Variância dos Tempos de Execução

Fundamentos dos Sistemas de Tempo Real

Rômulo Silva de Oliveira

eBook Kindle, 2018

www.romulosilvadeoliveira.eng.br/livrotemporeal

Outubro/2018

Sumário

- Introdução
- Variância Causada pelo Software
- Variância Causada pelo Hardware
  - Memória Cache
  - Pipeline
  - Branch Predictor
  - Memórias DRAM
  - Acesso Direto à Memória – DMA
  - Translation Lookaside Buffer – TLB
  - Controle de Frequência
  - Modo de Gerência do Sistema
  - Múltiplas Threads em Hardware
  - Impacto dos Tratadores de Interrupção e de Múltiplas Tarefas

Introdução 1/8

- Tempo de execução de uma tarefa corresponde ao tempo que ela precisa de processador para concluir
  - Considerando que a mesma está sozinha no computador.
  - Não existem outras tarefas
  - Nem mesmo tratadores de interrupções
  - Nem atividades no kernel do sistema operacional

- Isto é diferente do seu tempo de resposta
  - Inclui todas as atrapalhações que ela recebe de outras tarefas
  - e do sistema operacional

Introdução 2/8

- Suponha que uma tarefa é executada muitas vezes
- E o tempo de execução de cada ativação seja medido
- Tempo de execução varia

- Existem aspectos tanto de **software** como de **hardware** capazes de causar esta variação

Introdução 3/8

- Um aspecto fundamental é o **fluxo de controle da tarefa**, isto é, as linhas do código por onde a execução acontece
- Suponha que a tarefa tem um comando IF(EXPRESSÃO)
  - O código a ser executado caso a EXPRESSÃO seja verdadeira é composto por poucas linhas e rápido
  - O código a ser executado caso a EXPRESSÃO seja falsa é composto por muitas linhas e lento
- Mesma coisa para um comando do tipo laço onde o número de iterações é variável
  - Quanto mais iterações forem feitas no laço, a princípio maior será o tempo de execução
- As coisas ficam mais complicadas quando temos, por exemplo
  - um comando IF dentro do laço
  - um laço aninhado dentro de outro laço

Introdução 4/8

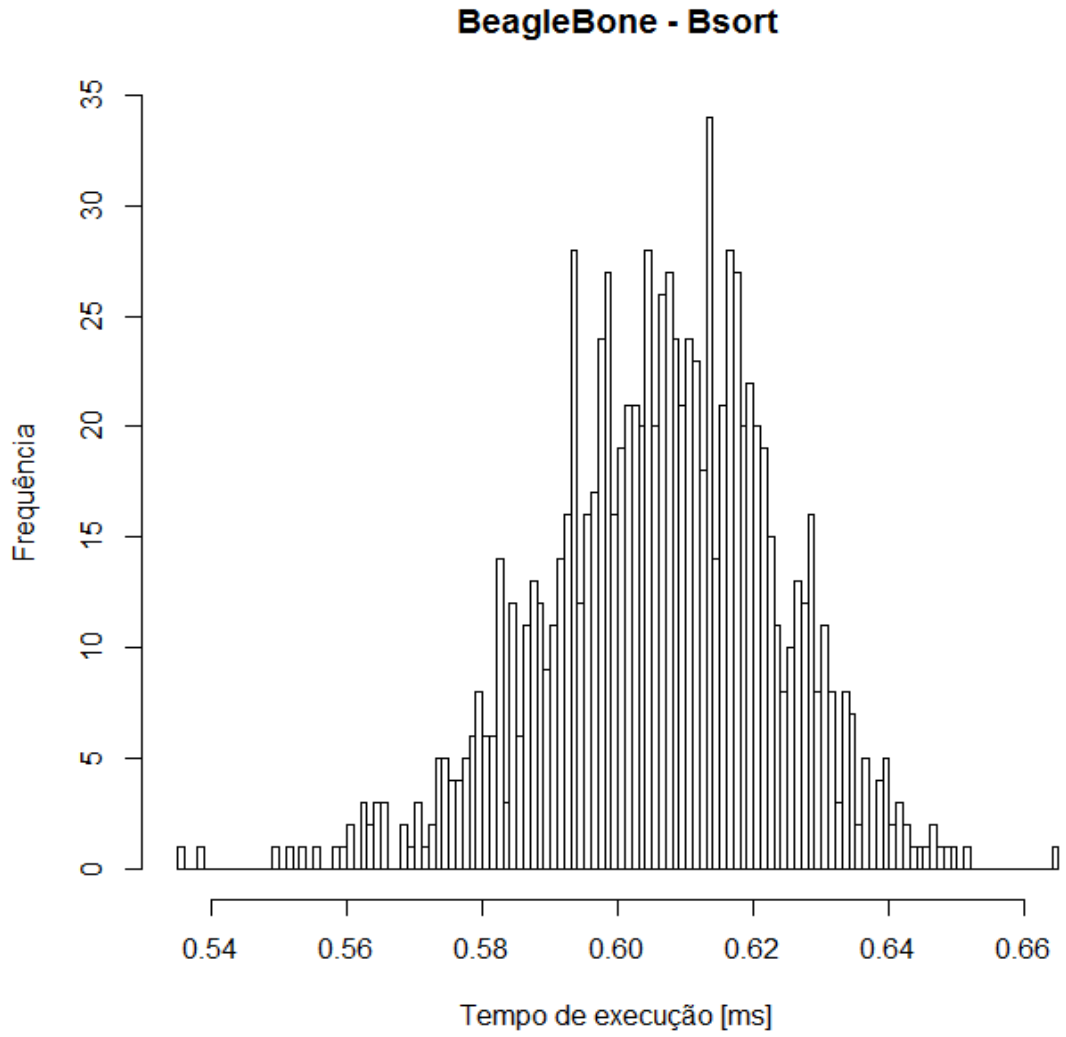
- Processadores modernos contam com vários **mecanismos de aceleração da execução**
  - Apresentam comportamento variável
  - Tempo de execução varia conforme o que foi executado antes
- Por exemplo, memórias cache
- Memória cache é uma memória mais rápida (e mais cara) que mantém dados recentemente acessados no passado
  - Se a tarefa precisar acessar em seguida um dado que está na cache (hit), o acesso será rápido
  - Se o dado não estiver na cache (miss), a memória mais lenta deverá ser acessada

Introdução 5/8

- Considere como exemplo uma tarefa programa na linguagem C
- Uma ordenação de acordo com o algoritmo Bubble Sort
- A cada ativação da tarefa ela ordena um vetor de 100 números inteiros
- A tarefa executa em uma plataforma BeagleBone White
- Após executar a tarefa 1000 vezes, cada vez com um vetor de entrada gerado aleatoriamente, foi construído um histograma

