Computação de Alto Desempenho 2013/14
Exame -- Parte A – 23/6/2014
Duração: 1h30

O exame é sem consulta e em caso de dúvida na interpretação do enunciado, deve explicitar todos os pressupostos na elaboração das suas respostas.

1. A aceleração ou *speedup* de um programa executado numa arquitectura com n processadores é o quociente entre o tempo de execução da versão sequencial e o tempo de execução da versão paralela com esses n processadores. De acordo com a lei de Amdalh, diga qual é a maior limitação para o speedup de programas embaraçosamente paralelos, e se esta limitação se mantém na perspectiva da lei de Gustafson-Baris. Justifique.

2. Considere o algoritmo de ordenação "*odd-even transposition sort*" estudado nas aulas. Aplique a metodologia de Foster ao desenvolvimento de uma solução com base em *pthreads.* Na fase de "mapping" assuma que faz o mapeamento para um multiprocessador de memória partilhada com 2 CPUs. Considere que o código base do algoritmo é o seguinte:

```
void Odd_even_sort(
        int  a[]  /* in/out */,
        int  n    /* in     */) {
   int phase, i, temp;

   for (phase = 0; phase < n; phase++)
      if (phase % 2 == 0) { /* Even phase */
         for (i = 1; i < n; i += 2)
            if (a[i-1] > a[i]) {
               temp = a[i];
               a[i] = a[i-1];
               a[i-1] = temp;
            }
      } else { /* Odd phase */
         for (i = 1; i < n-1; i += 2)
            if (a[i] > a[i+1]) {
               temp = a[i];
               a[i] = a[i+1];
               a[i+1] = temp;
            }
      }
} /* Odd_even_sort */
```

3. Considere o seguinte código em OpenMP:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

#define MAXC 256
#define MAXV 2048
```

```
int main()
{
	int i, vec[MAXV], h[MAXC];
	time_t tlock;

	srand(time(&tlock));

	for(i=0; i< MAXV; i++)
		vec[i] = rand()%MAXC;

	#pragma omp parallel for
	for(i=0; i<MAXV; i++)
		h[vec[i]] +=1;

	// outro codigo...

	return 0;
}
```

Diga se o código está correcto e se é o mais eficiente possível. Em caso negativo, implemente uma versão correcta que seja o mais eficiente possível.

4. Escreva um programa em *OpenMP* que implemente o produto interno de dois vectores *V1* e *V2*, ambos com dimensão *N*. Para tal complete o seguinte esqueleto de código em C:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define N  1000
int vec1[N], vec2[N];
int result;

// O seu código

int main (int argc, char *argv[])
{
   FILE *f;
   // Leitura dos resultados
   f = fopen("input.bin", "r");
   fread(vec1, sizeof(int), N, f);
   fread(vec2, sizeof(int), N, f);
   fclose(f);

   // O seu código

    // Escrita do resultado
   printf("Produto interno: %d\n", result);
   return 0;
}
```

5. Considere a paralelização do cálculo de uma superfície de Mandelbrot usando *OpenMP*, supondo que está disponível a seguinte função:

```
#define max_iterations 255
int compute_point(double ci, double cr) {
```

```
    int iterations = 0;
    double zi = 0;
    double zr = 0;
    while ((zr*zr + zi*zi < 4) && (iterations < max_iterations))
    {
         double nr, ni;
         nr = zr*zr - zi*zi + cr; ni = 2*zr*zi + ci;
         zi = ni; zr = nr;
         iterations ++;
    }
    return iterations;
}
```

Discuta as várias alternativas da cláusula *schedule,* do ponto de vista do desempenho obtido no programa paralelizado com *OpenMP*.

Anexo

A. Algumas funções da biblioteca de Pthreads

`int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`
`int pthread_join (pthread_t thread, void **retval)`
`int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)`
`int pthread_mutex_lock (pthread_mutex_t *mutex)`
`int pthread_mutex_unlock (pthread_mutex_t *mutex)`
`int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr)`
`int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)`
`int pthread_cond_signal(pthread_cond_t *cond)`