CONVERSA INICIAL

Olá, seja bem-vindo a sexta aula de Arquitetura de Computadores.
Nessa aula, vamos conhecer as **Arquiteturas Paralelas** e abordaremos os seguintes temas:

- Visão geral do paralelismo
- Paralelismo no chip
- Coprocessadores
- Multiprocessadores *versus* multicomputadores
- Escalonamento
- Desempenho

CONTEXTUALIZANDO

Embora os computadores continuem a ficar cada vez mais rápidos, as demandas impostas a eles estão crescendo no mínimo com a mesma rapidez. Astrônomos querem simular toda a história do universo, desde o *big bang* até o final do espetáculo. Cientistas farmacêuticos adorariam projetar em seus computadores medicamentos sob encomenda para doenças específicas, em vez de ter de sacrificar legiões de ratos. Projetistas de aeronaves poderiam inventar produtos mais eficientes no consumo de combustível, se os computadores pudessem fazer todo trabalho sem a necessidade de construir protótipos físicos, para testar em tuneis aerodinâmicos. Em suma, seja qual for a capacidade de computação disponível, para muitos usuários, em especial nas áreas da ciência, engenharia e industrial, ela nunca será suficiente.

Portanto, para enfrentar problemas maiores, os arquitetos de computadores estão recorrendo cada vez mais a computadores paralelos. Apesar de talvez não ser possível construir uma máquina com uma única CPU e com tempo de ciclo de **0,001 ns**, pode ser perfeitamente viável construir uma com 1.000 CPUs com um tempo de ciclo de **1 ns** cada. Embora esse último projeto use CPUs mais lentas do que o primeiro, sua capacidade total de computação é teoricamente a mesma. E é aqui que reside a esperança.

O paralelismo pode ser introduzido em vários níveis. No nível mais baixo, ele pode ser adicionado ao chip da CPU por *pipeline* e projetos superescalares com várias unidades funcionais. Também pode ser adicionado por meio de palavras de instrução muito longas com paralelismo implícito. Características especiais podem ser adicionas a CPU para permitir que ela manipule múltiplos *threads* de controle ao mesmo tempo. Por fim, várias CPUs podem ser reunidas no mesmo chip. Juntas, essas características podem equivaler, talvez, a um fator de 10 vezes em desempenho em relação a projetos puramente sequenciais.

No nível seguinte, placas extras de CPU com capacidade de processamento adicional podem ser acrescentadas a um sistema. Em geral essas CPUs de encaixe têm funções especializadas, tais como processamento de pacotes de rede, processamento de multimídia ou criptografia. No caso de funções especializadas, o fator de ganho também pode ser de aproximadamente 5 a 10.

Contudo, para conseguir um fator de cem, de mil, ou de milhão, é necessário replicar CPUs inteiras e fazer que todas elas funcionem juntas com eficiência. Essa ideia leva a grandes multiprocessadores e multicomputadores (computadores em *clusters*). Nem é preciso dizer que interligar milhares de processadores em um grande sistema gera seus próprios problemas, que precisam ser resolvidos.

Por fim, agora é possível envolver organizações inteiras pela internet e formar grades de computação fracamente acopladas. Esses sistemas estão apenas começando a surgir, mas têm um potencial interessante para o futuro.
Quando duas CPUs ou dois elementos de processamento estão perto um do outro, têm alta largura de banda, o atraso entre eles é baixo e são muito próximos em termos computacionais, diz-se que são **fortemente acopladas.** Por outro lado, quando estão longe um do outro, têm baixa largura de banda e alto atraso e são remotos em termos computacionais, diz-se que são **fracamente acopladas.**

Toda essa questão do paralelismo, de uma extremidade do espectro a outra, é um tópico de pesquisa muito atual e concorrido.

PESQUISE

O paralelismo pode ser dividido em várias categorias, mas as principais são: **paralelismo no chip** e **paralelismo no nível de instrução**.