Introdução 6/8

- Tarefa executando bubble sort em uma Beaglebone
- Entradas aleatórias



Introdução 7/8

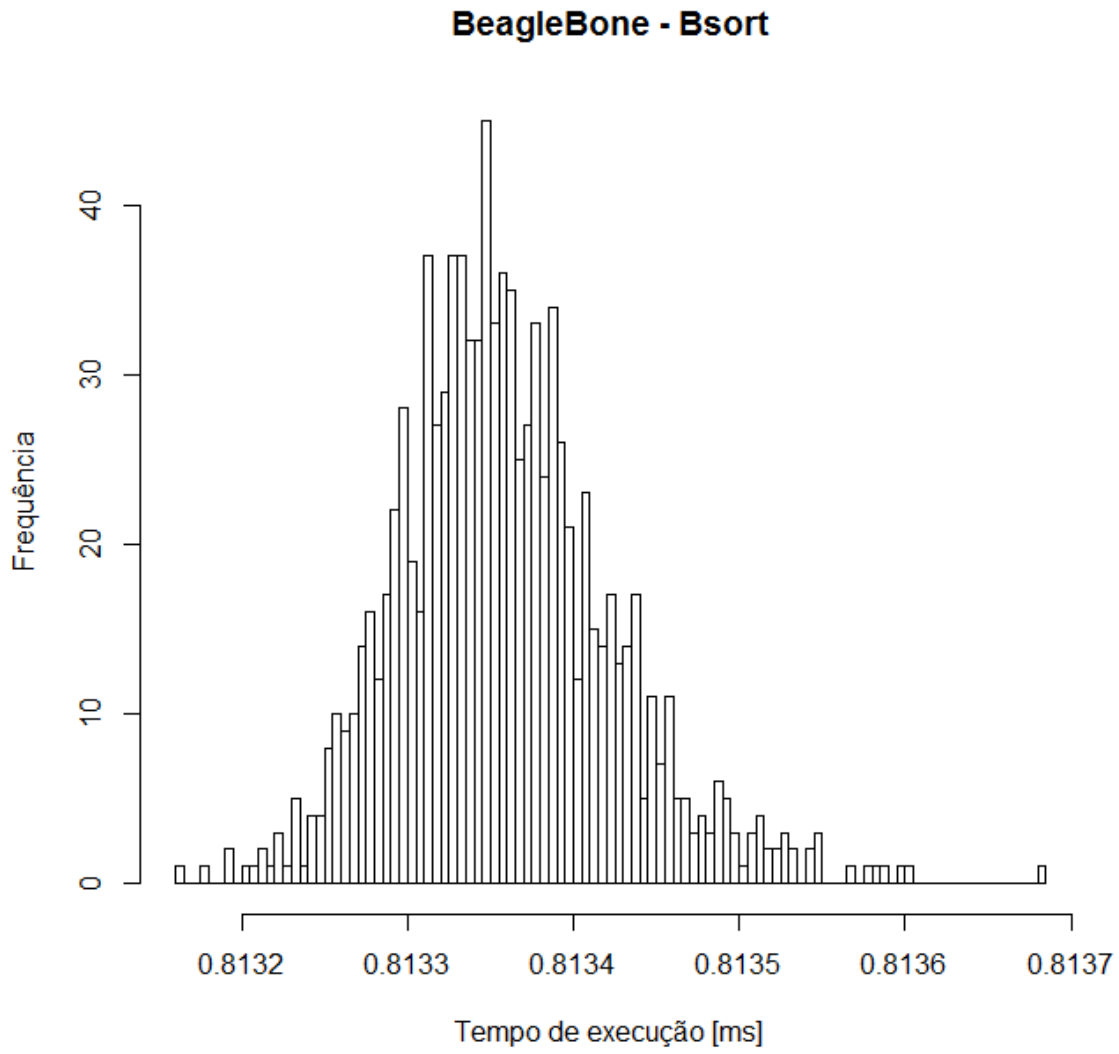
- O que aconteceria se executássemos a tarefa 1000 vezes, porém agora ela recebe sempre o mesmo vetor de entrada ?
- Histograma mostra o resultado para 1000 ativações da tarefa
- Precisa ordenar um vetor de entrada que já está inversamente ordenado
- Este é o cenário que gera o maior número de trocas no bubble sort

- Apesar da variância ser menor, ainda assim o tempo de execução não é constante. Lembre-se que os dados de entrada são sempre os mesmos

- Tempos de execução observados com o vetor invertido são maiores
- Gerar um vetor de entrada inversamente ordenado por acaso é muito improvável
- Os 1000 vetores de entrada gerados aleatoriamente foram sempre “mais fáceis” para o bubble sort do que quando forçamos um reversamente ordenado

Introdução 8/8

- Tarefa executando bubble sort em uma Beaglebone
- Entrada invertida



Sumário

- Introdução
- Variância Causada pelo Software
- Variância Causada pelo Hardware
  - Memória Cache
  - Pipeline
  - Branch Predictor
  - Memórias DRAM
  - Acesso Direto à Memória – DMA
  - Translation Lookaside Buffer – TLB
  - Controle de Frequência
  - Modo de Gerência do Sistema
  - Múltiplas Threads em Hardware
  - Impacto dos Tratadores de Interrupção e de Múltiplas Tarefas

Variância Causada pelo Software 1/11

- A variância no tempo de execução causada pelo software está relacionada com a idéia de fluxo de controle
- O fluxo de controle da tarefa indica por onde no código da tarefa a execução passa

- Praticamente todos os programas empregam comandos do tipo IF-THEN-ELSE
- Cada vez que cada IF é executado, o fluxo de controle pode seguir por um ou outro caminho

Variância Causada pelo Software 2/11

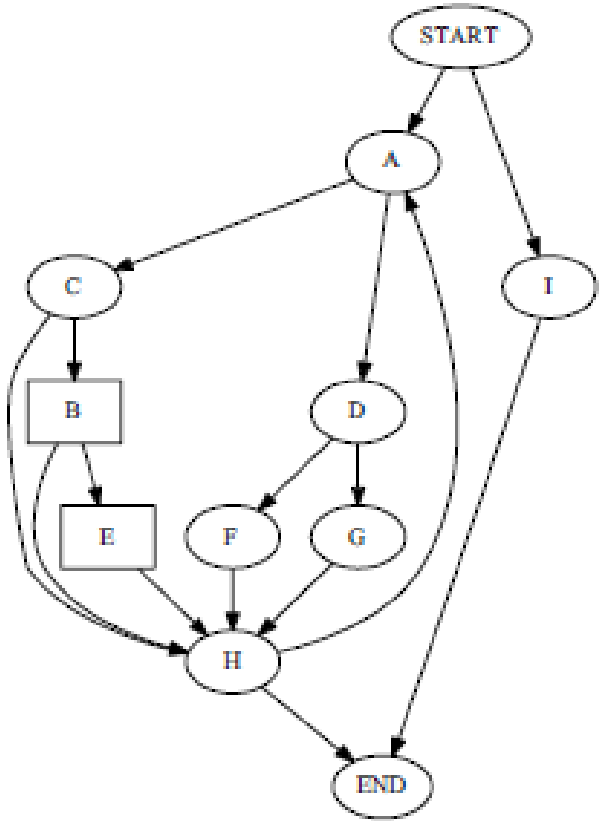
- Toda linguagem de programação inclui mecanismos para a criação de laços
- Aparecem nas mais variadas formas, tais como comandos FOR, WHILE, REPEAT-UNTIL, DO-WHILE, etc.
- Embora o número de iterações possa ser fixo no código, muitas vezes depende do valor de variáveis do programa
- Cada vez que o fluxo de controle entra no laço, seus comandos podem ser executados um número diferente de iterações

