SSC0510 Arquitetura de Computadores

Avaliação

Grupo 12

LUIZ PAULO SOUTO MONTEIRO	**N º USP: 10684889**
PEDRO JOSÉ GARCIA	**N º USP: 11846943**
JOÃO PEDRO ALONSO ALMEIDA	**N º USP: 11832343**

São Carlos - SP
13 de janeiro de 2022

1) Explique o que são, compare e exemplifique as arquiteturas SISD, SIMD, MISD, MIMD.

SISD, MISD, SIMD e MIMD são arquiteturas paralelas categorizadas no esquema de Flynn, baseado em fluxo de instruções e fluxo de dados. A diferença entre elas consiste nas diferentes formas como esses fluxos se apresentam em cada uma delas, da seguinte maneira:

SISD: único fluxo de instruções, único fluxo de dados. Um exemplo dessa máquina é o computador de Von Neumann. Apenas uma instrução é feita por vez;

MISD: múltiplos fluxos de instruções, único fluxo de dados. Possui muito menos aplicações práticas que as outras arquiteturas paralelas, porém a arquitetura de Pipeline pertence a essa categoria;

SIMD: único fluxo de instruções, múltiplos fluxos de dados. Ou seja, possui a capacidade de executar uma mesma instrução em um conjunto de dados de forma simultânea. Um exemplo dessa arquitetura é encontrada nos processadores vetoriais encontrados em hardware de videogame e aceleradores gráficos;

MIMD: múltiplos fluxos de instruções, múltiplos fluxos de dados. Nessa categoria, se encontram as máquinas de CPUs independentes, onde cada unidade atua com instruções distintas operando em dados distintos. Qualquer processador com mais de um núcleo cai nesse modelo.

2) Explique o que são, compare e exemplifique arquiteturas com memória compartilhada e memória distribuída.

Ainda dentro do conceito de arquiteturas paralelas, mais especificamente na arquitetura MIMD, há o conceito de modelos de acesso à memória, que é dividido em duas categorias: o modelo de memória distribuída e o de memória compartilhada. No modelo de memória distribuída, cada processador enxerga apenas seu espaço de memória. Já no modelo de memória compartilhada, as CPUs dividem o mesmo espaço de memória. Cada modelo possui suas vantagens e desvantagens, mas basicamente, o modelo de memória distribuída é altamente escalável e permite a construção de processadores maciçamente paralelos, pois a comunicação entre os processadores é feita através de trocas de mensagem, o que resolve tanto o problema de comunicação quanto de sincronização, mas ao mesmo tempo, surge o problema de ter de evitar os chamados “deadlocks” (quando um ou mais processos ficam esperando a execução do outro e vice-versa, travando a execução do programa), além do modelo de programação ser menos natural. Por outro lado, o modelo de memória compartilhada permite fácil programação, e não há necessidade de movimentar fisicamente os dados entre uma partição de memória e outra, tornando muito mais eficiente a comunicação entre processos, mas ao mesmo tempo, há necessidade do uso de primitivas especiais de sincronização quando do acesso a regiões compartilhadas na memória, e além disso, os sistemas deste tipo sofrem problemas de contenção e de alta latência para fazer acesso à memória compartilhada, o que degrada a performance e limita a escalabilidade das aplicações.

3) Explique o que são, compare e exemplifique as seguintes máquinas:

1. **Processador com pipeline de operações**
2. **Processadores Superescalares**
3. **Processadores Paralelos**

