

UNIVERSIDADE DE SÃO PAULO  
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE  
COMPUTAÇÃO

SSC0143  
PROGRAMAÇÃO CONCORRENTE - TURMA B

---

**Jacobi-Richardson**

---

PROFESSOR DR. JULIO ESTRELLA

*Grupo 06:*

Bruno Junqueira Adami

Lucas Junqueira Adami

Lucas Lobosque

*Números USP:*

6878762

6792496

6792645

2 de junho de 2012

## 1 Introdução

Sistemas lineares são descritos conforme a Equação 1. A matriz  $A$  do sistema linear pode ser decomposta na forma descrita pela Equação 2 e ser reduzida conforme a Equação 3.

$$Ax = b \quad (1)$$

$$A = L + D + R \quad (2)$$

$$A^* = L^* + I + R^* \quad (3)$$

Usando essas decomposições, é possível apresentar o processo iterativo chamado de método de Jacobi-Richardson definido na Equação 4. O processo de parada do método é definido na Equação 5. Para garantir a convergência, a matriz  $A$  deve ser estritamente diagonalmente dominante, satisfazendo o critério das linhas e colunas.

$$x^{(k+1)} = -(L^* + R^*)x^{(k)} + b^* \quad (4)$$

$$\frac{\|x^{(k)} - x^{(k-1)}\|_\infty}{\|x^{(k)}\|_\infty} < \epsilon \quad (5)$$

Apresentado os conceitos necessários, o objetivo do trabalho é a implementação do método de Jacobi-Richardson utilizando uma abordagem sequencial e uma paralela usando OpenMP e MPI. Ao final, há uma discussão dos resultados obtidos pelas duas abordagens.

## 2 A abordagem sequencial

A abordagem sequencial do método não tem segredos. Ela segue os passos do algoritmo fielmente. Primeiro, o programa carrega o arquivo de entrada. Logo após, ele aplica o método de Jacobi-Richardson. No final, o resultado da execução é mostrada de acordo com a especificação do trabalho.

## 3 A abordagem paralela

A abordagem paralela segue a linha do programa sequencial decompondo-o em partes paralelas. O método pode ser considerado como um algoritmo iterativo paralelo. Isto é, a cada iteração, cada  $x_i^{(k)}$  pode ser calculado independentemente das outras variáveis na mesma iteração. Portanto, é possível aplicar uma decomposição de dados, onde a cada passo, o vetor  $x$  é distribuído entre os nós para ser calculado, cada nó recebendo uma faixa de variáveis, conforme ilustra a Figura 1.

O fluxo de dados segue a lógica da Figura 2. O primeiro processo, o processo mestre é responsável pela administração de toda a execução. Ele carrega os arquivos de entrada, controla a cadeia de mensagens, envia aos outros processos pedaços do problema e verifica a condição de parada.

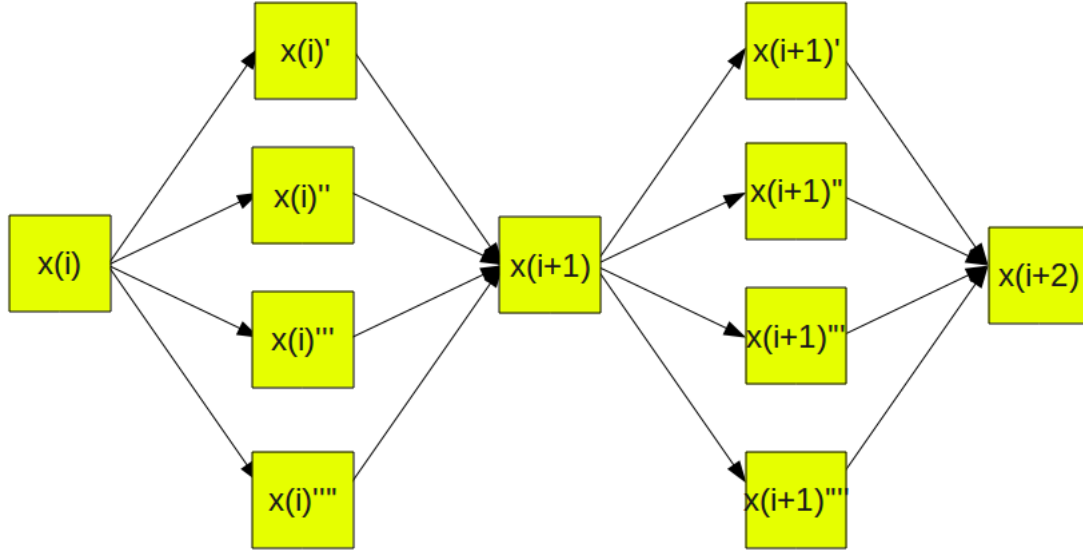


Figura 1: Decomposição de dados

Em relação à matriz  $A$ , o processo principal contém todos os seus valores, obtidos durante o carregamento do problema. Quanto aos processos secundários, o mestre envia a eles somente a faixa necessária para que eles possam executar o processo. Isto é feito através da chamada *scatter* do MPI. Essa abordagem é possível pelo fato de que para o cálculo de  $x_i$ , somente a linha  $i$  da matriz  $A$  é necessária.

