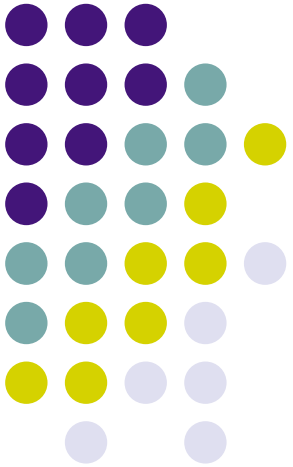
Paradigmas de Computação Paralela



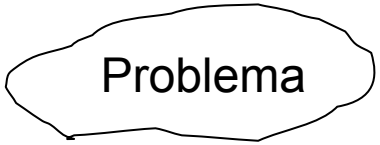
Desenvolvimento de Aplicações Paralelas

João Luís Ferreira Sobral
Departamento do Informática
Universidade do Minho

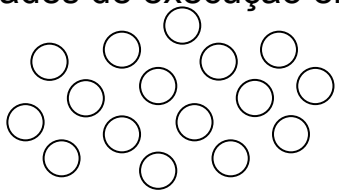
03-Nov-2015

Desenvolvimento de Aplicações Paralelas

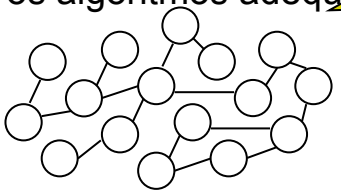
Uma Metodologia de Desenvolvimento de Algoritmos Paralelos



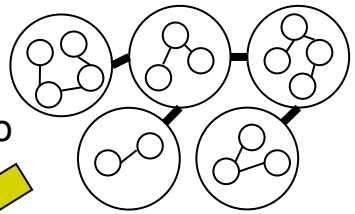
Partição da computação e dos dados a processar em tarefas, identificando oportunidades de execução em paralelo.



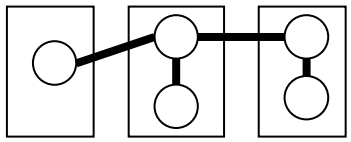
Identificação da **comunicação** necessária entre as várias tarefas, definindo as estruturas de dados e os algoritmos adequados.

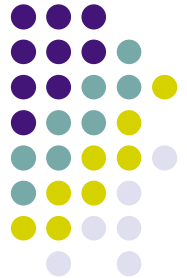


Aglomeração de tarefas e de estruturas de dados definidas nas duas primeiras fases, com vista à melhoria do desempenho ou redução dos custos de desenvolvimento.



Mapeamento das tarefas nos nodos de computação maximizando a utilização dos processadores e minimizando os custos de comunicação.



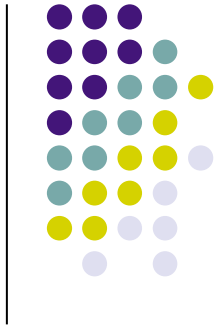


Desenvolvimento de Aplicações Paralelas

1. Partição do problema e dos dados a processar

- Identifica oportunidades de paralelismo:
  - Definindo um elevado número de tarefas de grão fino
  - Pode obter várias decomposições alternativas
- **Duas vertentes complementares:**
  - **Decomposição no domínio dos dados -** identifica dados que podem ser processados em paralelo
    - enfoque nos dados a processar e na sua divisão em conjuntos que podem ser processados em paralelo.
  - **Decomposição funcional –** identifica fases do algoritmo que podem ser efectuadas em paralelo.
    - enfoque no processamento a realizar, dividindo este processamento em tarefas independentes.
- A partição deve obter um número de tarefas, pelo menos, uma ordem de magnitude superior ao número de processadores
  - Introduz flexibilidade nas fases posteriores do desenvolvimento.
- Tarefas de dimensões idênticas facilitam a distribuição da carga
- O número de tarefas deve aumentar com a dimensão do problema.