Processador com pipeline de operações executam tarefas diferentes ao mesmo tempo sendo capazes de respeitar a ordem de instruções que chegam a ele, aumentando o seu desempenho e reduzindo o tempo global de execução de tarefas, como ele executa as tarefas em paralelo acaba maximizando o uso dos seus recursos, sendo que cada estágio pode executar uma instrução por vez, um exemplo do porquê esse processador diminui o tempo global de execução das tarefas, é compará-lo ao funcionamento de uma pizzaria, vamos dividi-la em 3 funções, cada uma levando 30 minutos para ser executada, tem uma pessoa para abrir a massa, uma para colocar o recheio e o forno, e deseja-se fazer 3 pizzas, em uma execução com pipeline, a massa da primeira pizza seria aberta, enquanto coloca o recheio da primeira pizza, a massa da segunda será aberta, quando colocar a primeira para assar, na segunda estará colocando o recheio e a massa da terceira será aberta, quando o forno liberar a primeira pizza, ela estará pronta, a segunda pizza estará sendo colocada para assar e na terceira estará pondo o recheio, quando a segunda estiver pronta, a terceira será colocada para assar e logo depois ficará pronta, portanto, as 3 tarefas (pizzas) foram concluídas em 5 etapas, totalizando 2 horas e 30 minutos, já em uma execução sem pipeline, seria necessário abrir a massa da primeira pizza, colocar o recheio e assar, só depois que assar a primeira, que poderia abrir a massa da segunda e assim por diante, então as 3 tarefas (pizzas) seriam executadas em 9 etapas, totalizando 4 horas e 30 minutos, portanto, processadores com pipelines executam tarefas com menos tempo global do que processadores sem pipelines.

Processadores superescalares possuem todos os aspectos do pipeline, contudo eles podem executar mais de uma instrução no mesmo estágio, assim, elas têm a habilidade de iniciarem múltiplas instruções no mesmo ciclo de clock, usando o mesmo exemplo da pizzaria, na etapa do forno poderia estar assando mais de uma pizza por vez, diminuindo ainda mais o tempo global de execução das tarefas. Processadores paralelos utilizam várias CPUs ao mesmo tempo, quebrando um processo em tarefas menores e dividindo essas tarefas em instruções menores, sendo distribuídas entre as CPUs para serem executados simultaneamente, um exemplo disso, é quando se tem um problema matemático muito grande, ele é dividido em vários cálculos que vão sendo resolvidos concomitantemente, com isso a velocidade de resolução de problemas será aprimorada exponencialmente.

Essa estratégia é muito boa nas operações que acumulam uma grande quantidade de dados e que precisam ser trabalhadas rapidamente. Comparando os três processadores, os superescalares conseguem executar tarefas mais rapidamente do que os com pipeline de operações, porque eles podem executar mais de uma instrução no mesmo estágio, o que os com pipeline de operações não conseguem, os processadores paralelos utilizam conceitos dos processadores com pipelines de operações e superescalares para conseguir executar grandes quantidades de dados rapidamente.

4) Explique as limitações intrínsecas do paralelismo: Dependência de dados, Dependência de desvio, Conflito de recurso (ULA) e o que pode ser feito para minimizar esses problemas.

A dependência de dados ocorre quando uma instrução pode ser obtida, decodificada mas não pode ser executada até que a instrução predecessora seja executada, porque ela necessita de dados gerados pela execução da predecessora, uma forma de minimizar isso é evitando executar instruções dependentes simultaneamente.

A dependência de desvio ou procedural, ocorre quando acontece desvios em um sequência de instruções, quando as instruções que vêm depois de um desvio possuem uma dependência procedural com o desvio e não podem ser executadas até que o desvio seja executado, podendo ser minimizados utilizando técnicas de previsão de desvio.

Um conflito de recursos é uma competição de duas ou mais instruções pelo mesmo recurso e ao mesmo tempo, esses recursos podem ser memória, cache, barramentos, entradas para banco de registradores e unidades funcionais, os conflitos de recursos podem ser minimizados a partir da duplicação de recursos e por meio da implementação da unidade como uma pipeline.

5) Explique renomeação de registradores:

Consiste em uma técnica feita em alguns compiladores para permitir a paralelização de instruções que foram alocadas ao mesmo registrador, otimizando, assim, o processamento e diminuindo o tempo necessário para execução da tarefa. Ela foi desenvolvida com base na técnica de tratamento de recursos.

