

Aula 15: Hierarquia de Memória— Reduzindo *Miss Penalty* (Cache Secundárias)

1. Reduzindo Miss Penalty: Aumentando a Prioridade de Reads sobre Writes

- Write back com write buffers oferecem conflitos RAW com leituras da memória principal em um miss de cache
- Parando leitura até buffer esvaziar pode aumentar MP em 50%
- Verifique write buffer antes de iniciar leitura; se não houverem conflitos, deixe leitura continuar
- Write Back?
 - Read miss troca bloco sujo
 - Normal: Escreva bloco sujo na memória, e leia novo bloco
 - Copie bloco sujo para write buffer, leia o novo bloco e então escreva bloco na memória
 - Stall é menor porque não precisa aguardar término da escrita

2. Subblock Placement para Reduzir *Miss Penalty*

- Não precisa carregar bloco inteiro durante miss
- Mantenha bits/sub-bloco para indicar quais estão válidos

100	1		1		1		1	
300	1		1		0		0	
200	0		1		0		1	
204	0		0		0		0	

Valid Bits



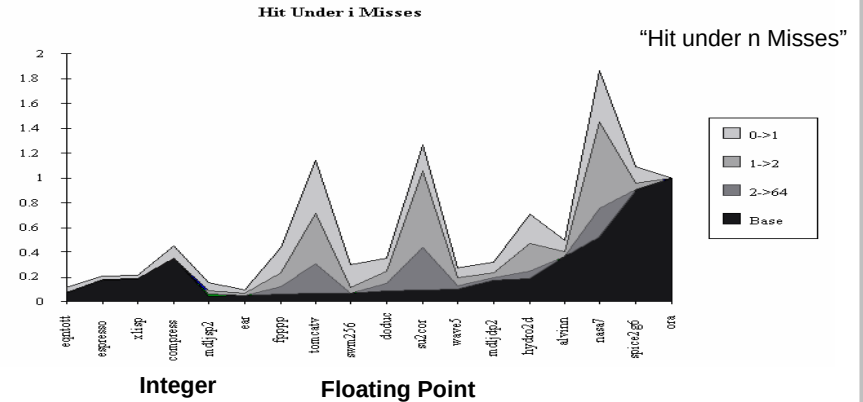
3. *Early Restart* e *Critical Word First*

- Não aguarde até bloco estar completamente carregado para liberar CPU
 - *Early restart*—Tão logo quanto palavra buscada estiver disponível, envie-a para CPU e deixe CPU continuar execução
 - *Critical Word First*—Busque palavra desejada primeiramente da memória e envie-a para CPU tão logo ela chegue; deixe CPU continuar execução enquanto restante do bloco é buscado. Também chamado de *wrapped fetch* e *requested word first*
- Bom para blocos grandes
- Localidade espacial é um problema; há uma tendência de se precisar buscar próxima palavra sequencial

4. Caches não-Blocantes para Reduzir Stalls de Misses

- **Non-blocking cache** ou **lockup-free cache** permite cache de dados continuar a suprir dados que sejam hits enquanto ela processa um miss
- **“hit under miss”** reduzem a penalidade de misses ao fazer alguma coisa útil durante tratamento de miss
- **“hit under multiple miss”** ou **“miss under miss”** pode melhorar ainda mais a penalidade efetiva de misses
 - Mas podem aumentar de maneira significativa complexidade do controlador da cache pois a cache deverá manter lista de acessos não terminados

Benefício de Hit Under Miss



- Programas FP na média: AMST= 0.68 -> 0.52 -> 0.34 -> 0.26
- Programas Ints. na média: AMST= 0.24 -> 0.20 -> 0.19 -> 0.19
- Cache de dados 8 Kb, DM, bloco de 32 bytes, 16 ciclos/miss

Caches Secundárias

- Equações para L2

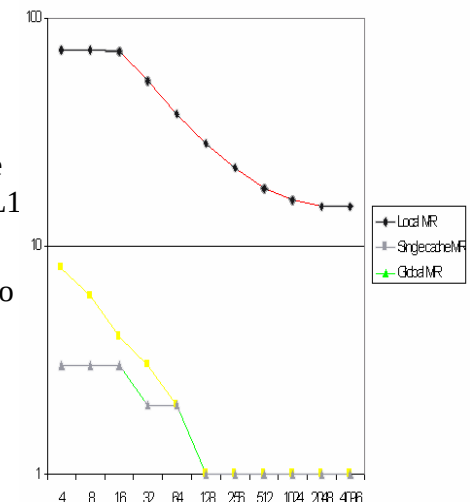
$$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$$

$$Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$$

$$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} + Miss\ Penalty_{L2})$$
- Definições:
 - **Local miss rate**— misses da cache divididas pelo número total de acessos para *esta* cache ($Miss\ rate_{L2}$)
 - **Global miss rate**— misses da cache divididos pelo número total de acessos de memória *gerados pela CPU* ($Miss\ Rate_{L1} \times Miss\ Rate_{L2}$)

Comparando Miss Rates Globais e Locais

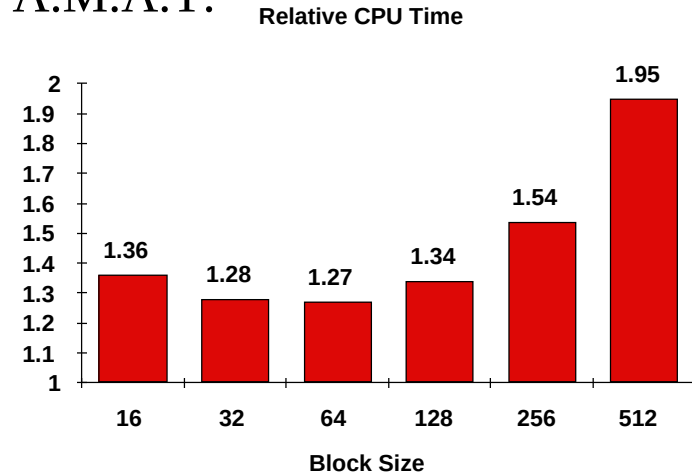
- L1: 32 KByte
Aumentar performance colocando L2 no sistema
- MR Global próximo a MR de cache única p/ L2, se $L2 \gg L1$
- Não use MR Local
- L2 não está vinculada a período de clock da CPU
- Custo & AMAT
- MR local de L2 é alto



Reduzindo Misses: O que se aplica a L2?

