Paralelização de Algoritmo do Produto Escalar

Wendel de Oliveira Santos[1], Dr. Leonardo Nogueira Matos[1]

[1]Departamento de Computação
Universidade Federal de Sergipe (UFS) – São Cristóvão, SE – Brazil
wendell_j.fox@hotmail.com, lnmatos@ufs.br

Abstract. *This article evaluates the implementation of a class library in C++ for algorithms concurrent and parallel to perform the scalar product, an operation used in the creation of linear classifiers for algorithms of machine learning. The classes were developed in language of parallel programming CUDA (Compute Unified Device Architecture) exploiting resources of the GPU's, and in C++ with an API of programming of shared memory OpenMP, using the resources of the CPU's. Finally, the article discusses the reduction of the computational cost of the scalar product comparing these approaches with the implementation conventional, sequential, held in CPU.*

Resumo. *Este artigo avalia a implementação de uma biblioteca de classes em C++ para algoritmos concorrente e paralelo para realizar o produto escalar, uma operação usada na criação de classificadores lineares para algoritmos de aprendizado de máquina. As classes foram desenvolvidas em linguagem de programação paralela CUDA (Compute Unified Device Architecture), explorando recursos de GPU's, e em linguagem C++ com uma API de programação de memória compartilhada OpenMP, utilizando os recursos das CPU's. Ao final, o artigo discute a redução do custo computacional do produto escalar comparando essas abordagens com a implementação convencional, sequencial, realizada em CPU.*

1. Introdução

Várias grandezas físicas, tais como, por exemplo, comprimento, área, volume, tempo, massa e temperatura são completamente descritas, uma vez que a magnitude (intensidade) é data [Biezuner 2009]. Tais grandezas são chamadas escalares e são modeladas por números reais [Biezuner 2009]. Outras grandezas físicas não são completamente caracterizadas até que uma magnitude, uma direção e um sentido sejam especificados. Tais grandezas são chamadas vetoriais e são modeladas por vetores. O produto escalar é uma operação entre dois vetores, cujo, resultado é um escalar. Essa operação é muito utilizada nos problemas de visão computacional, por exemplo, na criação de classificadores lineares para algoritmos de aprendizado de máquina.

CUDA (Compute Unified Device Architecture) é uma arquitetura de abstração como um modelo de programação embutido, desenvolvido para suportar a computação estrita e altamente paralelizada [Sanders and Kandrot 2011]. Com essa plataforma, o desafio de desenvolver uma implementação do produto escalar capaz de aumentar o seu paralelismo para otimizar a utilização do processador se torna viável, dando suporte para o uso de vários processadores ao mesmo tempo.

A utilização da API OpenMP (Open Multi Processing) tem crescido bastante nos últimos anos, uma vez que as funcionalidades do mesmo facilitam o desenvolvimento

de aplicações em memória compartilhada [Sena and Costa 2008]. O OpenMP funciona a partir de um conjunto de diretivas e uma API, aprimoradas por um conjunto de *threads*.

O objetivo desse trabalho é reduzir o custo computacional do produto escalar utilizando as abordagens da implementação de algoritmos concorrente empregando OpenMP e paralelo recorrendo CUDA. Ao final, será comparado essas visões com a implementação sequencial do produto escalar.

2. Metodologia do Produto Escalar Paralelo e Concorrente

Foi utilizado a IDE Qt Creator como ambiente de trabalho para a implementação da biblioteca de classes para os algoritmos do produto escalar. O método paralelo difere do concorrente por causa da estrutura das duas linguagens, CUDA e OpenMP, respectivamente. O produto escalar paralelo foi implementado de forma à paralelizar a multiplicação e não a soma, já em concorrente foi paralelizar todo o produto escalar. O experimento com produto escalar utilizando OpenMP foi realizado em uma máquina com processador de 8 núcleos de 2.93GHz e 7.8GiB de memória RAM.

2.1. Produto Escalar

O Cálculo do produto escalar entre dois vetores fica simples se estes vetores são dados em sistemas de coordenadas cartesianas ortogonais.