Variância Causada pelo Software 3/11

- A forma usual para representar os caminhos possíveis para o fluxo de controle é o **Grafo de Fluxo de Controle**
  - **GFC**, do inglês *control flow graph*
- Exemplo:
  - START precisa ser sempre executada.
  - Se for verdadeira será executado o comando “I” e depois “END”
  - Se for falsa, a expressão “A” é avaliada
  - “A” verdadeiro o fluxo de controle segue para uma cascata de comandos IF que inclui as expressões “C” e “B” e o comando “E”
  - “A” falso a execução desce para o IF com a expressão D e os comandos “F” e “G”
  - O laço acaba no comando WHILE onde a expressão “H” é avaliada e
  - “H” verdadeiro inicia uma nova iteração do laço (flecha para cima)
  - “H” falso, “END” é executado e a tarefa termina

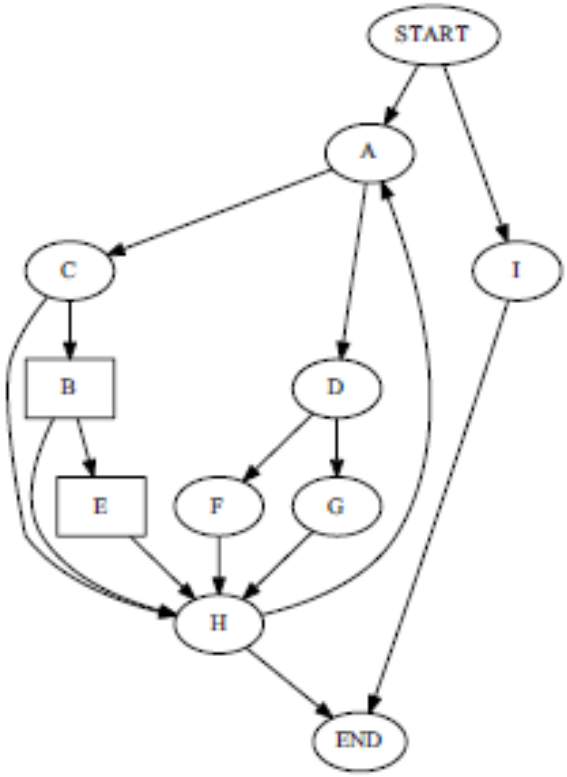
Variância Causada pelo Software 4/11

```
if( START )
   I;
else {
   do {
      if( A ) {
         if( C )
            if( B )
               E;
         }
      else {
         if( D )
            F;
         else
            G;
         }
      }while( H );
   }
END;
```



Variância Causada pelo Software 5/11

- Quantos caminhos diferentes existem entre START e END ?
  - Ramo da direita faz I
  - Ramo da esquerda tem um laço contendo 5 ramos:
    - A D F H
    - A D G H
    - A C H
    - A C B H
    - A C B E H
  - Precisa um limite para o número de iterações do laço



Variância Causada pelo Software 6/11

- Precisa um limite superior para o número de iterações do laço
  - Vamos supor 6, repete o laço de 1 a 6 vezes

  - Executa o laço 1 vez: 5^1 possibilidades
  - Executa o laço 2 vezes: 5^2 possibilidades
  - Executa o laço 3 vezes: 5^3 possibilidades
  - Executa o laço 4 vezes: 5^4 possibilidades
  - Executa o laço 5 vezes: 5^5 possibilidades
  - Executa o laço 6 vezes: 5^6 possibilidades

Variância Causada pelo Software 7/11

- Número de caminhos do laço: $\Sigma_{i=1}^{M} (N^i)$
  - Limite do laço são M vezes
  - Número de ramos no corpo do laço é N

- Dominado por N^M

- Por exemplo, N=4 e M=100
  - Resulta em 4^{100} o que é aproximadamente 10^{60} !!!
  - Completamente intratável processar cada caminho explicitamente

Variância Causada pelo Software 8/11

- **Quantos caminhos existem ?**

- Sem desvio nem laço
  - Apenas 1 caminho

- Com desvios mas sem laços
  - Um certo número de caminhos, depende da combinação dos desvios
  - Porém ainda um numero tratável explicitamente (não são muitos)

- Sem desvios mas com laços
  - Um certo número de caminhos, depende do número de iterações
  - Porém ainda um numero tratável explicitamente (não são muitos)

- Com desvios e com laços
  - Um imenso número de caminhos (ex: laço de 100 com 4 ramos, 4^100)
  - Número intratável explicitamente (ex: 10^60)
  - Pode ter laços dentro de laços

Variância Causada pelo Software 9/11

- O que define qual caminho é executado ?
  - Variáveis de entrada do programa
  - Variáveis globais alteradas em execuções anteriores
  - Data e hora correntes
  - Geração de números aleatórios

- Estes são os caminhos sintaticamente possíveis

- Alguns desses caminhos são semanticamente impossíveis
- Impossível pela semântica do programa
  - Análise de valor pode identificar (parcialmente)
- Impossível pela semântica do ambiente
  - Entradas impossíveis de acontecer na prática

Variância Causada pelo Software 10/11

- IF(A > B)
THEN X;
ELSE Y;
IF(A > B)
THEN Z;
ELSE W;

- A princípio 4 possibilidades:
  - “X Z” “X W” “Y Z” “Y W”
- Olhando as expressões nos dois comandos IF, somente 2 caminhos são possíveis:
  - “X Z” quando A>B
  - “Y W” quando A<=B
  - Os caminhos “X W” e “Y Z” são semanticamente impossíveis.
- É comum a existência de caminhos impossíveis, mas não a ponto de mudar a natureza da questão, que é a explosão do número de caminhos

Variância Causada pelo Software 11/11

- Se uma tarefa possui apenas um caminho, então o tempo de execução dela é constante ?
- Infelizmente não é garantido
- Quando a tarefa tem apenas um caminho, então não existe variância do tempo de execução causada pelo software
- Porém, pode ainda existir muita variância do tempo de execução causada pelo hardware

