Programação Paralela

Parte I

Fundamentos

Ricardo Rocha
ricroc@ncc.up.pt
DCC-FCUP

Tópicos Avançados de Informática
Mestrado em Informática 2003/04

Programação Paralela

"Se um único computador consegue resolver um problema em 10 segundos, podem 10 computadores resolver o mesmo problema em 1 segundo?"

No que consiste a **Programação Paralela**?

- **Concorrência:** identificar partes da computação que possam ser executadas em qualquer ordem sem alterar o resultado final.
- **Scheduling:** distribuir de forma eficiente a computação pelos vários processadores disponíveis.
- **Comunicação e Sincronização:** desenhar a computação para ser executada em simultâneo nos vários processadores.

Isto pressupõe a existência de **máquinas paralelas/distribuídas** que permitam ao programador explorar o paralelismo potencial existente nas aplicações. Existem vários modelos de programação paralela, mas nem todos se adequam a qualquer tipo de máquina. A arquitectura condiciona o tipo de problemas a explorar de forma eficiente.

Paralelismo Implícito

O paralelismo diz-se implícito quando cabe ao compilador e ao próprio sistema de execução:

- Detectar o paralelismo potencial do programa.
- Atribuir as tarefas para execução em paralelo.
- Controlar e sincronizar toda a execução.

Vantagens e inconvenientes:

- **(+)** Liberta o programador dos detalhes da execução paralela.
- **(+)** Solução mais geral e mais flexível.
- **(–)** Difícil conseguir-se uma solução eficiente para todos os casos.

Paralelismo Explícito

O paralelismo diz-se explícito quando cabe ao programador:

- Anotar as tarefas para execução em paralelo.
- Atribuir (possivelmente) as tarefas ao processadores.
- Controlar a execução, indicando nomeadamente os pontos de sincronização.
- Conhecer a arquitectura da máquina de forma a conseguir melhor performance (aumentar localidade, diminuir comunicação, etc.)

Vantagens e inconvenientes:

- **(+)** Programadores experientes produzem soluções muito eficientes para problemas específicos.
- **(–)** O programador é o responsável por todos os detalhes da execução (debugging pode ser deveras penoso).
- **(–)** Pouco portável entre diferentes arquitecturas.

Explorar Paralelismo: Decomposição e Partição

A computação a ser realizada bem como os dados sobre o qual a computação vai actuar podem ser decompostos em partições mais pequenas por forma a diminuir a complexidade do problema.

Estratégias comuns de decomposição:

- **Decomposição do Domínio:** determinar uma partição apropriada para os dados associados ao programa, e depois conceber um processo de associar a computação com a partição.

  Exemplo: multiplicação de matrizes ($A_{n\times m} \times B_{m\times p} = C_{n\times p}$)

- **Decomposição Funcional:** primeiro decompõe-se a computação a ser realizada e só depois se pensa nos dados.

  Exemplo: sucessão de Fibonacci (fib(n) = fib(n - 1) + fib(n - 2))

Decomposição do Domínio: Tipos de Partição

- **Partição Regular:** divisão dos dados num número fixo de partes, de igual tamanho, e com padrões de comunicação bem estruturados.
- **Partição Irregular:** uso de heurísticas na divisão dos dados, em domínios irregulares, e com padrões de comunicação não estruturados.
  - **Bissecção Recursiva:** partição do domínio (e.g. malha de elementos finitos) recursivamente (e.g. usando uma técnica *divide-and-conquer*) em subdomínios de custo aproximadamente igual e simultaneamente minimizando custos de comunicação.
- **Partição Adaptativa:** método mais complexo, em que o domínio é continuamente decomposto com base em resultados da computação.

O Problema da Granularidade

Num sistema distribuído ou paralelo, tiramos partido do paralelismo potencial quando dividimos um programa em tantas partes quanto possível e as executamos em simultâneo em mais do que um processador.

- Como dividir um programa em tarefas?

  Uma tarefa é, por definição, uma parte do programa que não pode ser subdividida.

- Qual a melhor estratégia para atingir máxima performance?