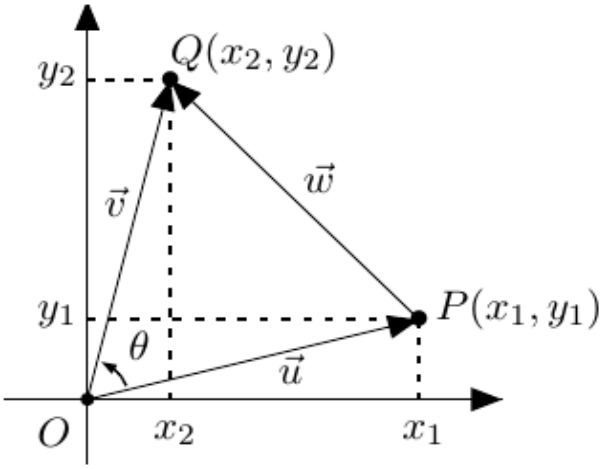

Figura 1. Plano no espaço R² [Lages 2007].

Sejam os pontos P e Q

$$P=(x_1, y_1) \quad (1)$$

$$Q=(x_2, y_2) \quad (2)$$

de modo que $\overline{OP}=\vec{u}\, e\, \overline{OQ}=\vec{v}$. θ é o ângulo entre $\vec{u}\, e\, \vec{v}$, porém, os vetores $\vec{u}$ e $\vec{v}$ são, respectivamente,

$$\vec{u}=(x_1, y_1) \quad (3)$$

$$\vec{v}=(x_2, y_2) \quad (4)$$

então

$$\vec{w}=x_1 y_1+x_2 y_2 \quad (5)$$

2.2. Implementação Concorrente

Para realizar o produto escalar concorrente foi definido a variável *prod_reduction* para armazenar o cálculo do produto escalar. Esse cálculo é feito de forma que cada *thread* calcula sua parcela do produto escalar e armazena na sua cópia da variável *prod_reduction*. No final da região paralela, a cópia original da variável *prod_reduction* é atualizada com a soma dos valores calculados por todas as *threads*.

A cláusula *reduction* especifica que uma ou mais variáveis que são privadas de cada *thread* serão submetidas a uma operação de redução no final da região definida pelo construtor ao qual à cláusula está associada [Sena and Costa 2008].

```
startPS = omp_get_wtime();
#pragma omp parallel private(tid,i) shared(v1,v2) reduction(+:prod_reduction) num_threads(nthreads){
      tid = omp_get_thread_num();
      #pragma omp for
            for (i = 0; i < tam; i++){
                  prod_reduction += v1[i]*v2[i];}}
endPS = omp_get_wtime();
```

Foram definidos dois vetores, *v1* e *v2* com tamanhos idênticos para o produto escalar com valores, desses vetores, aleatórios. O cálculo do tempo de execução desse algoritmo foi feito com a função de tempo *omp_get_wtime()* do OpenMP, essa função retorna um valor em precisão dupla igual ao tempo decorrido em segundos desde o tempo defino antes do construtor paralelo. A variável *startPS* inicializa o tempo e *endPS* finaliza o tempo decorrido em segundos. Abaixo ilustra a equação do tempo de execução em milissegundos.

$$tempoPS = (endPS - startPS) \times 100 \quad (6)$$

2.3. Implementação Paralela

A implementação do produto escalar paralelo foi dividido em duas partes: primeiro a multiplicação dos vetores que foi feita em paralelo em CUDA e depois a soma de todo o vetor resultante que foi feita sequencialmente. O código abaixo ilustra como foi feito a parte da multiplicação paralela em CUDA.

```
cudaEventRecord(start, 0);
__global__ void produtoEscalar(double *v1, double *v2, double *res, int tam){
      int tid = threadIdx.x + blockIdx.x* blockDim.x;
      if (tid < tam){
            res[tid] = v1[tid]*v2[tid];}}
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&time, start, stop);
```

Os parâmetros *v1*, *v2*, *res* e *tam* do construtor *__global__* representam,

respectivamente, os vetores *v1* e *v2*, o vetor resultante da multiplicação e o tamanho do vetor. Esse construtor é executado no dispositivo, ou seja, no *kernel*. Para calcular a identificação da *thread* chamada de *tid*, foi multiplicado a dimensão do bloco *blockDim.x* pela identificação do bloco *blockIdx.x*, por fim foi somado esse resultado com *threadIdx.x*. O vetor resultante será somado posteriormente de modo sequencial, fora do construtor paralelo.

