Trabalho 2 - Introdução à Computação Científica

Fernanda Yukari Kawasaki
Departamento de Informática Universidade Federal do Paraná
Curitiba, PR, Brasil
fyk18@inf.ufpr.br

Tiago Serique Valadares
Departamento de Informática Universidade Federal do Paraná
Curitiba, PR, Brasil
tsv19@inf.ufpr.br

1 INTRODUÇÃO

Durante o período letivo do terceiro Ensino Remoto Emergencial, a disciplina de Introdução à Computação Científica (CI1064) foi ministrada pelo Professor Doutor Guilherme Derenievicz. Ao longo desta, os alunos foram capazes de aprender algoritmos frequentemente utilizados para a resolução de problemas matemáticos, como o método da eliminação de Gauss, implementado na primeira parte do trabalho. Em seguida, foram abordadas diferentes formas de otimização de código em arquiteturas x64, sendo que algumas foram realizadas no trabalho. Este artigo visa explicar as otimizações realizadas, e traçar um comparativo entre as duas versões, utilizando-se de dados obtidos com a ferramenta LIKWID.

2 ARQUITETURA DO COMPUTADOR

O computador i30 do Departamento de Informática foi utilizado para a execução de todos os experimentos apresentados nas seções seguintes. Utilizando o comando "likwid-topology -g -c", temos a seguinte descrição da máquina.

CPU
CPU name: Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz
CPU type: Intel Coffeelake processor
CPU stepping: 9
Hardware Thread Topology
Sockets: 1
Cores per socket: 4
Threads per core: 1
Cache Topology
Level: 1
Size: 32 kB
Type: Data cache
Associativity: 8
Number of sets: 64
Cache line size: 64
Cache type: Non Inclusive
Shared by threads: 1
Cache groups: (0) (1) (2) (3)
Level: 2
Size: 256 kB
Type: Unified cache
Associativity: 4
Number of sets: 1024
Cache line size: 64
Cache type: Non Inclusive
Shared by threads: 1
Cache groups: (0) (1) (2) (3)
Level: 3
Size: 6 MB
Type: Unified cache
Associativity: 12
Number of sets: 8192
Cache line size: 64
Cache type: Inclusive
Shared by threads: 4
Cache groups: (0 1 2 3)

3 REPRODUÇÃO DOS EXPERIMENTOS

Para reproduzir os experimentos expostos, basta garantir que todos os requisitos sejam cumpridos e, depois, ir até o diretório “compare” e rodar o seguinte comando no terminal: "./compare.sh".

3.1 Requisitos

Utilizamos Python 3.8.10 para fazer a limpeza dos dados e plotar os gráficos. As bibliotecas de Python utilizadas são:

- kaleido==0.2.1
- pandas==1.3.4
- plotly==5.4.0

Caso o instalador de pacotes pip esteja disponível, é possível instalar todos os requisitos executando o seguinte comando: "pip install -r requirements.txt".

3.2 Estrutura do script

Todo o processo de geração de geração dos experimentos é englobado pelo script "compare.sh". Inicialmente, os códigos são compilados utilizando os respectivos makefiles, e a estrutura de diretórios é criada.

Depois, o script "broyden.sh"é executado, gerando as entradas para os programas. Para gerar os dados do LIKWID de cada dimensão, foi decidido que as entradas seriam divididas em diferentes arquivos, com base na dimensão. Os arquivos seguem o padrão "entradas/entrada_{dimensao}.txt".

Em seguida, um loop é realizado para gerar as análises com o LIKWID. Há também o programa em Python "data_strip.py", que é utilizado para buscar os dados dentro dos arquivos gerados pelo LIKWID, e depois os salva no diretório desempenho. É importante notar que o trabalho não otimizado não consegue ser executado para instâncias maiores ou iguais a 2000. Assim, rodamos somente o trabalho otimizado com todas as instâncias.

Após o loop, são gerados os gráficos com o programa "graph.py", e os outros arquivos são removidos. A frequência é retornada para a original.

