Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística

Disciplina SCE541 Arquitetura de Computadores

Grupo nr.: 2

Nome: Enzo Bustamante	Nº USP: 9863437
Nome: Gabriel Fontes	Nº USP: 10856803
Nome: Giovanna Fardini	Nº USP: 10260671
Nome: Thales Damasceno	Nº USP: 11816150
Nome: Vinicius Baca	Nº USP: 10788589

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

São arquiteturas baseadas na unicidade ou multiplicidade do fluxo de dados e instruções, também conhecido como classificação de Flynn. SISD (single instruction single data) representa computadores seriais, isto é, um único fluxo de instruções é enviado para a unidade de controle e um único fluxo de dados será processado pela unidade de processamento. É uma arquitetura mais antiga e onde temos apenas um processador. Como exemplo temos o arduíno.

SIMD (single instruction multiple data) se dá quando um único fluxo de instruções é enviado para a unidade de controle e opera sobre múltiplos fluxos de dados, representa processadores matriciais e como exemplo podemos citar uma GPU da placa de vídeo. É uma máquina de multiprocessadores, mas somente uma unidade de controle. É similar ao SISD no sentido que o fluxo de dados entre a unidade de processamento e a memória é único, mas por termos múltiplas unidades teremos múltiplos fluxos.

MISD (multiple instruction single data) significa que teremos um mesmo fluxo de dados sobre o qual estão sendo aplicados diferentes fluxos de operações – aqui temos também uma máquina de multiprocessadores e os dados entram em uma unidade de processamento e saem por outra (fluxo único). Seria como se múltiplos algoritmos decodificassem uma única mensagem, é o tipo de arquitetura menos comum e como exemplo temos máquinas de criptografia.

MIMD (multiple instruction multiple data) significa que múltiplos fluxos de instruções estão sendo aplicados sobre múltiplos fluxos de dados, não necessariamente dependentes ou relacionados, computadores multicore, superescalares tem essa arquitetura. Neste caso, temos uma unidade de controle para cada unidade de processamento como se fossem vários processadores SISD reunidos juntos. Similar ao MISD, porém enquanto o MISD tem somente uma unidade de processamento como entrada e uma como saída o MIMD tem ligação entre todas as suas unidades e a memória.

2ª Questão: Explique o que são, compare e exemplifique arquiteturas com memória compartilhada e memória distribuída.

No geral, dentro da computação paralela temos dois tipos de arquitetura de memória possível, única e compartilhada entre os multiprocessadores como ocorre em processadores de notebooks como da Intel por exemplo, ou distribuída, isto é, cada processador tem sua própria memória e todos estão conectados em uma rede única como se fosse um cluster.

Na memória compartilhada todos os processadores tem o mesmo direito de acesso, pois a memória é única e todos os processadores estão conectados fisicamente à ela. Neste caso, é preciso ter um controle de coerência para que um processo não modifique um endereço de memória que esteja sendo usado por outro. Aqui podemos ter dois tipos de máquinas, SMP – simétricas, mesmo nível de hierarquia de acesso entre todos os processadores ou NUMA – não uniforme, os processadores tem conexões com determinadas regiões da memória diferentes.

Na memória distribuída podem existir casos em que um processador deseja acessar um dado na memória de outro de modo que isso só poderá ser feito com autorização e via mensagens e sinais de comando. É como se tivéssemos uma rede em que cada processador representa um nó com sua própria memória, tem um custo maior porém maior escalabilidade, já que não depende do tamanho da placa mãe por exemplo como no caso da memória compartilhada.

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

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

Processadores com pipelines de operações são aqueles que em sua arquitetura de conjunto de instruções, podem ter uma sobreposição temporal dentre as fases de execução. O pipeline, permite que diferentes fases de execução sejam sobrepostas nos componentes do processador, desde que não haja concorrência no uso de um recurso específico. Com isso, apesar de o desempenho individual de uma instrução não ser melhorado, o desempenho conjunto de uma série de instruções pode ser impactado positivamente pelo pipeline.

Processadores superescalares por sua vez, também possuem uma lógica de pipeline implementada, que aliado ao fato de existirem diferentes linhas de pipeline, permite que múltiplas instruções de mesma natureza sejam executadas ao mesmo tempo. Para isso, a arquitetura superescalar se vale de: Uma unidade de busca de instruções que seja capaz de buscar mais de uma instrução por ciclo, com predição de desvios; Uma unidade de decodificação que seja capaz de ler múltiplos registradores em paralelo.

