Organização de Computadores – 2005/2006
Processamento Paralelo

Paulo Ferreira
paf@dei.isep.ipp.pt

Maio de 2006

Introdução

slide 2

Porquê?

- Uma forma de conseguir mais performance.
- ≪–Se um boi lavra um campo, porque não 40 galinhas?≫
- ≪–Porque não 40 bois, mais depressa?≫

ORGC Processamento Paralelo – slide 3

Definição de computação paralela

- Um conjunto de elementos de processamento que cooperam para resolver problemas grandes de uma forma rápida.
- E que problemas temos de resolver?

ORGC Processamento Paralelo – slide 4

Alocação de recursos

- Qual o número de elementos?
- Qual a performance dos elementos?
- Quantidade de memória?

ORGC Processamento Paralelo – slide 5

Acesso aos dados, comunicação e sincronização

- Como é que os elementos cooperam e comunicam?
- Como é que os dados são transmitidos entre processadores?
- Que primitivas e abstrações existem para a cooperação?

ORGC Processamento Paralelo – slide 6

Performance e escalabilidade

- Como é que isto se traduz em termos de performance?
- E em termos de escalabilidade?
- Escalabilidade quer dizer aumento de performance com o aumento do nº de elementos.

ORGC Processamento Paralelo – slide 7

Hoje em dia

- Microprocessadores de elevada performance
- Microprocessadores mais baratos
- Paralelismo ao nível da instrução já existe.
- Passo seguinte: Computação paralela

ORGC Processamento Paralelo – slide 8

História

- Processadores de 4, 8, 16, 32, 64 bits
- Uma unidade de execução, várias unidades de execução
- Passo seguinte?
  - Vários processadores?
  - Vários processos por processador?
  - VLIW?
- Já há:
  - *Hyperthreading* (Intel)
  - *Dual Core* (Intel+AMD)
  - *8 Cores * 4 threads* (Sun Ultra Sparc T1)

História

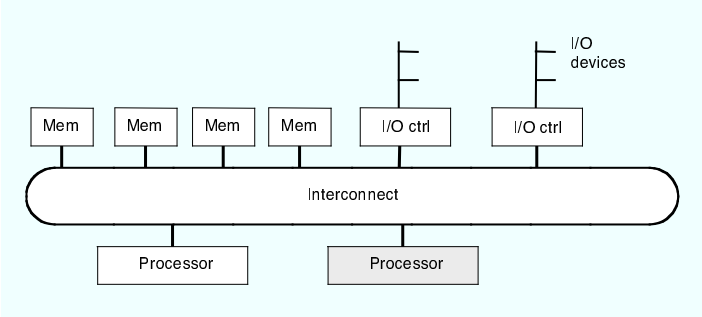
- Modelos divergentes
- Modelo de programação = Primitivas de comunicação = o que existia no hardware
- Hoje em dia já há mais flexibilidade

Taxonomia

Memória partilhada

- Mais simples
- Vários processadores partilham a mesma memória
- Parecido com o *time-sharing* nos S.O. multitarefa
- Extensão dos modelos de programação multitarefa (processos⇒processadores)
- Arquitectura *menos estranha* relativamente ao normal
  - Cuidado: Acesso à memória uniforme ou não?