4 OTIMIZAÇÕES PROPOSTAS

As otimizações efetuadas para este trabalho foram as seguintes:

(1) Estrutura de dados mais específica e eficiente:

Na segunda parte do trabalho, uma informação extra é fornecida: as entradas são sempre instâncias da Função de Broyden e, dessa forma, as matrizes jacobianas geradas são tri-diagonais. Assim, para armazenar os valores da jacobiana, foi possível utilizar apenas três vetores, dois de tamanho n-1 e um de tamanho n, o que totaliza 3n - 2 elementos. Já na versão não otimizada, uma matriz de n * n elementos era alocada.
Vale ressaltar que são utilizadas duas matrizes jacobianas: uma com as funções derivadas, e uma com os valores de fato. Além disso, para cada iteração do método de Newton, é preciso atualizar os valores.
Em ambos os casos, é possível encontrar a definição da estrutura no respectivo arquivo utils.h.

(2) Melhora no gauss e na retrossubstituição:
Agora que são usados três vetores para a armazenar a matriz de coeficientes do sistema linear que deve ser resolvido no método de Newton, a eliminação de Gauss e a retrossubstituição puderam ser melhoradas.
A melhora consiste no loop mais interno, usado para percorrer a linha inteira da matriz, não é mais necessário, o que representa uma boa melhora em relação ao trabalho anterior, no qual era feito o loop aninhado nos dois casos.

(3) Evitar alocação de tamanho de potências de 2:
Foi adicionada uma função que reconhece quando um número é potência de 2, dessa forma, todas as alocações de memória foram feitas evitando tais potências e, consequentemente, evitando a ocorrência de cache trashing.

(4) Loop Unrolling:
O loop unrolling diminui a quantidade de iterações em um laço, reduzindo a sobrecarga do loop e evitando os testes de parada, além de que, se as do loop não forem dependentes umas das outras, elas podem ser executadas em paralelo. A técnica foi utilizada para gerar a matriz jacobiana na versão otimizada.

(5) Alocar próximo de onde vai ser usado:
Usar a memória logo após ela ser alocada faz melhor aproveitamento dessa região na cache, evitando um futuro cache miss e a transferência entre níveis diferentes da hierarquia de cache.

(6) Remoção de cálculos e recursos desnecessários:
Com a troca das estruturas usadas, cálculos que resultaram em valores constantes ou nulos na versão não otimizada foram descartados. Também foram feitos ajustes e remoções em recursos que não possuíam necessidade, como o vetor termos_aux da versão não otimizada.

5 GRÁFICO DOS TESTES

Para valores acima de 1024 a versão não otimizada não funciona, então nos gráficos acima desse valor há apenas o resultado da versão otimizada.

??
??
??
??
??

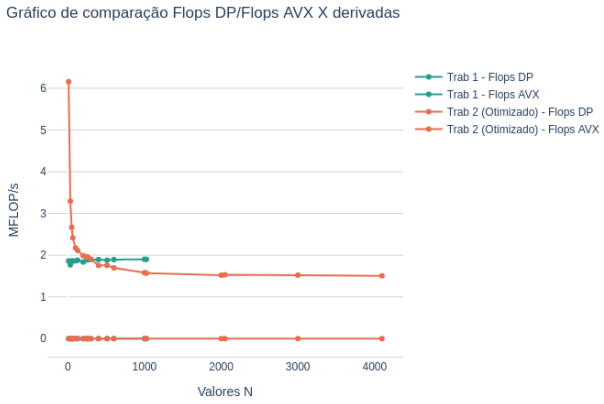


Figura 1

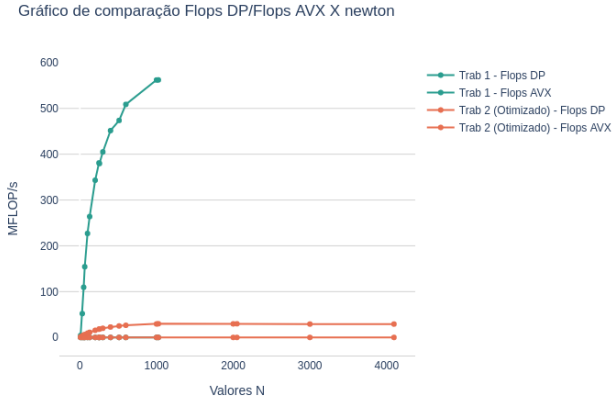


Figura 2

??
??
??
??
??
??
??
??
??
??
??

