UNIVERSIDADE FEDERAL DO PARANÁ

CHRISTIAN DEBOVI PAIM OLIVEIRA
VINÍCIUS TEIXEIRA VIEIRA DOS SANTOS

OTIMIZAÇÃO DO CÁLCULO DA SOLUÇÃO E RESÍDUO DE UMA EQUAÇÃO DIFERENCIAL PARCIAL

CURITIBA
2018

SUMÁRIO

1 INTRODUÇÃO

Este trabalho consiste em apresentar as otimizações feitas nos códigos das funções Gauss_Seidel() e normaL2residuo() da primeiro trabalho da disciplina de ICC , que agem sobre uma Equação Diferencial Parcial (EDP) de segunda ordem. A equação em questão é dada por:

$$\kappa\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\lambda\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)+\eta u(x,y)=f(x,y), \quad (x,y)\in\Omega$$

onde:

- $\kappa=-1,\ \lambda=1,\ \eta=4\pi^2$
- $f(x,y)=4\pi^2[\sin(2\pi x)\sinh(\pi y)+\sin(2\pi(\pi-x))\sinh(\pi(\pi-y))]$

com o domínio definido no intervalo:

- $\Omega=[0,\pi]\times[0,\pi]$

e os valores de contorno definidos por:

- $u(x,0)=\sin(2\pi(\pi-x))\sinh(\pi^2)$
- $u(x,\pi)=\sin(2\pi x)\sinh(\pi^2)$
- $u(0,y)=u(\pi,y)=0$

O cálculo da solução é baseado no número de pontos no eixo x e y da equação, dados por nx e ny, respectivamente. O tamanho do sistema linear da equação é dado por n = nx*ny, que é gerado através da discretização pelo Método das Diferenças Finitas. A função Gauss_Seidel() calcula a solução do sistema linear pelo Método de Gauss Seidel e a função normaL2residuo() calcula o resíduo do sistema linear utilizando a solução calculada. O número de iterações para os testes dessas funções foi especificado como i = 10, com os resultados dos testes tendo seu valor médio entre dez iterações.

2 CARACTERÍSTICAS DO COMPUTADOR DE TESTE

O computador utilizado para a realização dos teste é um HP EliteDesk, com processador *Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz* do tipo *Intel Coffeelake* , que teve sua frequência de *clock* fixada para performance máxima em 3.40 GHz. As informações da hierarquia de memória, como as características de capacidade e modelo de mapeamento das *caches* foram obtidas através do *software* Likwid, utilizando o comando likwid-topology -c -g. A execução do comando está representada nas imagens abaixo.

```
--------------------------------------------------------------------------------
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
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
3               0               3               0               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 1 2 3 )
--------------------------------------------------------------------------------
********************************************************************************
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 )
--------------------------------------------------------------------------------
********************************************************************************
```

Figura 1 - Execução do comando *likiwid-topology -c -g* , primeira parte.

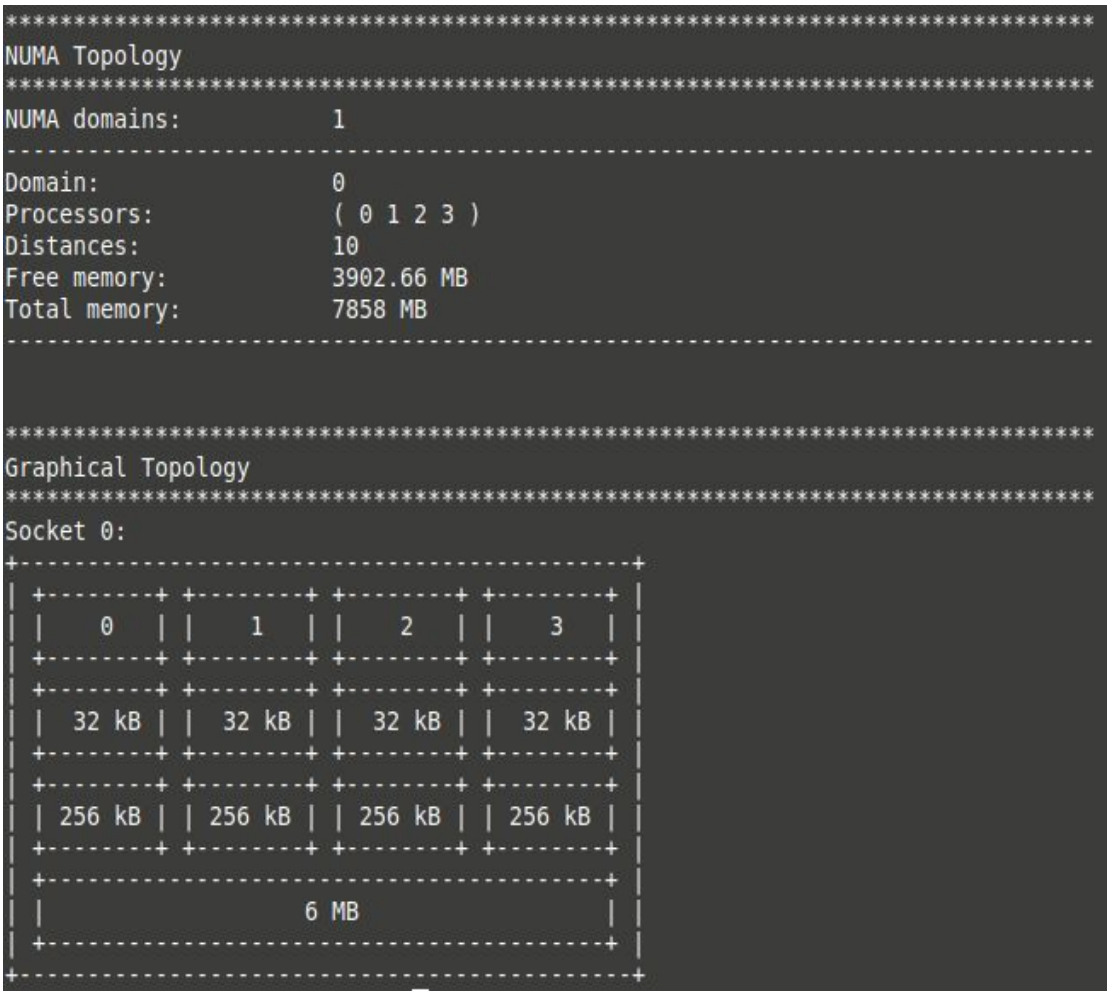

Figura 2 - Execução do comando *likiwid-topology -c -g* , segunda parte.

3 TESTES REALIZADOS

Os teste foram realizados pelo programa LIKWID (*Like I Knew What I'm Doing)*, sendo realizados apenas na terceira *core* do processador. Os resultados das medições foram divididos por dez (média em 10 iterações), com marcações que isolavam as funções Gauss_Seidel() e normaL2residuo(). O tempo de execução (RDTSC Runtime) na *core* especificada é sempre mostrado em todos os grupos de teste, sendo mostrado em escala logarítmica nos gŕaficos. Os grupos de teste utilizados no comando *likwid-perfctr* foram:

- **FLOPS_DP**: Número de Operações de Ponto Flutuante medidas em MFLOPS/s. Também mostra o número de operações de ponto flutuante utilizando registradores AVX (AVX DP FLOPS).
- **L2CACHE:** Mede a proporção de *cache misses* da cache L2 (*L2 cache miss ratio*).

- **L3:** Mede a vazão (ou largura de banda) de memória da cache L3 em Mbytes/s.

4 OTIMIZAÇÕES EFETUADAS

Em relação ao trabalho anterior, foram feitas algumas mudanças que serão discutidas nesta seção.

