

Otimização de Código Serial (EP04)

Izalorran Bonaldi & Yuri Tobias

October 29, 2023

1 Introdução

O resumo atual trata das conclusões obtidas após a utilização das técnicas conhecidas como unroll & jam e blocking para melhorar a performance de duas funções de multiplicação de matrizes e vetores, sendo elas (funções):

1. multiplicação de uma matriz do tipo MatRow¹ por um vetor;
2. multiplicação de duas matrizes de tipo MatRow¹.

A ideia é basicamente exibir e concluir, por meio dos gráficos que foram gerados, para quais métricas e para quais tamanhos da entrada (quantidade de linhas e colunas da matriz e do vetor), as otimizações foram mais eficientes.

2 Métodos

Para avaliar o desempenho das funções de multiplicação, uma série de testes foi conduzida. Cada teste foi documentado em um gráfico no qual o tamanho da matriz foi representado no eixo x (abscissa) e os indicadores de desempenho a seguir foram representados no eixo y (ordenada). Os seguintes tamanhos de matriz foram utilizados nos testes:

- N=64, 100, 128, 200, 256, 512, 600, 900, 1024, 2000, 2048, 3000, 4000.

Para cada função de multiplicação, foram realizados os seguintes testes:

1. Teste de Tempo:
 - Mostra o tempo médio do cálculo da função em milissegundos.
 - O tempo foi medido utilizando a função "timestamp()".
2. Banda de Memória:
 - Utilizou-se o grupo L3 do LIKWID para medir o desempenho de memória.

¹MatRow: tipo utilizado para representar uma matriz implementada como um único vetor cujo conteúdo são as linhas da matriz, em sequência.

- O resultado apresentado foi o "L3 bandwidth [MBytes/s]".

3. Cache Miss L2:

- Utilizou-se o grupo L2CACHE do LIKWID para medir a taxa de cache miss.
- O resultado apresentado foi o "L2 miss ratio".

4. Energia:

- Utilizou-se o grupo ENERGY do LIKWID para medir o consumo de energia durante a execução.
- O resultado apresentado foi a "Energy [J]".

5. Operações Aritméticas:

- Utilizou-se o grupo FLOPS_DP do LIKWID para medir a taxa na qual o programa está realizando operações de ponto flutuante de precisão dupla por segundo.
- Os resultados apresentados foram "FLOP_DP [MFLOPs]" e "FLOPS_AVX [MFLOPs]".

É fundamental observar que as seguintes condições foram seguidas durante os testes:

- Os códigos foram compilados com o GCC e as opções de compilação utilizadas foram: -O3 -mavx2 -march=native.
- Todos os testes foram realizados com os mesmos parâmetros e em igualdade de condições.
- Os códigos foram instrumentados com a biblioteca do LIKWID para separar os contadores de desempenho de cada função.
- A arquitetura do processador utilizada nos testes está documentada a seguir e foi obtida com o comando "likwid-topology -g -c".

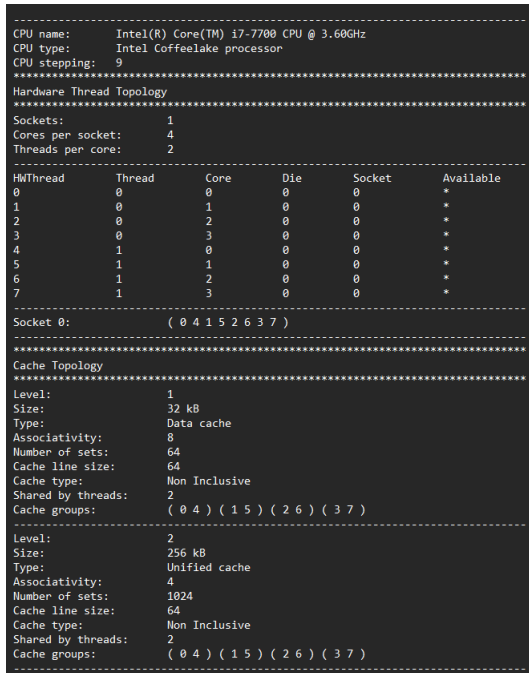


Figure 1

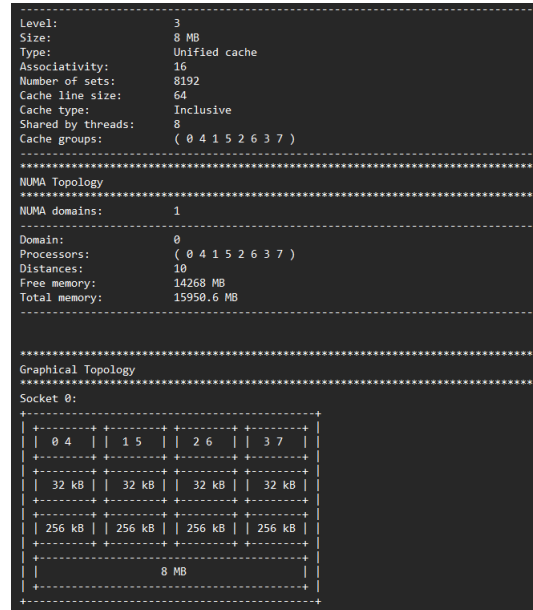


Figure 2

- Todos os códigos foram compilados na mesma máquina utilizada para os testes.

Essas diretrizes foram seguidas rigorosamente para garantir a precisão e comparabilidade dos resultados de desempenho das funções de multiplicação em diferentes tamanhos de matriz, tudo conforme o enunciado da atividade.

