Exame de Programação Paralela e Distribuída

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
22 de Janeiro de 2008
Duração: 2 horas e 30 minutos

Parte I
(responda em folhas separadas a cada uma das partes)

1. As arquitecturas MIMD (*Multiple Instruction Multiple Data*), tal como definidas pela taxonomia de Flynn, podem ser subdivididas em duas grandes classes: as arquitecturas multi-processador (*multiprocessors*) e as arquitecturas multi-computador (*multicomputers*). Indique as características, vantagens e inconvenientes que permitem diferenciar estas duas classes de arquitecturas.

2. Apesar da diversidade de problemas aos quais podemos aplicar a programação paralela, a implementação de algoritmos paralelos está condicionada pelo modelo de memória existente. Enumere as características que permitem diferenciar os modelos de programação em memória partilhada e de programação em memória distribuída.

3. Explique os conceitos e as diferenças entre comunicações síncronas e assíncronas no modelo de programação MPI. Justifique em qual destes dois tipos de comunicação se enquadra a função de envio *standard* de mensagens `MPI_Send`.

4. Em OpenMP existem 3 declarações principais de âmbito de variáveis: `shared`, `private` e `reduction`. Explique o significado de cada uma das declarações e dê um exemplo em que devam ser usadas. Diga ainda qual ou quais destas declarações podem dar origem a competição entre processos/threads?

5. Considere que uma determinada aplicação executa em 3200 segundos em 32 processadores. Segundo a lei de Gustafson-Barsis, qual é o speedup máximo da aplicação sabendo que por experimentação verificou-se que 10% do tempo de execução é passado em computações sequenciais?

Parte II
(responda em folhas separadas a cada uma das partes)

1. Dado um vector v de tamanho N, considere o problema de testar se cada valor do vector v é ou não primo num anel de P processadores (suponha N divisível por P) e escreva um programa que utilize e tire partido do modelo de programação MPI para implementar uma solução. Considere que N e o vector v são iniciados pelo processo 0 através da chamada à função `load_vector()`. Para testar se um número é primo pode utilizar a função `is_prime(int number)`.

2. Considere a implementação de uma fila de tarefas tal como se apresenta no código que se segue. Tendo por base o modelo Pthreads, complete os procedimentos `get_work()` e `put_work()` de modo a implementar um mecanismo de sincronização no acesso à fila sem espera activa quando a fila fica momentaneamente sem tarefas disponíveis. Utilize o procedimento `list_is_empty(my_list)` para testar se existem tarefas na fila. Caso pretenda utilizar variáveis globais, declare-as e inicie-as fora dos procedimentos.

```
void main_thread_function (LIST *my_list) {
  TASK *my_task;
  while (TRUE) {
    my_task = get_work(my_list);
    do_work(my_task);
  }
}

TASK *void get_work(LIST *my_list) {
  TASK *my_task;
  ...
  my_task = get_task(my_list);  // retira uma tarefa de my_list
  ...
  return my_task;
}

void put_work(TASK *my_task, LIST *my_list) {
  ...
  put_task(my_task, my_list);  // coloca a tarefa my_task em my_list
  ...
}
```