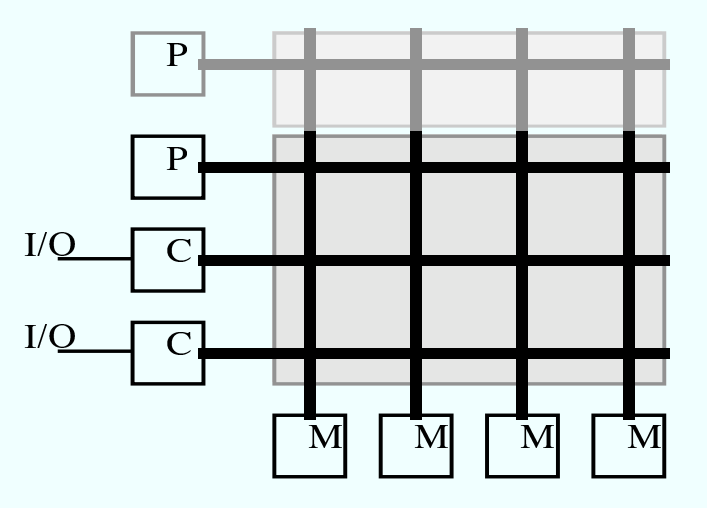
Hardware de comunicações



Memória partilhada

- Estilo *mainframe*
- Uso de um *crossbar*
- *Switch* que permite a ligação simultânea entre p processadores e m memórias
- Vantagens: Performance
- Desvantagens: Custo

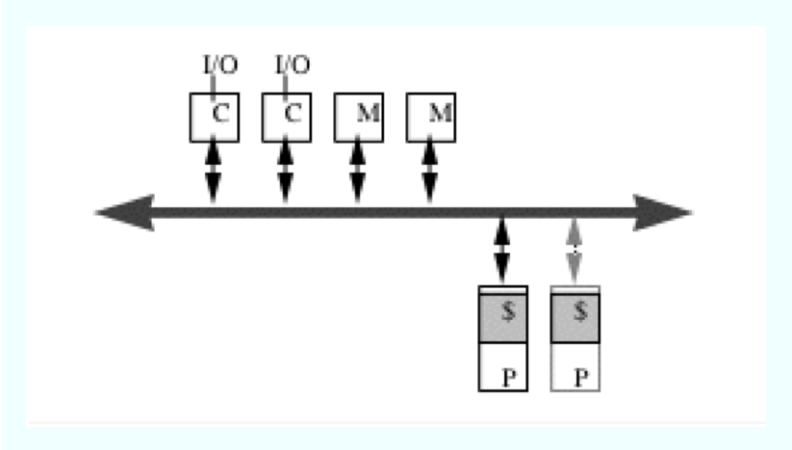
Exemplo de Hardware



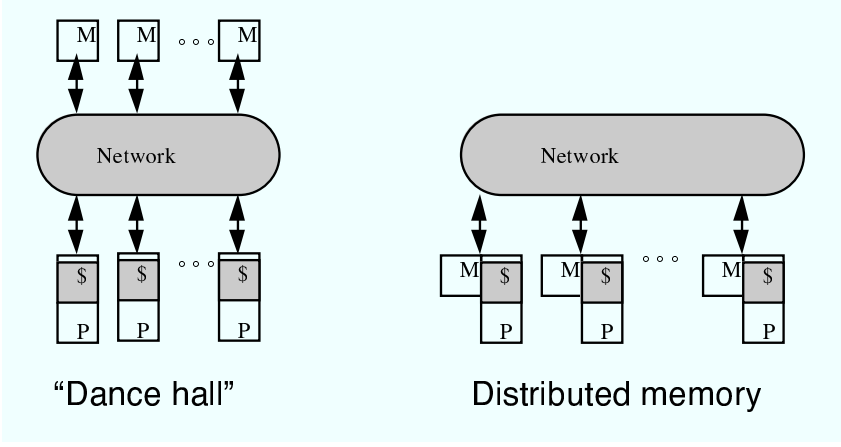
Memória partilhada

- (estilo PC)
- Partilha de um bus de acesso a memória por todos os processadores
- SMP – *Simmetric MultiProcessing*
- Vantagens: Custo
- Desvantagens: Performance
- A largura de banda do barramento é partilhada por todos os processadores.
- Caches: problema da coerência?

