**Relatório de Entrega de Trabalho**

**Disciplina de Programação Paralela(PP) – Prof. Marcelo Veiga Neves**

**Alunos:** Bruno Carvalho, João Otávio Fanti **Usuários:** pp12802, pp12813

**Exercicio:** Trabalho 2 de OpenMP **Entrega**: 28/11/2017

**Introdução**

Para o segundo trabalho da disciplina de programação paralela, foi proposto a criação de um programa paralelo utilizando o OpenMP. Para realizarmos esta tarefa, escolhemos o algoritmo de ordenação de vetores Bubblesort.

Criamos um algoritmo sequencial e outro paralelo para que realizarmos uma comparação de tempos de execução, variando o tamanho do vetor e o número de threads utilizadas no algoritmo paralelo.

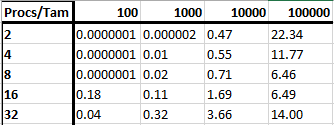
**Implementação**

A implementação deste trabalho foi feita em linguagem C, cujo compilador, “GCC”, possui suporte para utilização do “OpenMP”. Criamos uma implementação de Bubblesort diferente da clássica conhecida, para podermos realizar o aplicar o paralelismo.

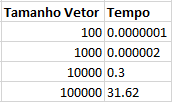
Primeiro comparamos os números a partir da posição 0, ou seja, 0 com 1, 2 com 3, 4 com 5, depois complementamos com as posições que não foram comparadas, 1 com 2, 3 com 4, 5 com 6, sem incluir o 0. Estas comparações devem acontecer uma quantidade de vezes igual à metade do tamanho do vetor, pois assim o vetor estará ordenado. Estas comparações são feitas desta maneira para evitar que haja concorrência ao utilizarmos múltiplas *threads* no algoritmo paralelo, porque assim quando um *thread* comparar 1 com 2, a próxima estará comparando 3 com 4, evitando a concorrência.

**Resultados**

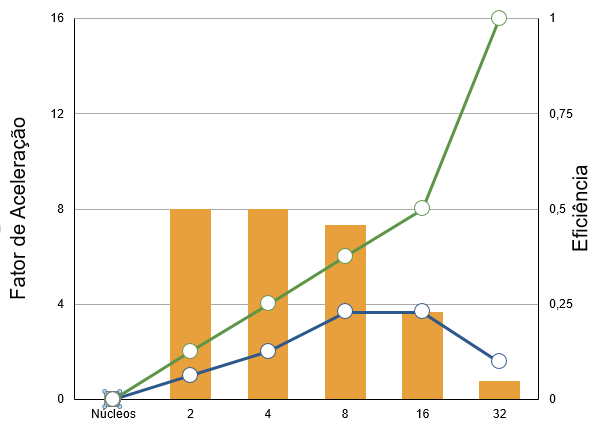
Os testes realizados, para obterem os resultados, envolveram vetores com 100, 1000, 10000 e 100000 elementos, e 2, 4, 8, 16, 32 *threads*. A planilha abaixo demonstra o tempo em segundos de cada uma das execuções:



Esta planilha representa o tempo em segundos do algoritmo paralelo. Para o algoritmo sequencial esses foram os valores:



Com estes valores é possível realizar o cálculo do *speed up*, para vermos o quanto que teve de ganho ao alterar o número de threads, utilizando 100000 elementos:



**Conclusao**

Com este trabalho, foi possível verificar que, com poucos elementos o aumento de *threads* não apresentou bom ganho, porém com mais elementos apresentou ganhos mais significativos.

**Código em C**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <time.h>

int main()

{

int ARRAY\_SIZE = 100000; //Tamanho do vetor

int vetor[ARRAY\_SIZE];

int i;

clock\_t t;

omp\_set\_num\_threads(1); //Numero de Threads

int debug = 0;

int j;

int controleVar = 0;

for (i=0 ; i<ARRAY\_SIZE; i++) //Populando o vetor

vetor[i] = ARRAY\_SIZE-i;

if(debug == 1){

printf("\nVetor Inicial: ");

for (i=0 ; i<ARRAY\_SIZE; i++)

printf("[%d] ", vetor[i]);

}

t = clock();

for (i = 0; i <= ARRAY\_SIZE; i++) //Executa as trocas uma quantidade

{ //de vezes igual ao tamanho do vetor

#pragma omp parallel for //Paraleliza as trocas

for (j = controleVar; j < ARRAY\_SIZE - 1; j = j + 2)

{

if (vetor[j] > vetor[j+1])

{

int aux1 = vetor[j];

vetor[j] = vetor[j+1];

vetor[j+1] = aux1;

}

}

if(controleVar == 1){ // Alterna entre as trocas a serem executadas

controleVar = 0;

}else{

controleVar = 1;

}

}

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

printf("\n\nOrdenado!!\n");

if(debug == 1){

printf("\nVetor Final: ");

for (i=0 ; i<ARRAY\_SIZE; i++)

printf("[%d] ", vetor[i]);

}

//printf("\nTempo de duracao: %f\n", time\_taken); //Tempo não está preciso quando tem //muitos elementos no array (100000)

printf("\n");

return 0;

}