

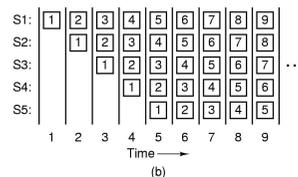
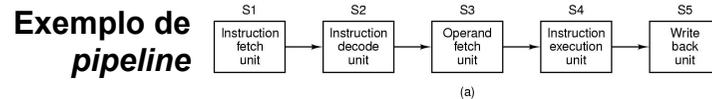
Estrutura do tema Avaliação de Desempenho (IA-32)

1. A avaliação de sistemas de computação
2. Técnicas de otimização de código (IM)
3. Técnicas de otimização de *hardware*
4. Técnicas de otimização de código (DM)
5. Outras técnicas de otimização
6. Medição de tempos ...

Otimização do desempenho (no h/w)

- com introdução de **paralelismo**
 - ao nível do processo (*multicore*/distribuídos/heterogéneos)
 - ao nível da instrução num *core* (**Instruction Level Parallelism**)
 - na execução do código:
 - » paralelismo desfasado (*pipeline*)
 - » paralelismo "real" (VLIW, superescalaridade)
 - só nos dados (processamento vectorial)
 - na transferência de informação de/para a memória
 - com paralelismo desfasado (*interleaving*)
 - com paralelismo "real" (>largura do *bus*, mais canais)
- com introdução de **hierarquia de memória**
 - *cache* dedicada/partilhada, acesso UMA/NUMA...

Paralelismo no processador Exemplo 1



Objectivo

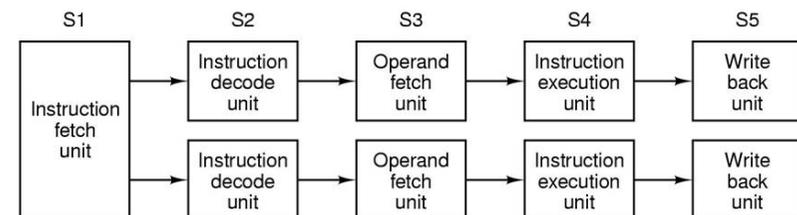
- CPI = 1

Problemas:

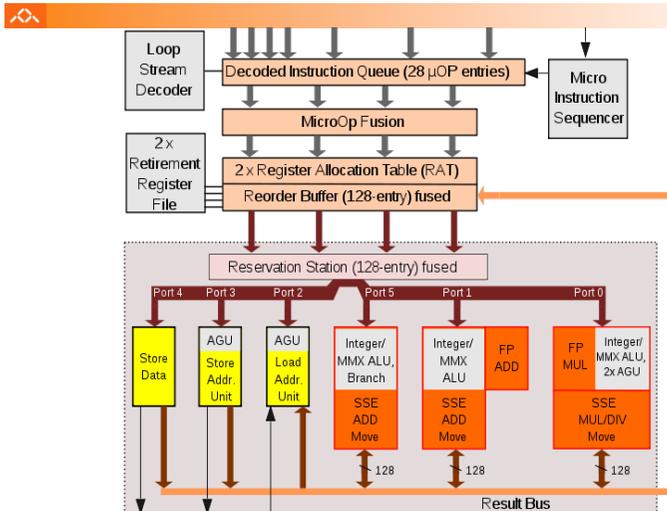
- dependências de dados
- latências nos acessos à memória
- saltos condicionais; propostas de solução para minimizar perdas:
 - executar sempre a instrução "que se segue"
 - usar o historial dos saltos anteriores (1 ou mais bits)
 - executar os 2 percursos alternativos até à tomada de decisão

Paralelismo no processador Exemplo 2

Exemplo de superescalaridade (nível 2)