O cálculo do tempo de execução do algoritmo foi feito somente na parte paralela que se refere à multiplicação, já a parte da soma não foi feito o cálculo do tempo. Foi usado a biblioteca *cudaEvent_t* para calcular a parte paralela, utilizando a função *cudaEventRecord()*. O código acima ilustra a parte da implementação do tempo de execução.

3. Comparação dos Tempos de Execução das Implementações Paralela e Concorrente com Sequencial

Foram feitos testes utilizando um tamanho fixo para os vetores e medindo o tempo para uma quantidade de execuções do produto escalar, e posteriormente, foi utilizado uma quantidade fixa de execuções e mediu-se o tempo para tamanhos diferentes dos vetores. Com os resultados, foram comparados os tempos de execução dos algoritmos paralelo e concorrente com o sequencial. Lembrando que os resultados esperados para o algoritmo paralelo refere-se ao tempo de execução da paralelização somente da multiplicação e ao tempo sequencial da soma do algoritmo paralelo.

Para a realização dos testes, foram utilizados dois vetores de tamanhos idênticos com valores aleatórios. Dessa forma, foram fixados os tamanhos dos vetores em 1000000 posições e variando a quantidade de execuções do produto escalar de 5 até 200 execuções com intervalos de 15 em 15 (5, 20, 35, …), e obtido os valores de tempo (em milissegundos) de execução dos algoritmos paralelo, concorrente e sequencial. A partir desses dados foram plotados os gráficos para uma análise mais detalhada. Na figura abaixo é ilustrado o gráfico do tempo de execução do algoritmo sequencial.

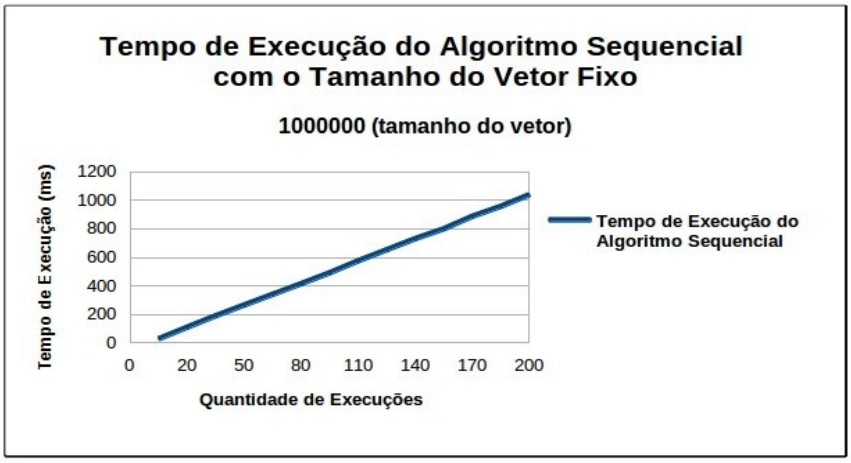

Figura 2. Tempo de execução do algoritmo sequencial com o tamanho dos vetores fixos.

Agora foi fixado um número de 200 execuções do produto escalar e variou-se o tamanho dos vetores de 10000 até 985000 posições com um intervalo de 75000 posições, e obtidos os valores de tempo (em milissegundos) de execução. A partir dos dados foram plotados os gráficos para que, de modo análoga ao anterior, se ter uma

análise mais detalhada dos tempos de execução. Na figura abaixo é ilustrado o gráfico do tempo de execução do algoritmo sequencial com o tamanho do vetor fixado.