Desenvolvimento de Aplicações Paralelas



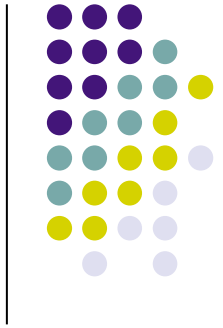
2. Identificação da comunicação necessária

- As tarefas geradas, não podem, em geral, executar de forma independente, uma vez que podem necessitar de dados associados a outras tarefas.
- A comunicação é **local** quando cada tarefa comunica com um pequeno conjunto de tarefas, ou **global**, quando cada tarefa comunica com um elevado número de tarefas.
- A comunicação é **estruturada** quando uma tarefa e as suas vizinhas formam uma estrutura **regular**, ou pode ser irregular quando a estrutura de comunicação pode ser um grafo arbitrário.
- A comunicação é **estática**, quando os parceiros não alteram durante a execução ou **dinâmica.**
- A comunicação pode ser síncrona ou assíncrona.

- Exemplo 1 - Resolver sistemas de equações pelo método numérico de Jacobi requer comunicação local, estruturada e estática: $$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8}$$

- Exemplo 2 – Somar *n* números pode requerer comunicação global (se for utilizado um algoritmo sequencial). $$S = \sum_{i=0}^{N-1} X_i$$

Desenvolvimento de Aplicações Paralelas

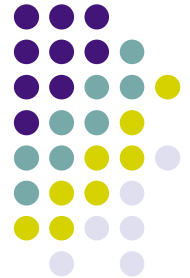


3. Aglomeração de computação e de comunicação

- Visa a execução eficiente da aplicação numa dada classe de sistemas, através da junção de tarefas e/ou comunicação
  - Também pode ser realizar replicação de dados ou computação.
- A aglomeração de comunicação pode ser efectuada juntando várias mensagens por forma a reduzir os custos de comunicação.
- A aglomeração da computação pode ser efectuada juntando várias tarefas por forma a eliminar a necessidade de comunicação remota entre tarefas.

4. Mapeamento das tarefas no sistema

- Especifica onde cada tarefa irá ser executada por forma a minimizar o tempo de execução
  - em geral é um problema complexo (problema NP-completo).
- Em problemas com paralelismo de dados, com uma estrutura regular e com tarefas de dimensão idêntica, o mapeamento é linear.
  - Em problemas com uma estrutura irregular, frequentemente, o mapeamento é efectuado através de heurísticas.
- O mapeamento pode ser estático ou dinâmico, utilizando uma política centralizada ou distribuída.



Padrões de Aplicações Paralelas

Decomposição do domínio dos dados - Algoritmos "*heartbeat*"

- Existe um conjunto de dados, geralmente representado por uma matriz (1D, 2D ou 3D), ao qual é aplicada uma mesma operação.
  - **O processo é iterativo, sendo o valor seguinte de cada ponto calculado com base nos valores das iterações anteriores.**
  - **Entre cada iteração pode ser necessário efectuar comunicação entre as várias tarefas, por exemplo, para determinar as condições de paragem.**

- Exemplos: multiplicação de matrizes, simulação do tempo, métodos numéricos para a resolução de sistemas de equações, etc.

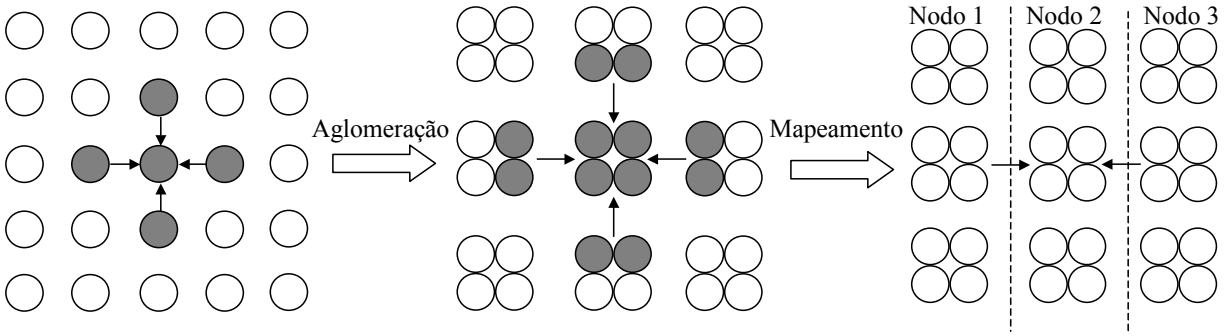
- **Partição** - cada tarefa pode ser constituída por um elemento da matriz, o que geralmente origina um elevado número de tarefas.