O mesmo ocorre com o vetor  $b$ . Para calcular  $x_i$ , somente o valor da posição  $i$  do vetor  $b$  é necessário. Uma chamada à função *scatter* é feita e cada processo secundário somente trabalha com a faixa que lhe diz respeito.

Para o vetor  $X$ , é necessário a cada iteração ter todo o seu conteúdo completo. Portanto, um *broadcast* do MPI é feito pelo processo mestre a todos os processos antes que as iterações comecem ( $x^{(0)}$ ).

Ao término de cada iteração, os valores obtidos estão armazenados no vetor  $X$ , cada processo contendo sua faixa de valores. Para prosseguir com a próxima iteração, o *allgather* do MPI é chamado sobre o vetor  $X$ , garantindo a atualização desse vetor a todos os nós.

Os macros OpenMP foram utilizados com 4 threads. Há um macro para paralelizar o processo de normalização da matriz  $A$  antes do início do algoritmo. Há também um macro que paraleliza o cálculo de todos os  $x_i$  pertencentes à faixa do processo. Para cada  $x_i$ , outro macro é utilizado na multiplicação feita entre o vetor  $x$  e a linha  $i$  da matriz  $A$ . Um macro foi criado para checar a condição de parada do algoritmo (definido na Equação 5). Finalmente, um último macro é usado para obter a solução do problema.

