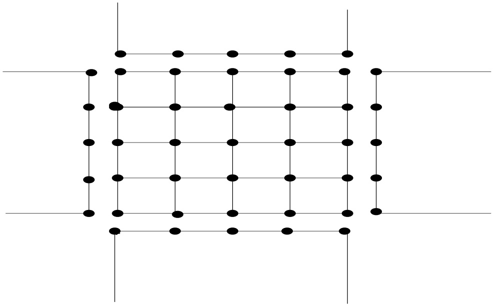
Parallelization of Jacobi and Gauss-Seidel method

- The parallelisation of Jacobi method is quite straight forward in contrast to the Gauss-Seidel method.
- Today, Jacobi and Gauss-Seidel method are rarely used as linear solvers due to their slow convergence.
- Krylov solvers like the conjugate gradient (CG) method are more appropriate.
- But Jacobi and Gauss-Seidel method are often used as preconditioners for Krylov methods or smoothers for multi grid approaches.

Decomposition

- For parallelisation we decompose the matrix $A \in \mathbb{R}^{n \times n}$ into sub-matrices.
- Assume that we have $m = p \cdot q$ processes. The sub matrices should have approximately the same size.
- Considering finite differences for each update the values of 4 neighboring points are needed. This information has to be sent.
- Point-to-Point communication is necessary.
- For the stopping criterion a global communication step is required.

Halo Points



Halo Points The used points on the neighboring domains are called *halo points* or *ghost points*.

Parallel Version of Jacobi method

Algorithm 5: **Jacobi method (parallel)**

```
Set error $> TOL$
WHILE error $> TOL$
Perform on process $i$:
    1 Compute all own components of the current iteration.
    2 Send all current values on the local boundary to the neighboring processes.
    3 Receive the current values on the ghost points from the neighboring processes.
Compute the current residual error $= \|b - Ax\|$.
EndWhile
```

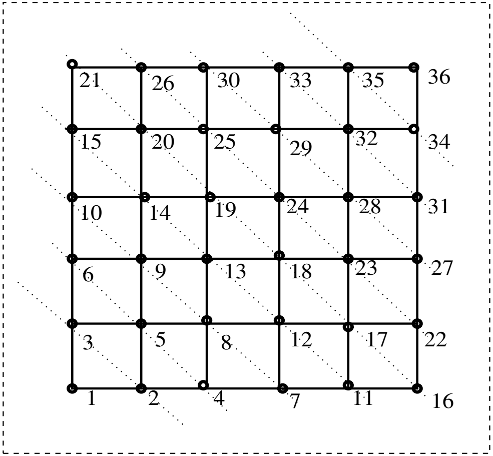
Remarks

- Normally, one uses for step 2 and step 3 blocking communication.
- The performance can be increased by using non-blocking communication. It is possible that on a processes several iteration steps are executed without updating the values on the ghost points.
- Then, one gets a different iteration sequence.
- This is especially attractive for shared-memory architectures because scheduling can be weakened.

Parallelisation of Gauss-Seidel method

- Considering our finite differences scheme for the k-th approximation at position (i, j), we need the already updated values $u^k_{i-1,j}$ and $u^k_{i,j-1}$.
- Therefore the parallelisation is not obvious.
- The solution is an appropriate sorting and renumbering of the unknowns.
- Two methods are proposed: Red-Black Numbering and Wavefront Numbering

Wavefront Numbering



Wavefront Numbering for a grid with 36 interior grid points

Wavefront Numbering

Using 36 unknowns and 6 processes we get

P_1 :	1	2	4	7	11	16	22	27	31	34	36
P_2 :		3	5	8	12	17	23	28	32	35	
P_3 :			6	9	13	18	24	29	33		
P_4 :				10	14	19	25	30			
P_5 :					15	20	26				
P_6 :						21					

Wavefront Numbering

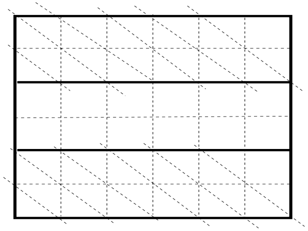
- The components of the diagonal can be computed in parallel.
- Essential disadvantage: the parallelisation is not uniform. First, the degree of parallelisation increases and finally decreases.
- Using a $P \times P$ grid with P processes, $2P-1$ steps are necessary for one iteration step.
- The average degree of parallelisation is

$$\frac{1}{2P-1}\left(\sum_{i=1}^{P} i + \sum_{i=1}^{P-1} i\right) = \frac{P^2}{2P-1}.$$

- Therefore the maximum speedup is ca. $p/2$ in this case.
- We obtain the same sequence of solutions as in the sequential algorithm.

Wavefront Numbering

- Now we consider $kP \times kP$ grid, $k \in \mathbb{N}$, with P processes.
- We split the grid in k stripes.



