12 – Programação Paralela (cont.)

Multicomputers

Um multicomputer é um conjunto de computadores ligados por rede em que cada computador tem acesso exclusivo à sua memória física.

- O espaço de endereçamento de cada computador não é partilhado pelos restantes computadores, ou seja, não existe o conceito de espaço de endereçamento global.
- As alterações sobre uma posição de memória realizada por um determinado processador não são visíveis pelos processadores dos restantes computadores, ou seja, não existe o conceito de coerência das caches.

12 – Programação Paralela (cont.)

Multicomputers

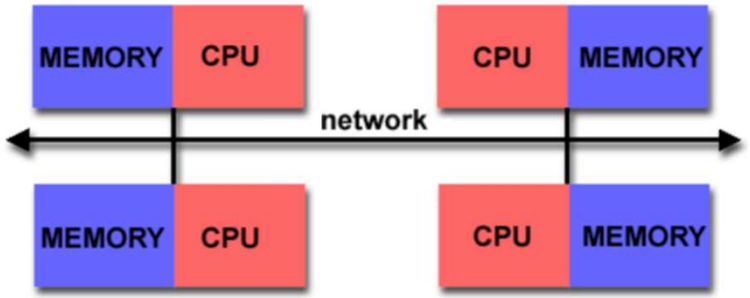
Existem duas grandes classes de multicomputers:

- Distributed Multicomputer
- Distributed-Shared Multicomputer

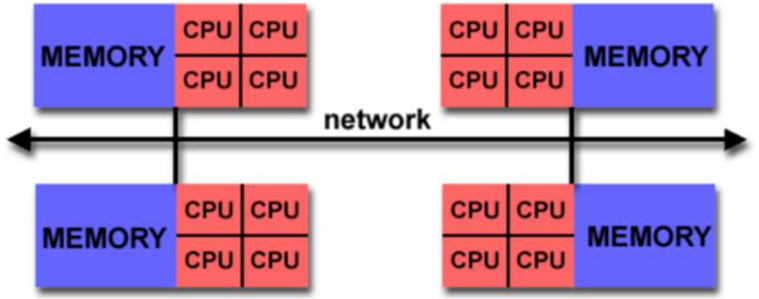
12 – Programação Paralela (cont.)

Multicomputers

- *Distributed Multicomputer*



- *Distributed-Shared Multicomputer*



12 – Programação Paralela (cont.)

Multicomputers

Vantagens e inconvenientes:

- (+) O aumento do número de computadores aumenta proporcionalmente a memória disponível sem necessitar de mecanismos de coerência das caches.
- (+) Fácil escalabilidade a baixo custo. O aumento do poder de computação pode ser conseguido à custa de computadores de uso doméstico.
- (–) Necessita de mecanismos de comunicação para partilha de dados entre tarefas de diferentes computadores.
- (–) O tempo de acesso aos dados entre diferentes computadores não é uniforme e é por natureza mais lento.
- (–) Pode ser difícil converter estruturas de dados previamente existentes para memória partilhada em estruturas de dados para memória distribuída.

12 – Programação Paralela (cont.)

Programação Paralela

Apesar das arquiteturas paralelas serem atualmente uma realidade, a programação paralela continua a ser uma tarefa complexa. Para além de depender da disponibilidade de ferramentas/ambientes de programação adequados para memória partilhada/distribuída, debate-se com uma série de problemas não existentes em programação sequencial:

12 – Programação Paralela (cont.)

Programação Paralela

- **Concorrência:** identificar as partes da computação que podem ser executadas em simultâneo.
- **Comunicação e Sincronização:** desenhar o fluxo de informação de modo a que a computação possa ser executada em simultâneo pelos diversos processadores evitando situações de deadlock e race conditions.
- **Balanceamento de Carga e Escalonamento:** distribuir de forma equilibrada e eficiente as diferentes partes da computação pelos diversos processadores de modo a ter os processadores maioritariamente ocupados durante toda a execução.

12 – Programação Paralela (cont.)

Metodologia de Programação Foster

Um dos métodos mais conhecidos para desenhar algoritmos paralelos é a metodologia de Ian Foster (1996). Esta metodologia permite que o programador se concentre inicialmente nos aspectos não-dependentes da arquitetura, como sejam a concorrência e a escalabilidade, e só depois considere os aspectos dependentes da arquitetura, como sejam aumentar a localidade e diminuir a comunicação da computação.