3 Resultados

3.1 Teste de tempo

Resultados obtidos:

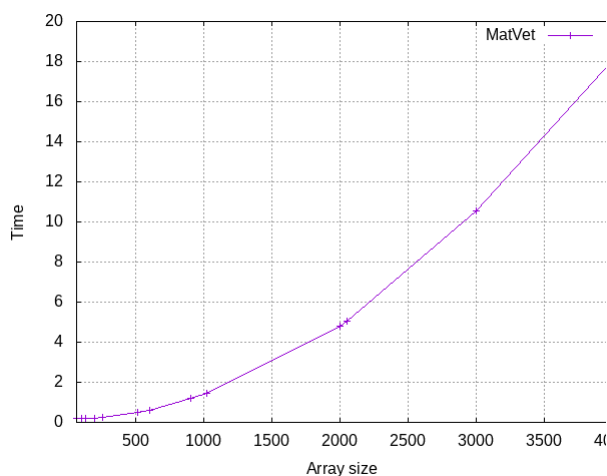


Figure 3

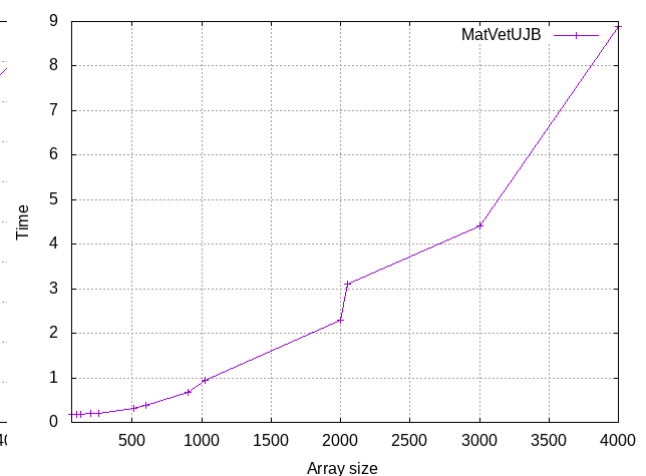


Figure 4

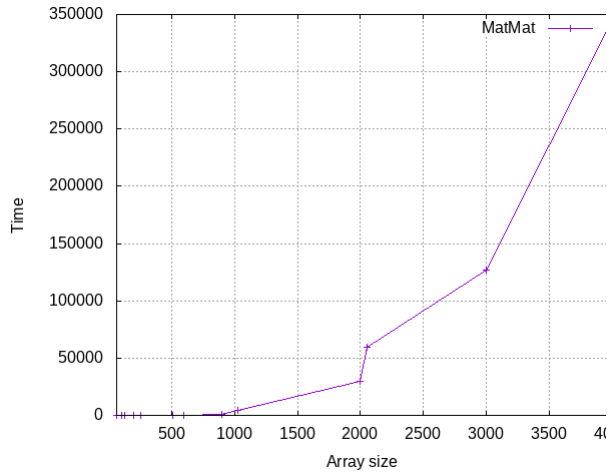


Figure 5

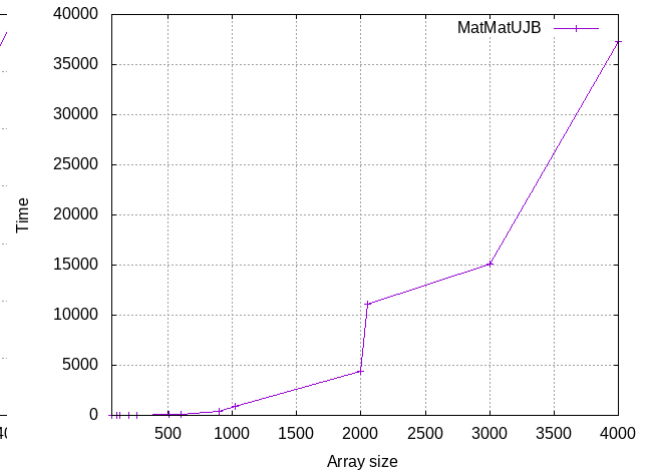


Figure 6

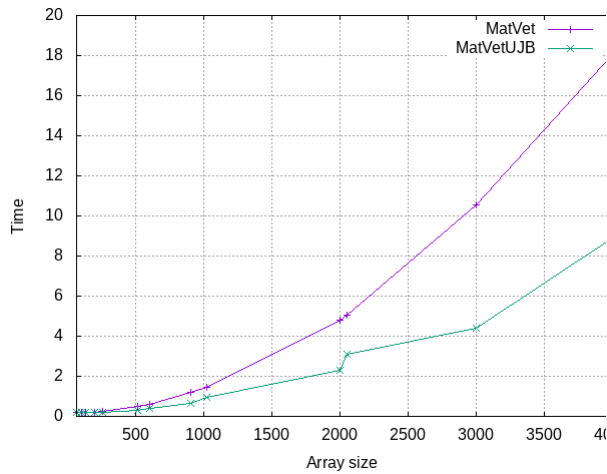


Figure 7

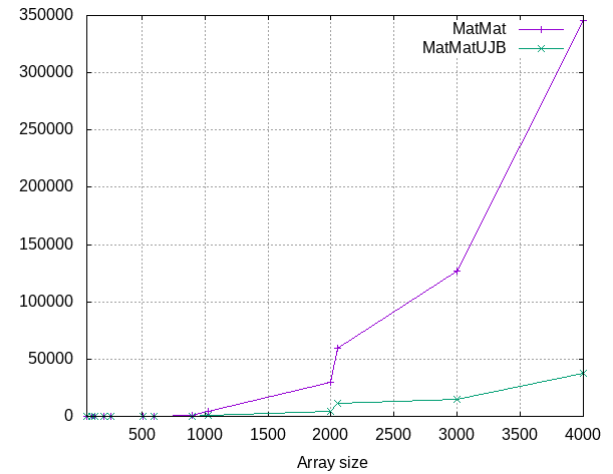


Figure 8

Com isso, é possível perceber, a partir das Figuras 3, 4 e 7, que para entradas não muito grandes a diferença não é tão expressiva assim do caso não otimizado pro otimizado na multiplicação de matriz por vetor. Já no caso da multiplicação de matriz por matriz, é possível perceber, vide Figuras 5, 6 e 8, que desde o início, ou seja, mesmo para entradas pequenas, a diferença de tempo já é relativamente grande!