- Mesmo que exatamente as mesmas instruções de máquina sejam sempre executadas pela tarefa
  - a cada execução o tempo de execução será diferente

Sumário

- Introdução
- Variância Causada pelo Software
- Variância Causada pelo Hardware
  - Memória Cache
  - Pipeline
  - Branch Predictor
  - Memórias DRAM
  - Acesso Direto à Memória – DMA
  - Translation Lookaside Buffer – TLBControle de Frequência
  - Modo de Gerência do Sistema
  - Múltiplas Threads em Hardware
  - Impacto dos Tratadores de Interrupção e de Múltiplas Tarefas

Variância Causada pelo Hardware 1/4

- Mesmo quando uma tarefa executa sempre exatamente as mesmas instruções de máquina
  - o seu tempo de execução pode variar
  - ou não
  - dependendo das características do processador

- Nos processadores mais antigos
  - Tempo necessário para executar cada instrução de máquina corresponde a um número inteiro de ciclos de clock
  - Basta inverter o valor da frequência do processador em Hertz
  - Obter a duração do ciclo de clock em segundos
  - E multiplicar pelo número de ciclos de clock necessários
  - Temos quanto tempo demora uma instrução de máquina

Variância Causada pelo Hardware 2/4

- Microcontrolador Intel MCS-51 (Intel 8051)
  - Faz parte de uma família de microcontroladores (*single chip microcontroller*)
  - 8 bits
  - Lançada em meados de 1980 para uso em *embedded systems*
- "MCS 51 MICROCONTROLLER FAMILY USER'S MANUAL"
  - Quando uma frequência de clock de 12MHz é empregada
  - Instrução de máquina **ADD A,<byte>** (soma ao acumulador um valor imediato de 8 bits) demora sempre 1 microsegundo
  - Instrução de máquina **MUL AB** (multiplicação inteira de dois registradores) demora sempre 4 microsegundos
  - **MOV <dest>,<src>** (cópia de um byte entre dois registradores) demora sempre 2 microsegundos.
  - **JZ rel** (jump condicional se zero) demora sempre 2 microsegundos
  - Etc

Variância Causada pelo Hardware 3/4

- No caso do microcontrolador 8051 da Intel,
  o hardware não introduz variância no tempo de execução da tarefa
- Se as mesmas instruções de máquina são executadas, o mesmo tempo de execução será obtido

- Processadores mais modernos empregam uma gama de mecanismos de hardware que aceleram a execução dos programas
  - Apresentando um comportamento probabilista
- Tais mecanismos tornam a execução das instruções de máquina mais rápida
  - Porém o tempo de execução de uma instrução de máquina é variável
  - Não mais uma constante como no caso do Intel 8051

Variância Causada pelo Hardware 4/4

- Exatamente quais mecanismos de aceleração são empregados varia de processador para processador
- Entre os mais importantes podemos citar:
  - Memória cache
  - Pipeline
  - Branch predictor
  - Memórias DRAM (Dynamic Random Access Memory)
  - DMA (Direct Memory Access)
  - TLB (Translation Lookaside Buffer)
- Por exemplo, o processador **ARM Cortex-M0** emprega um pipeline simples porém não emprega branch predictor
- **Intel Core i7** emprega um pipeline muito mais sofisticado, branch predictor e mais uma vasta gama de mecanismos
  - Tempo de execução das instruções de máquina consideravelmente variável

Memória Cache 1/11

- A memoria cache foi criada para explorar as propriedades de localidade espacial e temporal
- Cria-se uma ilusão de memória rápida de grande capacidade

- O princípio básico é colocar uma memória
  - de baixa capacidade
  - rápida
  - de alto custo
- entre o processador e a memória principal
  - Memória principal é mais barata e tem grande capacidade

- A memória cache de nível primário (L1) consegue satisfazer 90% das referências
  - Se está na cache é um acerto (hit)
  - Se não está na cache é uma falta (miss)

Memória Cache 2/11

- Cache é administrada em blocos ou linhas e não em bytes
  - Define a granularidade na qual a cache opera
- Cada bloco é uma sequência contínua de bytes e inicia em alinhamento comum
- O menor tamanho utilizável de um bloco possui o tamanho natural das palavras do processador
  - 4-bytes para máquinas de 32-bits

- Cache pode ter vários níveis
  - Para dar conta das grandes diferenças de custo e velocidade
- Cada nível implementa um mecanismo que permite uma procura
  - de baixa latência
  - para vericar se dado bloco está ou não contido na cache

Memória Cache 3/11

- Devido ao alinhamento dos blocos, os bits menos signicativos do endereco são sempre zero
- Se é preciso acessar um byte diferente do primeiro, os bits menos signicativos são utilizados como deslocamento para encontrar o byte correto

- Em termos de localização de blocos há três possíveis organizações

- Mapeamento direto
- Mapeamento totalmente associativo
- Mapeamento associativo por conjunto

Memória Cache 4/11

- **Mapeamento direto**
- Um endereço particular pode residir apenas em uma única localização na cache
- Esta localização é normalmente determinada extraindo **n bits** do endereço e os utilizando para indexar diretamente uma das 2^n possíveis localizações na cache

- Mapeamento de muitos endereços para somente um lugar na cache
  - Cada localização necessita de uma etiqueta (tag) que correspondem aos bits restantes do bloco gravado naquela localização
- Em cada procura, o hardware precisa
  - Ler a etiqueta e comparar com o endereco da referência
  - Para determinar se foi acerto ou falta

Memória Cache 5/11

- **Totalmente associativo**

- Permite um mapeamento de múltiplos endereços para múltiplas localizações na cache
  - Qualquer endereço de memória pode residir em qualquer localização na cache
- Todas as localizações da cache devem ser verificadas para encontrar os dados
- Cada linha da cache deve possuir uma etiqueta com o endereço dos dados que hospeda
  - Para o hardware comparar e detectar uma falta ou acerto

Memória Cache 6/11

- **Associatividade por conjunto**

- Existe um mapeamento de múltiplos endereços para algumas localizações na cache
- Em cada procura, um subconjunto de bits de endereço são utilizados para geração do índice
  - Como no caso de mapeamento direto
- Contudo, o índice corresponde a um conjunto de entradas que são procuradas em paralelo em busca da etiqueta correta
  - Existe associatividade por conjunto

Memória Cache 7/11

- Cada nível de cache possui uma capacidade finita
- Deve haver uma política para despejar ocupantes atuais possibilitando espaço para blocos com referências mais recentes
- Política de substituição: algoritmo para identificar quem será despejado

- Em mapeamento direto o problema é trivial
  - Somente um lugar para cada endereço

- Na associatividade completa ou por conjunto, precisa escolher
  - Um novo bloco pode ser colocado em várias possíveis localizações
- Três políticas básicas:
  - FIFO (primeiro a entrar, primeiro a sair)
  - LRU (menos recentemente utilizado)
    - Aproximada por não-mais-recentemente-utilizado (not-most-recently-used, NMRU)
  - Aleatoriamente