- **Comunicação** – cada tarefa deve, a cada iteração, comunicar com outras tarefas para obter os valores das iterações anteriores. Por exemplo, no método de Jacobi são necessárias 4 mensagens por iteração, cada mensagem contendo um valor.

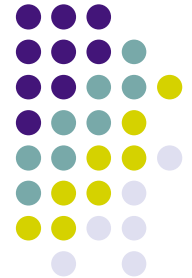
Padrões de Aplicações Paralelas

Algoritmos "*heartbeat*" (cont.)

- **Aglomeração** – a utilização de um ponto da matriz por tarefa origina um grão demasiado fino para a generalidade das arquitecturas, sendo conveniente aglomerar várias tarefas.
  - No método de Jacobi são efectuadas 6 operações por tarefa a cada iteração e geradas 4 mensagens por iteração. Se se juntarem 4 elementos da matriz por tarefa são efectuadas 4*6 operações por tarefa e geradas 4 mensagens por iteração, cada mensagem com dois valores.
    - Adicionalmente se for replicado o cálculo de pontos da matriz em várias tarefas podem ser executadas várias iterações sem comunicação entre tarefas.
- **Mapeamento** – se a quantidade de cálculos a realizar em cada elemento da matriz for idêntica pode ser efectuada uma partição regular da matriz pelos nodos do sistema. Caso contrário, essa partição pode ser irregular e ser alterada em função do curso da computação, obrigando à redistribuição dos dados.

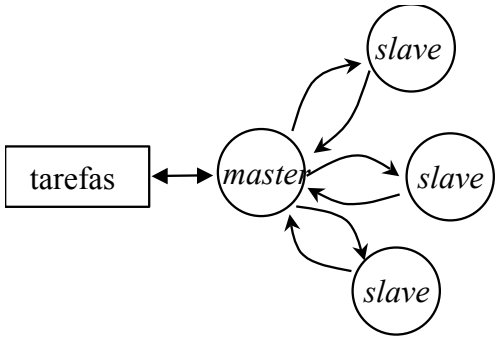


Padrões de Aplicações Paralelas



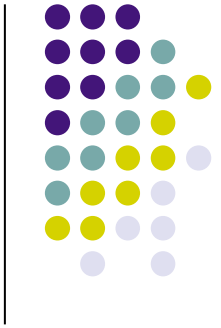
Decomposição do domínio dos dados - Algoritmos "*Master-Slave*"

- Existe um conjunto de dados a processar ao qual é aplicada uma mesma operação.
  - Um *master* é responsável pela gestão das tarefas a realizar e os *slave* são responsáveis por executar essas tarefas.
  - O *master* efectua a divisão do trabalho em tarefas e envia-as aos *slave*, recolhendo posteriormente os resultados.
  - Difere dos algoritmos *data-parallel* no sentido em que os dados a processar estão incluídos nas tarefas a processar pelos *slave*, sendo enviados entre o *master* e *slave*.



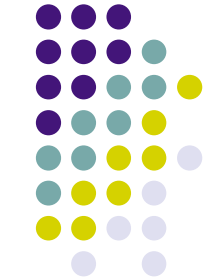
- Os algoritmos *master-slave* são adequados quando não existem dependências entre as tarefas e quando os custos de passagem dos dados entre o *master* e o *slave* não são excessivamente elevados.

Padrões de Aplicações Paralelas



Algoritmos "*Master-Slave*" (*cont.*)