3.2 Banda de memória

Resultados obtidos:

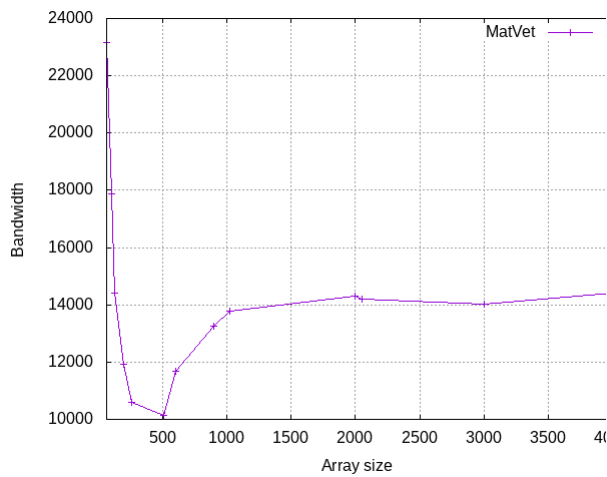


Figure 9

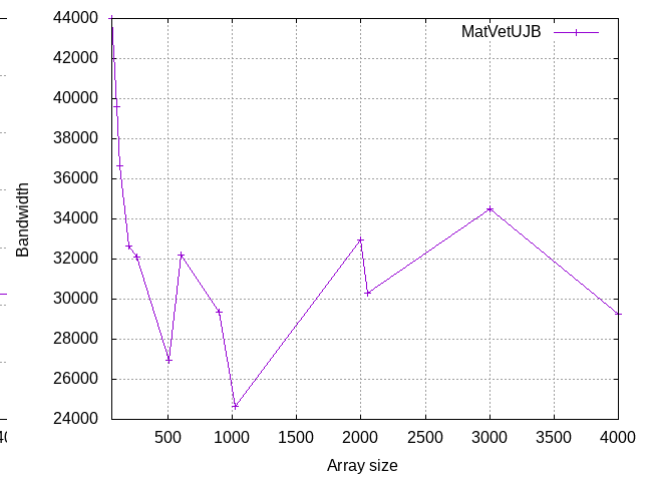


Figure 10

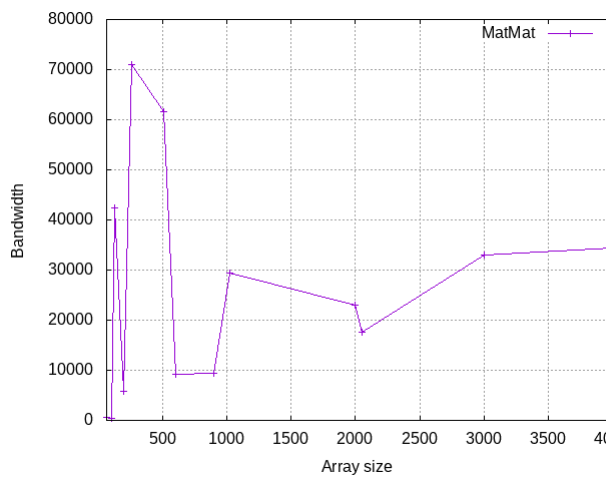


Figure 11

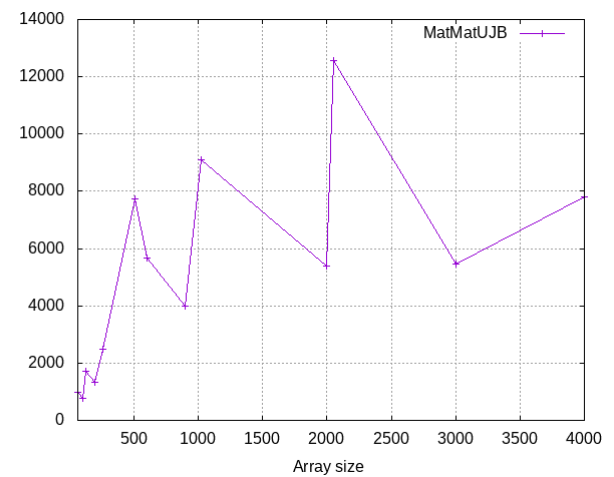


Figure 12

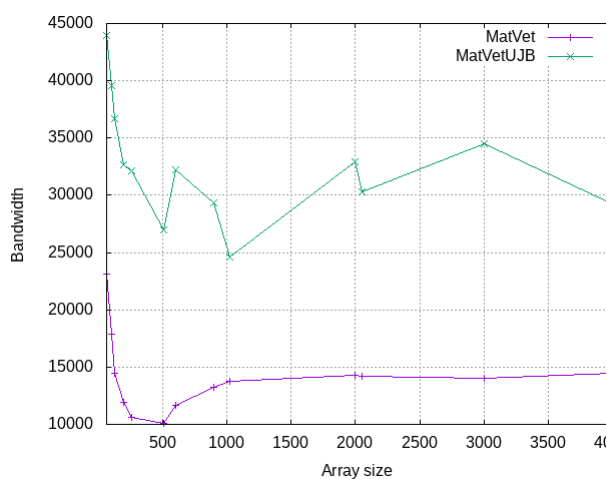


Figure 13

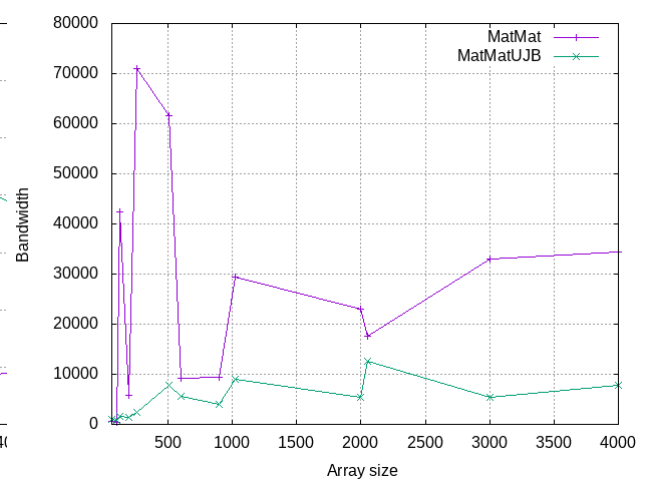


Figure 14

A partir dos gráficos acima é possível concluir que, no caso da multiplicação de matriz por vetor, a versão não otimizada é um pouco mais estável do que a versão otimizada,

vide 9, 10 e 13, porém em nenhum momento a primeira versão obteve valores melhores do que a segunda, o que era esperado. Já no caso da multiplicação de matriz por matriz os resultados obtidos foram praticamente os opostos, sem otimização a largura de banda passa a ser superior já a partir de uma entrada de tamanho 100, enquanto a versão otimizada fica bem pra trás, conforme gráficos 11, 12 e 14, o que de certa forma não era esperado (?).

3.3 Cache miss ratio

Resultados obtidos:

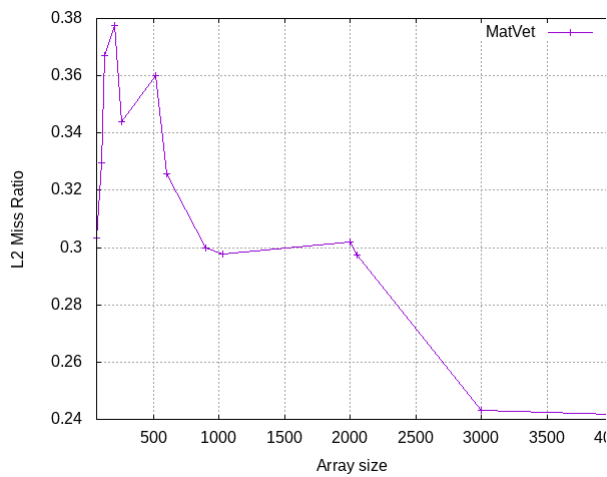


Figure 15

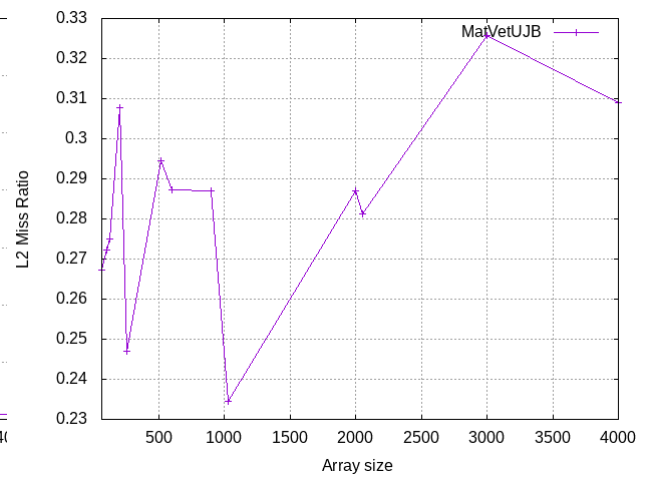


Figure 16

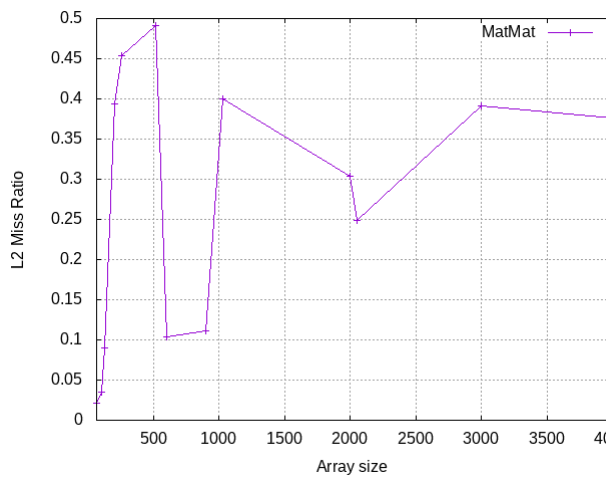


Figure 17

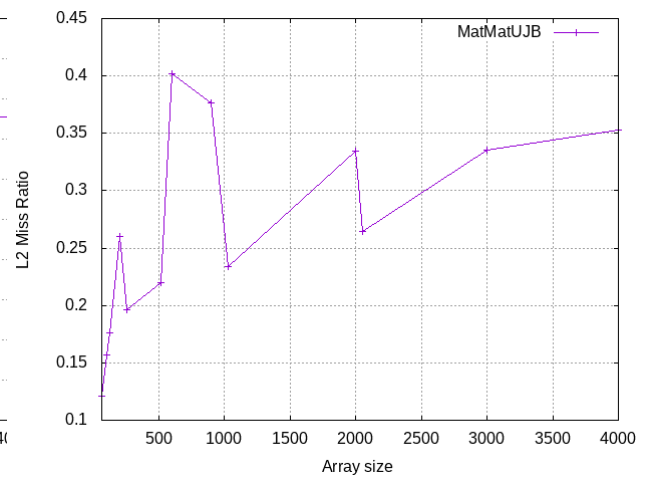


Figure 18

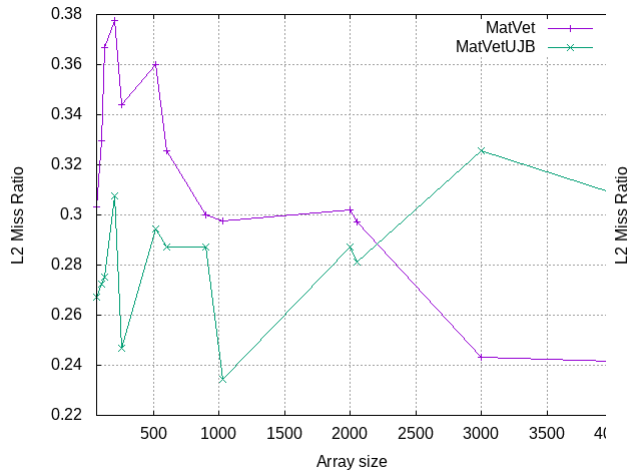


Figure 19

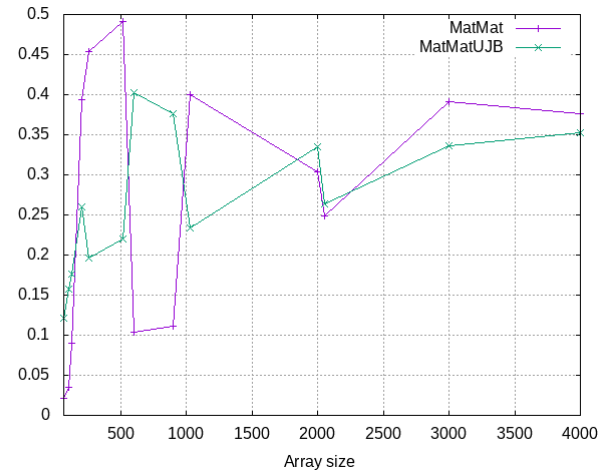


Figure 20

Observando os gráficos apresentados acima é possível concluir que, conforme Figuras 15, 16 e 19, no caso da multiplicação de matriz por vetor, para entradas de tamanho até 2048(valor aproximado), a taxa de cache miss é menor na versão otimizada, entretanto, a partir daí a versão não otimizada sofre menos cache misses. Já na versão da multiplicação de matriz por matriz, vide Figuras 17, 18 e 20, é possível perceber que na maior parte do tempo a taxa de cache miss é menor na versão otimizada, tendo oscilações apenas entre entradas de tamanho entre 500 e 2048, aproximadamente.

3.4 Energia

Resultados obtidos:

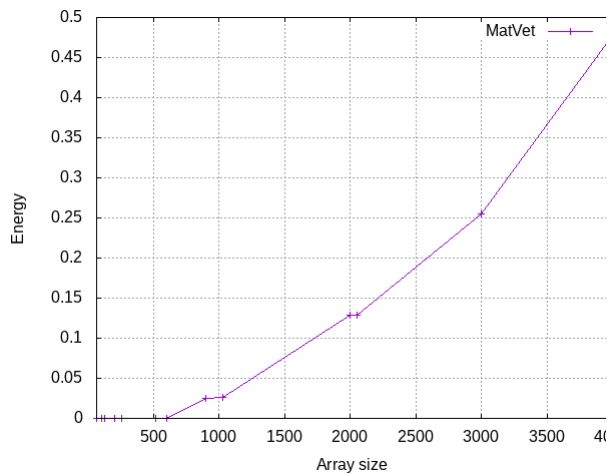


Figure 21

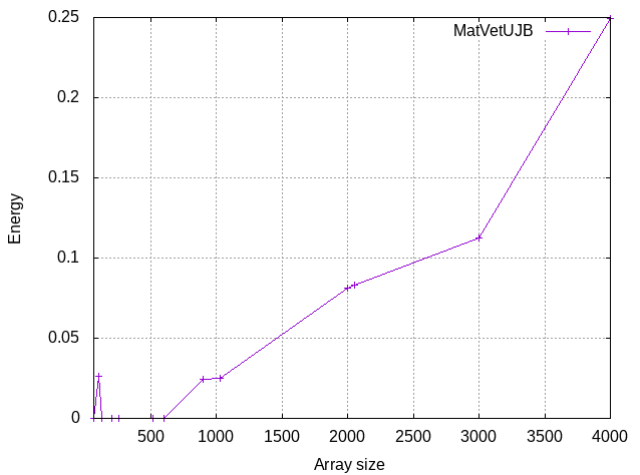


Figure 22

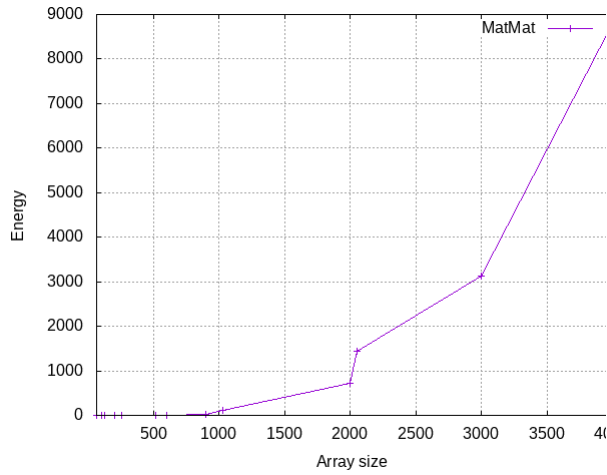


Figure 23

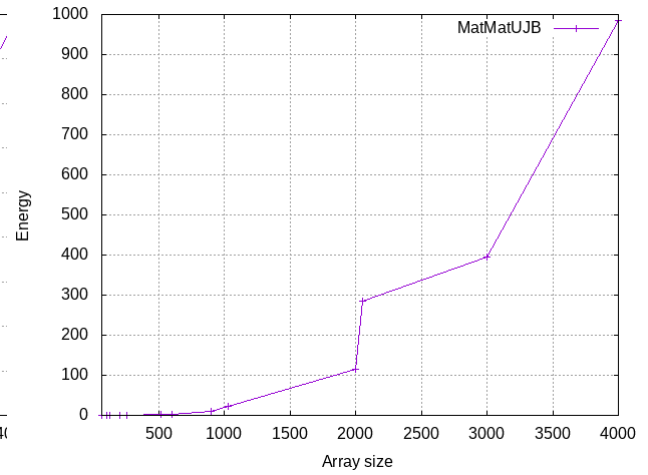


Figure 24

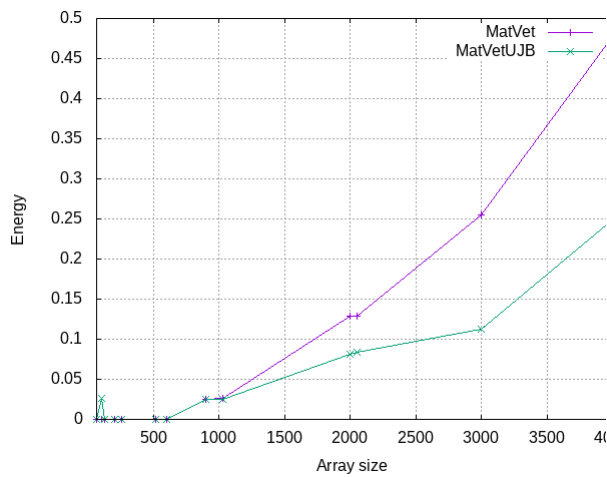


Figure 25

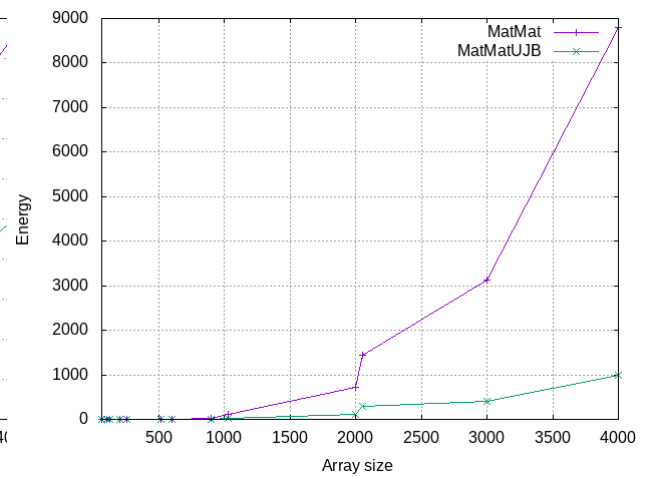


Figure 26

Com relação a métrica ENERGY [J] é possível concluir, a partir de todas as figuras acima, em especial das Figuras 25 e 26, que em todos os casos, ou seja, para todos os tamanhos de entrada, o consumo de energia durante a execução é consideravelmente menor na versão otimizada, o que acaba sendo mais expressivo para os valores de entrada maiores.

3.5 Operações aritméticas (FLOPS_DP)

Resultados obtidos:

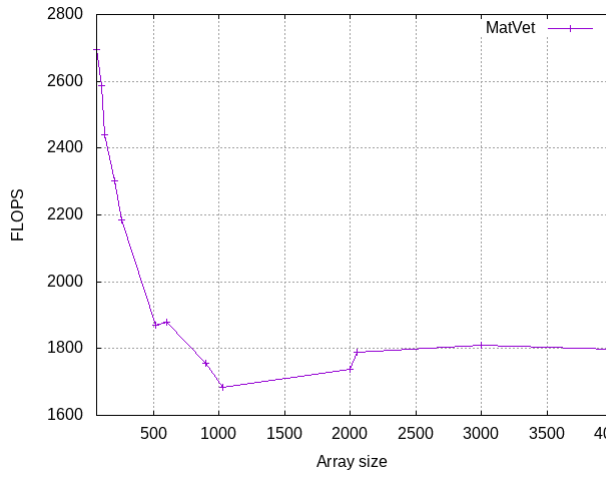


Figure 27

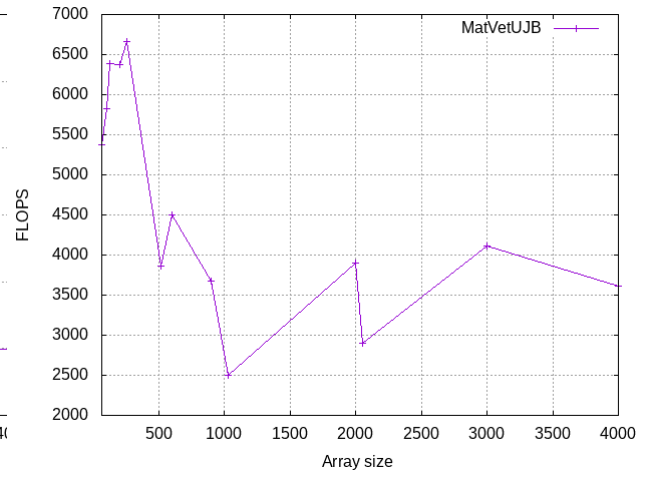


Figure 28

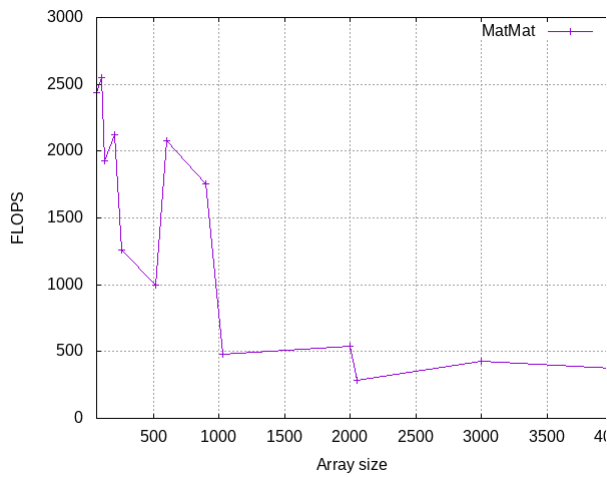


Figure 29

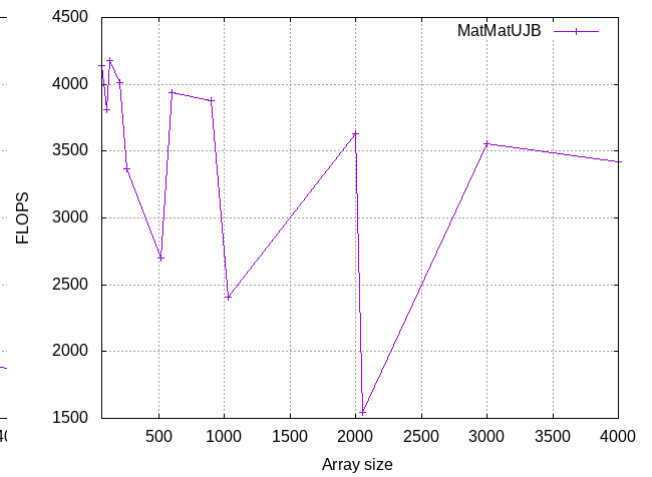


Figure 30

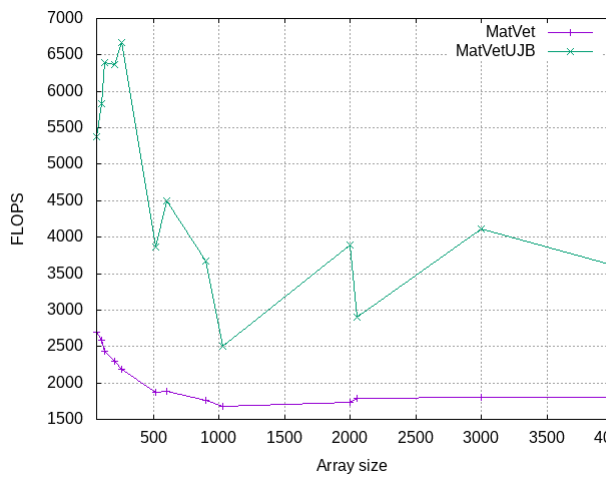


Figure 31

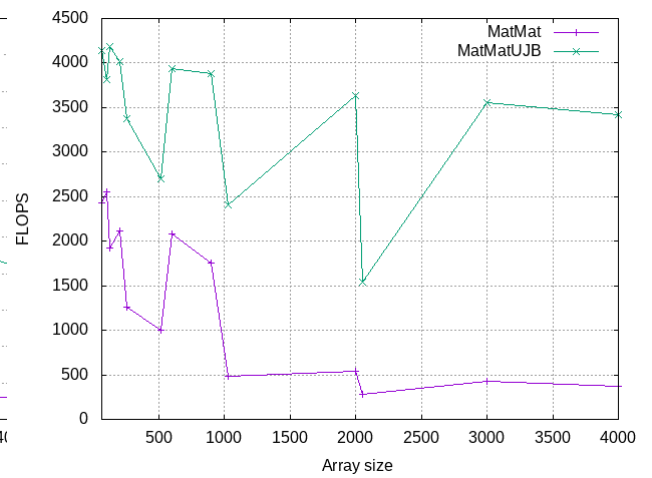


Figure 32

Com relação a métrica FLOPS_{DP} é possível concluir que, conforme os gráficos acima, em especial os das Figuras 31 e 32, as versões otimizadas, tanto no caso da multiplicação

de matriz por vetor quanto na multiplicação de matriz por matriz, acabam realizando um número consideravelmente maior de milhões de operações de ponto flutuante por segundo, o que acaba favorecendo a redução do tempo gasto para a execução das funções.

3.6 Operações aritméticas (FLOPS_AVX)

Resultados obtidos:

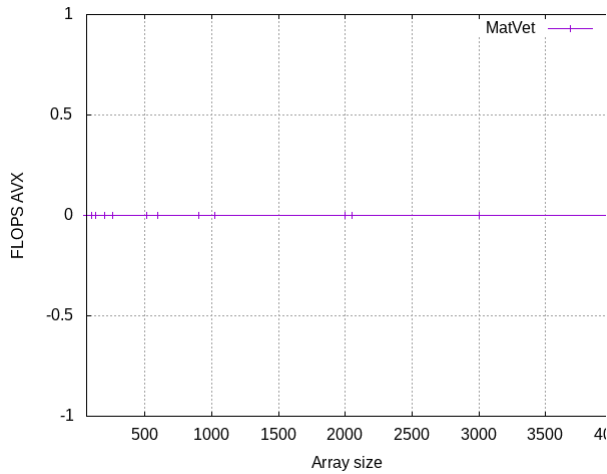


Figure 33

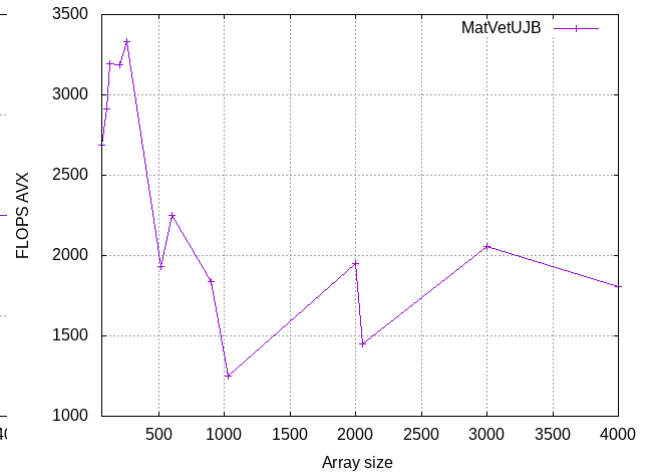


Figure 34

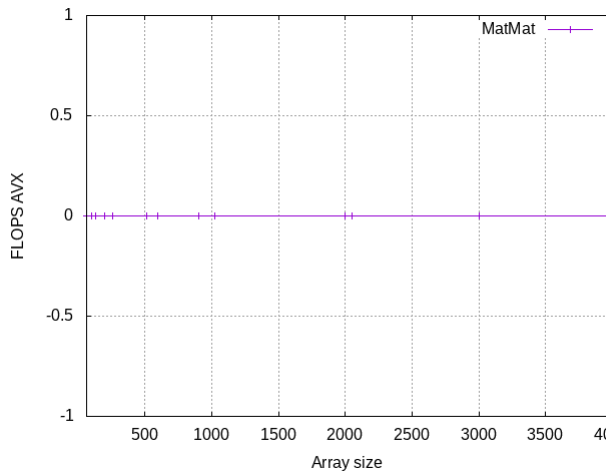


Figure 35

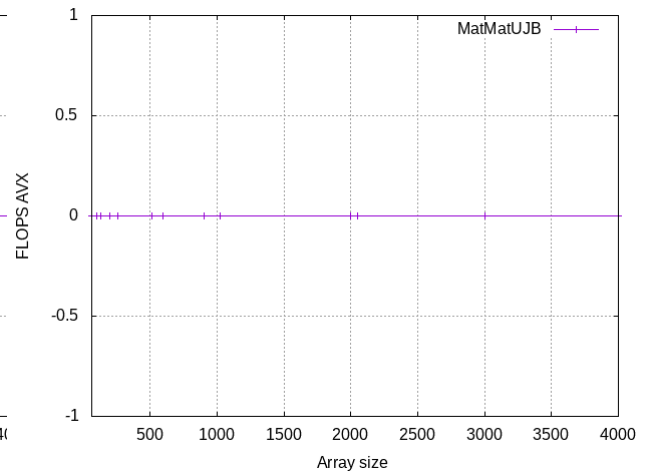


Figure 36

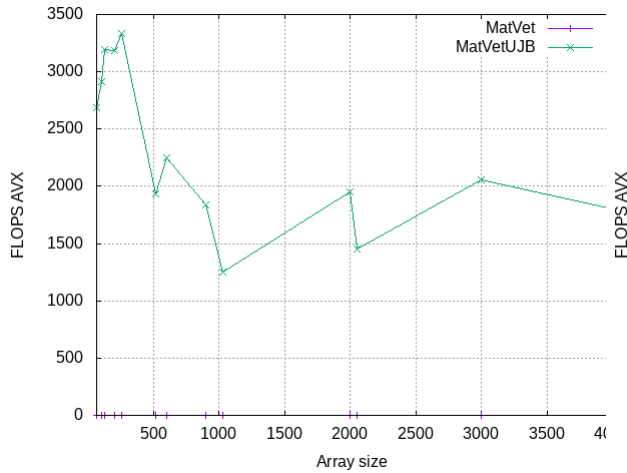


Figure 37

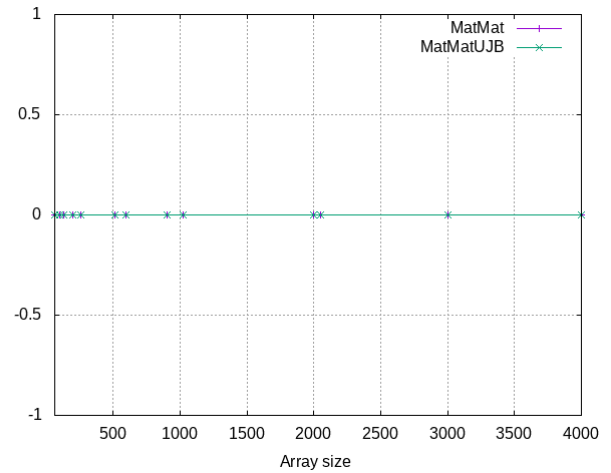


Figure 38

Por fim, com relação a métrica FLOPS_AVX é possível concluir, conforme as imagens acima, em especial as Figuras 37 e 38, que na versão de multiplicação de matriz por vetor a otimização foi feita utilizando o registrador avx para que fossem realizadas mais operações ao mesmo tempo, o que era esperado; além disso, nessa mesma versão é possível perceber que esse número cai conforme se aumenta o tamanho da entrada e depois fica razoavelmente estável. Já no segundo caso, ou seja, multiplicação de matriz por matriz, infelizmente a otimização utilizando o registrador avx não ocorreu.

4 Observações

4.1 Limitações

- Existem dois scripts: script.sh e extensiveScript.sh, o primeiro é utilizado para realizar testes unitários sem gerar gráficos, enquanto que o último executa o programa para todos os tamanhos de entrada fornecidos no enunciado do trabalho e gera os gráficos;
- O programa foi executado em uma máquina que não pertence aos laboratórios do departamento. Isso dito, a variável LIKWID_HOME possui como rota /usr/local;
- A saída do tempo em milissegundos segue o padrão indicado pela biblioteca utils.h;

4.2 Otimizações

- As otimizações feitas consistem basicamente em fazer unroll & jam + blocking tanto na função dada "MultMatVet", dando origem a função "MultMatVetUJB", e na função dada "MultMatMat", dando origem a função "MultMatMatUJB". Vale comentar também que em cada função existem dois laços extras para realizar as operações nas linhas e colunas que eventualmente possam ter ficado de fora por conta do fator de unroll;
- Foi adicionado "restrict" antes do tipo dos parâmetros nas funções para evitar que o compilador tenha dúvidas a respeito da existência ou não de dependências.