Memória Cache 8/11

- A utilização de memória cache implica em múltiplas cópias do bloco
- Enquanto há apenas leitura não ocorre nenhum problema
- Para escrita deve haver mecanismos para atualização dos blocos nos demais níveis da memória
- Existem basicamente dois mecanismos conhecidos como
  write-through e
  write-back

Memória Cache 9/11

- O mecanismo ***write-through*** simplesmente propaga a atualização para os níveis inferiores
  - É fácil implementar
  - Não existe ambiguidade sobre a versão dos dados

- Exige muita demanda de barramento
- Disparidade da frequência entre o processador e a memória principal torna impossível a utilização em todos os níveis da hierarquia

- Este mecanismo ainda deve especificar se vai ou não alocar espaço na cache quando ocorre uma falta em escrita de um bloco

- A política *write-allocate* implica em alocar o bloco na cache
- A política *write-no-allocate* evita a instalação do bloco na cache
  - Realiza esta operação apenas em faltas de leitura

Memória Cache 10/11

- O mecanismo ***write-back*** prorroga a atualização da memória principal
  - e dos outros níveis de cache se existirem
- até quando a linha que o bloco alterado ocupa for necessária para hospedar outro bloco
- ou quando existir um conflito entre caches de mesmo nível
  - no caso de multiprocessadores

Memória Cache 11/11

- Caches são muito empregadas na prática
- Quando um dado está na cache, o tempo de acesso a ele é muito menor do que o tempo para carregá-lo da memória principal
- Porém, afeta o tempo de acesso aos dados na memória
  - Afeta o tempo de execução das instruções de máquina

- A mesma instrução de máquina pode ter tempos de execução muito diferentes
- Depende se o dado necessário encontra-se ou não na cache no instante que ele é necessário

Pipeline 1/18

- Técnica poderosa para aumentar o desempenho do sistema
- Intel i486 foi a primeira implementação da arquitetura IA32 com pipeline

- A técnica do pipeline particiona o sistema entre múltiplos estágios
  - adicionando buffer entre eles
- A computação original é decomposta em K estágios
- Uma nova instrução pode ser inicializada assim que a anterior atravessar o primeiro estágio
- Ao invés de iniciar uma instrução a cada T unidades de tempo, inicializa-se uma a cada T/K unidades de tempo
  - Os processamentos das K instruções são sobrepostos pelo pipeline

Pipeline 2/18

- A princípio, o ganho de desempenho é proporcional ao comprimento do pipeline
  - Quanto mais estágios, maior desempenho

- Porém, existem limitações físicas relacionadas à frequência as quais determinam
  a quantidade de estágios que podem ser utilizados

Pipeline 3/18

- Um ciclo de instrução básico pode ser dividido em cinco subcomputações genéricas:

• Busca de instruções	*instruction fetch*
• Decodificação da instrução	*instruction decode*
• Busca dos operandos	*operand's fetch*
• Execução da instrução	*instruction execution*
• Escrita dos resultados	*operand store*

Pipeline 4/18

- Processador sem pipeline

Clock	Busca	Decodifica	Busca operandos	Execução	Escrita
1	Instrução 1				
2		Instrução 1			
3			Instrução 1		
4				Instrução 1	
5					Instrução 1
6	Instrução 2				
7		Instrução 2			
8			Instrução 2		
9				Instrução 2	
10					Instrução 2
11	Instrução 3				
12		Instrução 3			
13			Instrução 3		
14				Instrução 3	
15					Instrução 3

Pipeline 5/18

- Processador com pipeline de 5 estágios

Clock	Busca	Decodifica	Busca operandos	Execução	Escrita
1	Instrução 1				
2	Instrução 2	Instrução 1			
3	Instrução 3	Instrução 2	Instrução 1		
4		Instrução 3	Instrução 2	Instrução 1	
5			Instrução 3	Instrução 2	Instrução 1
6				Instrução 3	Instrução 2
7					Instrução 3
8					
9					
10					
11					
12					
13					
14					
15					

Pipeline 6/18

- As subcomputações genéricas correspondem a uma partição natural de um ciclo de instrução
  - Forma um pipeline genérico de cinco estágios

- Ainda múltiplas subcomputações com baixa latência podem formar um unico estágio
- ou ainda alguns dos cinco estagios genéricos também podem ser agrupados formando um melhor balanceamento do pipeline

Pipeline 7/18

- O aumento de desempenho em k consiste em três idealismos:

- Sub-computações uniformes
  - A computação a ser realizada é dividida em sub-computações com latências iguais

- Computações idênticas
  - A mesma computação é realizada repetidamente em uma grande quantidade de dados

- Computações independentes
  - As computações são independentes entre si

Pipeline 8/18

- As sub-computações uniformes mantem o idealismo que cada estágio possui o mesmo tempo de execução, ou seja, T/k
- Porém é impossível particionar computações em estágios perfeitos e balanceados

- Exemplo:
- Uma operação de 400ns é dividida em 3 estágios de 125ns, 150ns e 125ns
- Como a frequência do pipeline é determinada pelo estágio mais longo, esta será limitada pelos 150ns
- O primeiro e o terceiro estágio possuem uma ineficiência de 25ns
- E o tempo total da operação utilizando o pipeline será de 450ns ao invés de 400ns

Pipeline 9/18

- As computações idênticas assumem que todos os estágios do pipeline são sempre utilizados em todos os conjuntos de dados

- Esta suposição não é válida quando há execução de múltiplas funções,
  onde nem todos os estágios são necessários para suporta-las

- Como nem todos os conjuntos de dados precisarão de todos os estágios, estes ficaram esperando
  - mesmo que não precisem executar tal subfunção
  - devido a característica síncrona do sistema

Pipeline 10/18

- As computações independentes assumem que não há dependência de dados ou controle entre qualquer par de computações
- Esta suposição permite que o pipeline opere em fluxo contínuo
  - Nenhuma operação precisa esperar a completude da operação anterior
- Em alguns sistemas esta suposição é válida
- Mas para pipelines de instruções uma operação poderá precisar do resultado da anterior para prosseguir
- Se esta situação ocorre e as duas operações encontram-se em processamento,
  - Uma operação deverá esperar o resultado da anterior
  - Isto ocasiona uma flutuação do pipeline (pipeline stall)
  - Quando isto ocorre, todos os estágios anteriores também deverão esperar, ocasionando uma série de estágios ociosos

Pipeline 11/18

- Dadas duas instruções i e j sendo j precedido de i,
  j pode ser dependente de i em várias situações

- Instrução j requer um operando que está na imagem de i
- **Dependência read-after-write** (RAW) ou dependência verdadeira
- A instrução j não pode executar até a completude de i