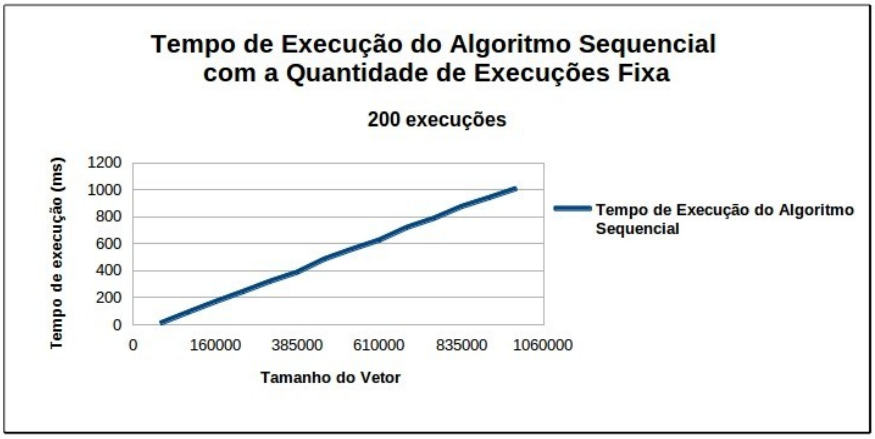

Figura 3. Tempo de execução do algoritmo sequencial com a quantidade de execuções fixa.

3.1. Utilizando OpenMP

Utilizando-se de linhas de tendência, foram encontrados polinômios de primeiro grau tanto para a curva de tempo de execução do algoritmo sequencial, mostrado na figura 2, como para concorrente mostrado na figura 4. Tais polinômios são mostrados abaixo, sendo que *ts* e *tc* representam, respectivamente, o tempo de execução do algoritmo sequencial e concorrente, e *Qe* representa a quantidade de execuções do produto escalar.

Figura 4. Tempo de execução do algoritmo concorrente com o tamanho do vetor fixo.

- Polinômio sequencial (percentual de exatidão 99,98%):

$$ts(Qe) = 77{,}5364\,Qe - 124{,}3094 \quad (7)$$

- Polinômio concorrente (percentual de exatidão 99,85%):

$$tc(Qe) = 15{,}4473\,Qe - 19{,}5829 \quad (8)$$

A partir dos tempos coletados foi calculado o melhor tempo de *ts* em relação à *tc*, ou seja, quando *ts* – *tc* tiver o menor valor possível, então

$$ts(Qe) - tc(Qe) = 77{,}5364\,Qe - 124{,}3094 - (15{,}4473\,Qe - 19{,}5829) \quad (9)$$

portanto,

$$ts(Qe) - tc(Qe) = 62{,}0891\,Qe - 104{,}7265 \quad (10)$$

Então temos o valor mínimo, ou seja, o melhor valor de *ts* em relação à *tc* para

$$Qe = \frac{104{,}7265}{62{,}0891} = 1{,}6867 \quad (11)$$

substituindo *Qe* em *ts* - *tc*

$$ts(1{,}6867) - tc(1{,}6867) = 62{,}0891\,(1{,}6867) - 104{,}7265 \quad (12)$$

portanto,

$$ts(1{,}6867) - tc(1{,}6867) = -8{,}1503 \times 10^{-04} \quad (13)$$

Sendo assim, o algoritmo sequencial nesse ponto (melhor desempenho possível em relação ao algoritmo concorrente) são $8{,}1503 \times 10^{-04}$, o que equivale à zero milissegundos para *Qe* = 1,6867, portanto, não existe melhor desempenho para o algoritmo sequencial.

Agora utilizando o método da variação percentual, chamado de *Vp*, para saber o ganho percentual em relação aos resultados dos algoritmos sequencial e concorrente. Partindo dos polinômios e *Qe* = 200 (quantidade de execuções máxima), temos

$$Vp(Qe) = \frac{ts(Qe) - tc(Qe)}{tc(Qe)} \times 100 \quad (14)$$

$$Vp(200) = \frac{ts(200) - tc(200)}{tc(200)} \times 100 \quad = 401{,}0940\ \% \quad (15)$$

ocorreu uma diminuição percentual de 401,0940 % do tempo de execução do algoritmo concorrente em relação ao sequencial. O gráfico abaixo ilustra a comparação dos tempos de execução entre o algoritmo concorrente e sequencial.

Figura 5. Comparação entre os tempos de execução concorrente e sequencial modificando apenas a quantidade de execuções do produto escalar.

Agora fixando a quantidade de execuções do procuto escalar e variando os tamanhos dos vetores, foi utilizado a linha de tendência e encontrados polinômios de primeiro grau tanto para o algoritmo concorrente como para sequencial. Os polinômios são mostrados abaixo, sendo que *Tvet* representa os tamanhos dos vetores.

Figura 6. Tempo de execução do algoritmo concorrente com a quantidade de execuções fixa.