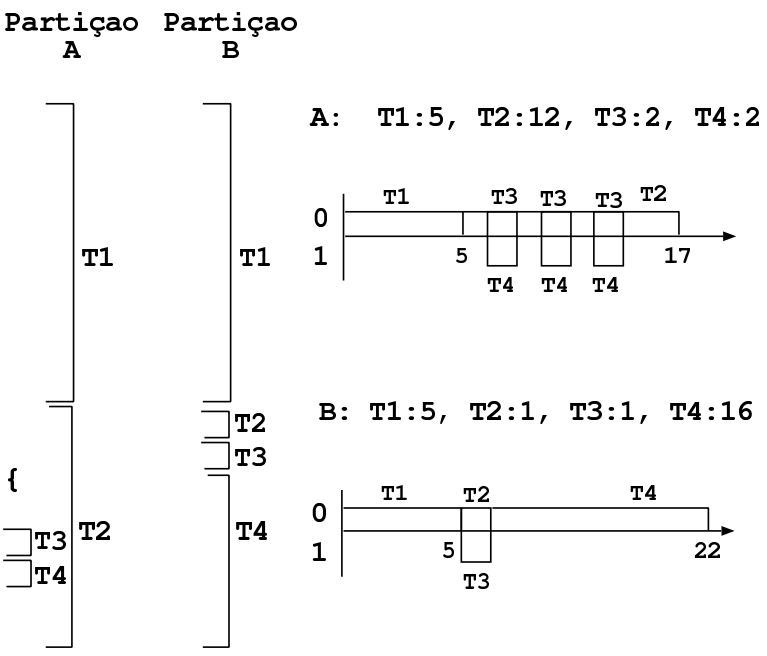
  Juntam-se algumas tarefas mais pequenas ou executam-se separadamente.

- O tamanho das tarefas condiciona a estratégia de execução?

  As tarefas podem ter granularidade baixa, média ou alta. A granularidade das tarefas é importante porque o seu tempo de execução tem de compensar os custos de criação e comunicação.

O Problema da Granularidade: Exemplo

```
nothing()
{
  input(A,B,C);
  if (A>B) {
    C= A-B;
    output(C);
  }
  else {
    C= B-A;
    output(A,B,C);
  }
  A= 0;
  B= 1;
  for (i=1; i<=3; i++) {
    input(C);
    A= A+C;
    B= B*C;
  }
  output(A,B,C);
}
```



O Problema da Granularidade: Outro Exemplo

Multiplicação de matrizes: $A_{n \times m} \times B_{m \times p} = C_{n \times p}$

Os elementos de C são calculados por: $C_{i,j} = \sum_{k=1}^{m} A_{i,k} \times B_{k,j}$

Numa implementação, os elementos de C poderiam ser calculados por:

```
for (i = 1; i <= n; i++)
  for (j = 1; j <= p; j++)
    for (k = 1; k <= m; k++)
      C[i][j] += A[i][k] * B[k][j];
```

Paralelização da multiplicação de matrizes:

- Paralelizar os dois ciclos exteriores (`i` e `j`): equivale a ter 1 processador a calcular 1 elemento da matriz resultado (granularidade muito baixa).
- Paralelizar o ciclo exterior (`i`): equivale a ter 1 processador a calcular 1 linha da matriz resultado (granularidade média).

Padrões de Comunicação

As tarefas geradas por uma partição nem sempre são independentes. A execução de uma tarefa pode envolver o acesso a valores calculados por outras tarefas. Para haver cooperação entre tarefas concorrentes é necessário existir comunicação.

Padrões de comunicação:

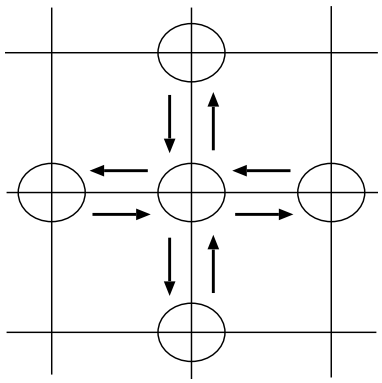
- Comunicação local x comunicação global
- Comunicação estruturada x comunicação não estruturada
- Comunicação estática x comunicação dinâmica
- Comunicação síncrona x comunicação assíncrona

Padrões de Comunicação

- **Comunicação Local:** comunicação restrita a tarefas vizinhas.
- Exemplo: método de Jacobi de diferenças finitas (2 dimensões).

$$X_{ij}^{t+1} = \frac{4X_{ij}^t + X_{i-1j}^t + X_{i+1j}^t + X_{ij-1}^t + X_{ij+1}^t}{8}$$

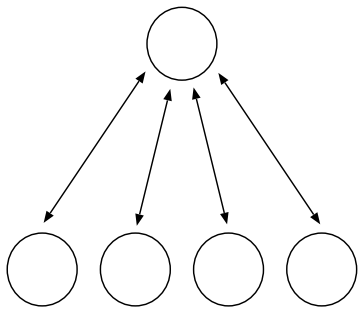
Estrutura da comunicação:



Padrões de Comunicação

- **Comunicação Global:** todas as tarefas podem comunicar entre si.
- Exemplo: modelo propagar/recolher.