12 – Programação Paralela (cont.)

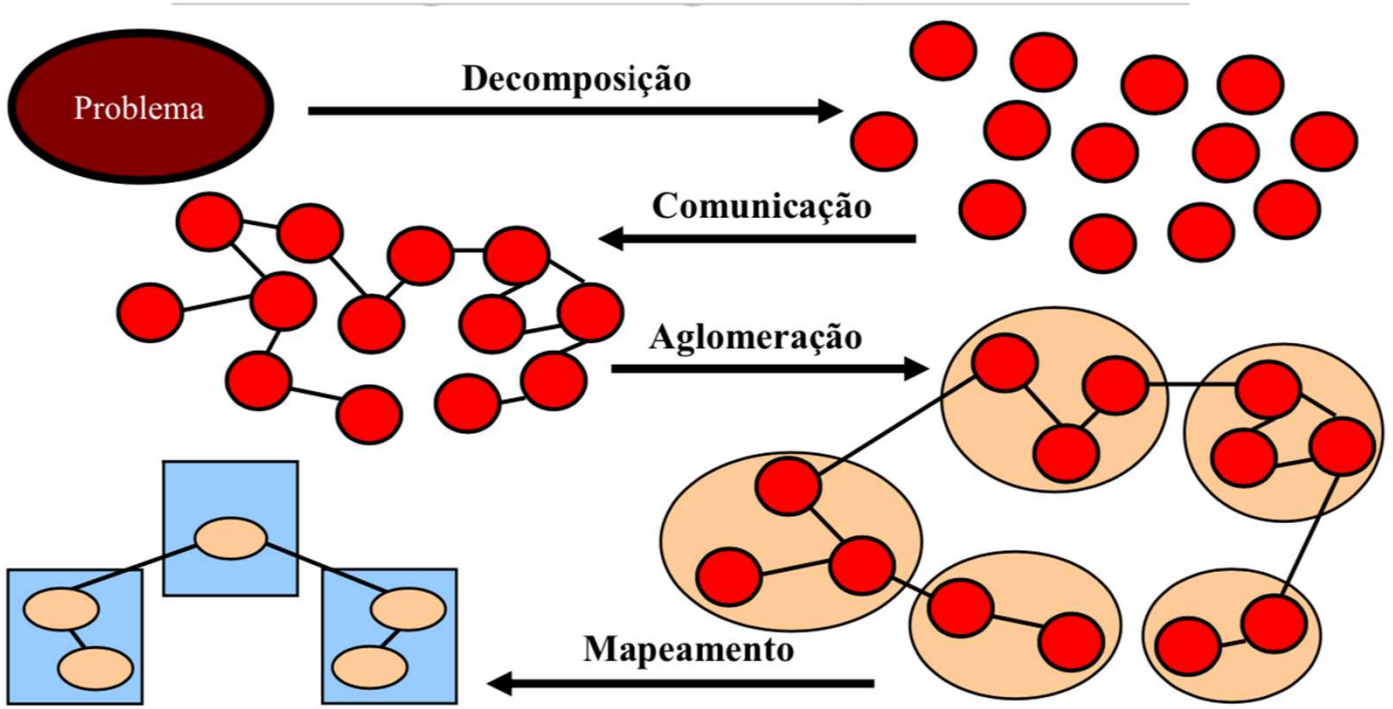
Metodologia de Programação Foster

A metodologia de programação de Foster divide-se em 4 etapas:

- Decomposição;
- Comunicação;
- Aglomeração;
- Mapeamento;

12 – Programação Paralela (cont.)

Metodologia de Programação Foster



12 – Programação Paralela (cont.)

Decomposição

Uma forma de diminuir a complexidade de um problema é conseguir dividi-lo em tarefas mais pequenas de modo a aumentar a concorrência e a localidade de referência de cada tarefa.

Existem duas estratégias principais de decompor um problema:

- **Decomposição do Domínio:** decompor o problema em função dos dados.
- **Decomposição Funcional:** decompor o problema em função da computação.

Um boa decomposição tanto divide os dados como a computação em múltiplas tarefas mais pequenas.

12 – Programação Paralela (cont.)

Decomposição do Domínio

- Tipo de decomposição em que primeiro se dividem os dados em partições e só depois se determina o processo de associar a computação com as partições.
- Todas as tarefas executam as mesmas operações.

12 – Programação Paralela (cont.)