A renomeação acontece da seguinte forma: quando o processador encontra uma instrução que aponta para um registrador de destino, invés de escrever nesse registrador ele escreve em um buffer alocado dinamicamente (chamado buffer de renomeação). Resolvendo, assim, problemas de dependências falsas no pipeline como, a antidependência (WAR/Write After Read) onde a instrução lê uma informação que é escrita por uma instrução sucessora, e a dependência de saída (WAW/Write After Write) onde a instrução escreve onde uma outra instrução sucessora também precisa escrever.

Através da renomeação de registradores, essas instruções com falsas dependências podem ser liberadas e executadas ao mesmo tempo da instrução raiz.

6) Explique o que são, compare e exemplifique as seguintes técnicas: Delayed Branch e Otimização do Branch.

Antes de tudo, é importante entendermos o que seria essa “branch”. Uma branch, então é uma ramificação (literalmente) da execução do programa. O exemplo mais comum disso seria o famoso “if”, ou uma operação condicional, onde o fluxo de execução pode tomar dois (ou mais) caminhos diferentes dependendo do resultado da operação. Como bem mencionado em aula, esses tipos de desvios em tempo de execução são desastrosos para os nossos queridos e amados pipelines, pois eles fazem com que todos os estágios já realizados para uma determinada instrução sejam completamente inutilizados (basicamente deixa o pipeline inútil, fazendo com que o tempo de execução aumente drasticamente). Para enfrentar esse tipo de problema nasceram as técnicas de Delayed Branch e Otimização do Branch.

Obs: Sabendo disso, fazemos aqui um apelo a todos programadores: não usem if. Sacanagem, mas caso hajam várias condições tente sempre organizá-las de forma que a execução não passe por um if desnecessariamente, uma boa forma de garantir isso (em linguagem que suportam) é através do elif, onde a condição não é executada caso uma anterior já tenha sido sucesso, evitando, assim, a geração de uma branch (com dois ifs em sequência, o segundo será sempre executado independente do resultado do primeiro gerando duas branches, com um if e um elif (ou else) somente uma branch é gerada).

Delayed Branch: Essa técnica consiste em “atrasar” o deslocamento do PC da instrução que testa a condição de forma que uma ou mais instruções sucessoras sejam sempre executadas independente do resultado da condição. Ou seja, quando o pipeline percebe uma instrução que gera uma branch, ele executa essa instrução mas não modifica o PC imediatamente, invés disso ele insere algumas instruções vazias (NOOP) na sequência dando um tempo para que as instruções sucessoras daquela condição sejam executadas. Embora pouco intuitivo à primeira vista, essa técnica mantém o pipeline cheio mesmo havendo desvios, caso contrário ele teria que ficar esperando a condição ser testada para poder executar as instruções sucessoras. As principais desvantagens dessa técnica são a necessidade de hardware adicional, a dificultação de interrupções dessas instruções e o aumento no tamanho no código (pela inserção de NOOPs).

Otimização do Branch (Branch Prediction): Aqui a solução é mais orientada ao hardware, onde temos a presença de circuitos que tentam prever o resultado da condição e começa a executar as instruções de acordo com essa previsão. Em um if, por exemplo, o pipeline tentaria adivinhar o resultado (verdadeiro ou falso) e seguiria um dos fluxos. Muitos sistemas implementam diferentes jeitos de melhorar a precisão dessa técnica, mas obviamente em algum momento o pipeline escolherá o caminho errado fazendo com que todo aquele processamento seja essencialmente perdido e o pipeline volte para a instrução da condição e siga o outro caminho. A grande desvantagem, então, fica bem clara por se tratar de uma técnica “High Risk High Rewards”, em caso positivo o pipeline basicamente anula as consequências do desvio mas em caso negativo essas consequências ficam ainda piores.