Prova da Disciplina SCE541 - Arquitetura de Computadores

Grupo 21
Nomes:
Álefe Alves Silva - 11218601
Gustavo Santos Schimiti - 7564002
Igor Guilherme Pereira Loredo – 11275071
Márcio Guilherme Vieira Silva - 11355786
Natã Silva Botelho - 11275105

1)

A Taxonomia de Flynn tenta categorizar as variações de computadores paralelos. Ela se baseia em dois conceitos principais: fluxo de instruções e fluxo de dados. Fluxo de instruções é o número de instruções que são executadas simultaneamente e o fluxo de dados é o conjunto de operações de cada uma destas instruções. Os fluxos são totalmente independentes, por tal podemos somá-los para chegar ao fim necessário. Temos 4 tipos de esquema principais:

- **SISD (*Single Instruction, Single Data*) -** Neste tipo de arquitetura mais simples não existe paralelismo, somente há um único fluxo de instruções e de dados, ela se assemelha muito à Arquitetura de Von Neumann. Ou seja, os processadores só conseguem aplicar uma instrução por ciclo nos dados de entrada e, ainda, com baixo poder de processamento. Exemplos: as máquinas com somente 1 core de processamento mais antigas e computadores mainframe.

- **SIMD (*Single Instruction, Multiple Data*) -** Neste tipo de arquitetura possui-se um único fluxo de instruções e múltiplas unidades de dados. Este, já possui a capacidade de executar uma instrução em conjunto com diversos dados simultâneos para produzir mais resultados. Exemplos: processadores matriciais e vetoriais e máquinas que necessitam aplicar uma única instrução em diversos elementos.

- **MISD (*Multiple Instruction, Single Data*) -** Neste tipo de arquitetura temos múltiplos fluxos de instruções que fazem operações diferentes em cima de uma única unidade de dados. Exemplos: Sistemas de tolerância de falhas para controle de voo de ônibus espacial e, na teoria, em alguns pipelines mais puristas, pois os dados após o processamento por cada fase do pipeline fazem eles se tornarem diferentes.

- **MIMD (*Multiple Instruction, Multiple Data*) -** Neste tipo de arquitetura existem múltiplos fluxos de instruções executando em múltiplos de dados simultaneamente. Ou seja, são processadores multi-core com sistemas distribuídos e podem ter compartilhamento ou não de memória. Exemplos: processadores atuais (i3, i5, i7, i9, Ryzen 3, 5, 7 e 9), processadores para servidor (Intel Xeon), etc.

A diferença aparente entre estas arquiteturas são em relação ao conjunto entre fluxo de instruções e de dados, como podemos ver nesta tabela:

Fluxo de Instruções	Fluxo de Dados: Único	Fluxo de Dados: Múltiplo
Único	SISD Single Instruction Single Data Instrução Simples de Dados Simples	SIMD Single Instruction Multiple Data Instrução Simples de Múltiplos Dados
Múltiplo	MISD Multiple Instruction Single Data Instrução Múltiplas de Dados Simples	MIMD Multiple Instruction Multiple Data Instrução Múltiplas de Dados Múltiplos

Mas vai muito além disto, o que mais se diferencia entre eles é como a arquitetura se integra com a memória, qual o desempenho dela para determinadas aplicações, se existe ou não paralelismo (para o caso da arquitetura SISD). Arquiteturas com múltiplas instruções geralmente têm mais desempenho para funções mais complexas e com menos instruções são mais eficientes em processar operações que demandam mais foco em um tipo de atividade.

2)

Arquiteturas de memória compartilhada: Utilizada em multiprocessadores, os processadores compartilham a memória, ou seja, a mesma memória é acessada pelos múltiplos processadores, o usuário é o responsável pela sincronização. A coerência de dados é protegida de modo que um lugar na memória não pode ser modificado por uma tarefa enquanto outra estiver acessando aquele local. Exemplo: arquiteturas SMP e NUMA.

Arquiteturas de memória distribuída: Utilizada em multicomputadores a memória é fisicamente distribuída entre os processadores, ou seja, cada processador tem uma certa quantidade de memória exclusiva para ele, e que apenas aquele processador pode acessá-la. Exemplo: arquiteturas MPP e COW.

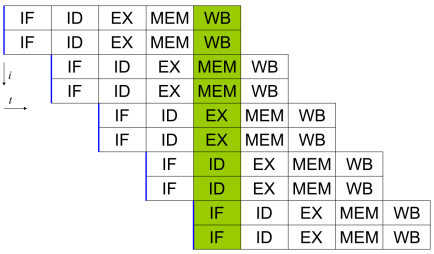
No entanto, estas arquiteturas não se diferem apenas em suas distribuições de memória, mas também na forma a qual as tarefas se comunicam, onde na arquitetura de memória compartilhada as tarefas se comunicam por meio de leitura/escrita na memória o que permite maior agilidade nas comunicações das tarefas, nas arquiteturas de memória distribuída se comunicam por troca de mensagens.

