Jacobi algorithm

In numerical linear algebra, the Jacobi method is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations. Let Ax=B, be a square system of n linear equations, where:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Then A can be decomposed into a diagonal component D, a lower triangular part L and an upper triangular part U:

A = D + L + U

Where D = $\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}$

And L + U = $\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}$.

The solution is then obtained iteratively via

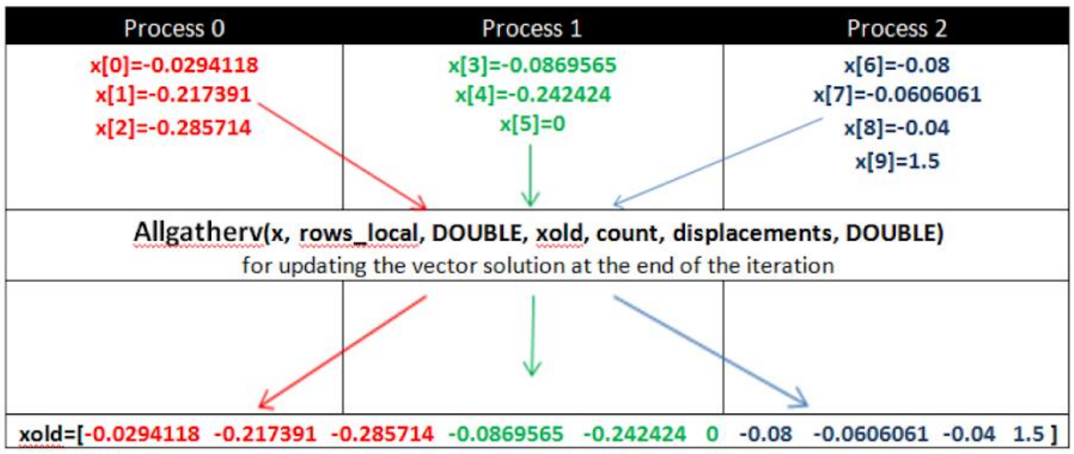
$$x^{(k+1)} = D^{-1}(b - (L + U)\, x^k$$

where x^k is the k^{th} approximation of x.

Hence the formula in terms of its elements can be given as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j^{(k)}\right), \quad i=1,2,\ldots,n.$$

The parallel algorithm for Jacobi


Process 0
Process 1
Process 2
x[0]=-0.0294118
x[1]=-0.217391
x[2]=-0.285714
x[3]=-0.0869565
x[4]=-0.242424
x[5]=0
x[6]=-0.08
x[7]=-0.0606061
x[8]=-0.04
x[9]=1.5
Allgatherv(x, rows_local, DOUBLE, xold, count, displacements, DOUBLE)
for updating the vector solution at the end of the iteration
xold=[-0.0294118 -0.217391 -0.285714 -0.0869565 -0.242424 0 -0.08 -0.0606061 -0.04 1.5]

This method assumes
we have all the input values of x in the previous iteration (k). But, usually all the x values are not given. Make an initial prepossession for x and to generate another group of solutions for x from the equation (1), which in fact
will represent the input values for the next iteration (k+1). After we have found the group of x values for the previous iteration we continue generating new groups again and again until we arrive at an acceptable solution. Jacobi puts borders between iterations; values of the vector-solution x are calculated only from the vector-solution of the previous iteration.

Based on Eq (1), we partition the problem as

D$_1$: sum$_1$=0

D$_2$: sum$_2$=j=2naijxj

D$_3$: x$_1$ = (-sum$_1$ - sum$_2$ + b$_1$)/A$_{11}$

D$_4$: sum$_1$ = a$_{11}$x$_1$

D$_5$: sum$_3$=j=3naijxj

D$_6$: x$_1$ = (-sum$_1$ - sum$_2$ + b$_2$)/A$_{22}$

D$_7$: sum$_1$ = a$_{11}$x$_1$+ a$_{12}$x$_2$

Likewise, we can decompose. On observation you will find that there are independent sub problems like 1, 2, 4, 5, 7, 8, etc. and dependent like 3 depends on 1 and 2, 6 depends on 4 and 5, and so on. A dependency graph can be constructed and strategy for parallelism can be made.

Process 0: sends the first three components of the vector solution x(0,1,2)

Process 1: sends the next coming three components of the vector solution x(3,4,5)

```
Repeat for k=0 to maxit
Initialize error_sum_local,sum1,sum2 ← 0.0
Repeat for j=0 to i_global
sum1 = sum1 + A[i][j]*xold[j]
Repeat for j=i_global+1 to N
sum2 = sum2 + A[i][j]*xold[j]
x[i] = (-sum1 - sum2 + b[i])/A[i][i_global]
error_sum_local += (x[i]-xold[i_global])*(x[i]-xold[i_global])
```

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

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

The communication cost is:

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