- Polinômio sequencial (percentual de exatidão 99,95%)

$$ts(Tvet) = 77{,}5579\,Tvet - 142{,}0909 \tag{16}$$

- Polinômio concorrente (percentual de exatidão 99,93%)

$$tc(Tvet) = 15{,}4566\,Tvet - 23{,}9366 \tag{17}$$

A partir dos polinômios, foi encontrada o tamanho o vetor *Tvet* = 1,9026, com esse valor, o algoritmo sequencial (melhor desempenho possível em relação ao algoritmo paralelo) são aproximadamente 0 milissegundos, portanto, não existe melhor desempenho para o algoritmo sequencial. Utilizando o método da variação percentual com *Tvet* = 985000 (tamanho máximo dos vetores), temos

$$Vp(985000) = \frac{ts(985000) - tc(985000)}{tc(985000)} \times 100 \quad = 401{,}7781\ \% \tag{18}$$

ocorreu uma diminuição percentual de 401,7781 % do tempo de execução do algoritmo concorrente em relação ao sequencial. O gráfico abaixo ilustra a comparação dos tempos de execução entre o algoritmo concorrente e sequencial.

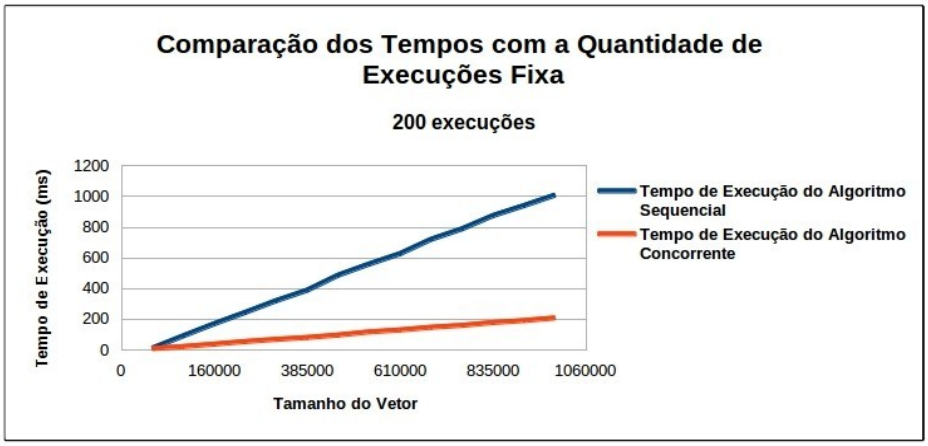

Figura 7. Comparação entre os tempos de execução concorrente e sequencial modificando apenas o tamanho do vetor.

3.2. Utilizando CUDA

Fixando-se os tamanhos dos vetores e variando a quantidade de execuções, foi utilizado

a linha de tendência e encontrados polinômios de primeiro grau tanto para o algoritmo paralelo como para sequencial. Os polinômios são mostrados abaixo, sendo que *ts* e *tp* representam, respectivamente, o tempo de execução do algoritmo sequencial e paralelo e, *Qe* representa a quantidade de execuções do produto escalar. O gráfico abaixo ilustra o tempo de execução do algoritmo paralelo.

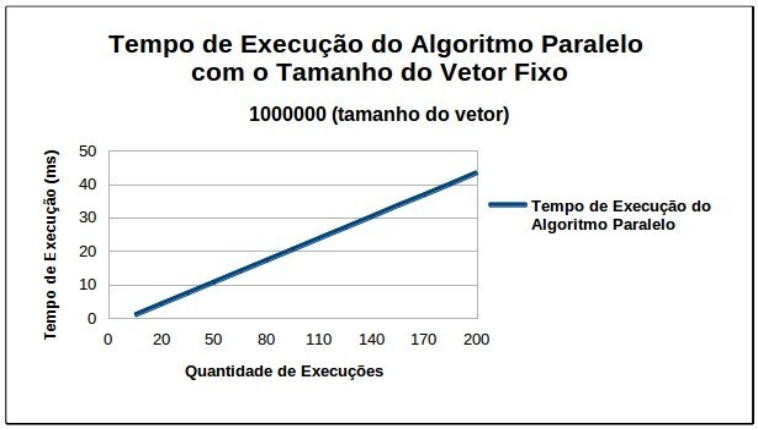

Figura 8. Tempo de execução do algoritmo paralelo com o tamanho do vetor fixo.

