Arquiteturas de Computadores

Multithreading

Fontes dos slides: **Curso *Advanced Computer Architectures***
Prof. Marco Ferretti - Università di Pavia

Multithreading

- Uma CPU multithreading não é uma arquitetura paralela estritamente falando; multithreading é obtida com uma única CPU, mas permite que um programador projete e desenvolva aplicações com um conjunto de programas chamados de threads que podem virtualmente executar em paralelo.
- Se estes programas são executados em uma "multithreaded" CPU, eles poderão explorar melhor as características de arquitetura da CPU.
- O que acontece com a execução destes programas em uma CPU que não suporta multithreading?

Multithreading

- Multithreading aborda um problema básico de qualquer CPU pipelined: a na cache causa uma "longa" espera, necessária para buscar da RAM informação que está faltando. Se nenhuma outra instrução independente estiver disponível para ser executada, o pipeline para.
- Multithreading é solução para evitar a perda de ciclos de relógio para obter os dados que não estão na cache: faz com que a CPU gerencie mais threads de uma tarefa concorrentemente; se uma thread for bloqueada, a CPU pode executar instruções de outra thread, mantendo as unidades funcionais ocupadas.
- Então, por que não é possível que threads de tarefas diferentes também sejam emitidas de forma concorrente?

Multithreading

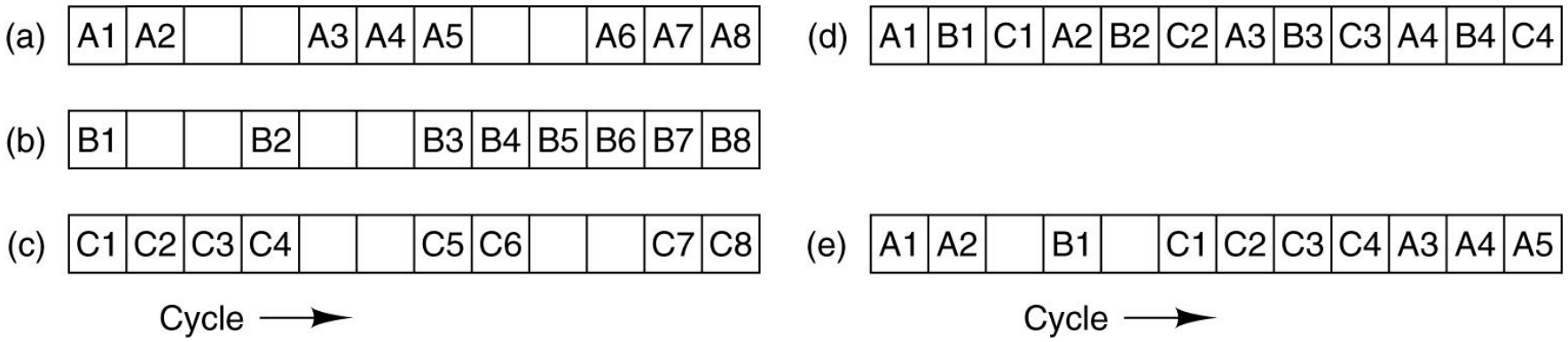
- Para realizar multithreading, a CPU deve gerenciar o estado da computação de cada thread.
- Cada thread deve ter um Contador de Instruções e um conjunto de registradores privados, separados das outras threads.
- Além disso, a troca de contexto das threads deve ser mais eficiente que a de processo, que geralmente requerem centenas ou milhares de ciclos de relógio
- Existem duas técnicas básicas de multithreading:
  - Multithreading de granulosidade fina
  - Multithreading de granulosidade grossa

Multithreading de granulosidade fina

- O chaveamento entre threads ocorre em cada instrução, independentemente do fato que uma thread tenha causado falta na cache
- O escalonamento de instruções entre as threads deve obedecer uma política round-robin, e a CPU deve realizar o chaveamento das threads sem overhead
- Se existir um número suficiente de threads, é provável que pelo pelos uma esteja ativa (não parada) e a CPU é mantida executando instruções

Multithreading de granulosidade fina

- (a)-(c) três threads e paradas associadas (slots vazios).
- (d) Multithreading de granulosidade fina. Cada slot é um ciclo de relógio, e assume-se que cada instrução pode ser completada em um ciclo de relógio, a não ser que ocorra uma parada



Neste exemplo, 3 threads mantém a CPU ocupada, mas e se a parada de A2 durar 3 ou mais ciclos de relógio?