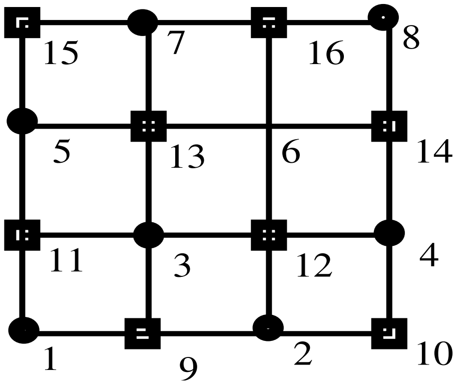
- In each stripe with kP^2 unknowns, $kP + P - 1$ steps are necessary.
- The average degree of parallelisation is

$$\frac{1}{kP + P - 1}\left(2\sum_{i=1}^{P-1} i + ((k-1)P + 1)P\right) = \frac{kP^2}{kP + P - 1}.$$

- Then, the theoretical speed-up is ca. $kP/(k+1)$.

Red-Black Numbering

Considering a quadratic grid the grid points are colored with red (●) and black (■) in such a way that neighboring nodes have different colors.



Red-Black Numbering

For the 16 above points we get the linear system

$$A = \begin{pmatrix} D_r & -C \\ -C^T & D_b \end{pmatrix}$$

with diagonal matrices $D_r := D_b := 4I_8$. The matrix C is given by

$$C = \begin{pmatrix}
1 & & 1 & & & & & \\
1 & 1 & & 1 & & & & \\
1 & & 1 & & 1 & 1 & & \\
 & 1 & & 1 & & 1 & & \\
 & & 1 & & 1 & & 1 & \\
 & & & 1 & 1 & 1 & & 1 \\
 & & & & 1 & & 1 & 1 \\
 & & & & & 1 & & 1
\end{pmatrix}.$$

Red-Black Numbering

Using the Gauss-Seidel method

$$(A_d + A_u)u^k = -A_o u^{k-1} + b.$$

and the special structure, we get

$$\begin{pmatrix} D_r & 0 \\ -C^T & D_b \end{pmatrix} \begin{pmatrix} u_r^k \\ u_b^k \end{pmatrix} = \begin{pmatrix} 0 & C \\ 0 & 0 \end{pmatrix} \begin{pmatrix} u_r^{k-1} \\ u_b^{k-1} \end{pmatrix} + \begin{pmatrix} b_r \\ b_b \end{pmatrix}$$

with

$$u^k = \begin{pmatrix} u_r^k \\ u_b^k \end{pmatrix}, b = \begin{pmatrix} b_r \\ b_b \end{pmatrix}.$$

Therefore we obtain the following two equations:

$$\begin{aligned} D_r u_r^k &= C u_b^{k-1} + b_r, \\ D_b u_b^k - C^T u_r^k &= b_b. \end{aligned}$$

Remarks

- If the 'black'-values u_b^{k-1} are known, the 'red'-values can be computed in parallel:

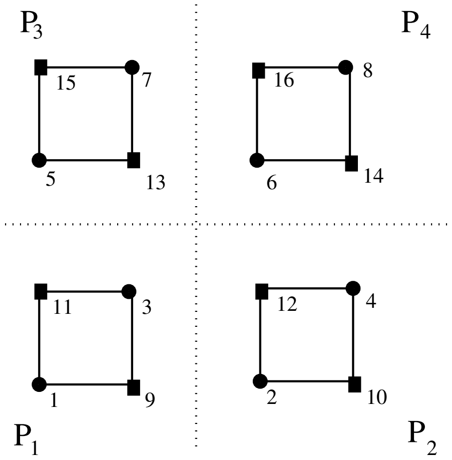
$$u_r^k = D_r^{-1}\left(Cu_b^{k-1} + b_r\right).$$

- Then, the 'black' points can be computed in parallel:

$$u_b^k = D_b^{-1}\left(C^T u_r^k + b_b\right).$$

- Thus, the algorithm scales perfectly.
- Notice, that the renumbering of the matrix A changes the iteration scheme.
- But one can prove that our application converges since the matrix A is symmetric and positive definite.

Parallel Distribution



Distribution of the nodes of a 4×4 grid on 4 processes

Parallel Gauss-Seidel method (Red-Black Numbering)

Algorithm 6: **Parallel Gauss-Seidel method (Red-Black Numbering)**

```
While error $>$ TOL; execute on process $i$:
    Compute for all values at 'red'-points a new approximation.
    Send the 'red'-values at the border of the sub-domains to the neighboring processes.
    Receive the new 'red'-values from the processes.
    Compute for all values at 'black'-points a new approximation.
    Send the 'black'-values at the border of the sub-domains to the neighboring processes.
    Receive the new 'black'-values from the processes.
Compute the residual error $= \|b - Au^k\|$.
EndWhile
```

Outline

1 Simple Iterative Methods for linear systems

2 Parallelisation of Iterative Methods

3 Programming MPI

Collective Communication

- Up to now we have seen Point-to-Point communication. There, only two processes are involved: a sender and a receiver.
- Communication patterns, where a group of processes is involved, is called *collective communication*. Usually, the number of involved processes is larger than two.
- Different classes of collective commands
  - Broadcast routines
  - Gather/Scatter routines
  - Reduction routines
  - Syncronization routines