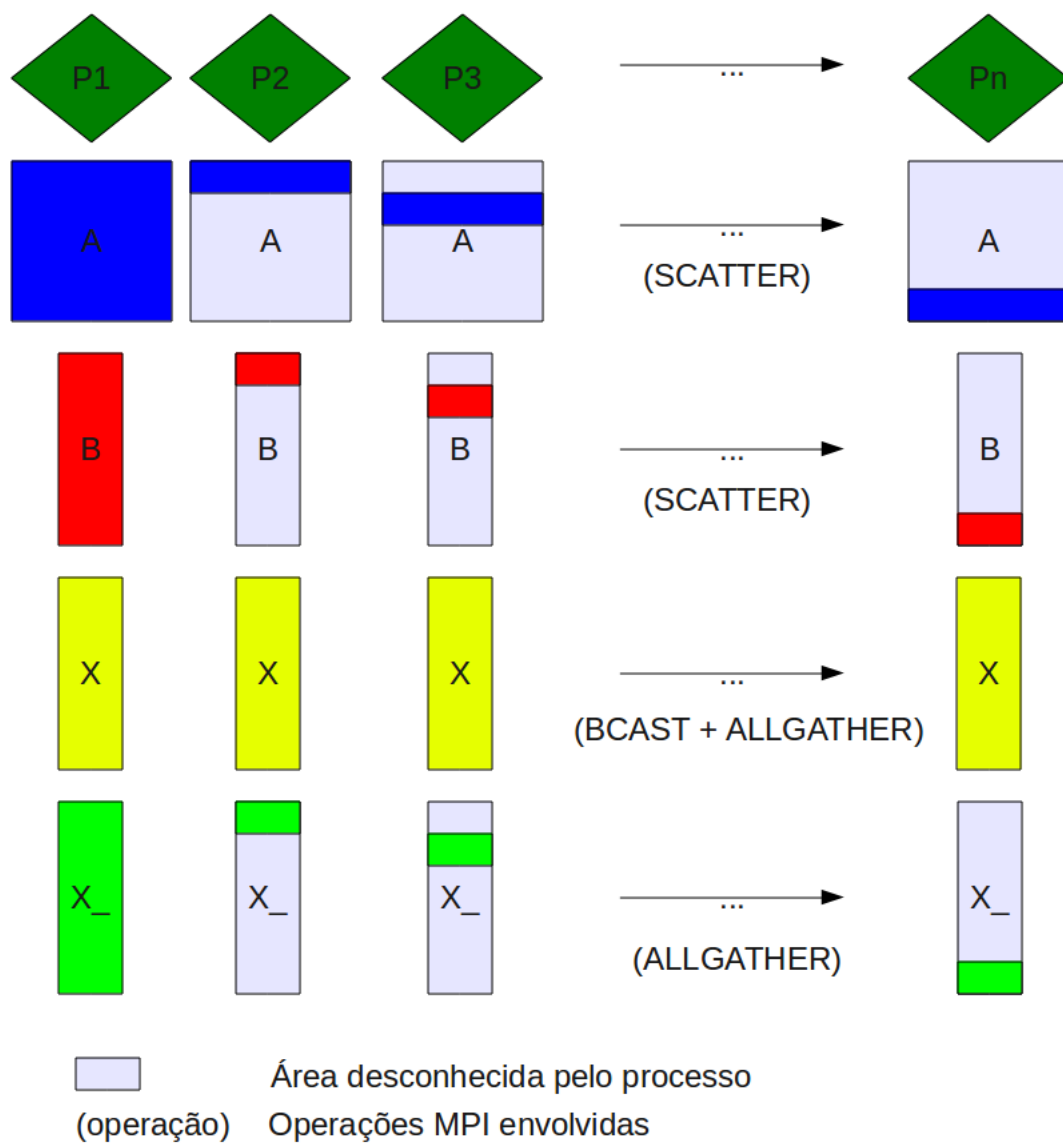


Figura 2: Fluxo de dados

## 4 Resultados e conclusões

Os experimentos consistem em 3 tipos de execução: sequencial, paralela com 5 nós e paralela com 10 nós. Cada configuração foi executada 7 vezes, seguindo uma ordem circular. As médias dos tempos de execução foram tiradas a partir dos resultados e a Tabela 1 mostra os speed-ups dos programas.

Segundo a tabela, o programa sequencial é melhor para matrizes pequenas, de tamanho até 500. Para a execução de 5 nós, o speed-up só foi bom quando esse tamanho chega a 1500. Já para 10 nós, é possível perceber que na matriz de tamanho 500, há uma competição entre os programas paralelo e sequencial. Para as matrizes maiores que esse tamanho, essa configuração obteve tempos muito bons.

Tamanho da matriz	Paralelo c/ 5 nós	Paralelo c/ 10 nós
250	38%	33%
500	75%	97%
1000	57%	142%
1500	117%	201%

Tabela 1: Speed-ups

A Figura 3 ilustra os tempos para o algoritmo resolver os problemas dos 3 tipos de execução. O eixo x representa o tamanho da matriz enquanto o eixo y diz respeito ao tempo de execução em segundos. Olhando para o gráfico, conclui-se que com o aumento das matrizes, a execução paralela é melhor que a sequencial, com a configuração de 10 nós sendo melhor que a de 5 nós.

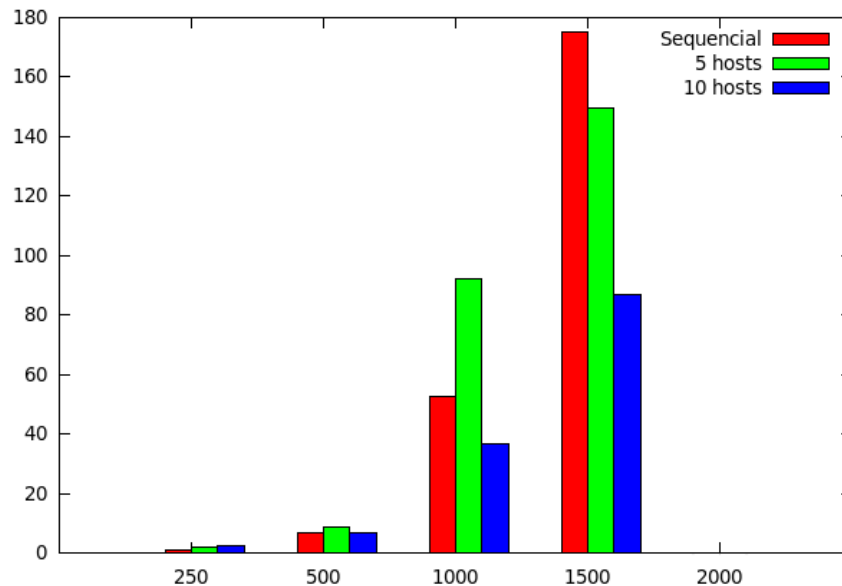


Figura 3: Gráfico de tempos

O fato de que para matrizes pequenas o programa sequencial é melhor que o paralelo pode ser explicado através da Equação 6. Isto é, o tempo que o algoritmo sequencial leva para calcular todo o vetor  $x$  é menor que o tempo que cada nó  $j$  leva para calcular sua faixa de  $x$  (começando em  $a_j$  e terminando em  $b_j$ ) mais o tempo gasto na propagação dos resultados dos nós.

$$t(x) \leq t\left(\sum_{i=a_j}^{b_j} x_i\right) + t(propagacao) \quad \text{todo } j \quad (6)$$

## Referências

[Jacobi-Richardson] Franco, Neide Bertoldi; Cálculo Numérico - São Paulo: Pearson Prentice Hall, 2006

[OpenMP] <http://bisqwit.iki.fi/story/howto/openmp/>

[OpenMP] <http://openmp.org/wp/>

[MPI] <http://www.slac.stanford.edu/comp/unix/farm/mpi.html>

[MPI] <http://www.eecis.udel.edu/~saunders/courses/372/01f/manual/manual.html>