4.1 MUDANÇA DE LOCAL DAS CONDIÇÕES DE CONTORNO

Na primeira versão do trabalho, o Método de Gauss Seidel recebia um sistema linear cujos coeficientes que multiplicam condições de contorno nas equações eram isoladas para o lado direito da equação, ou seja, eram subtraídas do vetor b. Na nova versão, o tamanho do sistema linear foi aumentado para (nx+2)*(ny+2), de forma que as condições de contorno pudessem ser adicionadas estrategicamente ao vetor x, utilizando a função adicionada Contorno(x, nx, ny), que também inicializa parte do vetor x com zeros para a resolução da EDP.

4.1.2 Justificativa

A mudança no armazenamento dos contornos teve intuito de tirar os comandos *if* dentro dos laços do cálculo do Método de Gauss Seidel e do cálculo do resíduo da solução. Anteriormente, existiam quatro comandos *if* para evitar que ambas as funções citadas subtraírem os contornos do vetor b, o que já havia sido feito antes em sua inicialização. Na versão otimizada do trabalho com os contornos em x, esses comandos se tornaram desnecessários, já que o problema gerado pela inicialização de b foi solucionado. Assim, apesar de utilizar uma quantidade maior de memória, o programa ganhou um desempenho significativo em seu tempo médio, já que os possíveis ocorrências de *stalls* no processador pelos comandos de desvio foram solucionadas.

4.2 TENTATIVA DE UNROLL AND JAM NO LAÇO DO GAUSS-SEIDEL E DO RESÍDUO

Para nos beneficiarmos do acesso à memória para calcular Gauss-Seidel e o resíduo de nossos cálculos, fizemos uso da técnica *Unroll & Jam*. Porém, percebemos em nossos resultados que isso não necessariamente trouxe benefícios.

Foi realizado *loop unrolling* com fatores de 2 até 6, construindo gráficos que

comparam as versões com *unroll* com o primeiro trabalho e o segundo trabalho (com as condições de contorno em x) sem nenhum *unroll*. O fator dos *unrolls* foi baseado no tamanho da linha de cache , que tem 64 Bytes e pode armazenar 8 *doubles*, então foi visado manter um fator abaixo de 8.

Os resultados obtidos em relação ao número de pontos e com 10 iterações foram dados por:

Gráfico de final	**Descrição**
-trab1	execução do trabalho original (T1)
-trab2	execução do T2 com aprimoramentos e sem *Unroll*
-u2	execução do T2 com *Unroll* de 2 no resíduo e Gauss-Seidel
-u3	execução do T2 com *Unroll* de 2 no resíduo e de 3 em Gauss-Seidel
-u4	execução do T2 com *Unroll* de 2 no resíduo e de 4 em Gauss-Seidel
-u5	execução do T2 com *Unroll* de 2 no resíduo e de 5 em Gauss-Seidel
-u6	execução do T2 com *Unroll* de 2 no resíduo e de 6 em Gauss-Seidel

Tabela 1 - Legenda para os gráficos de teste

Tempos médios de execução:

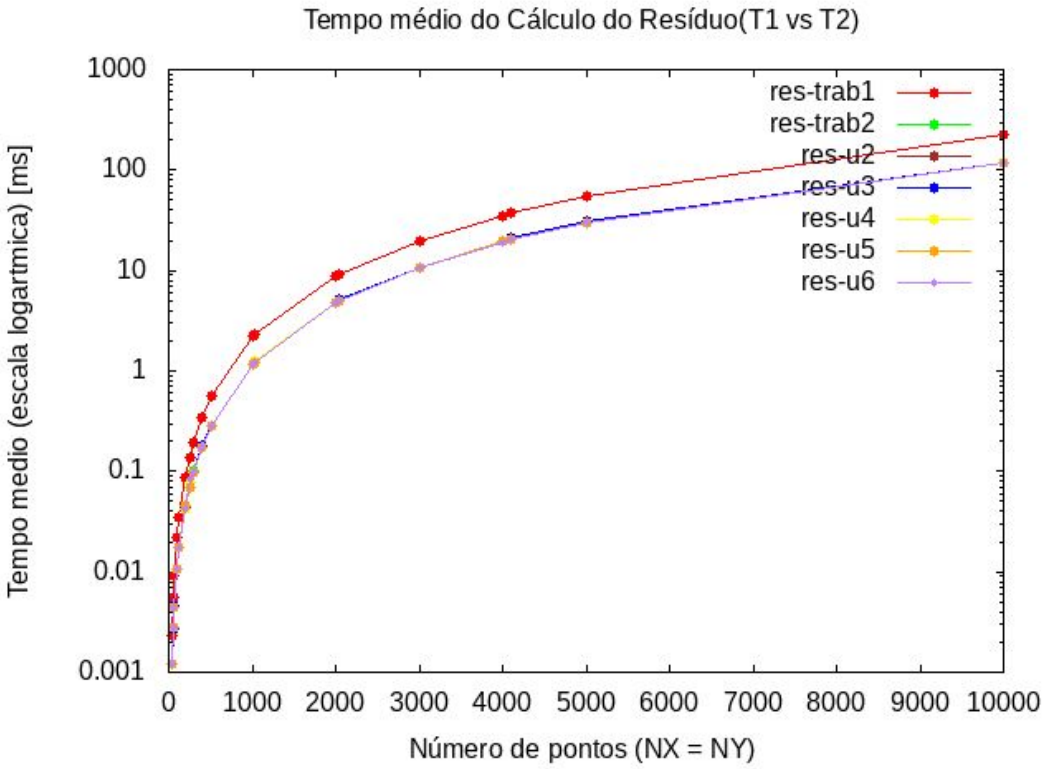

Gráfico 1 - Tempo médio do Método de Gauss Seidel com versões com *unroll*.

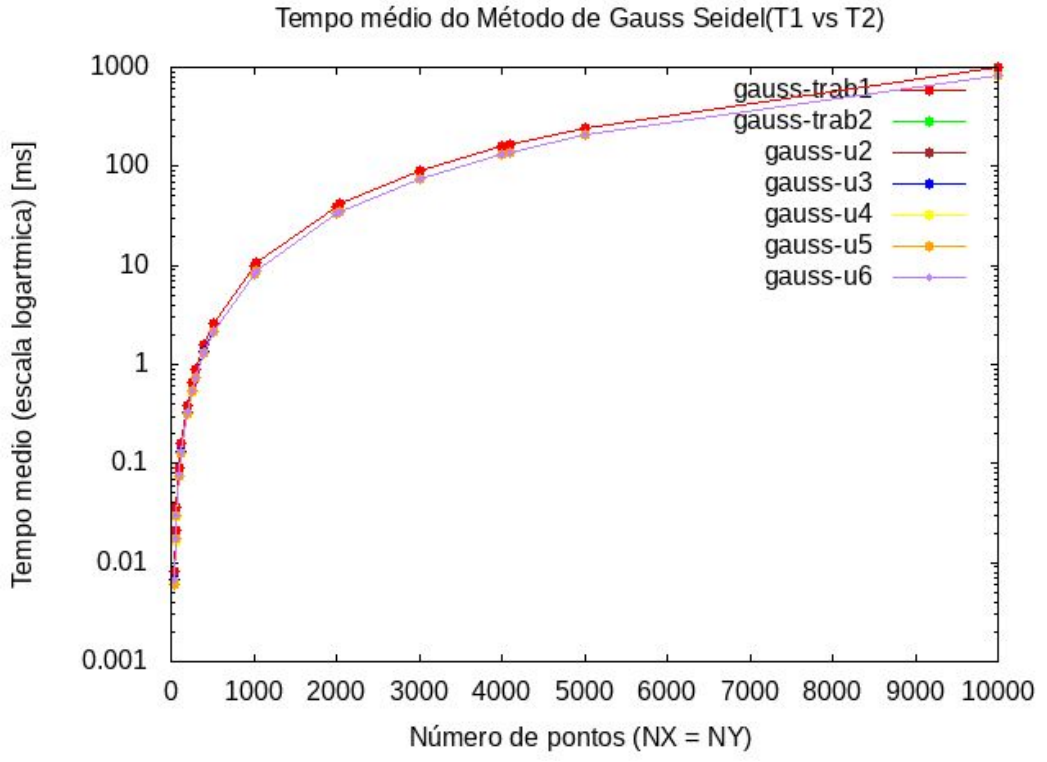

Gráfico 2 - Tempo médio do Cálculo do Reśiduol com versões com *unroll*.

