Exame de Programação Paralela e Distribuída

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
20 de Fevereiro de 2009
Duração: 2 horas e 30 minutos

1. Um dos métodos mais conhecidos para desenhar algoritmos paralelos é a metodologia de programação de Ian Foster. Diga em que consistem as etapas de *Aglomeração* e de *Mapeamento* da metodologia de Ian Foster e descreva de forma sucinta as principais características e conceitos relacionados com cada uma dessas etapas.

2. Explique a noção de *thread* e enumere as possíveis vantagens e inconvenientes de utilizar um modelo de programação *multithreaded*.

3. Tendo por base o modelo de programação MPI, considere o seguinte extracto de código em que se utiliza a função `MPI_Type_vector` para construir um novo tipo de dados para ser utilizado na comunicação entre os processos `source` e `dest` do comunicador `comm`:

```
MPI_Type_vector(COUNT, 1, STRIDE, MPI_INT, &new_datatype);
MPI_Type_commit(&new_datatype);
if (my_rank == source)
  MPI_Send(&buffer_source, 1, new_datatype, dest, tag, comm);
else if (my_rank == dest)
  MPI_Recv(&buffer_dest, COUNT, MPI_INT, source, tag, comm, &status);
```

   Explique o conceito e a importância do tipo de dados nas comunicações MPI e justifique porque é que no exemplo acima os processos `source` e `dest` conseguem comunicar correctamente, apesar de utilizarem tipos de dados diferentes nas funções de envio (`new_datatype`) e de recepção (`MPI_INT`).

4. Uma barreira é um ponto do programa onde um grupo de processos sincroniza à espera que todos os membros do grupo atinjam esse ponto. Escreva um procedimento de nome `my_barrier()` que utilize o modelo de programação MPI para implementar o mecanismo de barreira para os processos do comunicador MPI_COMM_WORLD. Não utilize comunicações colectivas, apenas funções de comunicação ponto-a-ponto.

5. No openMP, as variáveis duma região paralela podem ser privadas ou partilhadas. Indique justificando qual é o âmbito das variáveis no bloco de código que se segue e identifique eventuais incorrecções/problemas.

```
#pragma omp parallel for firstprivate(x) private(b)
for (i = 0; i < N; i++) {
  int c = 0;
  for (j = i; j <= N; j++)
    c += b[j];
  a[i] = c * x--;
}
```

6. Identifique as dependências *loop-carried* existentes no bloco de código que se segue, classifique-as quanto ao tipo de fluxo de dados (*flow dependence/anti dependence/output dependence*) e escreva uma versão paralela do código em OpenMP com as dependências removidas (assuma que o valor da variável `i` não é necessário na continuação da computação).

```
for (i = 0; i < N; i++) {
  x = x + a[i];
  b[i] = b[i] * x;
  c[i] = c[i + 1] + b[i];
}
```

7. Tanto a lei de Amdahl como a lei de Gustafson-Barsis são derivadas a partir da mesma expressão genérica de *speedup*. No entanto, à medida que se aumenta o número de processadores, pela lei de Amdahl o *speedup* está limitado a 1/f, enquanto que pela lei de Gustafson-Barsis aparentemente o *speedup* pode crescer indefinidamente. Explique por que é que isso acontece.