Exemplo de Hardware



Escalabilidade



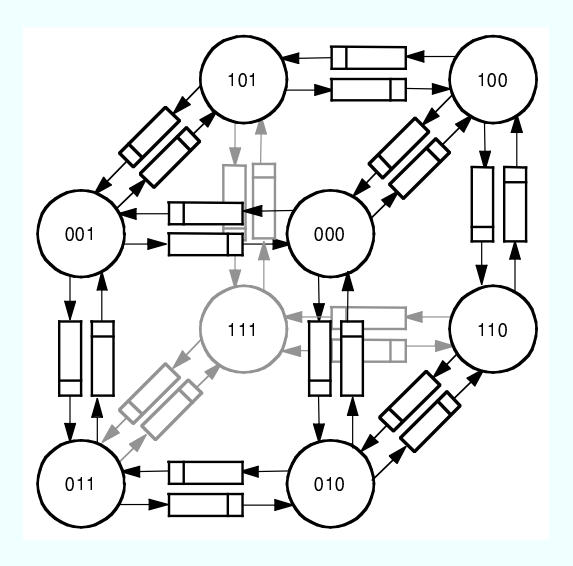
Memória partilhada não uniforme

- NUMA – *Non Uniform Memory Access*
- Cada processador tem a sua memória
- Todos os processadores acedem à memória dos outros, pela interligação

Interligação

- Em *bus*?
- Em rede?
- Em *switch*
- Coisas *novas* – *Hypertransport*

Message Passing



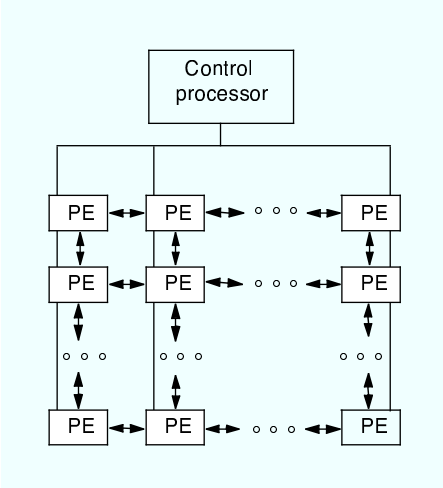
Message passing

- Entre os processadores circulam mensagens
- Originalmente implementação em hardware
- Mais flexível em software
- Muitos overheads
- Programação simples

Paralelismo de dados

- SIMD – *Single Instruction Multiple Data* processamento de vectores e matrizes
- As operações são feitas em paralelo em cada elemento da estrutura de dados.
- Existe apenas uma *instrução* para todos os *processadores*
- Um *processador* por elemento da estrutura de dados
- *Processador:* $\Rightarrow$ recebe instrução, executa instrução
- *Processador de controle:* $\Rightarrow$ lê programa e envia instruções para os outros

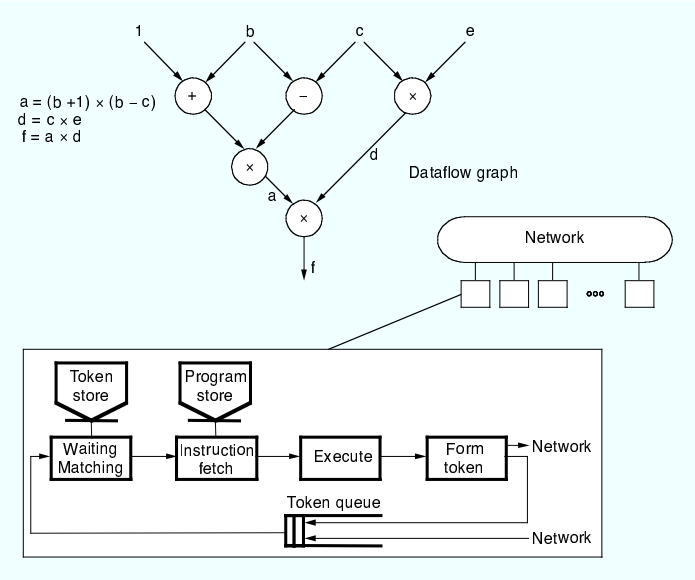
Exemplo de hardware



Dataflow

- Implementar em hardware as expressões matemáticas
- Cada operação é implementada num nó.
- Cada nó recebe os operandos e envia o resultado