Estrutura da comunicação:



Padrões de Comunicação

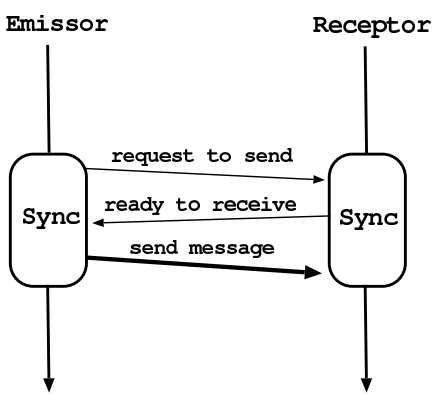
- **Comunicação Estruturada:** tarefas vizinhas constituem uma estrutura regular (e.g. árvore ou rede).
- **Comunicação Não Estruturada:** comunicação entre tarefas pode constituir um grafo arbitrário.

- **Comunicação Estática:** os parceiros de comunicação não variam durante a execução.
- **Comunicação Dinâmica:** a comunicação é determinada pela execução e pode ser muito variável.

Padrões de Comunicação

- **Comunicação Síncrona:** produtores e consumidores executam de forma coordenada, com pares produtor/consumidor a cooperarem na transferência de dados.
- Exemplo: protocolo das 3-fases ou *rendez-vous*. Comunicação através de um canal que apenas se concretiza quando os dois processos emissor e receptor estão sincronizados.

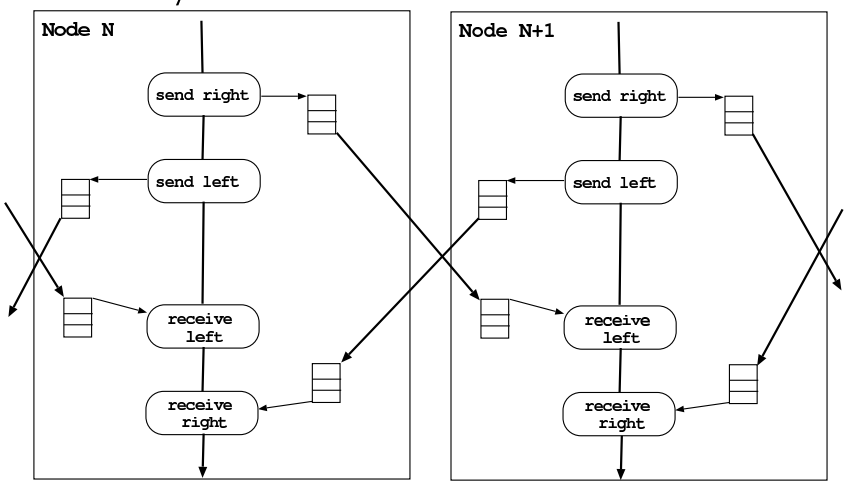
  Estrutura da comunicação:



Padrões de Comunicação

- **Comunicação Assíncrona:** produtores e consumidores executam de forma independente não necessitando de sincronizar para transferir dados.
- Exemplo: *buffering* de mensagens. O envio de mensagens através de um canal não interfere com o processo receptor. As mensagens são guardadas em *buffers*.

  Estrutura da comunicação:



Factores Limitativos de Performance

- **Granularidade das tarefas:** o tamanho das tarefas é importante porque o tempo que demoram a ser executadas tem de compensar os custos de execução em paralelo.
- **Código sequencial excessivo:** existem partes do código que são inerentemente sequenciais (iniciar/instanciar novas tarefas).
- **Contenção no acesso à memória:** resulta da partilha de grandes estruturas de dados por um grande número de processadores.
- **Atraso na comunicação/sincronização:** em sistemas de memória distribuída a comunicação é por natureza mais lenta o que pode afectar o desempenho.
- **Carga não balanceada:** ter os processadores igualmente ocupados durante a execução é muito importante.

Principais Modelos de Programação Paralela

Memória Partilhada

- Programação usando processos ou threads.
- Estratégias de decomposição: do domínio ou funcional.
- Granularidade das tarefas: alta, média ou baixa.
- Comunicação através de memória partilhada.
- Sincronização através de mecanismos de exclusão mútua (locks, semáforos, etc).

Memória Distribuída

- Programação usando mensagens.
- Paralelismo mais frequente: decomposição do domínio com granularidade alta.
- Comunicação e sincronização por troca de mensagens.