Brodcast

Broadcast is a collective communication, where one sender sends a message to all other processes in the communicator group.

	int MPI_Bcast(void* message, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
IN:	
message	message (only process root)
count	Length of message
datatype	datatype of message
root	process number of sender
OUT:	
message	message (not process root)

Remarks

- Such routines are usually used for distributing data.
- Please observe that the syntax is the same for all processes, although the sender provides data and all others receive data.
- Defining communicators broadcasts can be just sent to subgroups.
- Example

```
MPI_Bcast(a_ptr, 1, MPI_FLOAT,
    0, MPI_COMM_WORLD);
```

Reduction Routines

All processes in a group contributes data to reduction routines. The data is combined using binary operations like additions or computing maxima or minima.

	int MPI_Reduce(void* operand, void* result, int count, MPI_Datatype datatype, MPI_Op operator, int root, MPI_Comm comm)
IN:	
operand	(local) operand of operation
count	Length of the array for operand
datatype	data type of operand
operator	binary operator
root	receiver of result
comm	communicator
OUT:	
result	Result of operation

Reduction Routines

- All processes in a group have to start `MPI_Reduce`.
- The process will only continue after finishing the reduction routine. The parameters `count`, `datatype`, `perator` and `root` must be identical on all processes.
- Why do all processes have an entry for the result? The function call is the same on all processes. Use a 'dummy parameter' for all non-root processes.
- Example

```
MPI_Reduce(&integral, &total, 1, MPI_FLOAT,
       MPI_SUM, 0, MPI_COMM_WORLD);
```

Binary Operations

The binary operation is fixed by the choice of `operator`. The data type of `operator` is `MPI_Op`.

MPI - data type	Meaning
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Procukt
MPI_LAND	Logical and
MPI_BAND	Bit-wise and
MPI_LOR	Logical or
MPI_BOR	Bit-wise or
MPI_LXOR	Logical exclusive or
MPI_BXOR	Bit-wise exclusive or
MPI_MAXLOC	Maximum and Location of Maximums
MPI_MINLOC	Minimum and Location of Minimums

Reduction Routines

Very often the result of the reduction routine is required on all processes. Then, you can use

	int MPI_Allreduce(void* operand, void* result, int count, MPI_DATATYPE datatype, MPI_Op operator, MPI_Comm comm)
IN:	
operand	(local) operand of operation
count	Length of array for operand
datatype	data type of Operand
operator	binary Operator
comm	communicator
OUT:	
result	result of binary operation

In contrast to `MPI_Reduce` we do not need have an entry for `root`.

Barrier Function

All processes can be synchronized using `MPI_Barrier`.

int MPI_Barrier(MPI_Comm comm)	
IN:	
comm	communicator
OUT:	

- The call is started on all processes.
- All processes wait until the last process has called the function.
- This command is especially used for time stopping issues.

Scatter and Gather Routines

- Now we consider functions for distributing ('*scattering*') and collecting ('*gathering*') data.
- Example: $A \cdot \mathbf{b}$ with $A \in \mathbb{R}^{n \times m}$ and $\mathbf{b} \in \mathbb{R}^m$.
  This can be rewritten as

$$A\mathbf{b} = \begin{pmatrix} (a_1, \mathbf{b}) \\ (a_2, \mathbf{b}) \\ \vdots \\ (a_n, \mathbf{b}) \end{pmatrix}$$

  using the rows a_i of A.
  - Scatter the matrix A.
  - Broadcast b to all processes.
  - Compute (a_i, b) locally.
  - Gather results and send it to process 0.

Gather data

	int MPI_Gather (void* send_data, int send_count, MPI_Datatype send_type, void* recv_data, int recv_count, int recv_type, int root, MPI_Comm comm)
IN:	
send_data	sent data
send_count	array length of sent data
send_type	data type of sent data
recv_count	length of received data from each process
recv_type	data type of received data
root	process number of receiver
comm	communicator
OUT:	
recv_data	received message

Gather data

- All processes send data `send_data` of data type `send_type` to process `root`.
- The data will be combined using the sequence given by the process numbers.The result `recv_data` with data type `recv_type` is returned to `root`.
- Normally, `recv_type` and `send_type` resp. `recv_count` and `send_count` are equal.
- The total size of the received message `recv_data` is not a parameter.
- Example

```
MPI_Gather(local, n/p, MPI_FLOAT,
           global,n/p, MPI_FLOAT,
           0, MPI_COMM_WORLD);
```

Scatter data

	int MPI_Scatter(void* send_data, int send_count, MPI_Datatype send_type, void* recv_data, int recv_count, MPI_Datatype recv_type, int root, MPI_Comm comm)
IN:	
send_data	data
send_count	Length of split data
send_type	data type of sent data
recv_count	length of received message
recv_type	data type of received message
root	process number of root
comm	communicator
OUT:	
recv_data	received data