Multithreading de granulosidade fina

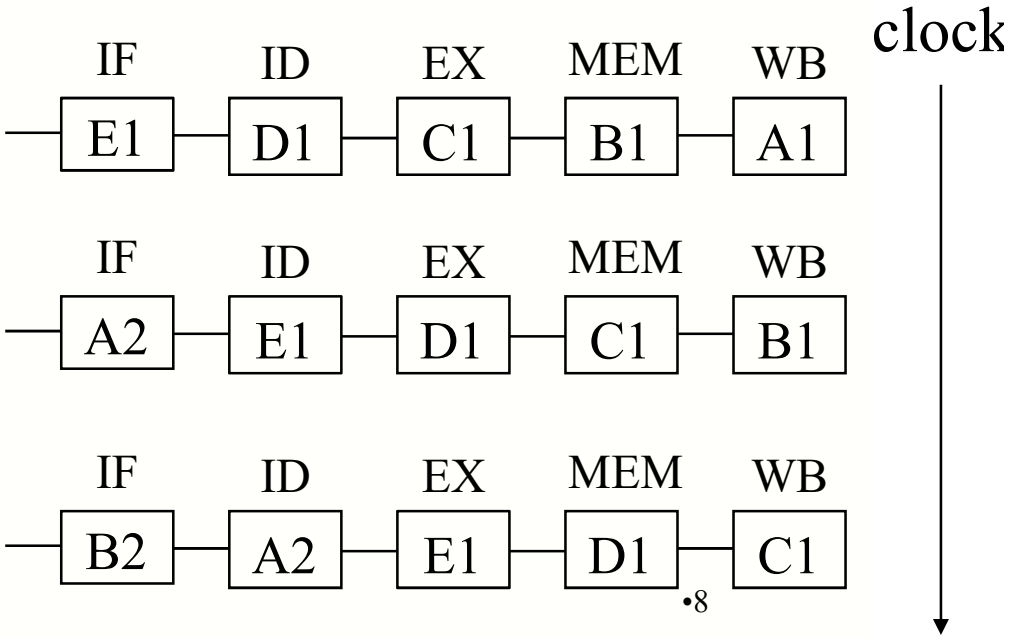
- A parada da CPU pode ser devido a uma falta na cache CPU, mas também devido a uma real dependência de dados ou a um desvio: técnicas ILP dinâmicas dynamic não garantem sempre que uma parada do pipeline possa ser evitada
- Em uma arquitetura pipelined com multithreading de granulosidade, se:
  - o pipeline tem *k* estágios,
  - existem pelo menos *k* threads para serem executadas,
  - a CPU pode executar uma troca de thread em cada ciclo de relógio
- então não pode nunca ter mais de uma instrução por thread no pipeline em qualquer instante, de modo que não haja conflitos devido às dependências, e o pipeline nunca pare

Multithreading de granulosidade fina

Exemplo de uma CPU multitherading com pipeline de 5 estágio :
nunca tem 2 instruções da mesma thread ativa concorrentemente no pipeline. Se as instruções podem ser executadas fora de ordem, então é possível manter a CPU ocupada mesmo no caso de falta da cache.

5 threads in execution:

A	A	A	A	A	A	...
B	B	B	B	B	B	...
C	C	C	C	C	C	...
D	D	D	D	D	D	...
E	E	E	E	E	E	...
1	2	3	4	5	6	



Multithreading de granulosidade fina

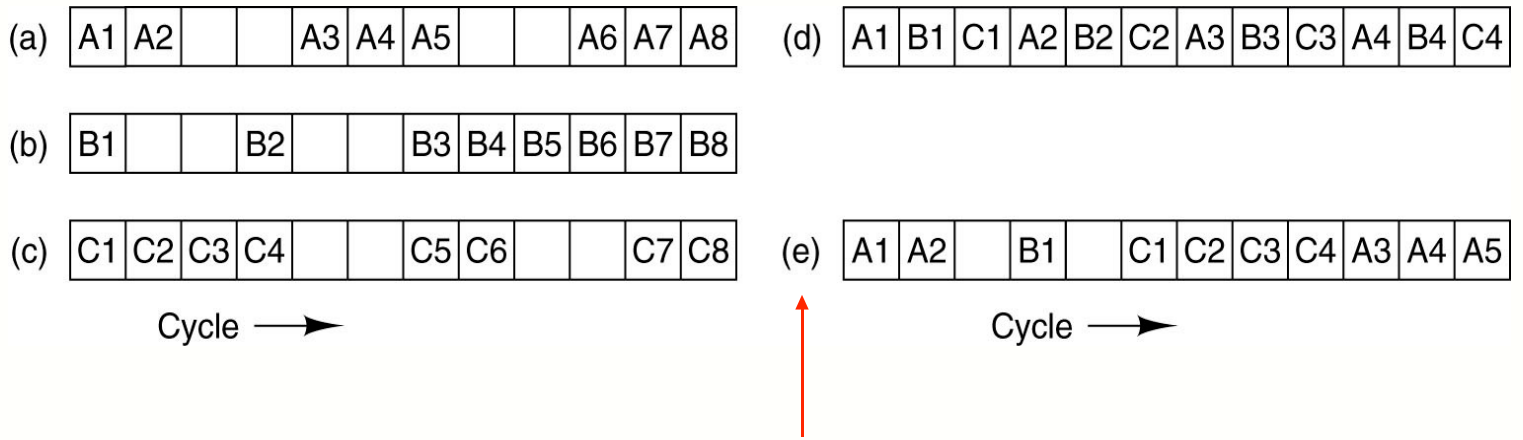
- Além de requerer uma troca de contexto eficiente o escalonamento de cada instrução desacelera uma thread mesmo que a thread possa continuar sua execução sem provocar uma parada
- Além disso, pode haver menos threads que estágios do pipeline, fazendo com que seja difícil manter a CPU ocupada
- Devido a esses problemas, uma nova abordagem foi desenvolvida denominada multithreading de granulosidade grossa