Operações em ponto flutuante por segundo:

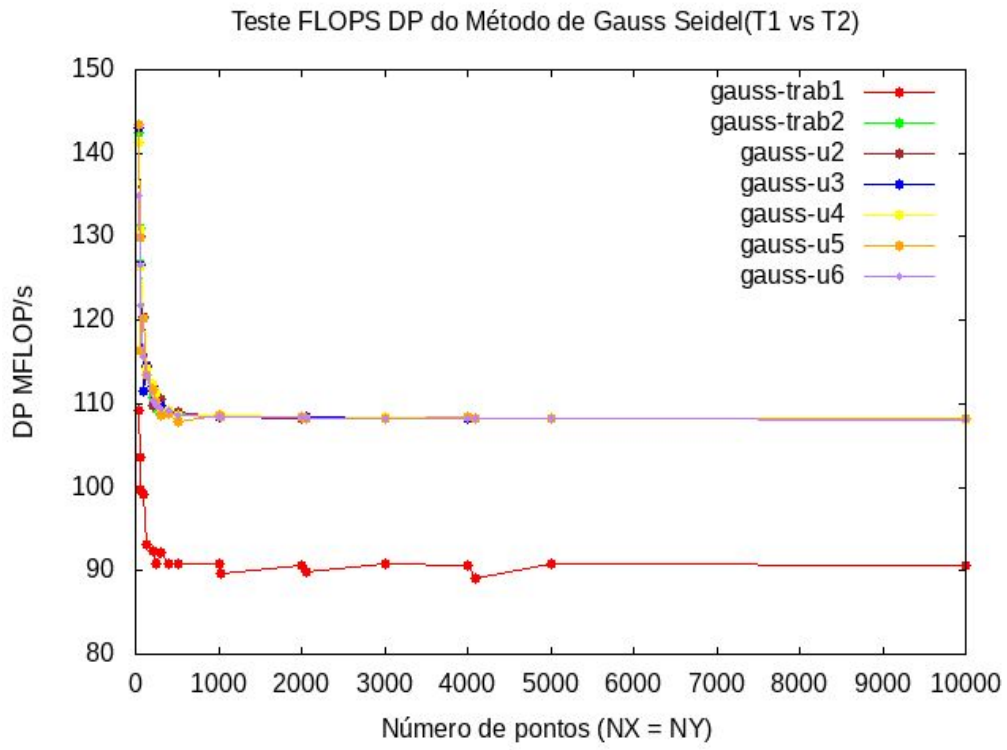

Gráfico 3 - FLOPS_DP do Método de Gauss Seidel com versões com *unroll*.

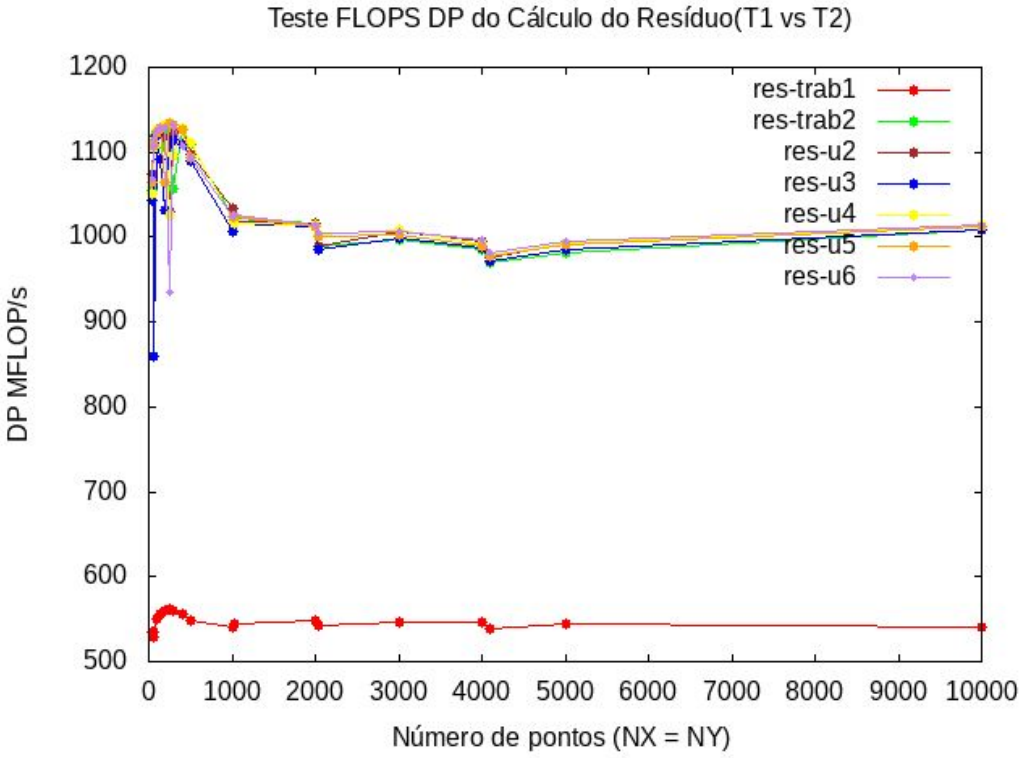

Gráfico 4 - Teste FLOPS_DP do Cálculo do Resíduo com versões com *unroll*.

Miss ratio da memória L2:

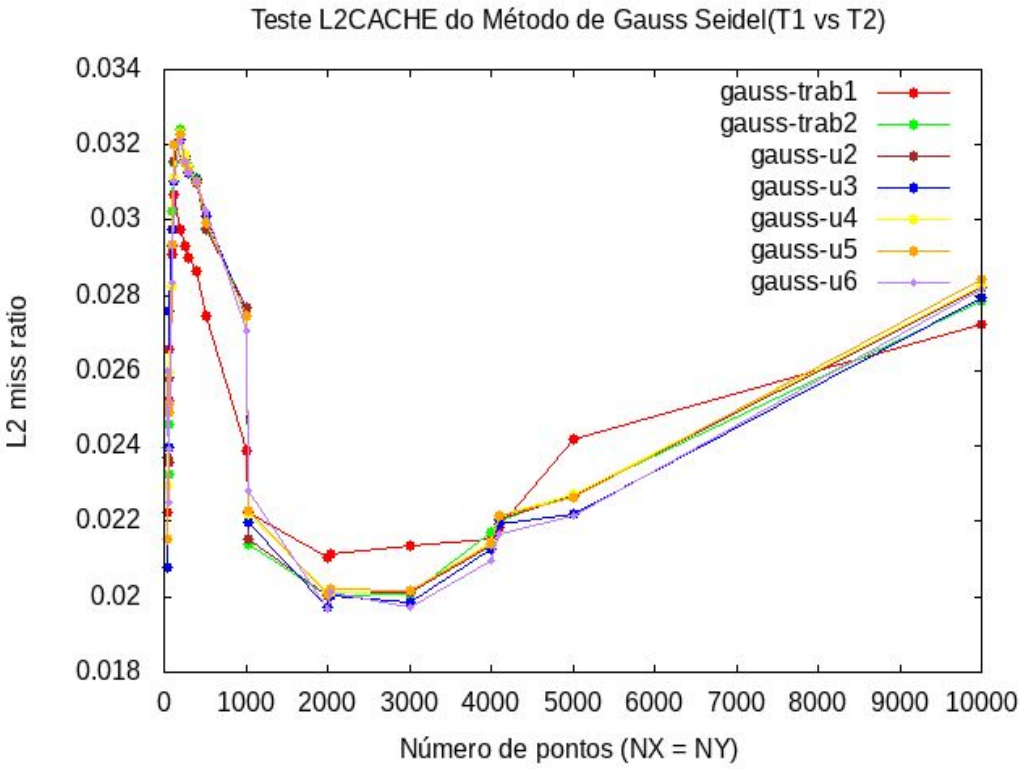

Gráfico 5 - Teste L2CACHE do Método de Gauss Seidel com versões com *unroll*.