- i : R3 <- R1 op R2
  j : R5 <- R3 op R4

- Outras situações existem, não serão mostradas

Pipeline 12/18

- Pipeline sofre um stall em função de dependência de dados

Clock	Busca	Decodifica	Busca operandos	Execução	Escrita
1	Instrução 1				
2	Instrução 2	Instrução 1			
3	Instrução 3	Instrução 2	Instrução 1		
4	Instrução 4	Instrução 3	Instrução 2	Instrução 1	
5	Instrução 5	Instrução 4	Instrução 3	Instrução 2	Instrução 1
6				**espera**	Instrução 2
7	Instrução 6	Instrução 5	Instrução 4	Instrução 3	
8	Instrução 7	Instrução 6	Instrução 5	Instrução 4	Instrução 3
9					
10					
11					
12					
13					
14					
15					

Pipeline 13/18

- Além das dependências de dados
  existem as **dependências de controle**
- Dadas as instruções i e j, com j precedido por i, a execução de j depende do resultado de i

- As dependências de controle são resultados da estrutura de controle de fluxo do programa
- Saltos condicionais geram incerteza no sequenciamento das instruções

Pipeline 14/18

- Pipeline sofre um stall em função de dependência de controle

Clock	Busca	Decodifica	Busca operandos	Execução	Escrita
1	Instrução 1				
2	Instrução 2	Instrução 1			
3	Instrução 3	Instrução 2	Instrução 1		
4	***Instrução 4***	Instrução 3	Instrução 2	Instrução 1	
5	***Instrução 5***	***Instrução 4***	Instrução 3	Instrução 2	Instrução 1
6	***Instrução 6***	***Instrução 5***	***Instrução 4***	Instrução 3	Instrução 2
7	***Instrução 7***	***Instrução 6***	***Instrução 5***	***Instrução 4***	**Instrução 3!!!**
8	Instrução 88				
9	Instrução 89	Instrução 88			
10		Instrução 89	Instrução 88		
11			Instrução 89	Instrução 88	
12				Instrução 89	Instrução 88
13					Instrução 89
14					
15					

Pipeline 15/18

- Como a semântica do programa requer que as dependências sejam respeitadas
- e a execução de instruções por um pipeline pode facilmente romper o sequenciamento
- deve haver mecanismos de identicação e resolução de dependências

- Uma potencial violação das dependências é conhecida como pipeline hazzards
- As dependências verdadeiras, anti-dados e de saída devem ser identificadas e resolvidas através de mecanismos de hardware:
  Pipeline interlock

Pipeline 16/18

- Máquinas superescalares são capazes de avançar múltiplas instruções pelos estágios do pipeline
  - Elas incorporam múltiplas unidades funcionais
- Aumentam a capacidade de processamento concorrente a nível de instrução aumentando o throughput
- Podem executar instruções em ordem diferente do programa
- A ordem sequencial das instruções no programa implica em algumas precedências desnecessárias entre as instruções
- A capacidade de executar as instruções fora de ordem
  - alivia a imposição sequencial
  - permite mais processamento paralelo sem modificação do programa original

Pipeline 17/18

- Um pipeline de K estágios teoricamente pode aumentar o desempenho em um fator K
  - paralelismo temporal
- De modo alternativo, pode-se atingir o mesmo fator de velocidade empregando K cópias do mesmo pipeline processando K instruções em paralelo
  - paralelismo espacial

- **Processadores superescalares** utilizam as duas técnicas de paralelismo

Pipeline 18/18

- Para evitar ciclos de flutuação desnecessários:
  é permitido que novas instruções sejam adiantadas
  quando há ciclos de flutuação por dependências
  ou faltas de memória
- Este adiantamento pode mudar a ordem de execução das instruções com relação ao código do programa
- A execução fora de ordem permite que estas sejam executadas assim que os operandos estejam disponveis

- Um pipeline paralelo que permite execução fora de ordem é chamado de pipeline dinâmico
  - Emprega buffers multi-entrada
  - As instruções entram e saem em ordens diferentes

Branch Predictor 1/3

- As dependências de controle induzem uma quantidade signicativa de flutuações no pipeline
- **Brach Prediction**: técnicas de previsão do controle de fluxo
- O comportamento desse tipo de instrução é altamente previsível
- A técnica chave para minimizar as penalidades e maximizar o desempenho é especular tanto o endereço quanto a condição do salto nas instruções de controle
- Obviamente que a utilização de técnicas especulativas requerem mecanismos de recuperação de erros de previsão

Branch Predictor 2/3

- A especulação do endereço envolve o uso de um buffer
- **Branch Target Buffer (BTB)**
- Armazena o endereço alvo das instruções de controle anteriores

- O BTB é uma memória cache pequena
  - acessada durante o estágio de busca de instruções
- Cada entrada no BTB contem dois campos:
  - endereço da instrução - Branch Instruction Address (BIA)
  - endereço alvo - Branch Target Address (BTA)

- O BTB é acessado concorrentemente com a busca das instruções
- Quando o contador de programa (CP) bate com o BIA, o campo BTA é utilizado para a busca da próxima instrução
  - se a condição é especulada como verdadeira para o salto

Branch Predictor 3/3

- A avaliação do endereço e condições são comparadas com a versão especulada
- Caso concordem,
  - A previsão estava correta, não há penalidade
- Caso contrário,
  - Um erro de previsão ocorreu, deve-se iniciar a recuperação
- O resultado da execução não especulativa atualiza o BTB
- Um erro de previsão é custoso
  - Ciclos perdidos no processamento de instruções não utilizadas
  - Limpeza dos efeitos das instruções falsamente previstas
- Cerca de 90% das previsões são corretas

Memórias DRAM 1/2

- **Memórias DRAM** (***Dynamic Random Access Memory***)
- São divididas em bancos que contêm uma ou mais matrizes de células que armazenam 1 bit cada
- Cada célula é composta de um transistor e um capacitor
- Esse capacitor sofre perdas de energia e precisa ser periodicamente "recarregado" para que sua tensão permaneça distinguível
  - *Refresh*

- Durante o refresh a memória não pode ser acessada, portanto acessos realizados durante ele precisam ser contidos (atrasados)

Memórias DRAM 2/2

- Cada linha da matrix de células precisa ser transferida para um buffer antes de ser acessada
- Células de uma mesma linha que já está no buffer podem ser acessadas sem necessitar de uma nova transferência para o buffer
- Diferentes bancos são independentes e, portanto, podem ser acessados de forma paralela se a arquitetura permitir

- Acessos consecutivos a uma memória DRAM podem ter latência (atraso) diferente, em função dos bancos, das linhas e das colunas acessadas
  - A latência depende do histórico de acessos anteriores