Decomposição Funcional

- Tipo de decomposição em que primeiro se divide a computação em partições e só depois se determina o processo de associar os dados com cada partição.
- Diferentes tarefas executam diferentes operações.

12 – Programação Paralela (cont.)

Comunicação

A natureza do problema e o tipo de decomposição determinam o padrão de comunicação entre as diferentes tarefas. A execução de uma tarefa pode envolver a sincronização/acesso a dados pertencentes/calculados por outras tarefas.

Para haver cooperação entre as tarefas é necessário definir algoritmos e estruturas de dados que permitam uma eficiente troca de informação. Alguns dos principais fatores que limitam essa eficiência são:

12 – Programação Paralela (cont.)

Comunicação

A natureza do problema e o tipo de decomposição determinam o padrão de comunicação entre as diferentes tarefas. A execução de uma tarefa pode envolver a sincronização/acesso a dados pertencentes/calculados por outras tarefas.

12 – Programação Paralela (cont.)

Comunicação

- **Custo da Comunicação:** existe sempre um custo associado à troca de informação e enquanto as tarefas processam essa informação não contribuem para a computação.
- **Necessidade de Sincronização:** enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.
- **Latência** (tempo mínimo de comunicação entre dois pontos) e Largura de Banda (quantidade de informação comunicada por unidade de tempo): é boa prática enviar poucas mensagens grandes do que muitas mensagens pequenas.

12 – Programação Paralela (cont.)

Padrões de Comunicação

Comunicação Global

- Todas as tarefas podem comunicar entre si.

Comunicação Local

- A comunicação é restrita a tarefas vizinhas (e.g. método de Jacobi de diferenças finitas).

12 – Programação Paralela (cont.)

Padrões de Comunicação

Comunicação Estruturada

- Tarefas vizinhas constituem uma estrutura regular (e.g. árvore ou rede).

Comunicação Não-Estruturada

- Comunicação entre tarefas constituí um grafo arbitrário.

Comunicação Estática

- Os parceiros de comunicação não variam durante toda a execução.

Comunicação Dinâmica

- A comunicação é determinada pela execução e pode ser muito variável.

12 – Programação Paralela (cont.)

Padrões de Comunicação

Comunicação Síncrona

- As tarefas executam de forma coordenada e sincronizam na transferência de dados (e.g. protocolo das 3-fases ou rendez-vous: a comunicação apenas se concretiza quando as duas tarefas estão sincronizadas).

Comunicação Assíncrona

- As tarefas executam de forma independente não necessitando de sincronizar para transferir dados (e.g. buffering de mensagens: o envio de mensagens não interfere com a execução do emissor).

12 – Programação Paralela (cont.)

Aglomeração

Aglomeração é o processo de agrupar tarefas em tarefas maiores de modo a diminuir os custos de implementação do algoritmo paralelo e os custos de comunicação entre as tarefas.

12 – Programação Paralela (cont.)

Aglomeração

Custos de implementação do algoritmo paralelo:

- O agrupamento em tarefas maiores permite uma maior reutilização do código do algoritmo sequencial na implementação do algoritmo paralelo.

- No entanto, o agrupamento em tarefas maiores deve garantir a escalabilidade do algoritmo paralelo de modo a evitar posteriores alterações (e.g. optar por aglomerar as duas ultimas dimensões duma matriz de dimensão 8×128×256 restringe a escalabilidade a um máximo de 8 processadores).

12 – Programação Paralela (cont.)

Aglomeração

Custos de comunicação entre as tarefas:

- O agrupamento de tarefas elimina os custos de comunicação entre essas tarefas e aumenta a granularidade da computação.
- O agrupamento de tarefas com pequenas comunicações individuais em tarefas com comunicações maiores permite aumentar a granularidade das comunicações e reduzir o número total de comunicações.

12 – Programação Paralela (cont.)

Granularidade

- Períodos de computação são tipicamente separados por períodos de comunicação entre as tarefas. A granularidade é a medida qualitativa do rácio entre computação e comunicação.
- O numero e o tamanho das tarefas em que a computação é agrupada determina a sua granularidade. A granularidade pode ser fina, media ou grossa.

“Como agrupar a computação de modo a obter o máximo desempenho?”

12 – Programação Paralela (cont.)

Granularidade Fina

- A computação é agrupada num grande número de pequenas tarefas.
- O rácio entre computação e comunicação é baixo.
- (+) Fácil de conseguir um balanceamento de carga eficiente.
- (–) O tempo de computação de uma tarefa nem sempre compensa os custos de criação, comunicação e sincronização.
- (–) Difícil de se conseguir melhorar o desempenho.