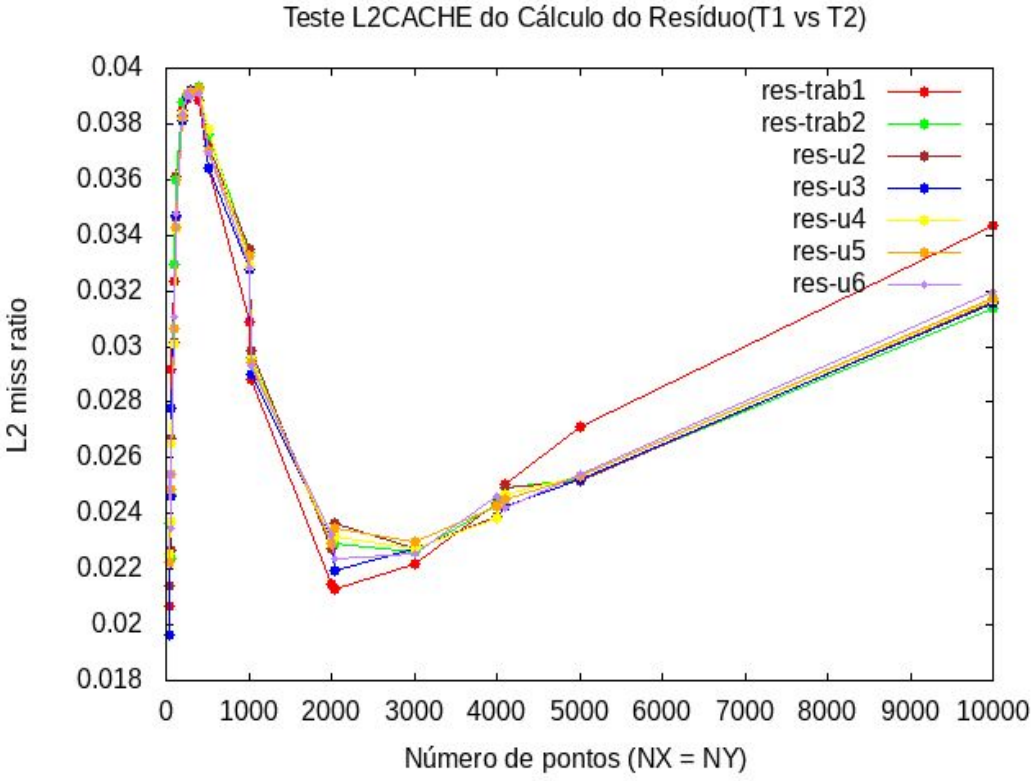

Gráfico 6 - Teste L2CAHCE do Cálculo do Resíduo com versões com *unroll*.

Largura de banda da memória L3:

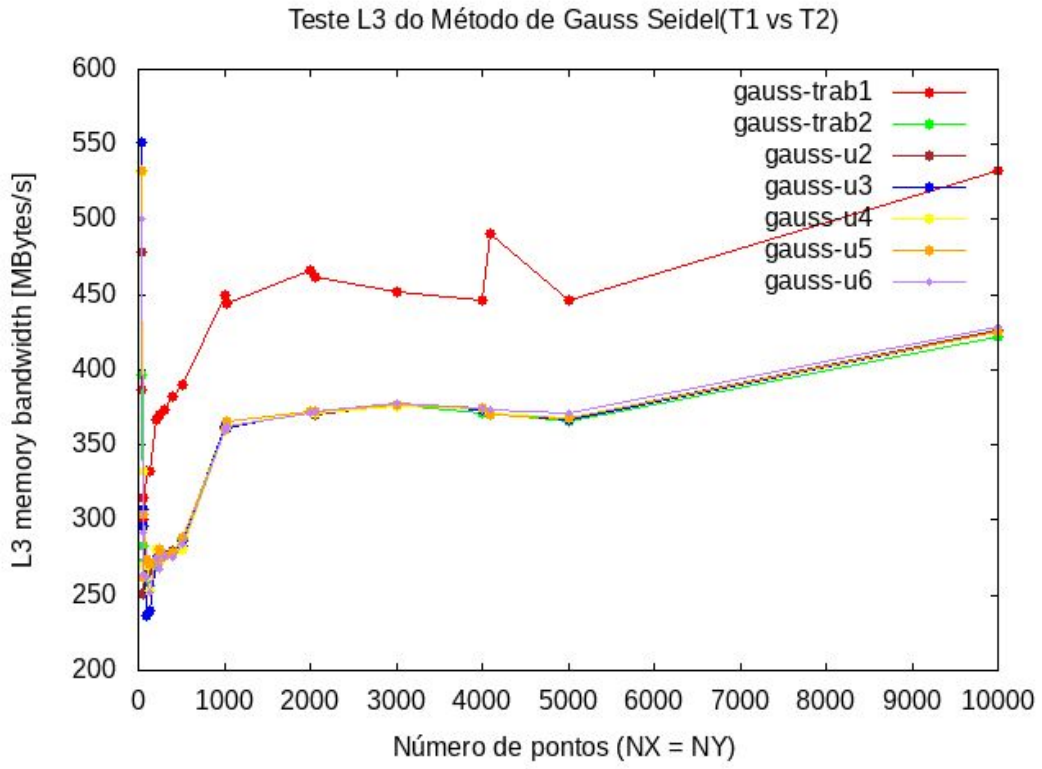

Gráfico 7 - Teste L3 do Método de Gauss Seidel com versões com *unroll*.

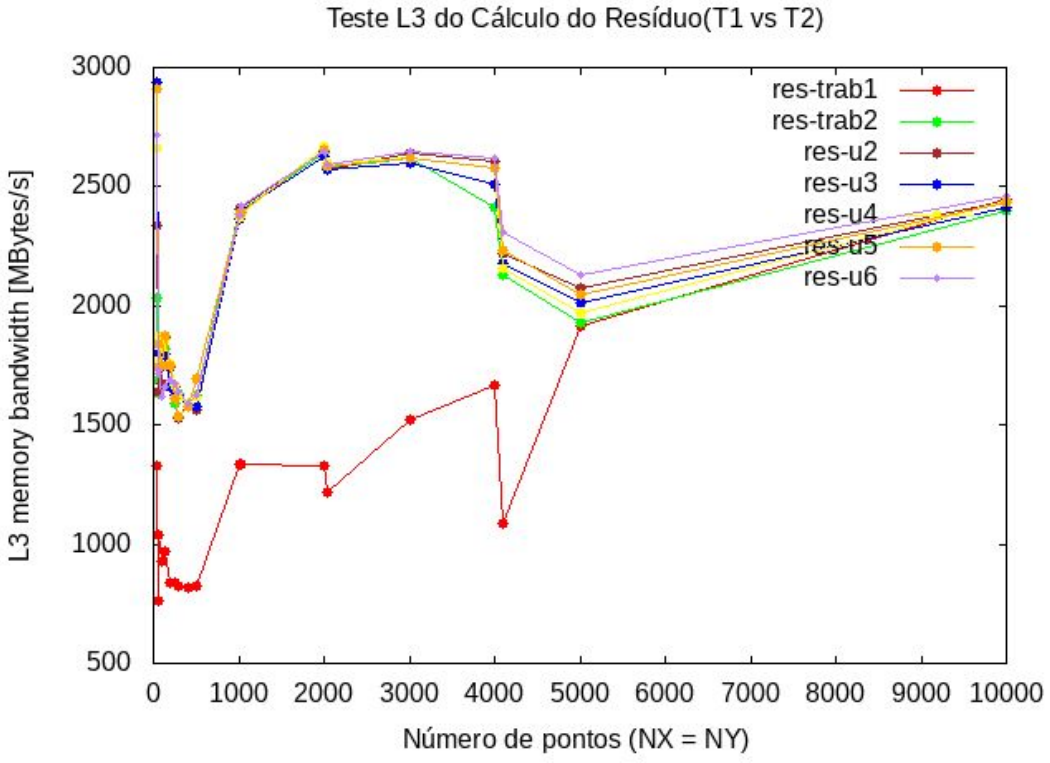

Gráfico 8 - Teste L3 do Cálculo do Resíduo com versões com *unroll*.

