UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação - ICMC-USP

Fabio Alves Martins Pereira (NUSP 7987435)

Naylor Garcia Bachiega (NUSP 5567669)

Trabalho 1: algoritmo sequencial e paralelo utilizando o método de Jacobi-Richardson

SÃO CARLOS

2015

Índice

[Lista de Figuras III](#_Toc429241013)

[Lista de Tabelas IV](#_Toc429241014)

[1 Introdução 5](#_Toc429241015)

[1.1 Objetivos 6](#_Toc429241016)

[2 Método de Jacobi-Richarson 7](#_Toc429241017)

[2.1 Descrição do Método 7](#_Toc429241018)

[2.2 Algoritmo 8](#_Toc429241019)

[*3* *Pthreads* 9](#_Toc429241020)

[3.1 Funcionamento 9](#_Toc429241021)

[4 Desenvolvimento e Metodologia 11](#_Toc429241022)

[4.1 MAIN 11](#_Toc429241023)

[5 Resultados e Discussões 11](#_Toc429241024)

[Referências 12](#_Toc429241025)

# Lista de Figuras

[Figura 1 – Benchmark em um único computador (LAMMPS, 2015). 6](#_Toc429233920)

# Lista de Tabelas

Nenhuma entrada de índice de ilustrações foi encontrada.

# Introdução

A ciência da computação é uma área abrangente envolvendo vários aspectos nas mais variadas esferas do conhecimento. Ainda segundo Brookshear (2013):

A ciência da computação é uma disciplina que busca construir uma base científica para tópicos como projeto e programação de computadores, processamento de informação, soluções algorítmicas de problemas e o próprio processamento algorítmico.

Dentro dessa área de conhecimento existem os algoritmos, os quais são importantes para resolver problemas ou criar soluções para os mais diversos paradigmas computacionais. Eles são um conjunto de passos que definem como uma ou mais tarefas serão realizadas (BROOKSHEAR, 2013).

Como o surgimento dos computadores e os algoritmos, o tempo de processamento das tarefas foi reduzido substancialmente em processadores de um núcleo. Com o acoplamento de mais núcleos no processador, algoritmos que dividem suas tarefas entre esses núcleos, tendem a ter um melhor desempenho, de acordo com o tipo de dados e sua possibilidade de paralelização (YANO, 2010).

Sendo assim, é importante que o algoritmo desenvolvido avalie todas as possibilidades de paralelização, para extrair um melhor tempo de execução. Atualmente, existem diversas linguagens de programação e bibliotecas que fornecem ferramentas para paralelização, entre elas pode-se citar *Pthreads*[[1]](#footnote-1), *OpenMP* [[2]](#footnote-2)e *MPI* [[3]](#footnote-3)(SATO, GUARDIA, 2013)

Além da paralelização de processos na *CPU*, é possível enviar trabalho para a *GPU* através de *CUDA*, que é uma plataforma de computação paralela e um modelo de programação desenvolvido pela NVIDIA. Ela permite aumentos significativos de desempenho computacional ao aproveitar a potência da Unidade de Processamento Gráfico (GPU) (NVIDIA, 2015).

Conforme pode ser observado na Figura 1, há um aumento significativo no tempo de execução para algoritmos que utilizam processamento paralelo, tanto para trabalhos enviados para a *CPU* quanto para a *GPU*. Porém na *GPU*, o tempo de resposta foi menor, pelo desempenho da unidade.