12 – Programação Paralela (cont.)

Granularidade Grossa

- A computação é agrupada num pequeno número de grandes tarefas.
- O rácio entre computação e comunicação é grande.
- (–) Difícil de conseguir um balanceamento de carga eficiente.
- (+) O tempo de computação compensa os custos de criação, comunicação e sincronização.
- (+) Oportunidades para se conseguir melhorar o desempenho.

12 – Programação Paralela (cont.)

Mapeamento

Mapeamento é o processo de atribuir tarefas a processadores de modo a maximizar a percentagem de ocupação e minimizar a comunicação entre processadores.

- A percentagem de ocupação é ótima quando a computação é balanceada de forma igual pelos processadores, permitindo que todos começam e terminem as suas tarefas em simultâneo. A percentagem de ocupação decresce quando um ou mais processadores ficam suspensos enquanto os restantes continuam ocupados.

12 – Programação Paralela (cont.)

Mapeamento

- A comunicação entre processadores é menor quando tarefas que comunicam entre si são atribuídas ao mesmo processador. No entanto, este mapeamento nem sempre é compatível com o objetivo de maximizar a percentagem de ocupação.

"Como conseguir o melhor compromisso entre maximizar ocupação e minimizar comunicação?"

12 – Programação Paralela (cont.)

Balanceamento de Carga

- O balanceamento de carga refere-se à capacidade de distribuir tarefas pelos processadores de modo a que todos os processadores estejam ocupados todo o tempo. O balanceamento de carga pode ser visto como uma função de minimização do tempo em que os processadores não estão ocupados.
- O balanceamento de carga pode ser estático (em tempo de compilação) ou dinâmico (em tempo de execução).

12 – Programação Paralela (cont.)

Fatores Limitativos do desempenho

- **Código Sequencial:** existem partes do código que são inerentemente sequenciais
- (e.g. iniciar/terminar a computação).
- **Concorrência:** o número de tarefas pode ser escasso e/ou de difícil definição.
- **Comunicação:** existe sempre um custo associado à troca de informação e enquanto as tarefas processam essa informação não contribuem para a computação.
- **Sincronização:** a partilha de dados entre as várias tarefas pode levar a problemas de contenção no acesso à memória e enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.
- **Granularidade:** o número e o tamanho das tarefas é importante porque o tempo que demoram a ser executadas tem de compensar os custos da execução em paralelo (e.g. custos de criação, comunicação e sincronização).
- **Balanceamento de Carga:** ter os processadores maioritariamente ocupados durante toda a execução é decisivo para o desempenho global do sistema.

12 – Programação Paralela (cont.)

Principais Modelos de Programação Paralela

Programação em Memória Partilhada

- Programação usando processos ou threads.
- Decomposição do domínio ou funcional com granularidade fina, media ou grossa.
- Comunicação através de memória partilhada.
- Sincronização através de mecanismos de exclusão mútua.

12 – Programação Paralela (cont.)

Principais Modelos de Programação Paralela

Programação em Memória Distribuída

- Programação usando troca de mensagens.
- Decomposição do domínio com granularidade grossa.
- Comunicação e sincronização por troca de mensagens.

12 – Programação Paralela (cont.)

Principais Paradigmas de Programação Paralela

Apesar da diversidade de problemas aos quais podemos aplicar a programação paralela, o desenvolvimento de algoritmos paralelos pode ser classificado num conjunto relativamente pequeno de diferentes paradigmas, em que cada paradigma representa uma classe de algoritmos que possuem o mesmo tipo de controle:

- Master/Slave
- Single Program Multiple Data (SPMD) Data Pipelining
- Divide and Conquer
- Speculative Parallelism

12 – Programação Paralela (cont.)

Principais Paradigmas de Programação Paralela

A escolha do paradigma a aplicar a um dado problema é determinado pelo:

- Tipo de paralelismo inerente ao problema: decomposição do domínio ou funcional.
- Tipo de recursos computacionais disponíveis: nível de granularidade que pode ser eficientemente suportada pelo sistema.

12 – Programação Paralela (cont.)