- Polinômio sequencial (percentual de exatidão 99,98%)

$$ts(Qe) = 77{,}5564\,Qe - 124{,}3094 \tag{19}$$

- Polinômio paralelo (percentual de exatidão 99,99%)

$$tp(Qe) = 3{,}2625\,Qe - 5{,}4426 \tag{20}$$

Foi encontrada uma quantidade de execução *Qe* = 1,5999, com esse valor, o algoritmo sequencial (melhor desempenho possível em relação ao algoritmo paralelo) são aproximadamente 0 milissegundos, portanto, não existe melhor desempenho para o algoritmo sequencial. Utilizando o método da variação percentual com *Qe* = 200 (quantidade de execução máxima), temos

$$Vp(200) = \frac{ts(200) - tp(200)}{tp(200)} \times 100 \quad = 2278{,}0833\ \% \tag{21}$$

ocorreu uma diminuição percentual de 2278,0833 % do tempo de execução do algoritmo paralelo em relação ao sequencial. O gráfico abaixo ilustra a comparação dos tempos de execução entre o algoritmo paralelo e sequencial.

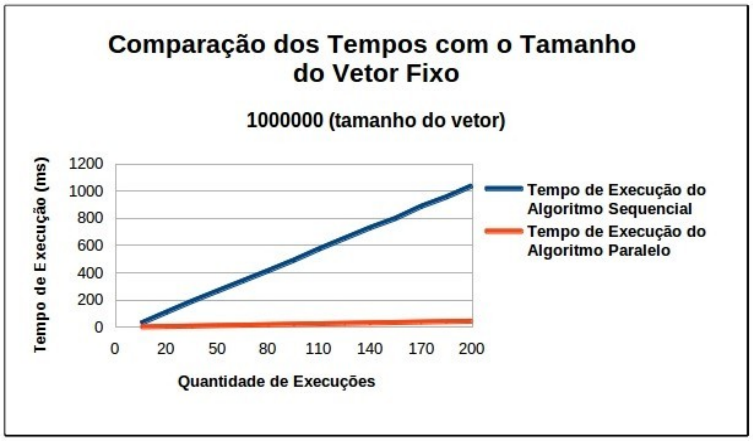

Figura 9. Comparação entre os tempos de execução paralelo e sequencial modificando apenas a quantidade de execuções do produto escalar.

Fixando-se a quantidade de execuções do procuto escalar e variando os tamanhos dos vetores.

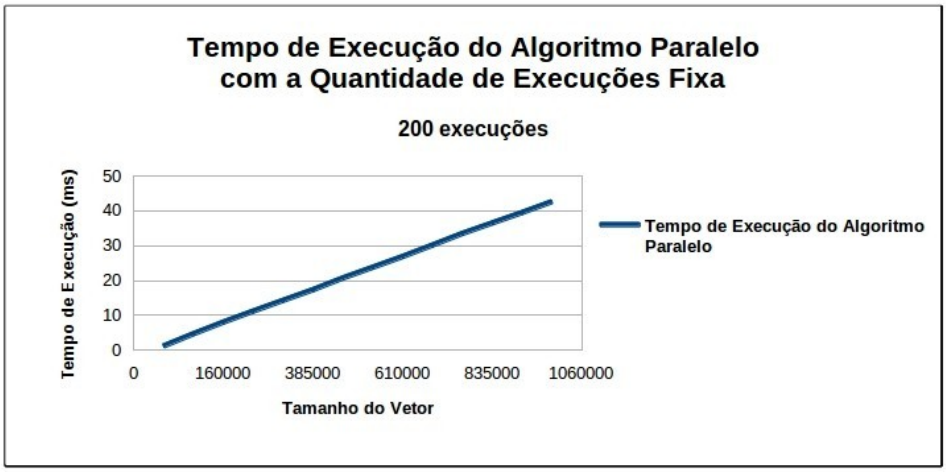

Figura 10. Tempo de execução do algoritmo paralelo com a quantidade de execuções fixa.