Através da análise dos gráficos de tempo de execução, percebe-se que os valores das versões com unroll de fator 2 (gauss-u2 e res-u2) até fator 6 (gauss-u6 e res-u6) tiveram desempenho quase idêntico ao código sem *loop unrolling* (gauss-trab2 e res-trab2), a ponto das linhas das funções se sobreporem.

Nos outros gráficos, a diferença do crescimento dos valores acaba também não sendo tão significativa.

4.2.1 Justificativa

Uma justificativa plausível para que o *Unroll & Jam* não trouxe benefícios significativos para o desempenho dos códigos das funções se dá pelo fato de não serem utilizadas matrizes para armazenar os dados. A constantes das diagonais da matriz A são armazenadas em 5 variáveis e a solução foi vetorizada (é acessada por um único índice k, no lugar de um acesso por i e j), assim evitando problemas como *strided memory access*, que poderia ocorrer no caso de acessar os valores das diagonais na matriz A[i][j] (*strided access* em j).

Outra razão seria o fato da memória ser acessada irregularmente pelas duas funções (especificamente pelo acesso distante de x[k+(nx+2)] e x[k-(nx+2)], que aumenta com o tamanho de nx), diminuindo a eficiência do *unroll*.

5 RESULTADOS FINAIS

No final das otimizações, o resultado da comparação entre a primeira versão no trabalho (gauss-trab1 e res-trab1) e a versão otimizada (gauss-trab2 e res-trab2) é descrita pelos gráficos a seguir.

Média de tempo de execução:

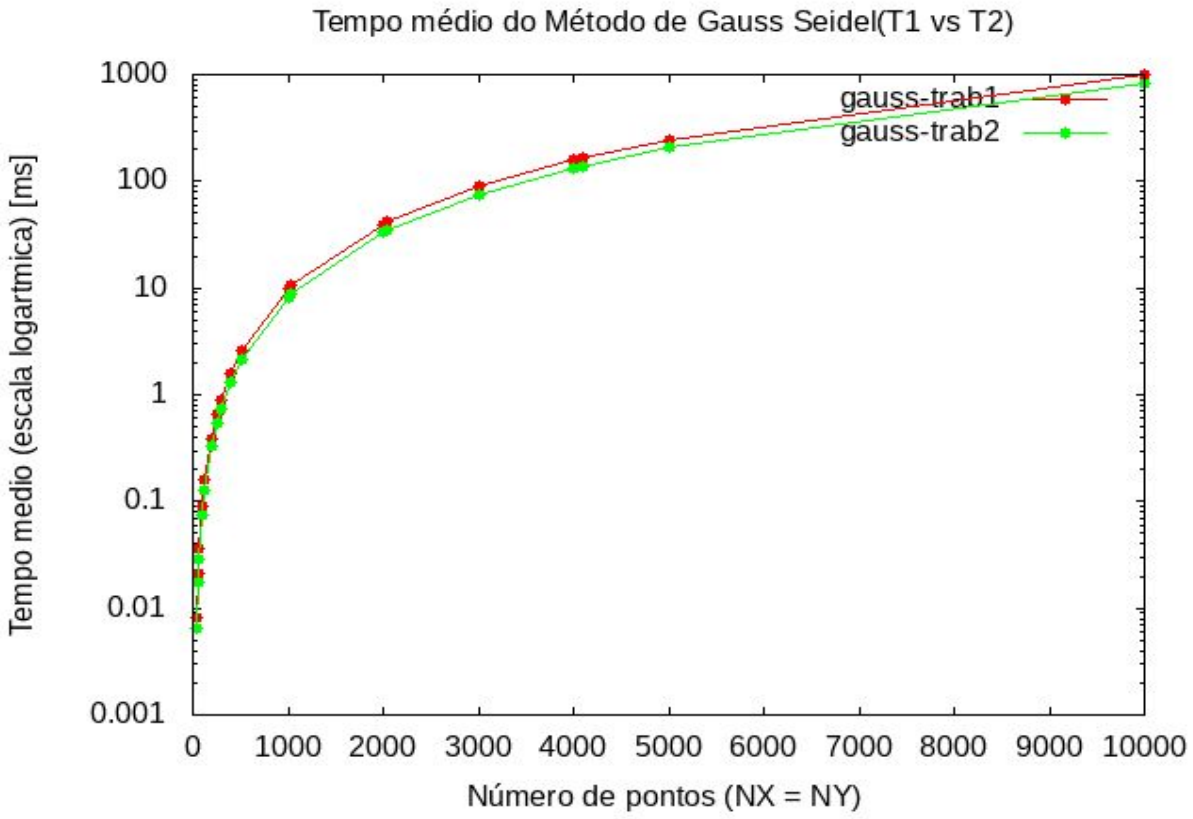

Gráfico 9 - Tempo Médio do Método de Gauss Seidel do trabalho anterior e atual.