Acesso Direto à Memória – DMA 1/3

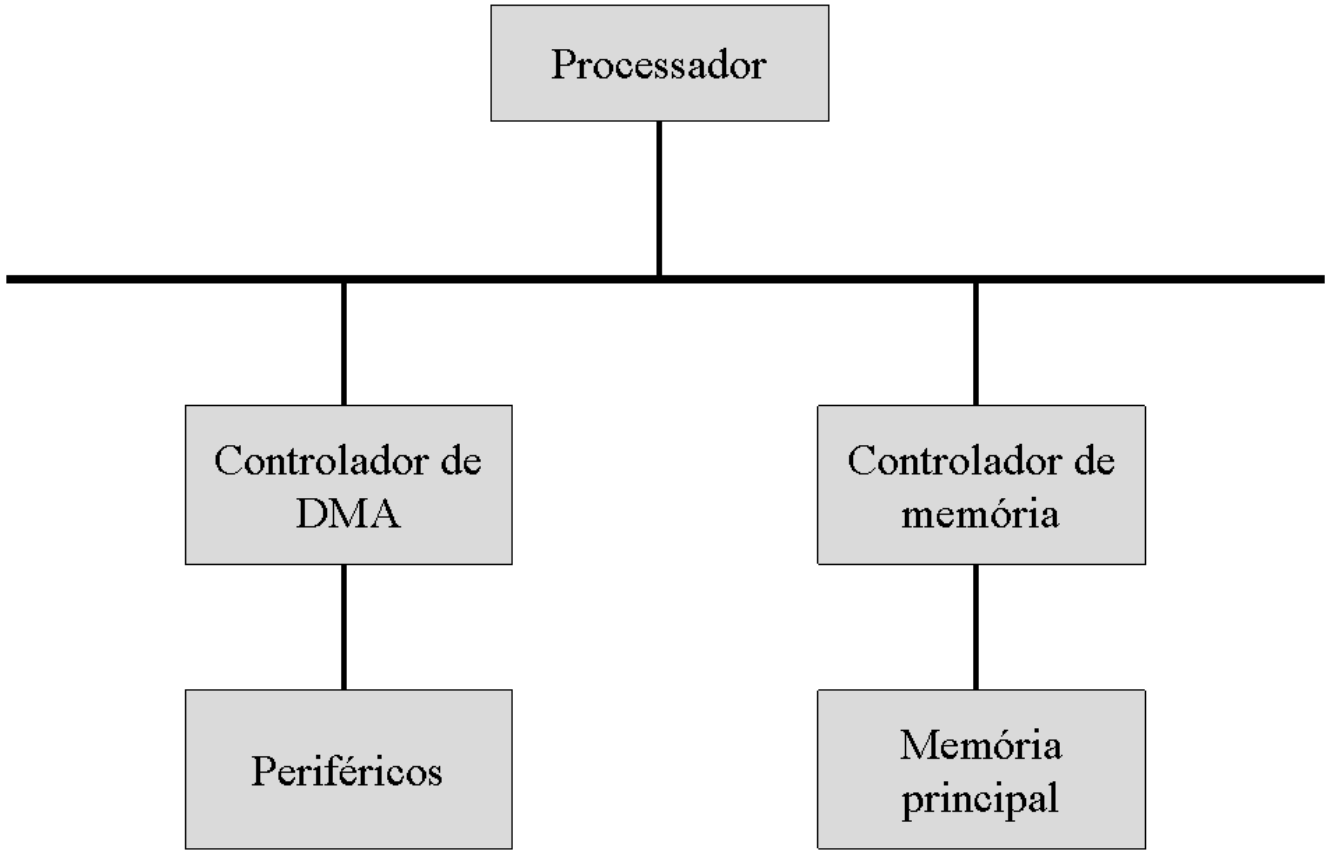
- **Controladores DMA (*Direct Memory Access)***
  - Permitem controladores de dispositivos periféricos acessar diretamente a memória
  - Para a transferência rápida de dados
  - Sem a necessidade do processador
- Mecanismo muito eficiente para o caso de periféricos que fazem a leitura ou escrita de grandes blocos de dados
  - Disco magnético
  - Ethernet

Acesso Direto à Memória – DMA 2/3

- Controladores DMA são conectados à memória através dos mesmos barramentos que o processador
- Podem causar atrasos devido a contenção
  - disputa pelo barramento
- Qualquer acesso do processador à memória sofrerá atrasos se ocorrer quando o DMA estiver operando
- O acesso do controlador DMA impede o acesso do processador à memória
  - Afeta os tempos de execução das instruções de máquina

Acesso Direto à Memória – DMA 3/3

- Barramento da memória compartilhado por processador e controlador de DMA



Translation Lookaside Buffer – TLB 1/4

- O mapeamento da memória lógica para a física deve ser gravado em uma memória de tradução
- O sistema operacional é responsável por atualizar esse mapeamento quando necessário
- O processador deve acessar a memória de tradução para determinar o endereço físico de cada acesso lógico que ocorre
- Cada entrada de tradução deve conter
  - o endereço lógico
  - o endereço físico
  - bits de permissão para leitura, escrita e execução
  - bits adicionais para a gerência da tabela

Translation Lookaside Buffer – TLB 2/4

- As memórias de tradução são chamadas de tabelas de páginas
- Podem ser organizadas em tabelas de páginas diretas
  - *Forward page tables*
  - Tabelas de páginas diretas são mais simples
  - Possui uma entrada para cada bloco do espaço de endereçamento lógico
  - Estrutura grande, com muitas entradas não utilizadas
  - As tabelas diretas são implementadas em múltiplos níveis
- Ou tabelas de páginas invertidas
  - *Inverted page tables* ou *hashed page tables*
  - Existem apenas entradas suficientes para mapear todos os endereços da memória física
  - Pode ser armazenada mais confortavelmente
  - É basicamente uma tabela hash onde o endereço lógico é a chave
  - No caso de memória virtual, ela não comporta os endereços de disco

Translation Lookaside Buffer – TLB 3/4

- Estrutura chamada de *translation lookaside buffer* (TLB) é usada
- TLB contem um número pequeno de entradas de páginas
  - Tipicamente 64 a 256
  - Organizadas por associatividade completa
- O processador provê um rápido hardware de pesquisa associativa para converter referências à memória
- Faltas no TLB resultam em acesso à tabela de páginas
- Arquiteturas MIPS, Alpha e Sparc empregam
  software TLB miss handler
- Arquiteturas PowerPC e Intel IA-32 empregam
  hardware TLB miss handler
  - Incluem uma máquina de estados para acesso à memória por hardware para procura da tabela de páginas

Translation Lookaside Buffer – TLB 4/4

- Mapeamentos de endereço lógico para endereço físico
  - Acontecem com informações presentes na TLB
  - São significativamente mais rápidos que aqueles que precisam ser carregados da memória principal
- Este mapeamento é necessário a cada acesso à memória
- Produz efeitos semelhantes aos das memórias cache sobre os tempos de execução
- Por exemplo, possivelmente a leitura da primeira instrução de uma função C chamada requererá um acesso extra à tabela de páginas na memória
  - Copia as informações de mapeamento para a TLB
- A segunda instrução da função C já terá suas informações de mapeamento na TLB
  - MMU muito mais rápida
- Tempo de execução de uma instrução de máquina depende de quais instruções foram executadas antes

