ERAD-SP 2017 - Exercícios em OpenMP (curso básico)

1. Considerando a identidade matemática a seguir:

$$\int_1^u \frac{1}{x} dx = \ln u, \quad u > 1$$

Usando o método do trapézio para integração numérica tem-se o seguinte código demonstrado abaixo. Deve-se implementar um programa em C ou C++, usando *OpenMP*, que paralelize o código citado. Deve-se também elaborar um gráfico ou tabela indicando o tempo gasto obtido, *speedup* e eficiência, considerando $N=10^6$, $N=10^7$, $N=10^8$, e o número de *threads* variando de 1 até 8.
Obs.: Rode o programa pelo menos 3 vezes para cada quantidade de *threads* e assuma o tempo gasto como sendo a média dos tempos obtidos.

```
/* ********************************************************** */

#include <stdio.h>
#define N 100000000
#define u 2.0
int main(void) {
double passo, soma,x;
int i;
 passo = (u-1) / (double) N;
 for (i=0; i<N;i++) {
   x=1+i*passo;
   soma=soma+0.5*(1/x+1/(x+passo)); }
 printf("ln %f = %20.15f\n", u, passo*soma);
 return 0; }
/* ********************************************************/
```

2. Implemente através de OpenMP uma solução para calcular o Produto Escalar entre 2 vetores (do tipo *double*) de tamanho N (PE = A1*B1 + A2*B2 + ... + AN*BN). Calcule a taxa de MFLOPS, o *Speed-up* e a eficiência, considerando N variando de 10^3 até 10^5 e o número de *threads* de 1 até 8.
Obs: Preencha o vetor com números aleatórios (o maior valor pode ser determinado pelo programador), entretanto, ao invés de usar a função *rand()*, a qual não tem comportamento "*thread safe*", utilize a função *rand_r()* (http://uw714doc.sco.com/en/SDK_sysprog/PTL_ThdsMgmt_example.html)
3. O métodos dos mínimos quadrados é uma técnica padrão de otimização matemática para encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados

observados. O algoritmo consiste em encontrar uma equação do tipo: $y = mx + b$, onde, dado um conjunto de n pontos $\{(x1,y1), x2,y2),...,xn,yn)\}$, deve-se calcular:
* $SUMx = x1 + x2 + ... + xn$
* $SUMy = y1 + y2 + ... + yn$
* $SUMxy = x1*y1 + x2*y2 + ... + xn*yn$
* $SUMxx = x1*x1 + x2*x2 + ... + xn*xn$
* Sendo os valores de "m" e "b" calculados da seguinte forma:
* $m = (SUMx*SUMy - n*SUMxy) / (SUMx*SUMx - n*SUMxx)$
* $b = (SUMy - m*SUMx) / n$

A atividade deve ser feita a partir do programa serial disponibilizado, o qual lê um arquivo contendo:
a) na primeira linha a quantidade total de elementos.
b) nas linhas seguintes o conjunto de dados para os pares x e y.
Paralelize com OpenMP o código serial e verifique o desempenho obtido: tempo de execução, Speed-up e Eficiência. Mostre o resultado através de uma tabela ou gráfico.

4. Construir um programa paralelo com *OpenMP* para multiplicar duas matrizes quadradas de *NxN* elementos, onde $N=10^3$ e 10^4 de números do tipo *float*.
   Variar a quantidade de *threads*, no mínimo, da seguinte forma: 1, 2 e 4.
   Coloque em um gráfico ou tabela os tempos de execução, *Speedup*, Eficiência e *MFlops* obtidos variando-se o número de *threads* conforme pedido.

5. Paralelize o algoritmo de ordenação *Odd-Even Sort* (baseado no *bubble-sort*) com *OpenMP*:

```
void OddEven (int a[], int n) {
   int i, j, nHalf, lastEven, lastOdd;
   nHalf = n/2;
   if (n%2==0) {lastEven=nHalf-1; lastOdd=nHalf-2;}
   else        {lastEven=nHalf-1; lastOdd=nHalf-1;}

   for (i=0; i<n-1; i++) {
       for (j=0; j<=lastOdd; j++)
           ce(&a[2*j+1], &a[2*j+2]); /* odd */
       for (j=0; j<=lastEven; j++)
           ce(&a[2*j], &a[2*j+1]); /* even */ }
}
```

   Teste com diferentes números de *threads* (mínimo: 1, 2 e 4). Use tamanhos de conjuntos de dados onde o tempo de processamento serial demore, se possível, até um segundo.

6. Construa um algoritmo que, dada um vetor preenchido com números inteiros aleatórios cujos valores variam no intervalo de 0 até 999, construa um algoritmo

paralelo em *OpenMP* que conte quantas ocorrências de cada número foram encontrados. Utilize vetores com tamanho $N=10^8$. Faça testes variando o número de *threads* em 1, 2 e 4 e mostre tempo de processamento, *Speedup* e Eficiência. Ao final do programa, deve-se somar a quantidade de ocorrências para cada número encontrado e este valor deverá ser igual ao tamanho do vetor, ou seja, 10^8. Dica: ao paralelizar, tome cuidado para que duas ou mais *threads* ao achar a ocorrência de um dado número ao mesmo tempo não tenha problemas de inconsistência ao computar esta ocorrência.

7. Christian Goldbach (1690-1764) foi um matemático nascido na Prussia contemporânio de Euler. Um das suas mais famosas conjecturas ainda não devidamente provadas estabelece que todo número par maior que dois é a soma de dois números primos, por exemplo, 28 = 5 + 23.
   O programa disponibilizado calcula a primeira ocorrência de cada soma de dois primos encontrados para números que variam de 2 até 32000. Paralelize-o usando *OpenMP* introduzindo a cláusula "*for*" no primeiro laço (linha 19), usando a cláusula *Schedule* para modificar as diferentes formas de balanceamento de carga entre os processadores:
   a) STATIC,10
   b) STATIC, 5
   c) STATIC, 2
   d) DYNAMIC
   e) GUIDED
   Compare os tempos de processamento com 1, 2 e 4 *threads*. Compare ainda com o uso do laço paralelo sem nenhuma cláusula que mude o escalonamento.