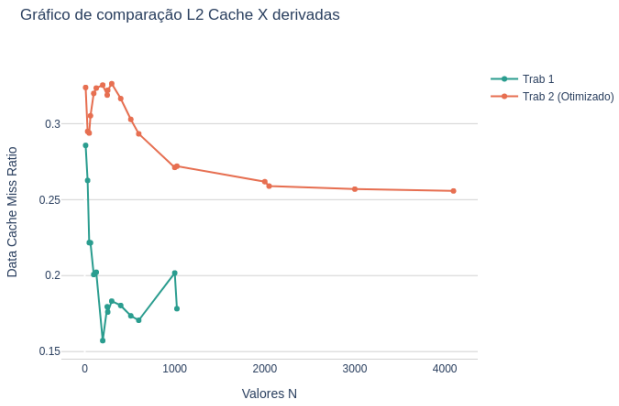


Figura 5

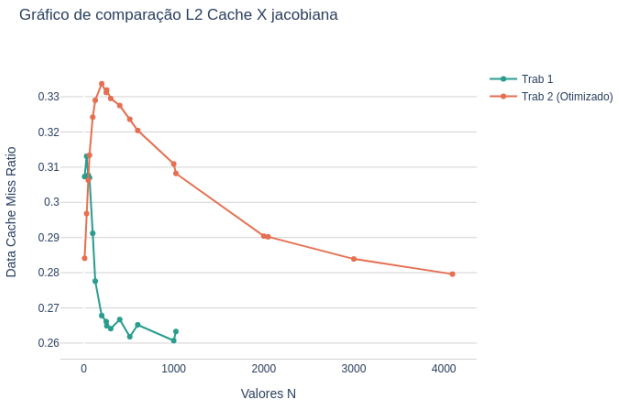


Figura 7

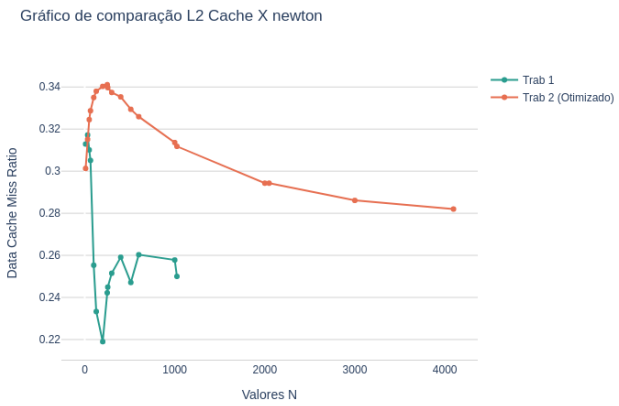
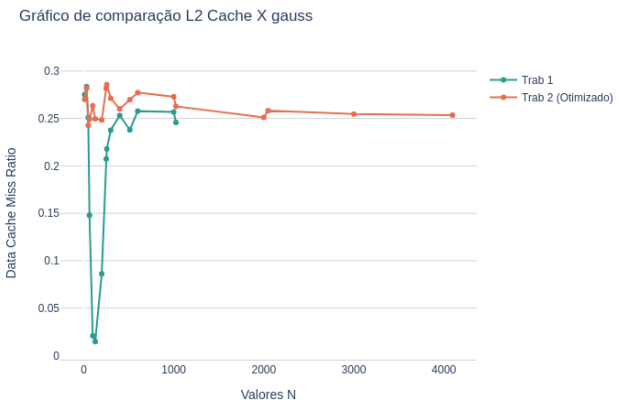