- Reduzindo MR
 1. Reduzindo misses com blocos maiores
 2. Reduzindo misses com maior associatividade
 3. Reduzindo misses com *victim cache*
 4. Reduzindo misses com pseudo-associatividade
 5. Reduzindo misses com HW prefetching de instruções e dados
 6. Reduzindo misses com SW prefetching de dados
 7. Reduzindo misses com otimizações do compilador

Tamanho de bloco em L2 & A.M.A.T.



- 32KB L1, largura do barramento = 32 bits
1 ciclo/end, 6 para acessar dados, 1 ciclo/palavra

Resumo: Reduzindo Miss Penalty

- Cinco técnicas
 - Aumentando a prioridade de leituras sobre escritas em um miss
 - *Subblock placement*
 - *Early Restart* e *Critical Word First*
 - Caches não-blocantes (*Hit Under Miss*)
 - Caches secundárias
- Podem ser aplicadas recursivamente p/ caches multinível
 - Tempo entre CPU e DRAM pode subir com o aumento do número de níveis

Melhorando a Performance de Caches

1. Reduzindo MR,
2. Reduzindo MP, ou
3. Reduzindo tempo de hit.

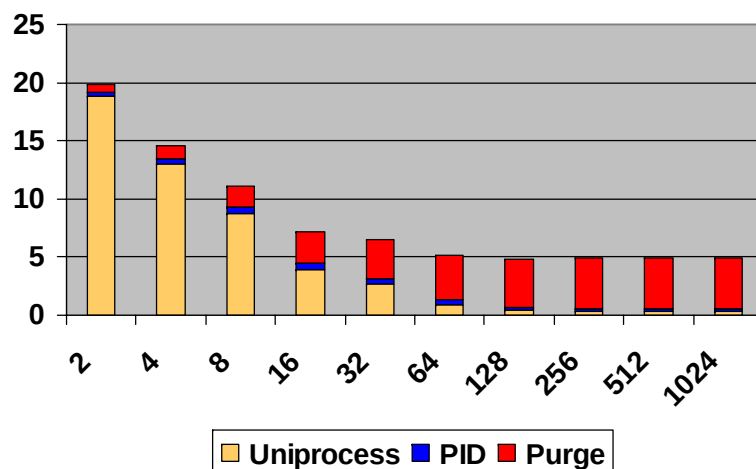
1. Caches Menores e Mais Simples

- Por que Alpha 21164 possui 8KB I-cache e 8KB D-cache + 96KB L2
- Mapeamento direto no chip

2. Evitando Tradução de Endereço

- Envio de endereço virtual p/ cache? Chamado *Virtually Addressed Cache* ou *Virtual Cache* vs. *Physical Cache*
 - Cada vez que processo é trocado, precisamos limpar (*flush*) a cache; caso contrário podemos ter hits falsos
 - Custo = tempo para “flush” + misses “compulsórios” causados pela limpeza
 - *aliases* (ou *sinônimos*);
Dois endereços virtuais apontando para mesmo endereço físico
 - I/O deve interagir com cache
- Solução para aliases
 - HW garante aliases são mapeados para mesma linha (DM) se eles possuem campos de índices iguais (*page coloring*)
- Solução para flush
 - Adicione em campo de tag identificador do processo: hit não pode ocorrer se processo for diferente

2. Impacto do *Process ID*

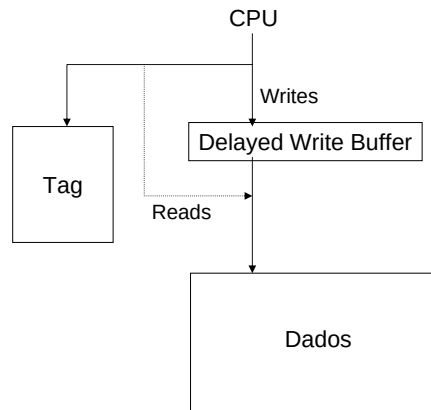


2. Evitando Tradução: Índice com Parte Física de Endereço

- Se índice é parte física de endereço, podemos iniciar acesso à cache em paralelo com tradução de endereço
- Limita cache à tamanho da página: O que podemos fazer se quisermos caches maiores com a mesma característica?
 - Maior associatividade
 - *Page coloring*

3. *Pipelined Writes*

- Pipeline verificação de tag com update de cache



Resumo: Otimização de Caches

	Miss Rate	Miss Pen.	Hit Time	Complexity
miss rate				
Larger Block Size	+	-		0
Higher Associativity	+		-	1
Victim Caches	+			2
Pseudo-Associative Caches	+			2
HW Prefetching of Instr/Data	+			2
Compiler Controlled Prefetching	+			3
Compiler Reduces Misses	+			0
miss penalty				
Priority to Read Misses		+		1
Subblock Placement		+	+	1
Early Restart & Critical Word 1 st		+		2
Non-Blocking Caches		+		3
Second Level Caches		+		2
Small and simple caches	-		+	0
hit time				
Avoid address translation			+	2
Pipelining writes			+	1