- Polinômio sequencial (percentual de exatidão 99,95%)

$$ts(Tvet) = 77{,}5579\,Tvet - 142{,}0909 \tag{22}$$

- Polinômio paralelo (percentual de exatidão 99,98%)

$$tp(Tvet) = 3{,}1798\,Tvet - 4{,}8626 \tag{23}$$

Foi encontrada o tamanho o vetor *Tvet* = 1,8450, com esse valor, o algoritmo sequencial (melhor desempenho possível em relação ao algoritmo paralelo) são aproximadamente 0 milissegundos, portanto, não existe melhor desempenho para o algoritmo sequencial. Utilizando o método da variação percentual com *Tvet* = 985000 (tamanho máximo dos vetores), temos

$$Vp(985000) = \frac{ts(985000) - tp(985000)}{tp(985000)} \times 100 \quad = 2339{,}0803\ \% \tag{23}$$

ocorreu uma diminuição percentual de 2339,0803 % do tempo de execução do algoritmo paralelo em relação ao sequencial. O gráfico abaixo ilustra a comparação dos tempos de execução entre o algoritmo paralelo e sequencial.

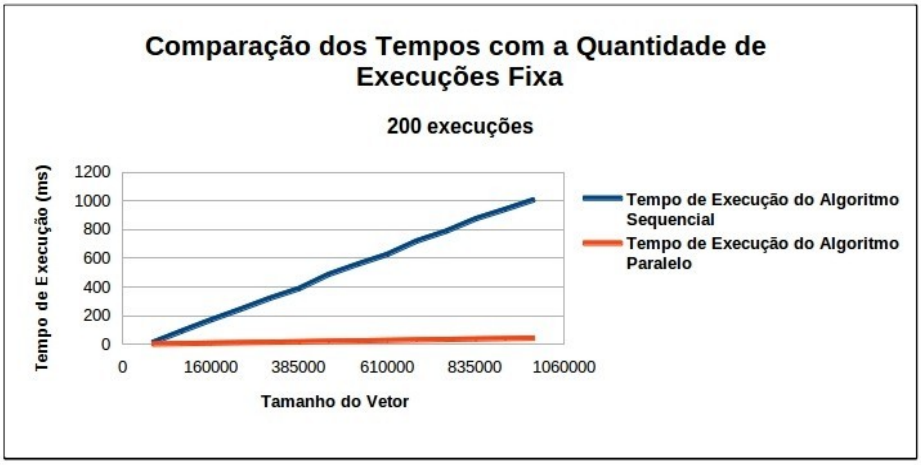

Figura 11. Comparação entre os tempos de execução paralelo e sequencial modificando apenas apenas o tamanho do vetor.

4. Conclusão

Esse artigo apresentou uma implementação do produto escalar em GPU utilizando linguagem de programação paralela CUDA e uma implementação concorrente usando a biblioteca OpenMP. A implementação com OpenMP gerou desempenho satisfatório, sendo bastante superior àquela usando programação sequencial, com melhoria de desempenho de cerca de 400% em um dos experimentos realizados. Utilizando CUDA, mesmo sendo feito somente a paralelização da multiplicação, foi satisfatório com uma diminuição percentual cerca de 2000% do tempo de execução do algoritmo paralelo em relação àquela usando programação sequencial. O uso de paralelismo voltado para aprendizado de máquina, particularmente para criação de classificadores, a paralelização do produto escalar mostrou-se apropriado.

Referências

Biezuner, R. J. (2009). Vetores, Universidade Federal de Minas Gerais, Departamento de Matemática, Belo Horizonte, Minas Gerais.

Sanders, J. and Kandrot, E. (2011). CUDA by Example: An Introduction to General-Purpose GPU Programming, Pearson Education, United States.

Sena, Maria C. R. and Costa, Joseaderson A. C. (2008). TUTORIAL OpenMP C/C++, Universidade Federal de Alagoas, SUN microsysterms.

Lages, E. N. (2007). Vetores – Produto Escalar, Universidade Federal de Alagoas, Faculdade de Arquitetura e Urbanismo, Alagoas.

Yano, G. A. (2010). Avaliação e comparação de desempenho utilizando tecnologia CUDA. São José do Rio Preto, São Paulo. Monografia (Ciência da Computação) – Universidade Estadual Paulista.