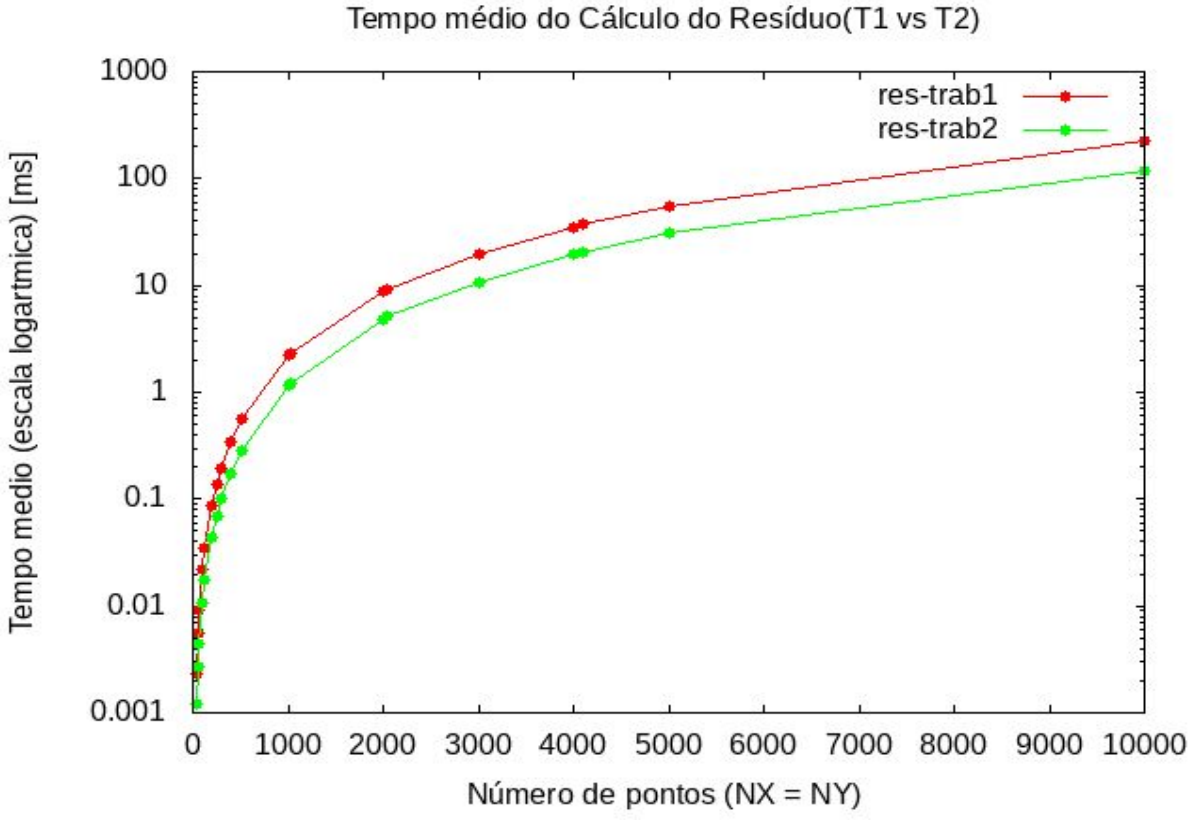

Gráfico 10 - Tempo Médio do resíduo do trabalho anterior e atual.

Operações em ponto flutuante por segundo:

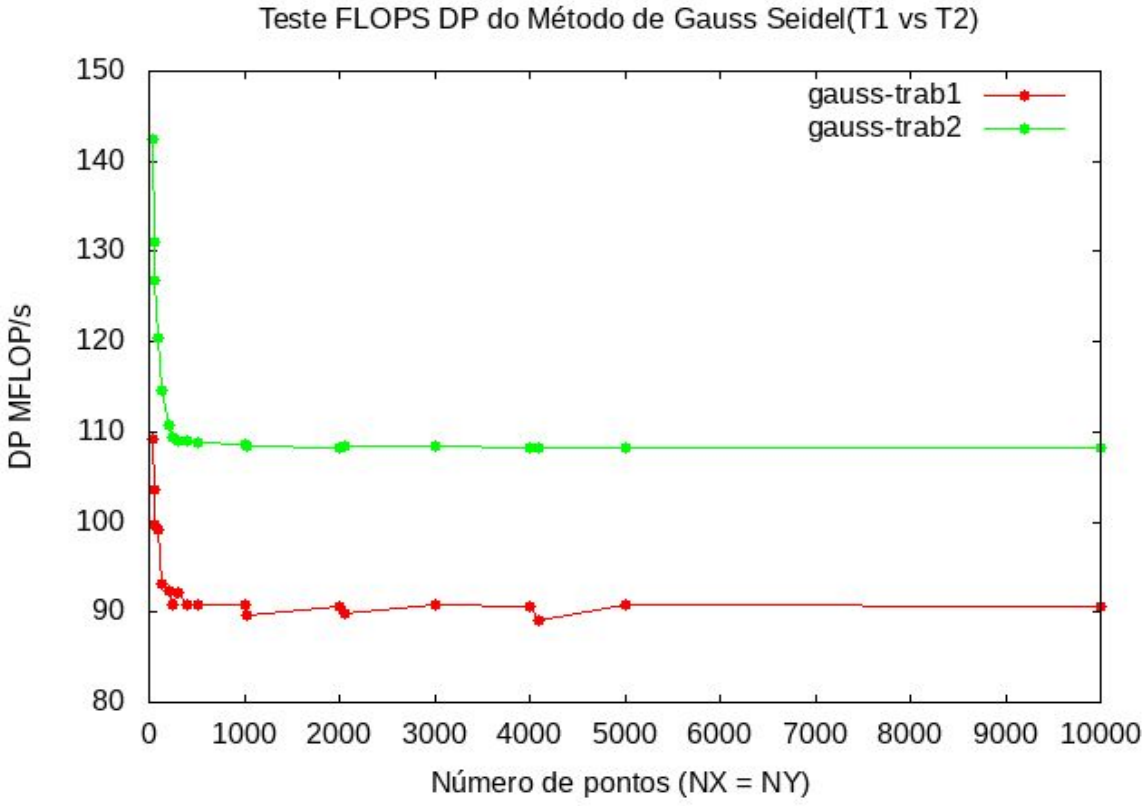

Gráfico 11 - FLOPS_DP do Método de Gauss Seidel do trabalho anterior e atual.

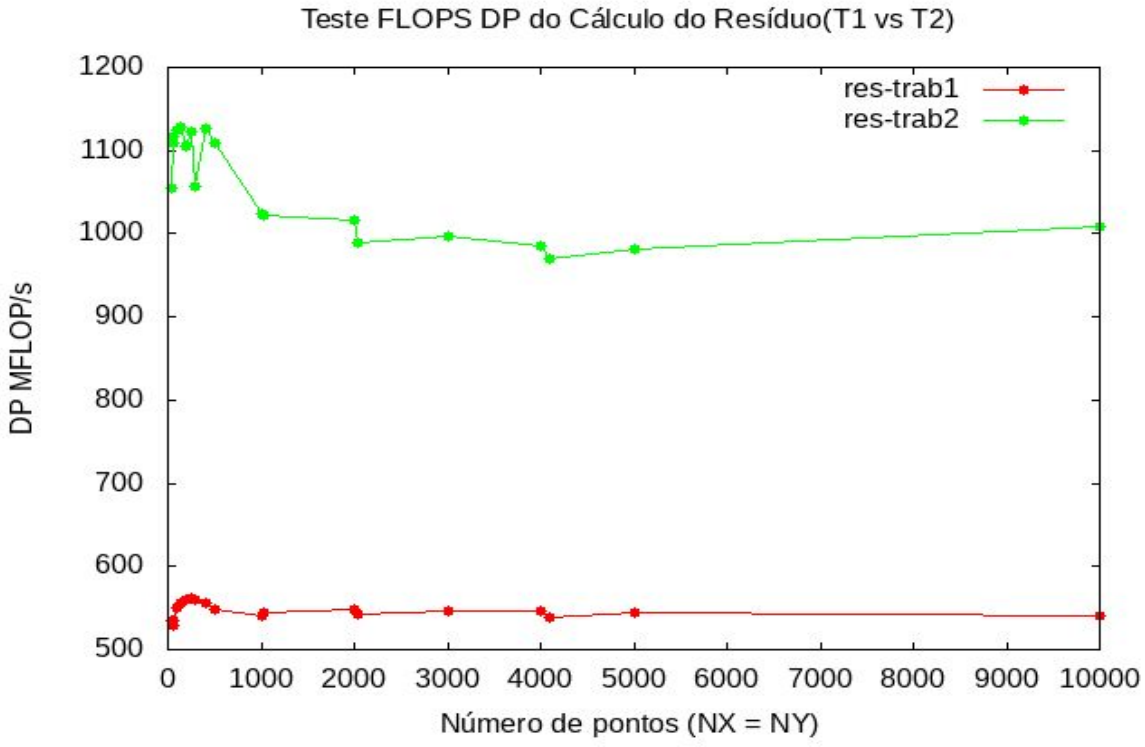

Gráfico 12 - FLOPS_DP do resíduo do trabalho anterior e atual.