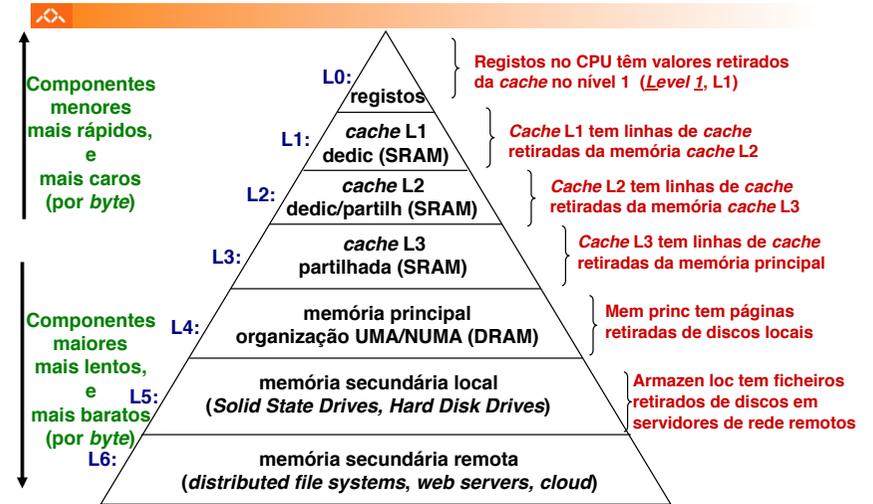
Paralelismo no processador Exemplo 3 (superescalaridade nível 4 no Intel Nehalem)



AJProença, Sistemas de Computação, UMinho, 2011/12

5

Organização hierárquica da memória



AJProença, Sistemas de Computação, UMinho, 2011/12

6

Sucesso da hierarquia de memória: o princípio da localidade

A cache numa hierarquia de memória: introdução

Princípio da Localidade:

- programas tendem a re-usar dados e instruções próximos daqueles que foram recentemente usados, ou que foram recentemente referenciados por eles
- **Localidade Espacial:** itens em localizações contíguas tendem a ser referenciados em tempos próximos
- **Localidade Temporal:** itens recentemente referenciados serão provavelmente referenciados no futuro próximo

Exemplo da Localidade :

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

•Dados

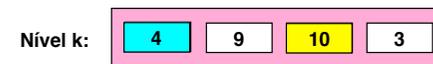
- os elementos do *array* são referenciados em instruções sucessivas: **Localidade Espacial**
- a variável *sum* é acedida em cada iteração: **Localidade Temporal**

•Instruções

- as instruções são acedidas sequencialmente: **Localidade Espacial**
- o ciclo é repetidamente acedido: **Localidade Temporal**

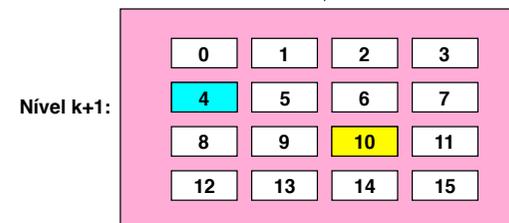
AJProença, Sistemas de Computação, UMinho, 2011/12

7



O componente no nível k, menor, +rápido e +caro, faz o *cache* de um sub-conj de blocos do nível k+1

Os dados são copiados entre níveis em blocos (unidades de transferência)

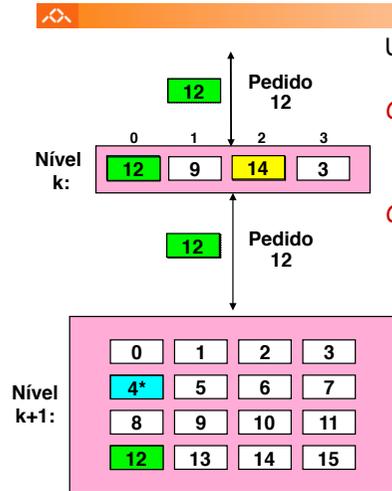


O componente no nível k+1, maior, +lento e +barato, está organizado em blocos

AJProença, Sistemas de Computação, UMinho, 2011/12

8

A cache numa hierarquia de memória: conceitos



Um programa pede pelo objecto *a*, que está armazenado num bloco *b*

Cache hit

- o programa encontra *b* na cache no nível *k*.
Por ex., bloco 14

Cache miss

- b* não está no nível *k*, logo a cache do nível *k* deve buscá-lo do nível *k+1*.
Por ex., bloco 12
- se a cache do nível *k* está cheia, então um dos blocos deve ser substituído (retirado); qual?