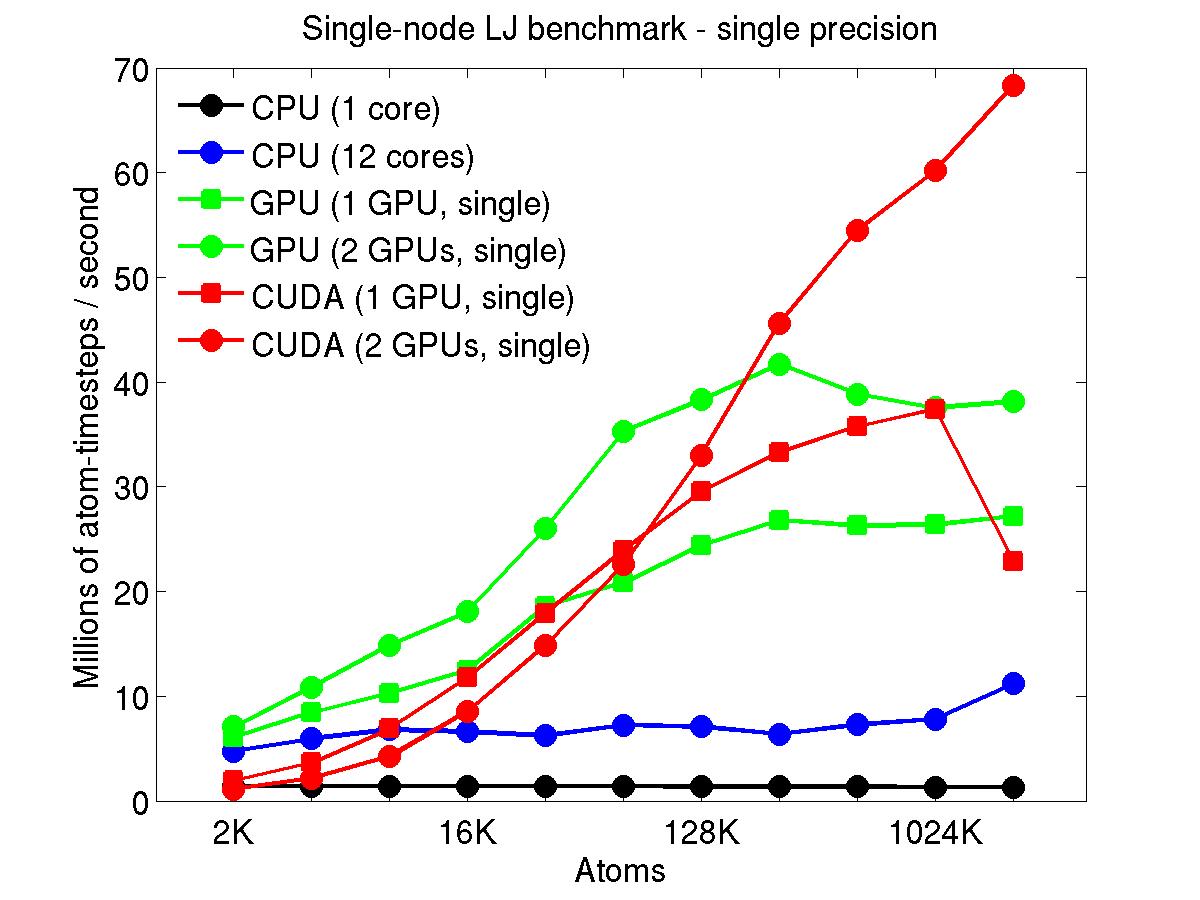


Figura 1 – Benchmark em um único computador (LAMMPS, 2015).

## Objetivos

Tendo em vista os benefícios da paralelização de algoritmos, esse trabalho mostra o desenvolvimento de dois algoritmos para o processamento de matrizes lineares, utilizando o método de Jacobi-Richarson. Após o desenvolvimento, os resultados serão demonstrados através de gráficos e tabelas.

# Método de Jacobi-Richarson

Métodos numéricos para solução de sistemas de equações lineares são divididos principalmente em dois grupos (YONA, 2010):

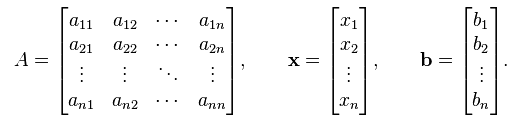
• Métodos Exatos: são aqueles que forneceriam a solução exata, se não fossem os erros de arredondamento, com um número finito de operações.

• Métodos Iterativos: são aqueles que permitem obter a solução de um sistema com uma dada precisão através de um processo infinito convergente e que possui um erro de truncamento.