Paralelismo no chip

Um modo de aumentar a produtividade de um chip é conseguir que ele faça mais coisas ao mesmo tempo. Em outras palavras, explorar o paralelismo. São apresentados alguns modos de conseguir aumentar a velocidade por meio do paralelismo no nível do chip, incluídos paralelismo no nível da instrução, *multithreading* e a colocação de mais de uma CPU no chip. Essas técnicas são bem diferentes, mas cada uma delas ajuda à sua própria maneira. Mas, em todos os casos, a ideia é conseguir que mais atividades aconteçam ao mesmo tempo.

Paralelismo no nível da instrução

Um modo de conseguir paralelismo no nível mais baixo é emitir múltiplas instruções por ciclo de *clock*. Há duas variedades de CPUs de emissão múltipla: processadores superescalares e processadores VLIW.

Na configuração mais comum, em certo ponto do *pipeline* há uma instrução pronta para ser executada. CPUs superescalares são capazes de emitir múltiplas instruções para as unidades de execução em um único ciclo de *clock*. O número real de instruções emitidas depende do projeto do processador, bem como das circunstâncias correntes. O *hardware* determina o número máximo que pode ser emitido, em geral, duas a seis instruções. Contudo, se uma instrução precisar de uma unidade funcional que não está disponível ou de um resultado que ainda não foi calculado, ela não será emitida.

A outra forma de paralelismo no nível da instrução é encontrada em processadores VLIW (*Very Long Instruction Word* - Palavra de Instrução Muito Longa). Na forma original, máquinas VLIW de fato tinham palavras longas que continham instruções que usavam múltiplas unidades funcionais. Considere, por exemplo, um *pipeline* no qual a máquina tem cinco unidades funcionais e pode efetuar, simultaneamente, duas operações com inteiros, uma operação de ponto flutuante, um carregamento e um armazenamento.

Uma instrução VLIW para essa máquina conteria cinco ***opcodes*** e cinco pares de operandos, um *opcode* e um par de operandos por unidade funcional. Com 6 bits por *opcode*, 5 bits por registrador e 32 bits por endereço de memória, as instruções poderiam facilmente ter 134 bits - ou seja, bem comprida.

Contudo, esse projeto revelou ser muito rígido porque nem toda instrução podia utilizar todas as unidades funcionais, o que resulta em muitas NO-OP inúteis usadas como filtro, por conseguinte, modernas máquinas VLIW têm um modo de marcar um grupo de instruções que formam um conjunto com um bit de "final de grupo". Então, o processador pode buscar o grupo inteiro e emiti-lo de uma vez só e cabe ao compilador preparar grupos de instruções compatíveis.

Computação de *Cluster*

O outro estilo de multicomputador é o **computador de *cluster***. Em geral, consiste em centenas de milhares de PCs ou estações de trabalho conectadas por uma placa de rede. A diferença entre um *cluster* e os multicomputadores é análoga à diferença entre um mainframe e um PC. Ambos possuem uma CPU, possuem memória RAM, Sistema Operacional e assim por diante. Porém, os componentes do mainframe são mais rápidos (exceto talvez o Sistema Operacional). No entanto, em termos qualitativos, eles são considerados diferentes e são usados e gerenciados de modos diferentes. Essa mesma diferença vale para os multicomputadores em relação aos *clusters*.

Embora existam muitos tipos de *clusters,* são dois os que dominam: o centralizado e o descentralizado.

O centralizado é um *cluster* de estações de trabalho ou PCs montados em uma grande estante dentro de uma sala. Às vezes, eles são empacotados de um modo bem mais compacto do que o usual, para reduzir o tamanho físico e o comprimento dos cabos. Em geral, as máquinas são homogêneas e não tem periféricos, a não ser placas de rede e, possivelmente, discos.

Clusters descentralizados consistem em estações de trabalho ou PCs espalhados por um prédio ou campus. Grande parte deles fica ociosa por muitas horas do dia, em especial a noite. Costumam ser

conectados por uma LAN. Em geral, são heterogêneos e têm um conjunto completo de periféricos, embora ter um *cluster* com 1.024 mouses na verdade não é muito melhor que ter um *cluster* sem mouse algum. O mais importante é que muitos deles têm proprietários que têm um apego emocional às suas máquinas e tendem a olhar com desconfiança algum astrônomo que queira simular o *big bang* nelas.