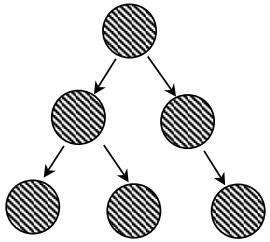
- A grande parte dos algoritmos *data-parallel* pode ser também implementada como um *master-slave*. Os algoritmos "embaraçosamente paralelos" também são geralmente implementados com esta estratégia.
- **Partição** – Na fase de partição são identificadas as tarefas que são geridas pelo *master*.
- **Comunicação** – Neste tipo de algoritmo a comunicação é essencialmente entre o *master* e os *slave* para a envio de tarefas e para o retorno de resultados.
  - Quando existem dependências de dados entre tarefas, ou quando o volume de dados é elevado, pode ser introduzido um gestor de dados, ao qual os *slave* podem pedir a informação necessária para o processamento.
- **Aglomeração** – Visando a redução dos custos de comunicação; podem ser enviadas várias tarefas em cada mensagem.
- **Mapeamento** – O mapeamento é efectuado através da distribuição dos *slave* pelos processadores.
  - Podem ser colocados vários *slave* por nodo, incluindo o processador do *master*.
  - A distribuição de carga pode ser efectuada através de uma atribuição estática das tarefas aos *slave* ou através de uma distribuição em que as tarefas são atribuídas a pedido dos *slave*.
- A escalabilidade pode ser melhorada com a introdução de vários *master*, uma vez que este pode constituir um estrangulamento na aplicação.



Padrões de Aplicações Paralelas

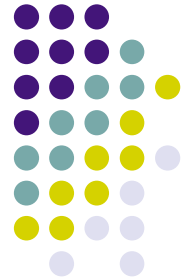
Decomposição do domínio dos dados - Algoritmos "*divide & conquer*"

- Este algoritmo pode ser visto como uma generalização do *master-slave*, onde existe uma hierarquia de *master*.
  - os dados a processar são sucessivamente divididos até se atingir uma dimensão pré-definida para os dados.
  - aos dados com a dimensão pré-definida é aplicado o algoritmo, sendo os dados processados juntos pela ordem inversa.



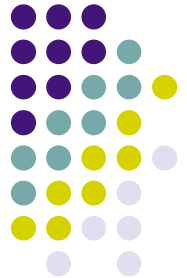
- O algoritmo pode ser visto como a travessia de uma árvore, existindo uma variante, designada por *branch & bound*, onde é pesquisada uma solução num determinado espaço de soluções (ex. problema do caixeiro viajante).
- A generalidade das funções com recursividade múltipla pode ser implementada com esta estratégia.
  - o cálculo de cada uma das "sub-funções" pode ser realizado em paralelo.
  - Exemplos típicos são a soma de um conjunto de número e a ordenação de números.

Padrões de Aplicações Paralelas



Algoritmos "*divide & conquer*" (*cont.*)

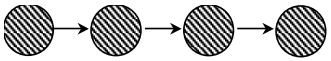
- A **partição** é efectuada identificando as partições dos dados que podem ser recursivamente divididas.
- A **comunicação** é efectuada entre cada elemento da árvore e os seus nodos directos
  - nos algoritmos de *branch & bound* e necessária a comunicação da melhor solução já obtida, por forma a permitir o corte de ramos que já não conduzem à melhor solução, reduzindo assim a quantidade de processamento efectuada.
- A **aglomeração** é efectuada através da junção de ramos em tarefas, o que pode ser implementado através do controlo da dimensão pré-definida para que os dados sejam divididos e processados em paralelo, eliminando o aparecimento de tarefas demasiado pequenas.
- O **mapeamento** é efectuado atribuindo o processamento de ramos da árvore aos processadores.
  - Se a árvore for regular pode ser efectuado um mapeamento estático.
  - O mapeamento deve ser dinâmico quando a quantidade de nodos da árvore depende dos dados a processar
    - Pode ser implementado um algoritmo onde os nodos sem trabalho requerem entre si ramos da árvore para processar.



Padrões de Aplicações Paralelas

Decomposição funcional - Algoritmos em "*pipeline*"

- O processamento a aplicar aos dados é dividido em fases, sendo cada fase executada em paralelo.



- Surge normalmente associado ao processamento de fluxos dados, por exemplo na compressão de vídeo.
- A **partição** corresponde à identificação das fases de algoritmo que podem ser executadas em paralelo.
- A **comunicação** é efectuada entre os elementos da cadeia para transmissão dos dados a processar.
- A **aglomeração**
  - junção de elementos consecutivos da cadeia
  - junção de vários dados numa só mensagem.
- O **mapeamento** pode ser regular se as fases do processamento forem de complexidade equivalente
  - O mapeamento pode ser complexo se cada fase não for de complexidade semelhante
  - O início e o final do processamento implicam algum tempo de ócio