Exame de Programação Paralela e Distribuída

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

1. Em programação paralela é habitual designar a execução de uma aplicação como sendo de granularidade fina ou grossa. Diga em que consiste a granularidade de uma aplicação, o que caracteriza o facto da granularidade ser fina ou grossa e enumere as principais vantagens/inconvenientes associadas a cada tipo de granularidade.

2. Considere o seguinte extracto de código MPI:

```
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
x = compute_something(my_rank);
if (my_rank != 0)
  MPI_Send(&x, 1, MPI_INT, 0, world_tag, MPI_COMM_WORLD);
else {
  y = x;
  for (i = 1; i < world_size; i++) {
    MPI_Recv(&x, 1, MPI_INT, i, world_tag, MPI_COMM_WORLD, &status);
    y += x;
  }
}
```

   Diga qual é o tipo de operação sobre a variável `y` que o código acima implementa e elabore uma solução alternativa utilizando funções específicas da biblioteca MPI 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.

3. Um *saddle point* de uma matriz é um elemento que satisfaz a condição de ser o menor elemento da sua coluna e o maior elemento da sua linha. Dada uma matriz $M_{N\times N}$ de valores inteiros, considere o problema de calcular os *saddle points* dessa matriz, caso existam. 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 da matriz M através da chamada à função `load_values()`. Para apresentar o valor e as coordenadas de um *saddle point* utilize a função `show_saddle_point(int val, int x, int y)`.

4. 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 apenas o valor da variável `i` não é necessário na continuação da computação).

```
for (i = 1; i <= N; i++) {
  x = 2 * i;
  a[i] += x;
  b[i] += a[i - 1];
}
```

5. Por experimentação, verificou-se que a versão paralela de uma determinada aplicação permite obter um *speedup* de 6 quando executada numa máquina multiprocessador com 11 processadores. Considerando que a versão paralela da aplicação é constituída por partes puramente sequenciais e por partes totalmente paralelizáveis (sem qualquer custo de iniciação, comunicação ou sincronização), determine qual é a fracção da aplicação inicial que foi paralelizada.

6. O algoritmo *bublesort* (método da bolha) é um algoritmo muito conhecido de ordenação com ordem de complexidade $\mathcal{O}(n^2)$. Descreva e ilustre o funcionamento de um método relativamente eficiente de paralelizar este algoritmo.

7. Descreva qual a vantagem de uma estratégia de *scheduling* dinâmica, distinga entre estratégias centralizadas e distribuídas e dê um exemplo de estratégias em cada uma destas categorias.