Largura de banda da memória L3:

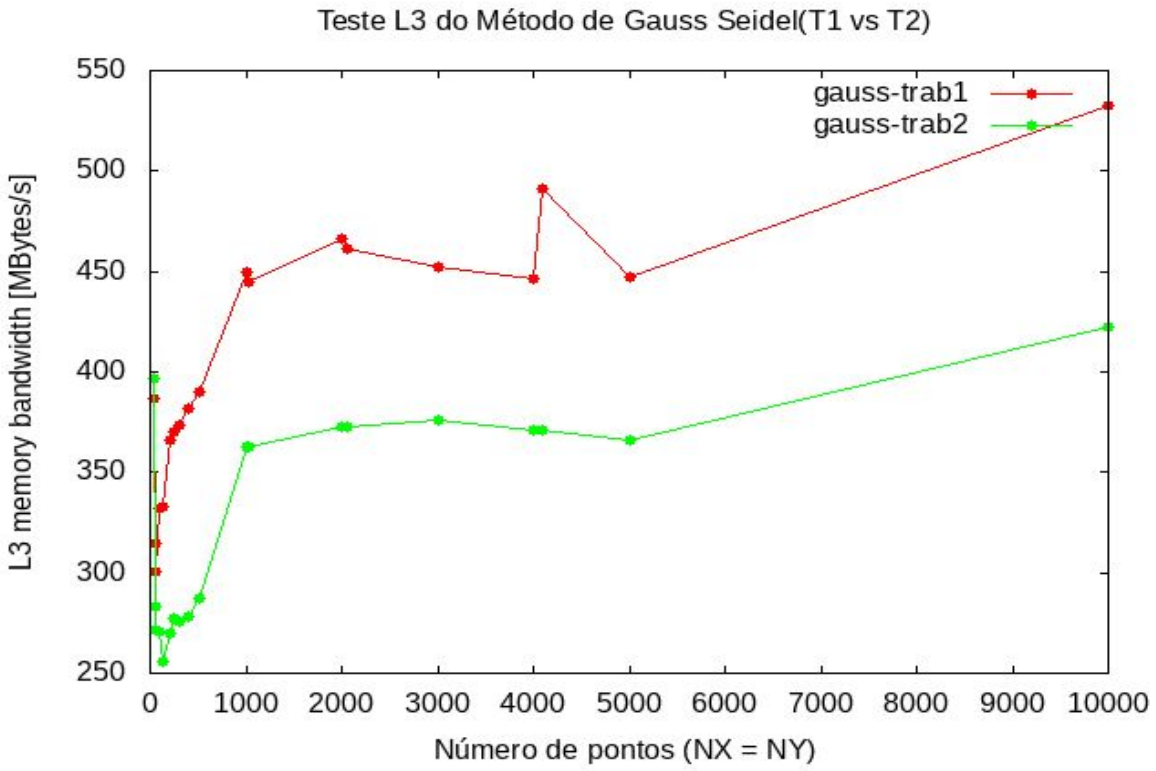

Gráfico 13 - Teste L3 do Método de Gauss Seidel do trabalho anterior e atual.

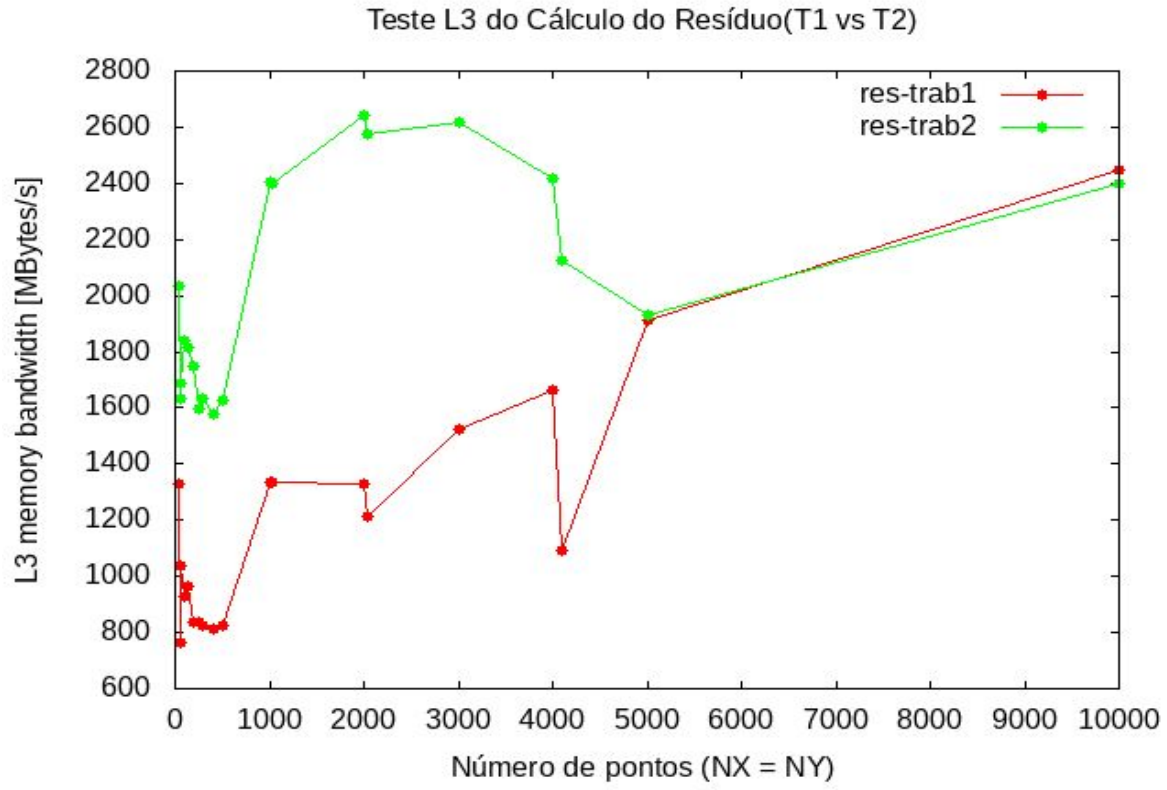

Gráfico 14 - Teste L3 do resíduo do trabalho anterior e atual.

Miss ratio da memória L2:

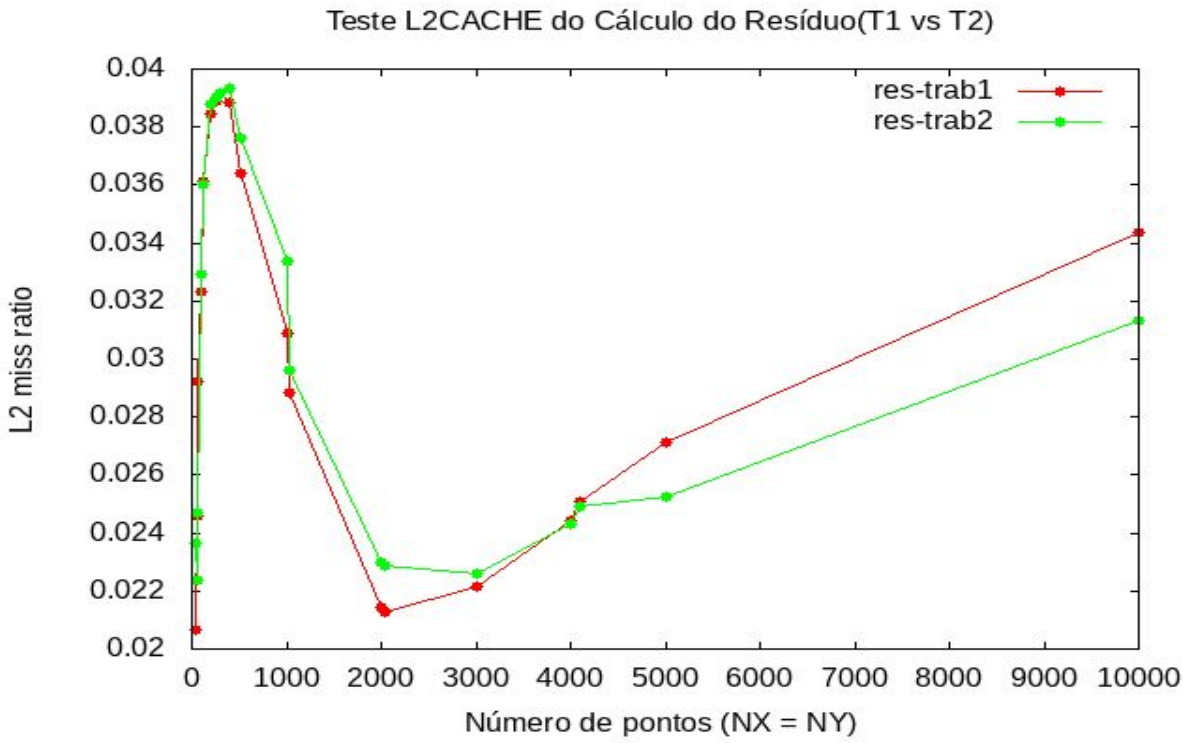

Gráfico 15 - Teste L2CACHE do Método de Gauss Seidel do trabalho anterior e atual.