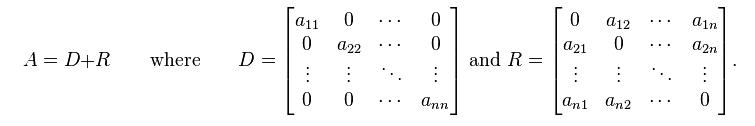
O método iterativo de Jacobi-Richardson se baseia inicialmente numa verificação de convergência, ou seja, verifica-se se a matriz é estritamente diagonalmente dominante. Se ela atender o critério de convergência, define-se a margem de erro máximo (truncamento) que o resultado terá. Assim, o método inicia-se através da solução inicial (no caso de um sistema linear, zero é uma das soluções possíveis), em seguida, calcula-se o resultado e ocorre a substituição no vetor resultado na próxima iteração, e assim em diante até que se tenha a solução que atenda a margem de erro (YONA, 2010).

## Descrição do Método

Dado um sistema quadrado de equações lineares n: , onde:



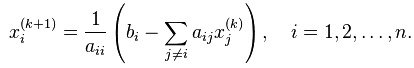
Então, A pode ser decomposto numa diagonal D, e o restante de R:



A solução é, então, obtida iterativamente:



Onde \mathbf{x}^{(k)} é a aproximação de ordem *k* ou iteração de \mathbf{x} e \mathbf{x}^{(k+1)} é o próximo ou a iteração *k* + 1 de \mathbf{x}. A fórmula base é:



## Algoritmo

Para o trabalho, foi utilizado o algoritmo básico, conforme pode ser visualizado:

Inicialmente, é estimado um valor inicial para x^{(0)} para a solução  k = 0 

enquanto erro não é alcançado

para i := 1 até n faça

 \sigma = 0 

para j := 1 até n faça

se j ≠ i então

 \sigma  = \sigma  + a_{ij} x_j^{(k)} 

fim se

fim (j-loop)

  x_i^{(k+1)}  = {{\left( {b_i  - \sigma } \right)} \over {a_{ii} }} 

fim (i-loop)

k = k + 1

repete (enquanto erro não é alcançado)

# *Pthreads*

*Pthread* é uma maneira simples e eficaz de criar uma aplicação paralelizada. Quando uma *thread* é criada usando **pthread\_create**, a *thread* original e a nova compartilham da mesma base de código e a mesma memória - é como fazer duas chamadas de função ao mesmo tempo (TIM, 2010).

Todos os programas em C usando *pthreads* precisam incluir o arquivo de *header* pthread.h (#include <pthread.h>). No sistema operacional Ubuntu Desktop 15.04 é necessário instalar um pacote através do *apt-get* (ferramenta de instalação e atualização de pacotes):

* sudo apt-get install build-essential
* sudo apt-get install libpthread-stubs0-dev

## Funcionamento

Há quatro etapas para a criação de um programa básico utilizando Pthreads (TIM, 2010):

* Definir a variável thread de referência: a variável do tipo pthread\_t é uma forma de referenciar *threads*. É preciso haver uma variável pthread\_t na existência de cada segmento que está sendo criado. Algo como pthread\_t thread0.
* Criar um ponto de entrada para a *thread*: ao criar a *thread* usando *pthreads*, é necessário apontá-la para uma função para ela iniciar a execução. A função deve retornar void \* e tomar um único argumento void \*. Por exemplo, para que a função pegue um argumento inteiro, é necessário passar o endereço do inteiro. Um exemplo de função seria: void \* my\_entry\_function (void \* param);
* Criar a thread: uma vez que a variável pthread\_t foi definida e a função de ponto de entrada criada, deve criar o segmento usando pthread\_create. Este método tem quatro argumentos: um ponteiro para a variável pthread\_t, os atributos extras, um ponteiro para a função a ser chamada e o ponteiro que está sendo passado como argumento para a função. Esta chamada será algo parecido com pthread\_create (&thread0, NULL, my\_entry\_function, e &parameter);

# Desenvolvimento e Metodologia

Esse capítulo aborda o desenvolvimento dos algoritmos e a metodologia utilizada. Para esse trabalho, foi utilizada linguagem C e Code::Blocks como interface de desenvolvimento. Como o trabalho foi em grupo, o GitHub foi utilizado para compartilhar o código e controlar o versionamento.