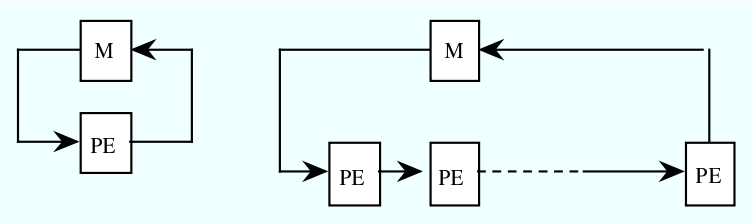
Exemplo de hardware



Arquitecturas sistólicas

- *Pipeline* de elementos de processamento
- Diferente de *dataflow* porque cada elemento tem (pode ter) programa local, e memória local
- Motivação inicial: VLSI permite chips de baixo custo
- Ligar diferentes chips para fazer um algoritmo
- Poupar na largura de banda do acesso à memória

Exemplo de hardware



Resumo (memória partilhada)

- UMA (mesma memória para todos)
  - Bus (aka SMP) – PCs
  - *Rede*
  - *Crossbar* – Mainframes
- NUMA (cada processador tem uma memória *mais sua*)
  - Interligação pode ser de vários tipos

ORGC Processamento Paralelo – slide 29

Resumo (outros)

- Paralelismo de dados
  - extensões SIMD dos processadores actuais
- Message passing
  - entre computadores diferentes é o mais fácil
- DataFlow e arquitecturas sistólicas
  - *Naming*: como se acedem aos dados?
  - só se estivermos a *fazer* hardware

ORGC Processamento Paralelo – slide 30

Detalhes slide 31

Problemas fundamentais

- *Naming*: como se referenciam os dados partilhados
- Operações: que operações são permitidas nesses dados
- Ordenamento: como são coordenados e ordenados os acessos
- Replicação: como os dados são (ou não) copiados
- Custo das comunicações: Latência, largura de banda, overhead, ocupação do canal

ORGC Processamento Paralelo – slide 32

Modelo de programação sequencial

- *Naming*: Espaço de endereços virtual
- Hardware (e compiladores) fazem tradução
- Operações: Leituras e escritas
- Ordenamento: Ordem sequencial do programa

ORGC Processamento Paralelo – slide 33

Performance (programação sequencial)

- Dependências baseadas em referências feitas a variáveis.
- Compiladores e hardware desrespeitam as ordens
- Compilador: reordenamento e alocação de registos
- Hardware: execução fora de ordem
- Caches: replicação transparente

Modelo de memória partilhada

- *Naming*: Qualquer processo usa a memória
- Operações: R/W mais as necessárias para ordenação
- Modelo mais simples de Ordenação:
  - Dentro de um processo: ordem sequencial
  - Entre processos: concorrência
- Podemos ter sincronização
- Compiladores e hardware não cumprem as ordens

Sincronização

- Exclusão Mútua
  - *Locks*
  - Assegurar que certas operações em certos dados são feitas apenas por um processo de cada vez
  - Não dá garantias nenhumas de ordenação
- Sincronização de eventos
  - Ordenar os eventos para salvaguardar dependências
    - ex: produtor $\Rightarrow$ consumidor
  - 3 tipos principais
    - ponto a ponto
    - global
    - grupo

Message Passing

- *Naming*: Apenas dos dados privados (não existem dados partilhados)
- Operações: *send* e *receive*
  - *send*: dados privados são copiados para outro processo
  - *receive*: dados do processo são copiados para memória
  - Temos obrigatoriamente *nomes* para os processos
- Ordenação:
  - Ordem normal dentro de um processo
  - *send* e *receive* podem fazer sincronismo ponto a ponto
  - Exclusão mútua garantida
- Podemos construir *endereços globais*
  - *Process Id* + endereço dentro do processo
  - Mas não teremos operações directas nesses *endereços*