Figura 6



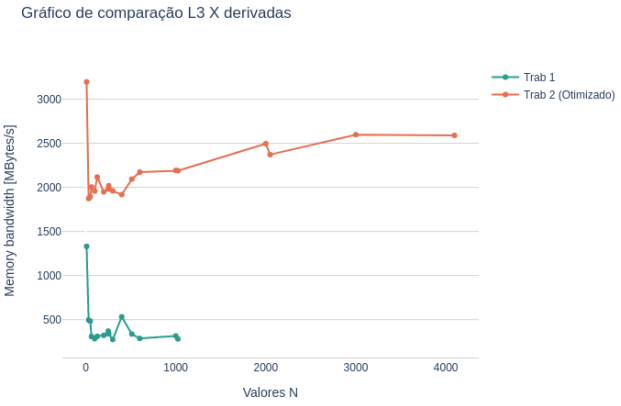


Figura 8

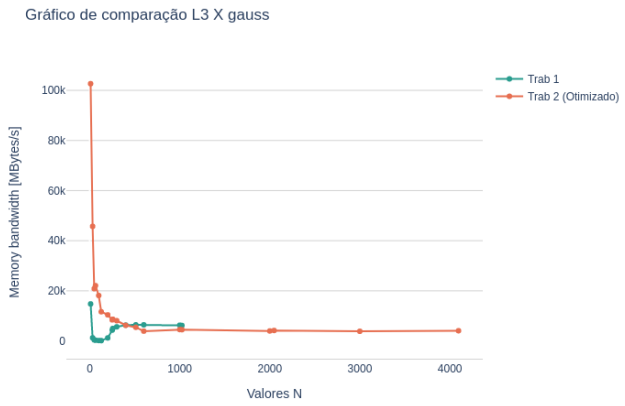


Figura 11

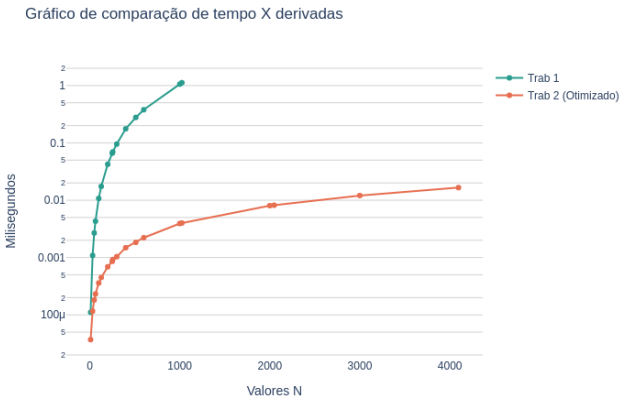


Figura 12

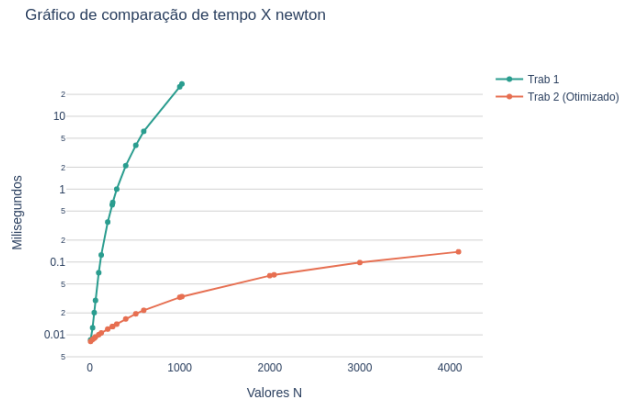


Figura 13

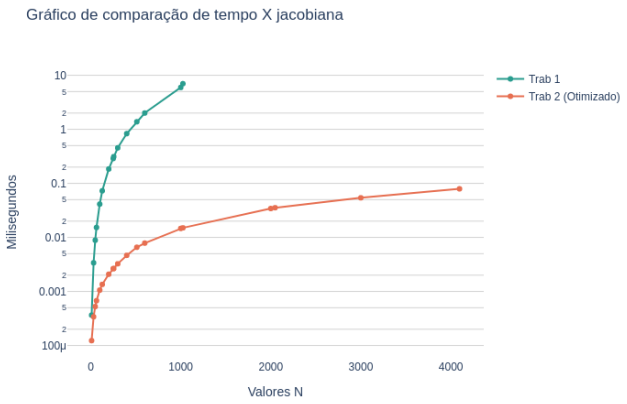


Figura 14

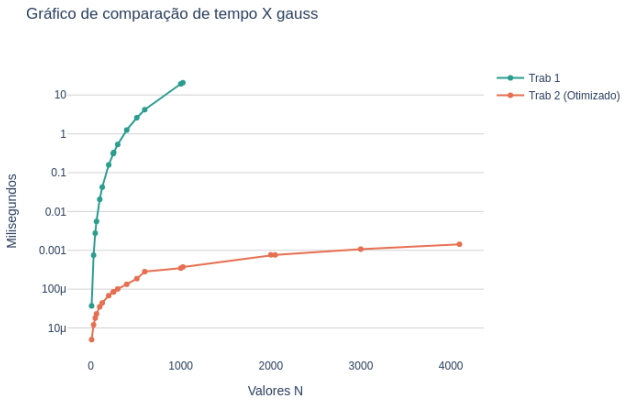


Figura 15