Usar estações de trabalho ociosas para formar um *cluster* sempre significa dispor de algum meio de migrar tarefas para fora das máquinas, quando seus donos as reivindicarem. A migração é possível, mas aumenta a complexidade do *software*.

Se modo geral, os *clusters* são conjuntos pequenos, com cerca de 500 PCs. Contudo, também é possível construir *clusters* muito grandes com PCs de prateleira, como o Google faz. Leia mais sobre os clusters, acesse os artigos indicados e fique por dentro!

https://s3info.wordpress.com/2011/01/08/o-cluster-e-as-consultas-do-google-licoes-a-aprender/

http://www.ice.edu.br/TNX/encontrocomputacao/artigos-internos/prof_andersown_computacao_em_cluster.pdf

TROCANDO IDEIAS

Depois de ler os artigos indicados, o que você tem a dizer sobre os clusters?
Poste suas considerações no fórum da nossa aula e também comente a postagem de seus colegas! Essa troca de informações é muito valiosa!

NA PRÁTICA

Vamos falar sobre escalonamentos?

Programadores podem criar *jobs* com facilidade requisitando várias CPUs e executando durante períodos substanciais de tempo. Quando várias requisições independentes estão disponíveis vindas de diferentes usuários, cada uma necessitando um número diferente de CPUs por períodos de tempos diferentes, o *cluster* precisa de um escalonador para determinar qual *job* é executado quando.

No modelo mais simples, o escalonador de *jobs* requer que cada um especifique quantas CPUs necessita. Então os *jobs* são executados em ordem. Nesse modelo, após um *job* ser iniciado, verifica-se se há número suficiente de CPUs disponíveis para iniciar o próximo *job* que está na fila de entrada. Se houver, este é iniciado e assim por diante. Caso contrário, o sistema espera até que mais CPUs fiquem disponíveis.

Um algoritmo de escalonamento melhor evita o bloqueio de cabeça de fila saltando *jobs* que não cabem e escolhendo o primeiro que couber. Sempre que um *job* termina, uma fila de *jobs* remanescentes é verificada em ordem.

Um algoritmo ainda mais sofisticado requer que cada *job* apresentado especifique seu formato, isto é, quantas CPUs ele quer durante quantos minutos. Com essa informação, o escalonador de *jobs* pode tentar montar um esquema em lajotas com o tempo da CPU. Esse esquema é especialmente eficaz quando os *jobs* são apresentados durante o dia para execução a noite, de modo que o escalonador tem todas as informações sobre os *jobs* com antecedência e pode executá-los na melhor ordem.

Desempenho

O ponto principal da construção de um computador paralelo é fazer com que ele execute mais rápido do que uma máquina com um único processador. Se ele não cumprir esse simples objetivo, não vale a pena o ter. Além disso, ele deve cumprir o objetivo de uma maneira eficiente em relação ao custo. Uma máquina que é duas vezes mais rápida do que um uniprocessador a 50 vezes o custo muito provavelmente não será um sucesso de vendas.

Para que possa ser mensurada, são utilizadas as métricas de *hardware* e *software*.

SÍNTESE

Na aula de hoje vimos que está ficando cada vez mais difícil conseguir que os computadores funcionem com mais rapidez apenas aumentando a velocidade de *clock*, em vez disso, os projetistas estão buscando o paralelismo para conseguir ganhos de velocidade.

No nível baixo está o paralelismo no chip e outra forma é o paralelismo no nível da instrução, no qual uma instrução, ou uma sequência delas, emite diversas operações que podem ser executadas em paralelo por diferentes unidades funcionais. E assim vimos vários outros tipos de paralelismo hoje.

COMPARTILHANDO

Por meio das redes sociais ou mesmo em uma conversa informal, leve adiante o que você aprendeu nesta aula. Veja o que cada um tem a dizer sobre o assunto e discutam, troquem ideias, pois é fundamental para sua aprendizagem que haja troca de informações.
Até a próxima aula!