Master/Slave

Este paradigma divide a computação em duas entidades distintas: o processo master e o conjunto dos processos slaves:

- O master é o responsável por decompor o problema em tarefas, distribuir as tarefas pelos slaves e recolher os resultados parciais dos slaves de modo a calcular o resultado final.
- O ciclo de execução dos slaves é muito simples: obter uma tarefa do master, processar a tarefa e enviar o resultado de volta para o master.

12 – Programação Paralela (cont.)

Master/Slave

O balanceamento de carga pode ser estático ou dinâmico:

- É estático quando a divisão de tarefas é feita no inicio da computação. O balanceamento estático permite que o master também participe na computação.
- É dinâmico quando o número de tarefas excede o número de processadores ou quando o número de tarefas ou o tempo de execução das tarefas é desconhecido no início da computação.

12 – Programação Paralela (cont.)

Master/Slave

Como só existe comunicação entre o master e os slaves, este paradigma consegue bons desempenhos e um elevado grau de escalabilidade.

- No entanto, o controle centralizado no master pode ser um problema quando o número de slaves é elevado. Nesses casos é possível aumentar a escalabilidade do paradigma considerando vários masters em que cada um controla um grupo diferente de slaves.

12 – Programação Paralela (cont.)

Single Program Multiple Data (SPMD)

Neste paradigma todos os processos executam o mesmo programa (executável) mas sobre diferentes partes dos dados. Este paradigma é também conhecido como:

- Geometric Parallelism
- Data Parallelism

12 – Programação Paralela (cont.)

Single Program Multiple Data (SPMD)

Tipicamente, os dados são bem distribuídos (mesma quantidade e regularidade) e o padrão de comunicação é bem definido (estruturado, estático e local):

- Os dados ou são lidos individualmente por cada processo ou um dos processos é o responsável por ler todos os dados e depois distribui-los pelos restantes processos.
- Os processos comunicam quase sempre com processos vizinhos e apenas esporadicamente existem pontos de sincronização global.

12 – Programação Paralela (cont.)

Single Program Multiple Data (SPMD)

Como os dados são bem distribuídos e o padrão de comunicação é bem definido, este paradigma consegue bons desempenhos e um elevado grau de escalabilidade.

- No entanto, este paradigma é muito sensível a falhas. A perca de um processador causa necessariamente o encravamento da computação no próximo ponto de sincronização global.

12 – Programação Paralela (cont.)

Data Pipelining

Este paradigma utiliza uma decomposição funcional do problema em que cada processo executa apenas uma parte do algoritmo total. Este paradigma é também conhecido como:

- Data Flow Parallelism

12 – Programação Paralela (cont.)

Divide and Conquer

Este paradigma utiliza uma divisão recursiva do problema inicial em sub- problemas independentes (instâncias mais pequenas do problema inicial) cujos resultados são depois combinados para obter o resultado final.

12 – Programação Paralela (cont.)

Divide and Conquer

A computação fica organizada numa espécie de árvore virtual:

- Os processos nos nós folha processam as subtarefas.
- Os restantes processos são responsáveis por criar as subtarefas e por agregar os seus resultados parciais.

12 – Programação Paralela (cont.)

Divide and Conquer

O padrão de comunicação é bem definido e bastante simples:

- Como as subtarefas são totalmente independentes não é necessário qualquer tipo de comunicação durante o processamento das mesmas.
- Apenas existe comunicação entre o processo que cria as subtarefas e os processos que as processam.

12 – Programação Paralela (cont.)

Divide and Conquer

No entanto, o processo de divisão em tarefas e de agregação de resultados também pode ser realizado em paralelo, o que requer comunicação entre os processos:

- As tarefas podem ser colocadas numa fila de tarefas única e centralizada ou podem ser distribuídas por diferentes filas de tarefas associadas à resolução de cada subproblema.

12 – Programação Paralela (cont.)

Speculative Parallelism

É utilizado quando as dependências entre os dados são tão complexas que tornam difícil explorar paralelismo usando os paradigmas anteriores.

Este paradigma introduz paralelismo nos problemas através da execução de computações especulativas:

- A ideia é antecipar a execução de computações relacionadas com a computação corrente na assumpção optimista de que essas computações serão necessariamente realizadas posteriormente.
- Quando isso não acontece, pode ser necessário repor partes do estado da computação de modo a não violar a consistência do problema em resolução.

12 – Programação Paralela (cont.)

Speculative Parallelism

Uma outra aplicação deste paradigma é quando se utiliza simultaneamente diversos algoritmos para resolver um determinado problema e se escolhe aquele que primeiro obtiver uma solução.