Controle de Frequência 1/1

- Muitos processadores oferecem mecanismos para o **escalonamento dinâmico de frequência** (***dynamic frequency scaling***)
- Permite alterar a frequência do clock do processador dinamicamente
  - Durante a execução do sistema
- Para economizar energia
- Reduzir a quantidade de calor gerado
- Reduzir o nível de ruído gerado
- Através da redução da frequência do clock
  - Consequente redução do desempenho
- No caminho inverso, a frequência pode ser elevada
  - Em momentos de grande demanda por processamento
- Escalonamento dinâmico de frequência tem um impacto direto e enorme nos tempos de execução das tarefas
- Prática comum: Desligar o escalonamento dinâmico de frequência

Modo de Gerência do Sistema 1/1

- **Modo de Gerência do Sistema**
  - **SMM** – ***System Management Mode***)
- Modo de operação do processador criado para atividades de controle geral
  - Proteção contra sobreaquecimento e gestão de energia
- Invisíveis mesmo para o sistema operacional
- Código que implementa fica em memória permanente (*firmware*)
  - Fora do acesso do sistema operacional
- Dependendo do fabricante do computador, este recurso é usado ou não, e pode ser usado de diferentes maneiras
- Fabricantes de computadores não divulgam o seu uso
  - Qual o impacto temporal dele sobre o sistema operacional e as aplicações
- Procurar computadores sem SMM

Múltiplas Threads em Hardware 1/2

- **Hiperprocessamento** (***hyper-threading***)
- Duplica alguns de seus componentes em hardware
- Executa vários (normalmente dois) fluxos de controle concorrentes e independentes
- Nem todos os componentes de hardware são duplicados
  - Não trata-se de um processador com dois núcleos (*multicore*)
  - Trata-se de duas threads em hardware que compartilham um processador parcialmente duplicado
- Vantagem está em obter mais processamento médio

Múltiplas Threads em Hardware 2/2

- O tempo de execução de uma tarefa depende profundamente do que sua companheira de processador está fazendo
- Uma tarefa sofrerá mais ou menos atrasos dependendo dos conflitos que ela tiver com a outra tarefa que executa no mesmo processador e compartilha componentes de hardware

- A mesma tarefa pode exibir tempos de execução muito diferentes caso execute em um momento sozinha e em outro momento com outra tarefa pesada como companheira de *hyperthreading*

- *Hyperthreading* não é recomendada para sistemas de tempo real
- Variância gerada nos tempos de execução torna a verificação dos requisitos temporais muito difícil

Impacto de Múltiplas Tarefas ou Threads 1/6

- O tempo de execução de uma instrução de máquina depende do que aconteceu antes no processador
  - com mecanismos de aceleração com comportamento probabilista

- O fato de uma instrução ou dado ter sido acessado recentemente aumenta as chances dele ser encontrado na memória cache
- O fato da instrução anterior ser um desvio condicional que provocou o esvaziamento do pipeline aumenta o tempo de execução da instrução imediatamente após a execução do desvio
- O fato do branch predictor acertar a decisão de um desvio condicional evita o esvaziamento do pipeline e diminui o tempo de execução da instrução seguinte

Impacto de Múltiplas Tarefas ou Threads 2/6

- O tempo de execução de uma instrução de máquina depende do que aconteceu antes no processador
  - com mecanismos de aceleração com comportamento probabilista

- O fato de dois dados estarem na mesma linha do mesmo banco de uma memória DRAM reduz o tempo de acesso
- O fato do controlador de DMA ocupar o barramento de memória exatamente no momento que uma instrução de máquina requer acesso à memória aumenta o seu tempo de execução
- O fato de uma instrução de máquina residir em uma página lógica cuja informação de mapeamento se encontra na TLB da MMU reduz o tempo de acesso

Impacto de Múltiplas Tarefas ou Threads 3/6

- Todos estes mecanismos de hardware operam simultaneamente
- O que um deles faz acaba afetando o comportamento dos demais
- Torna complexo tentar prever quanto tempo exatamente cada instrução de máquina demora

- Todos os exemplos apresentados consideraram a execução de apenas uma tarefa
- O tempo de execução de cada instrução de máquina depende do histórico da tarefa, ou seja, de quais instruções de máquina ela executou antes

Impacto de Múltiplas Tarefas ou Threads 4/6

- Na grande maioria dos sistemas de tempo real uma tarefa não está sozinha no computador
  - Tratadores de interrupções de periféricos
  - Tarefas de mais alta prioridade

- No caso de uma tarefa X ser interrompida por um tratador de interrupção ou para a execução de outras tarefas
  - Todos os exemplos de como o tempo de execução de uma instrução de máquina pode variar continuam válidos
  - Porém, agora a situação é muito pior
  - A execução de um trecho de código que não pertence à tarefa X em questão (pertence ao tratador de interrupções ou a outras tarefas) vai destruir a história da tarefa X nos vários mecanismos de hardware

Impacto de Múltiplas Tarefas ou Threads 5/6

- Instruções e dados da tarefa X serão removidos da memória cache para abrir espaço para tratadores de interrupções e/ou outras tarefas

- Quando ocorre o desvio da execução, interrompendo a tarefa X, o conteúdo do pipeline é esvaziado
  - No retorno da tarefa X o pipeline inicia vazio

- A memória do *branch predictor* é limitada e as informações relativas à tarefa X serão substituídas pelas informações dos outros códigos
  - No retorno da tarefa X o *branch predictor* precise "aprender" novamente como são os desvios condicionais da tarefa X

Impacto de Múltiplas Tarefas ou Threads 6/6

- Uma sequência de acessos pela tarefa X a dados da mesma linha e banco da memória DRAM seria interrompida
- As demais tarefas do sistema podem solicitar acessos a periféricos que aumentam a atividade do controlador de DMA
- Informações de mapeamento relativas à memória lógica da tarefa X serão removidas da TLB da MMU para abrir espaço para as informações de mapeamento das outras tarefas executando
- **Múltiplas tarefas, threads e tratadores de interrupção <u>aumentam</u>** a variância causada pelo hardware

Sumário

- Introdução
- Variância Causada pelo Software
- Variância Causada pelo Hardware
  - Memória Cache
  - Pipeline
  - Branch Predictor
  - Memórias DRAM
  - Acesso Direto à Memória – DMA
  - Translation Lookaside Buffer – TLB
  - Controle de Frequência
  - Modo de Gerência do Sistema
  - Múltiplas Threads em Hardware
  - Impacto dos Tratadores de Interrupção e de Múltiplas Tarefas