Exame de Programação Paralela e Distribuída

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
4 de Fevereiro de 2010
Duração: 2 horas e 30 minutos

1. Em programação paralela é habitual designar a execução de uma aplicação como utilizando uma estratégia de decomposição do domínio ou como utilizando uma estratégia de decomposição funcional. Diga em que consiste o processo de decomposição de uma aplicação e enumere o que caracteriza e distingue a estratégia de decomposição do domínio da estratégia de decomposição funcional.

2. Considere o seguinte extracto de código MPI:

```
if (my_rank == 0) {
  MPI_Send(&rows, 1, MPI_INT, 1, tag, comm);
  MPI_Send(&cols, 1, MPI_INT, 1, tag, comm);
  MPI_Send(matrix, rows * cols, MPI_INT, 1, tag, comm);
} else if (my_rank == 1) {
  MPI_Recv(&rows, 1, MPI_INT, 0, tag, comm, &status);
  MPI_Recv(&cols, 1, MPI_INT, 0, tag, comm, &status);
  matrix = (int *) malloc(rows * cols * sizeof(int));
  MPI_Recv(matrix, rows * cols, MPI_INT, 0, tag, comm, &status);
}
```

   Utilizando outras funções da biblioteca MPI, elabore uma solução alternativa em que exista apenas uma única comunicação entre os processos 0 e 1. Apresente uma breve discussão sobre como o código acima se compara em termos de eficiência com a sua solução alternativa.

3. A matriz transposta de uma matriz $M_{L\times C}$ é a matriz $T_{C\times L}$ que resulta da troca das linhas por colunas da matriz M, ou seja, em que todos os elementos de M verificam a condição $M(i,j) = T(j,i)$. Dadas duas matrizes $M_{N\times N}$ e $T_{N\times N}$ de valores inteiros, considere o problema de determinar o número de elementos que não verificam a condição de T ser a transposta de M e vice-versa, ou seja, o número de elementos que não verifica a condição $M(i,j) = T(j,i)$. Escreva um programa que tire partido do modelo de programação MPI para implementar uma solução utilizando P processadores (assuma que P divide N) e explique resumidamente a sua solução. O processo 0 deve ser o responsável por carregar os valores de N e das matrizes M e T através da chamada à função `load_values()` e de apresentar o resultado através da chamada à função `show_result(int n_elements)`.

4. Considere o seguinte extracto de código OpenMP:

```
#pragma omp parallel private(i)
{ int x = 0;
  #pragma omp for
  for (i = 0; i < N; i++)
    x += func(i);
  #pragma omp critical
  y += x;
}
```

   Diga qual é o tipo de operação sobre a variável `y` que o código acima implementa e elabore uma solução alternativa utilizando cláusulas específicas da biblioteca OpenMP para esse tipo de operação. Apresente uma breve discussão sobre como o código acima se compara em termos de eficiência com a sua solução alternativa.

5. Considere que $n >= f(p)$ representa a relação de isoeficiência de uma determinada aplicação paralela (para um problema de tamanho n e p processadores) e que $M(n)$ representa a memória necessária para resolver um problema de tamanho n. Em termos de memória, ordene da mais para a menos escalável as seguintes aplicações paralelas:

   - $f(p) = C * p \quad$ e $\quad M(n) = n^2$
   - $f(p) = C * \sqrt{p} \quad$ e $\quad M(n) = n^2$
   - $f(p) = C * p * \log p \quad$ e $\quad M(n) = n$

6. Descreva e exemplifique o funcionamento do método *bitonic-sort*.

7. Diga o que entende por uma estratégia de *scheduling* e que factores influenciam o desenho de uma boa estratégia.