Multithreading de granulosidade grossa versus fina

- Só ocorre a troca de execução para uma outra thread quando a thread em execução causa uma parada, portanto gastando ciclo de relógio
- Neste ponto, ocorre uma troca de contexto para executar uma outra thread. Quando esta thread para, uma terceira thread é escalonada (ou possivelmente a primeira é re-escalonada) e assim por diante
- Esta abordagem potencialmente gasta mais ciclos de relógio que a granulosidade fina, porque a troca de threads ocorre quando acontece uma parada
- Mas se tiver poucas threads, elas podem ser suficientes para manter a CPU ocupada

Multithreading de granulosidade grossa versus fina

- (a)-(c) três threads com paradas associadas (slots vazios).
  - (d) Multithreading de granulosidade fina.
  - (e) Multithreading de granulosidade grossa (Tanenbaum, Fig. 8.7)



Multithreading de granulosidade grossa versus fina

- No slide anterior parece que multithreading com granulosidade fina funciona melhor, mas não é sempre assim .
- Especificamente, uma troca de threads não pode ocorrer sem nenhum gasto de ciclo de relógio.
- Então se as instruções das threads não causam paradas frequentemente, um esquema de granulosidade grossa pode ser mais conveniente do que um de granulosidade fina, onde existe troca de contexto em cada ciclo de relógio

Multithreading de granulosidade média

- Uma abordagem intermediária entre granulosidade fina e grossa que consiste em fazer uma troca de contexto para outra thread somente quando a thread que está executando vai emitir uma intrução que pode causar uma parada longa, tal como, um load (que requer um dado que não está na cache) ou um desvio condicional.
- A instrução é emitida, mas o processador chaveia para outra thread. Com esta abordagem, é possível eonomizar até a perda de um ciclo de relógio devido à parada do load (inevitável em multithreading com granulosidade grossa).

Multithreading

- Como o pipeline sabe a qual thread a instrução pertence?
- Em granulosidade fina, cada instrução possui um identificador de thread
- Em granulosidade grossa, pode-se usar o identificador de thread acima, ou o pipeline pode ser esvaziado em cada troca de contexto de thread: deste modo o pipeline só contém instruções de uma thread
- A opção de esvaziamento do pipeline só é razoável, se a troca de contexto acontecer em intervalos que são muito maiores que o tempo necessário para esvaziar o pipeline.

