**INE5645 – Prova 1 – Parte A - Paralelização em OpenMP - 12/04/2017
Nome : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Identifique os seguintes loops

Versão sequencial
double res[10000];
for (i=0 ; i < 10000 ; i++)
 calculo\_pesado(&res[i]);

Versão paralela
double res[10000];
#pragma omp parallel for
 for (i=0 ; i < 10000 ; i++)
 calculo\_pesado(&res[i]);

1. Explique, brevemente, a diferença de execução existente entre estes dois exemplos de loops.
2. Se um usuário estiver usando um processador *quad-core*, o que pode você afirmar com relação ao desempenho com a adição de apenas uma linha de código #pragma omp parallel for na versão paralela ?
3. Explique a diferença entre as cláusulas (atributos) shared e private

quanto ao compartilhamento dos dados entre threads, usando-se numa diretiva omp parallel.

1. Explique como funciona as funções:

(a) omp\_set\_num\_threads() e omp\_get\_thread\_num(), quando se constrói uma região paralela omp parallel.

(b) Antes da execução do programa, será necessário especificar o número de threads que irão participar da execução paralela. No caso de você não usar omp\_set\_num\_threads(), qual a maneira mais fácil de se obter o mesmo resultado num ambiente Linux ?
2. Definição de região paralela. Indique para os itens abaixo (Verdade/Falso).

OMP PARALLEL double A[10000];
omp\_set\_num\_threads(4);
#pragma omp parallel
 { int th\_id = omp\_get\_thread\_num();
 calculo\_pesado(th\_id, A);
 }
 printf(“Terminado”);
3. o início da execução das threads é sinalizado;
4. as threads são sincronizadas;
5. o vetor A é compartilhado;
6. Usou-se funções OpenMP, além das diretivas.
7. Sobre o código seguinte pode afirmar (Verdade/Falso)

#define N 10000;
int i;
#pragma omp parallel
 #pragma omp for
 for (i=0 ; i < 10000 ; i++)
 {
 calculo();
 }
 printf(“Terminado”);
8. As iterações são distribuídas entre as threads ?
9. Tem uma barreira implícita de sincronização entre threads no final do loop ?
10. omp for pode ser complementado pela schedule para especificar como fazer a distribuição da carga do for (i=0 ; i < 10000 ; i++) ?
11. Mais uma cláusula: reduction(op : list). Usada para operações tipo “all-to-one”. A cláusula reduction efetua uma operação de redução em uma lista de variáveis (pode ser somente uma veriável). Essas variáveis devem ser escalares, no contexto em que elas participarão.
 - exemplo com a operação de soma: op = ’+’
 - cada thread terá uma cópia da(s) variável(is) definidas em ’list’ com a
 devida inicialização;
 - ela efetuará a soma local com sua cópia;
 - ao sair da seção paralela, as somas locais serão automaticamente adicionadas na
 variável para proporcionar um soma total, ou seja, no final da construção paralela do
 loop, todas as threads irão adicionar o seu valor do resultado para thread mestre
 com a variável global.
#include <omp.h>
#define NUM\_THREADS 4
...
int i, tmp, res = 0;
#pragma omp parallel for reduction(+:res) private(tmp)
{
 for (i=0 ; i< 15 ; i++)
 {
 tmp = Calculo( );
 res += tmp ;
 }
 printf(“O resultado vale %d´´, res) ;
}

Obs: Os índices de loops sempre são privados.

1. A variável res deve ser shared ou private ? Explique.
2. Veja o código acima e explique o que se passa quanto a variável res, indicada em reduction(+:res). Dê um exemplo de como você entende, para 4 threads e i variando de 0 a 15.
3. Distribuição de trabalho com seções paralelas. Pode-se usar omp sections, quando não se usam loops:

#pragma omp parallel
{
 #pragma omp sections
 {
 Calculo1( );
 #pragma omp section
 Calculo2( );
 #pragma omp section
 Calculo3( );
 }
}
4. As seções são distribuídas entre as threads ou as thread são distribuídas nas seções paralelas ?
5. Explique como funciona a região paralela e as seções paralelas definidas.
6. Sincronizações - Existem algumas instruções para sincronizar os acessos à memória compartilhada:

 (a) **Seção crítica**
 - #pragma omp critical {. . . }
 - Apenas uma thread pode executar a seção crítica num dado momento.

 (b)**Atomicidade
 -** #pragma omp atomic

 **- <instrução atômica>;** - versão “light” da seção crítica.
 - funciona apenas para uma próxima instrução de acesso à memória.

 (c) **Barreira**
 #pragma omp barrier
 - barreiras implícitas nos fins das seções paralelas! master. Pode-se provocar
 barreiras.

 Procure dar um exemplo de cada caso (a), (b) e (c).

10 . Funções de biblioteca para *run-time* para locks:

(a) São diretivas ?

omp\_lock\_t , [omp\_init\_lock](https://msdn.microsoft.com/en-us/library/8xybk13s.aspx), [omp\_destroy\_lock](https://msdn.microsoft.com/en-us/library/5ksya513.aspx), [omp\_set\_lock](https://msdn.microsoft.com/en-us/library/heh3f0a3.aspx), [omp\_unset\_lock](https://msdn.microsoft.com/en-us/library/8w00t74d.aspx), [omp\_test\_lock](https://msdn.microsoft.com/en-us/library/6e1yztt8.aspx)

#include <stdio.h>

#include <omp.h>

omp\_lock\_t my\_lock;

int main() {

 omp\_init\_lock(&my\_lock);

 #pragma omp parallel num\_threads(4)

 {

 int tid = omp\_get\_thread\_num( );

 int i, j;

 for (i = 0; i < 5; ++i) {

 omp\_set\_lock(&my\_lock);

 printf\_s("Thread %d - starting locked region\n", tid);

 printf\_s("Thread %d - ending locked region\n", tid);

 omp\_unset\_lock(&my\_lock);

 }

 }

 omp\_destroy\_lock(&my\_lock);

}

Explique o que ocorre no código acima.