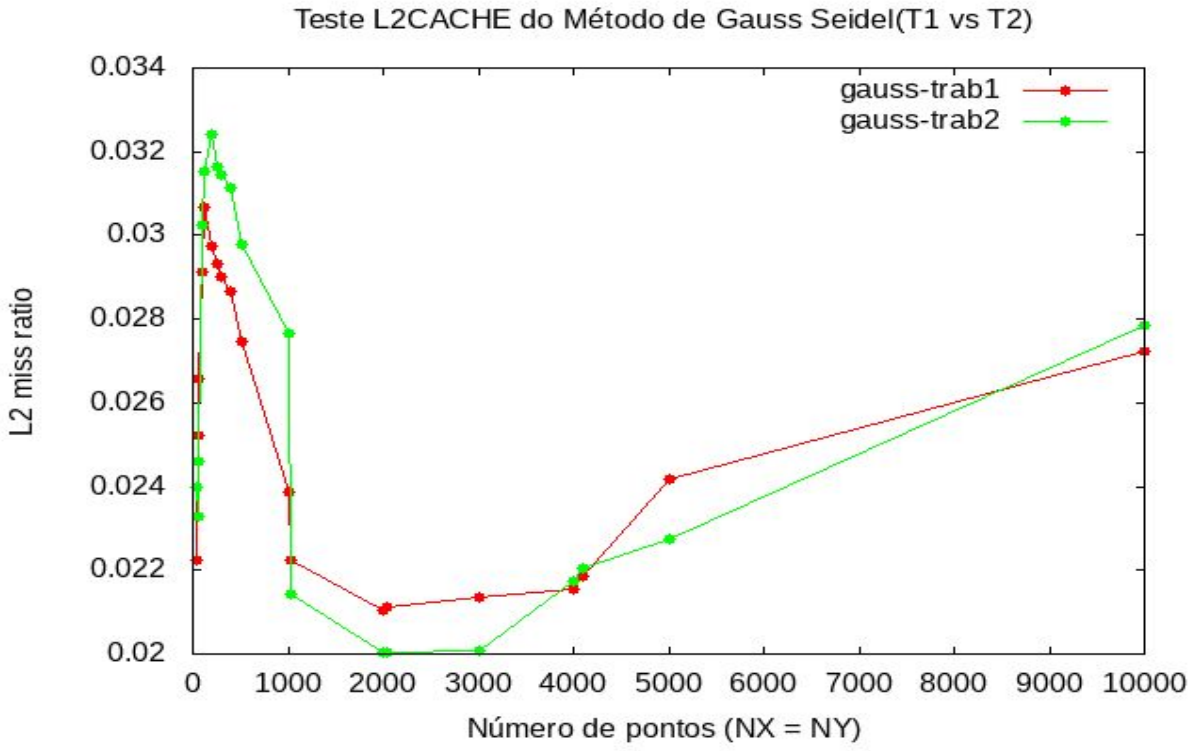

Gráfico 16 - Teste L2CACHE do resíduo do trabalho anterior e atual.

Piores casos de execução:

Gráfico 17 - Tempo Médio para os maiores casos de teste do Método de Gauss Seidel do trabalho anterior e atual.

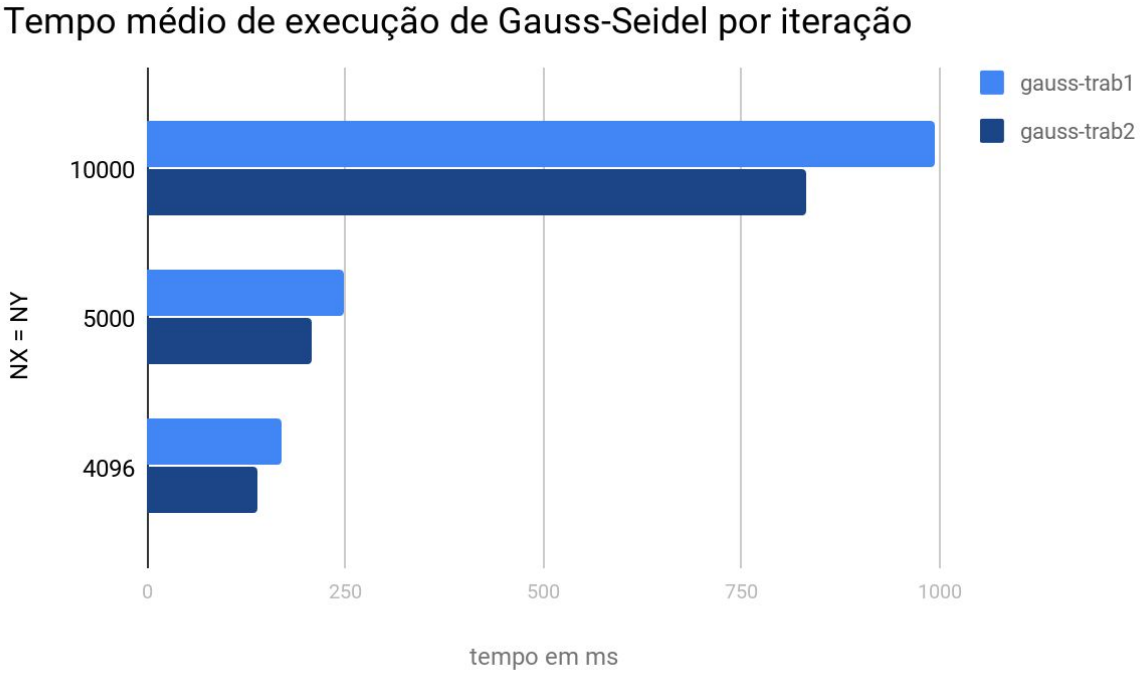

Gráfico 18 - Tempo Médio para os maiores casos de teste do resíduo do trabalho anterior e atual.

6 CONCLUSÃO

Concluímos com o trabalho que, apesar de aumentarmos o tamanho de nosso vetor-solução x, ao retirar condições de desvio em nossos *loops* para o cálculo de Gauss-Seidel e de nosso resíduo, tivemos um ganho significativo no tempo de execução do programa, chegando a uma melhoria aproximada de 47% no resíduo e 16% no Gauss-Seidel no melhor caso (NX = NY = 10000).

Além disso, notamos um considerável aumento no número de operações de pontos flutuantes em nossa nova versão, aproveitando melhor a capacidade de processamento disponível.

Com relação à largura de banda da memória L3, como passamos a utilizar o vetor x com os valores de contorno armazenados no próprio vetor, acabamos fazendo menor uso de acesso à memória, já que os valores estavam em uma linha de *cache* vizinha**.** Já no cálculo do resíduo, como o mesmo é realizado após o Gauss Seidel, os valores iniciais que a serem usados no resíduo já saíram dos níveis da cache L1 e L2 no final da resolução do sistema. Ou seja, o tráfego de memória da cache L3 (vazão) aumenta.

Um fato curioso é o *miss ratio* da memória L2 , tanto na execução de Gauss-Seidel quanto na do resíduo, não apresentam um resultado em definitivo, sendo ora melhor, ora pior.