Naming e Operações

- Existindo no modelo de programação podem ser suportadas:
  - Directamente pelo hardware
  - Pelo sistema operativo
  - Por bibliotecas
  - Pelo compilador

Exemplo

- Memória partilhada no modelo de programação
  - Hardware tem memória físicamente partilhada
    - Suporte directo pelo hardware
  - Hardware tem memórias independentes
    - Memória partilhada pode ser feita pelo sistema operativo
    - Pode ser feita pelo compilador/bibliotecas

Exemplo

- Message passing
  - Suporte directo pelo hardware
    - mais flexível com *matching* e *buffering*
  - Suporte pelo S.O. ou acima
    - Hardware faz o transporte
    - Send/Receive em SW (protecção,buffering)
    - S.O.: custo das chamadas ao S.O.

Ordenação

- Message Passing
  - Não existe nenhuma ordenação exceptuando aquela que é imposta pelos pares *send/receive*
- Memória partilhada
  - Importante e subtil
  - Uniprocessadores fazem reordenação para ganhar paralelismo ou localidade
  - Estes truques são mais importantes em multiprocessadores
  - Quais são os truques que continuam a ser válidos?
  - Quais são os novos truques?

Replicação

- Importante para reduzir transferência de dados/comunicação
- Depende do modelo de *naming*
- Uniprocessador: automática
- Message passing: a replicação tem de estar explícita no SW
- Memória partilhada: problema da coerência das caches

Comunicação

- Performance determina o uso das operações
- Três características fundamentais:
  - Latência: tempo necessário para o início
  - Largura de Banda: Velocidade de transferência
  - Custo: Tempo de CPU necessário
- Erros normais:
  - Não contar com a latência
  - Não contar com a possibilidade de *overlap*

Redes de interconexão

- Cada vez mais importantes:
  - Entre computadores
  - Entre chips (Interconnection Networks)
  - Intra chips

Paralelização slide 45

Lei de Amdahl

- Um programa tem uma percentagem do seu tempo total de execução série S e o resto do tempo de execução é paralelizável P.
- Se o total do tempo de programa é 100% então o tempo total é igual a 1, o que implica que $1 = P + S$ ou de outra forma $P = 1 - S$.
- Temos assim que o tempo normal de execução será igual a $S + (1 - S)$.
- Se pusermos N processadores a trabalhar na parte paralela e o tempo de execução da parte paralela diminuir de uma forma ideal o tempo de execução será igual a $S + \frac{(1-S)}{N}$.
- Se N for infinito, então o tempo de execução será apenas de S.
- Logo o aumento de performance que teremos será o inverso dos tempos ou $\frac{(S+(1-S))}{S}$ o que dá $\frac{1}{S}$, isto supondo que a paralelização é perfeita.

Decomposição

- A partir de um programa sequencial gerar tarefas
- O que é uma tarefa: parte do trabalho global que é processada sequencialmente.
- Muitas ou poucas tarefas?

Atribuição

- Atribuir (distribuir) tarefas pelos processos.
- O que é um processo: entidade que executa as tarefas, tem a obrigação de comunicar com os outros processos para executar as tarefas.
- Atribuição estática ou dinâmica? Vantagens e desvantagens?
- Se for dinâmica temos uma fila de tarefas global ou uma fila por processo? Vantagens e desvantagens?

ORGC Processamento Paralelo – slide 48

Orquestração

- Objectivo: estruturar a comunicação e o acesso aos dados.
- Que comunicacão vai haver entre que processos?
- Reduzir os custos da comunicação.
- Escalonar as tarefas correctamente.
- Depende muito das primitivas oferecidas e do tipo de comunicação.
- Depende de onde (em que processo) pomos os dados.

ORGC Processamento Paralelo – slide 49

Mapeamento

- Atribuir processos aos processadores.
- Mais simples e usual: cada processo tem o seu processador.
- Normalmente o utilizador pode especificar a sua intenção mas o sistema operativo pode decidir de outra maneira.

ORGC Processamento Paralelo – slide 50