3)

- **Processador com pipeline de operações:** As instruções são executadas simultaneamente mas em estágios de execução diferentes. Cada estágio é executado independentemente das etapas sendo executadas em outras instruções, o que possibilita otimização do tempo para execução das instruções, salvo o caso de dependência entre as mesmas.
  Exemplo

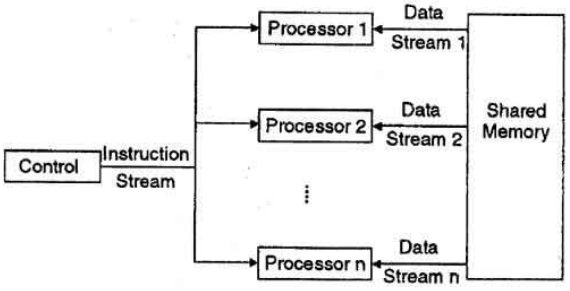
(Os número acima representam os ciclos, cada linha uma instrução e cada bloco um estágio)

- **Processadores superescalares:** As instruções podem ser executadas sem necessidade de estarem em diferentes estágios, de forma que seja possível a execução paralela de mais de uma instrução por ciclo de clock, isso é feito através de múltiplos pipelines independentes e buscando por instruções que não possuam dependência, sendo possível executar instruções fora da ordem original.
  Exemplo:



(Pipeline Superescalar do MIPS)

- **Processadores paralelos:** Processadores paralelos são os que permitem que um único fluxo de instruções seja enviado por uma unidade de controle, esse fluxo será recebido ao mesmo tempo por vários processadores, que irão executá-lo atuando sobre diferentes fluxos de dados, permitindo a execução de várias tarefas dentro de uma mesma instrução de forma paralela.
  Exemplo:



(No exemplo o fluxo de instruções é enviado pela unidade de controle e executado em N fluxos de dados por N processadores disponíveis que utilizam de uma memória compartilhada)

4)

Resposta:

- Dependência de dados: Ocorre quando uma instrução depende do resultado de outra instrução que ainda está em outro processo, seja dependente dos dados de alguma instrução anterior. A solução nesse caso trata-se do controle dos registradores e a técnica de renomeação de registradores, por exemplo.

- Dependência ou conflito de recurso: Se dar por recurso que são acessados ao mesmo tempo, ex: instruções diferentes precisam acessar o mesmo recurso do processador. A solução seria controle de congestionamento e prevenção de colisão, duplicação de recursos.

- Dependência de desvio é comando de desvio condicional que depende de algum dado modificado em instrução anterior, e que se alterar a ordem mudar o execução, ou seja dificulta a execução das tarefas em uma pipeline, uma das soluções seria a otimização da pipeline e também prevenção de desvio (descobrir mais cedo se um desvio será tomado ou não tomado).

5)

Renomeação de registradores é uma técnica utilizada para evitar serialização desnecessária na execução de instruções, buscando aumentar o paralelismo disponível. Ela pode ser classificada em parcial ou total:

- **Renomeação parcial:** restrita a um ou poucos tipos de instrução, por exemplo, apenas instruções de ponto flutuante;
- **Renomeação total:** inclui todas as instruções que possuem registrador de destino.

A renomeação de registradores remove dependências de dados falsas, escrita após leitura (WAR) e escrita após escrita (WAW). Uma dependência de dados é gerada quando instruções estão interligadas de forma que a ordem de execução em um pipeline mudaria a ordem de acesso aos operandos envolvidos na dependência. Esse tipo de dependência exige a preservação da ordem que as instruções seriam executadas se a execução fosse serial, afetando o paralelismo.

O princípio da técnica de renomeação de registradores é direto: se o processador encontra uma instrução que endereça um registrador de destino, ele temporariamente escreve o resultado da operação em um buffer de renomeação alocado dinamicamente ao invés de escrever no próprio registrador de destino. A escolha do tipo de buffer de renomeação para ser usado em um processador tem impacto direto na implementação do processo de renomeação.

6)

Tanto Delayed Branch, como Otimização de Branch são técnicas de otimização de pipeline para lidar com Branches.

O Delayed Branch é um delay criado para atrasar a execução, onde são inseridos NOOPS (instruções que não fazem nada) para que haja tempo suficiente para que todas as instruções sejam realizadas e os dados não sejam sobrescritos incorretamente durante a execução. N-1 NOOPS devem ser inseridas, sendo N o número de estágios do pipeline.

A Otimização de Branch muda operações de lugar (troca o branch) para que as instruções sejam realizadas e os dados não sejam sobrescritos incorretamente durante a execução. N-1 instruções devem ser alteradas, sendo N o número de estágios do pipeline.