Multithreading

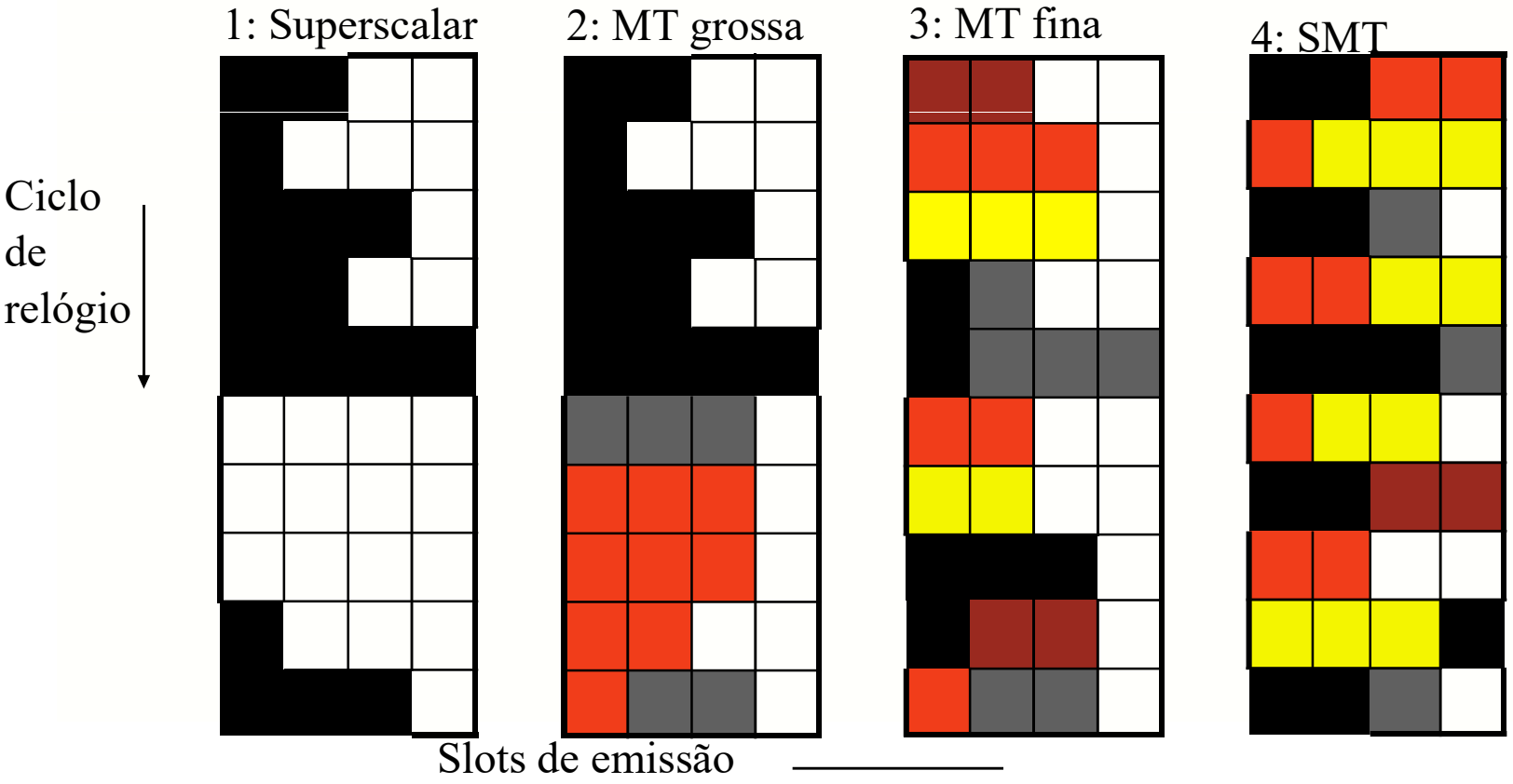
- As instruções das threads que estão em execução devem estar na cache de instrução o máximo possível, seão, cada troca de contexto irá causar uma falta e todas as vantagens de multithreading serão perdidas.

Multithreading simultâneo

- Arquiteturas superescalares modernas, com emissão múltipla e escalonamento dinâmico permitem explorar o paralelismo tanto no nível de instrução (ILP) como no vível de threads (TLP)
- **ILP + TLP = Multi-Threading Simultâneo (SMT)**
- SMT é conveniente porque CPUs modernas com múltiplas emissões possuem um número de unidades funcionais que não podem ser mantidas ocupadas com instruções de uma única thread.
- Aplicando renomeação de registradores e escalonamento dinâmico, instruções pertencentes a threads diferentes podem ser executadas de forma concorrente.

Multithreading simultâneo

- Múltiplas instruções são emitidas em cada ciclo de relógio, possivelmente pertencentes a threads diferentes, possibilitando um aumento da utilização dos recursos da CPU (cada slot colorido representa uma única instrução de uma thread).



Multithreading simultâneo

- Em CPUs superescalares sem multithreading, a emissão múltipla pode ser inútil se não existir paralelismo de instrução suficiente em cada thread, e uma parada muito longa pode congelar todo o processador.
- Em MT com granulosidade grossa, paradas longas são escondidas realizando-se um chaveamento de thread, mas se cada thread tiver pouco paralelismo de instrução ocorrerá uma baixa utilização dos recursos da CPU pois nem todos slots de emissão de instrução poderão ser utilizados.
- Mesmo em MT com granulosidade fina, se cada thread tiver pouco paralelismo de instrução ocorrerá uma baixa utilização dos recursos da CPU
- Em SMT, as instruções pertencentes a threads diferentes são provavelmente independentes, e emitindo-as de forma concorrente possibilitará uma maior utilização dos recursos da CPU.

Multithreading simultâneo

- Mesmo com MT, não há garantia de da emissão do número máximo de instryçõe spossível, devido ao núemro limitado de unidades funcionais, estações reserva e capacidade da cache de instruções.
- SMT só é viável se existir uma grande quantidade de registradores que podme ser renomeados
- Além disso, em uma CPU que suporta especulação, é necessário existir um buffer de reordenamento (ROB) diferente para cada thread, de modo que a confirmação da execução das instruções seja realizada de forma independente para cada thread