Por sua vez, os processadores paralelos realizam o ganho de performance em outra escala arquitetural. O processamento paralelo de instruções realiza uma alta segregação (em termos de processamento) das tarefas entre as camadas do SO e da Arquitetura do processador em si. Os processadores paralelos realizam as instruções de forma completamente independente entre si mas compartilham outros elementos da arquitetura de computadores, como I/O e memória, e necessitam de um agente sincronizador (seja em software ou hardware) para a divisão dos macro blocos de instruções que irão processar.

4ª Questão: 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.

1 Dependência de dados:

Ocorre quando uma instrução depende do resultado de outra instrução que ainda está no pipeline. Este tipo de dependência é originado na natureza sequencial do código em execução, cuja ordem normal é alterada dentro do pipeline.
O cálculo só pode ser executado caso a outra função for executada previamente. Portanto, caso exista uma precedência obrigatória de operações no tratamento de dados, quaisquer que sejam, o paralelismo não pode ser aplicado, por definição.
Neste sentido, o paralelismo só pode ser aplicado dividindo os dados, porém essa limitação continua presente.

2- Dependência de desvio:

Dependência de desvio é quando à título de exemplo, um BNE, BEQ, BGE, BLE, ou seja, precisa saber o resultado de uma condição, porque dependendo do valor vai desviar para um trecho ou outro do programa. Então, se há dois números em processamento, e chega num ponto que é tipo BLE (Brench Lesser or Equal), ou seja: vai para um caminho se for menor ou igual, senão continua em outro. É preciso saber os dois números que vão entrar nessa comparação, senão vai mudar o caminho que o programa irá seguir.
Neste sentido há problemas de várias ordens; à nível de programa, como no primeiro exemplo, à nível de instrução, o que implica em diferentes níveis de paralelismo, porque há paralelismo em que a mesma CPU tem vários núcleos, exemplo: dual core, quad core e etc.
Uma forma de minimizar problemas neste caso é a utilização de Semáforos, ou seja, restringir o acesso aos recursos à uma ULA por vez. Pode ser feito à nível de thread porém a nível de Assembler é mais eficiente.

3- Conflito de recurso (ULA):

O conflito de recursos ocorre quando duas ULA's estão efetuando suas operações mas precisam ter acesso ao mesmo dado. Se duas ULA's receberem a instrução para carregar um dado que se encontra na mesma posição de memória no mesmo ciclo de clock, esperamos que ocorra um erro pois a memória é compartilhada entre os sistemas. Este problema também é conhecido como problema do acesso concorrente.

5ª Questão: Explique renomeação de registradores.

Permite abstrair registradores lógicos de registradores físicos, tornando-os opacos, impedindo que seus respectivos registradores físicos sejam referenciados diretamente.

A grande sacada desse impedimento é eliminar falsas dependências (estas surgem com o reuso de registradores por instruções sucessivas que na verdade não possuem dependências de dados entre si). Usar registradores lógicos permite expressar mais claramente quais instruções possuem, de fato, dependências de dados. Eliminar dependências desnecessárias permite mais paralelismo, já que uma não precisa aguardar finalização da outra, permitindo, por exemplo, que processadores superescalares atinjam seu maior potencial.

Essa transposição usualmente ocorre em nível de hardware. Porém, em arquiteturas sem esse recurso, é usual que os (bons) compiladores (durante o processamento de representação intermediária) detectem as sequências e troquem os registradores.

6ª Questão: Explique o que são, compare e exemplifique as seguintes técnicas:

Delayed Branch e Otimização do Branch.

Delayed branch e Otimização de branch são duas maneiras diferentes de mitigar os efeitos de um pipeline de execução longa. Para que o pipeline não precise parar sempre que um branching condicional é executado, pois não é possível prever qual instrução deve ser executada após a instrução de branch até que suas computações pendentes sejam concluídas.

Delayed branch simplesmente significa que algumas instruções apareçam após que a branch do fluxo de instruções seja executada, independente do caminho que o branching realmente seguir. Essa abordagem mantém o hardware simples, mas sobrecarrega o compilador. Um exemplo é que na arquitetura MIPS a operação da branch é atrasada por uma instrução, em por padrão o compilador insere uma instrução vazia, chamada de slot de delay, para que termine de executar o branch antes de trocar a instrução.

Otimização de branch é uma abordagem mais baseada em hardware, em que as instruções simplesmente tentam prever para qual branch irá, executando as instruções neste fluxo. Existem diversas formas que os sistemas utilizam para aperfeiçoar a performance da previsão, como por exemplo o hardware manter um histórico de que forma cada branch seguiu no passado. Um exemplo é que no assembler MAL (Micro Assembly Language), o compilador é capaz de analisar as dependências entre as instruções e identificar quais instruções não afetam os testes de branch, podendo reorganizar o código para diminuir as dependências de controle, resultando no mesmo output.

Obs. Procure cobrir os pontos abordados em aula em suas respostas, mantendo o limite de UMA folha para cada questão.