## MAIN

O programa principal foi criado para listar as matrizes disponíveis no diretório “matrizes/” e disponibilizar a opção de escolha para o usuário. O usuário, após escolher a matriz, pode especificar se a execução será sequencial ou paralela. No caso da execução paralela, ainda é possível digitar a quantidade desejada de *threads*.

# Resultados e Discussões

# Referências

ABREU, Luciano. **Calculo Numérico - Método de Jacobi-Richardson**. Disponível em: <https://www.youtube.com/watch?v=2AORqeCrQEc>. Acesso em: 30 ago. 2015

ALVES, Carlos J. S. **Métodos Iterativos para Sistemas Lineares**. Disponível em: < http://www.math.ist.utl.pt/~calves/cursos/SisLin-Iter.htm>. Acesso em: 30 ago. 2015

ANDRETTA, Marina. **Sistemas lineares - Método Iterativo de Jacobi-Richardson**. São Carlos, 2008.

BALBO, Antonio Roberto. **Métodos iterativos de solução de sistemas lineares**. Disponível em: <http://wwwp.fc.unesp.br/~arbalbo/Iniciacao\_Cientifica/sistemaslineares/teoria/jacobi\_richardson.pdf>. Acesso em: 26 ago. 2015.

BROOKSHEAR, J. Glenn. **Ciência da computação: uma visão abrangente**. 11ª ed. Porto Alegre: Bookman, 2013.

LAMMPS, Molecular Dynamics Simulator. ***GPU and USER-CUDA package benchmarks on Desktop system with Fermi GPUs***. Disponível em: <https://computing.llnl.gov/tutorials/pthreads/#Pthread>. Acesso em: 05 set. 2015

LLNL. ***POSIX Threads Programming***. Disponível em: <http://lammps.sandia.gov/bench.html>. Acesso em: 02 set. 2015

NVIDIA. **O que é *CUDA*?**. Disponível em: <http://www.nvidia.com.br/object/cuda\_home\_new\_br.html>. Acesso em: 02 set. 2015

SATO, Liria Matsumoto; GUARDIA, Hélio Crestana. **Programando para múltiplos processadores: *Pthreads, OpenMP* e *MPI*.** Disponível em: <http://erad.dc.ufscar.br/mc/eradsp2013-multiproc-3.pdf>. Acesso em: 02 set. 2015.

SOUZA, Marcone Jamilson Freitas. **Sistemas Lineares**. Disponível em: <http://www.decom.ufop.br/marcone/Disciplinas/CalculoNumerico/Sistemas.pdf>. Acesso em: 03 set. 2015.

OPENMP. ***What is OpenMP*?**. Disponível em: <http://www.openmp.org/mp-documents/paper/node3.html>. Acesso em: 02 set. 2015.

OPEN-MPI. ***General information about the Open MPI Project***. Disponível em: <https://www.open-mpi.org/faq/?category=general>. Acesso em: 05 set. 2015.

RICARDO, Luis; PAULINO, Diogo; CARVALHO, Paulo. **Paralelização do algoritmo do Método Jacobi**. Disponível em: <https://drive.google.com/file/d/0B639uUhZ62fgV012UkZvSDdZeWs/view>. Acesso em: 30 ago. 2015.

TIM, C. ***Pthreads in C – a minimal working example***. Disponível em: <http://timmurphy.org/2010/05/04/pthreads-in-c-a-minimal-working-example/>. Acesso em: 30 ago. 2015.

YANO, Luis Gustavo Abe. **Avaliação e comparação de desempenho utilizando tecnologia CUDA**. São José do Rio Preto: 2010.

1. *Pthreads* são definidos como um conjunto de tipos de linguagem de programação C e chamadas de procedimento (LLNL, 2015). [↑](#footnote-ref-1)
2. *OpenMP* é um conjunto de diretivas do compilador e bibliotecas chamadas através de rotinas para expressar o paralelismo de memória compartilhada (OPENMP, 2015) [↑](#footnote-ref-2)
3. *MPI* é uma *API* padronizada normalmente utilizada para computação paralela e/ou distribuída. [↑](#footnote-ref-3)