- Replacement policy:** que bloco deve ser retirado? Por ex., LRU
- Placement policy:** onde colocar o novo bloco? Por ex., $b \bmod 4$

A cache numa hierarquia de memória: métricas de desempenho

Miss Rate

- percentagem de referências à memória que não tiveram sucesso na cache (*misses* / acessos)
- valores típicos:
 - 3-10% para L1
 - pode ser menor para L2 (< 1%), dependendo do tamanho, etc.

Hit Time

- tempo para a cache entregar os dados ao processador (inclui o tempo para verificar se a linha está na cache)
- valores típicos:
 - 1-2 ciclos de clock para L1
 - 3-10 ciclos de clock para L2

Miss Penalty

- tempo extra necessário para ir buscar uma linha após *miss*
 - tipicamente 25-100 ciclos para aceder à memória principal

A cache numa hierarquia de memória: regras na codificação de programas

Referenciar repetidamente uma variável é positivo!

(localidade temporal)

Referenciar elementos consecutivos de um array é positivo!

(localidade espacial)

Exemplos:

– cache fria, palavras de 4-bytes, blocos (linhas) de cache com 4-palavras

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

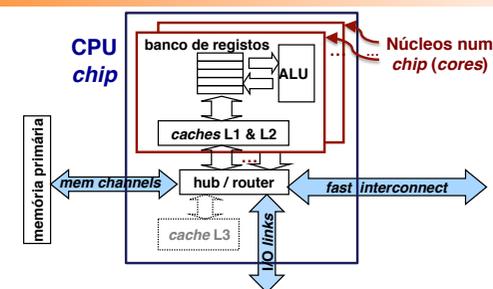
Miss rate = $1/4 = 25\%$

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Miss rate = 100%

A cache em arquiteturas multicore

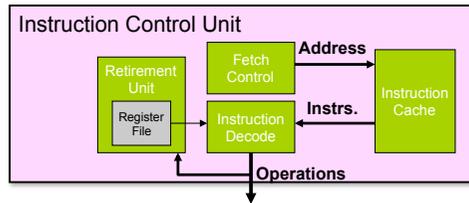


Notas:

- as caches L1 de dados e de instruções são normalmente distintas
- as caches L2 em multi-cores podem ser partilhadas por outras cores
- muitos cores partilhando uma única memória traz complexidades:
 - manutenção da coerência da informação nas caches
 - encaminhamento e partilha dos circuitos de acesso à memória

Papel da ICU:

- Lê instruções da *InstCache*
 - baseado no IP + previsão de saltos
 - antecipa dinamicamente (por *h/w*) se salta/não_salta e (possível) endereço de salto
- Traduz Instruções em *Operações*
 - *Operações*: designação da Intel para instruções tipo-RISC
 - instrução típica requer 1–3 operações
- Converte referências a Registos em *Tags*
 - *Tags*: identificador abstracto que liga o resultado de uma operação com operandos-fonte de operações futuras



Versão de combine4

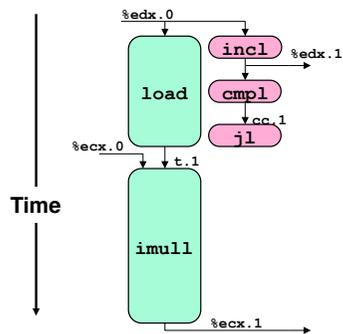
– tipo de dados: *inteiro* ; operação: *multiplicação*

```
.L24:                # Loop:
imull (%eax,%edx,4),%ecx # t *= data[i]
incl %edx              # i++
cpl %esi,%edx         # i:length
jl .L24                # if < goto Loop
```

Tradução da 1ª iteração

<pre>.L24: imull (%eax,%edx,4),%ecx incl %edx cpl %esi,%edx jl .L24</pre>	<pre>load (%eax,%edx.0,4) → t.1 imull t.1,%ecx.0 → %ecx.1 incl %edx.0 → %edx.1 cpl %esi,%edx.1 → cc.1 jl -taken cc.1</pre>
--	--

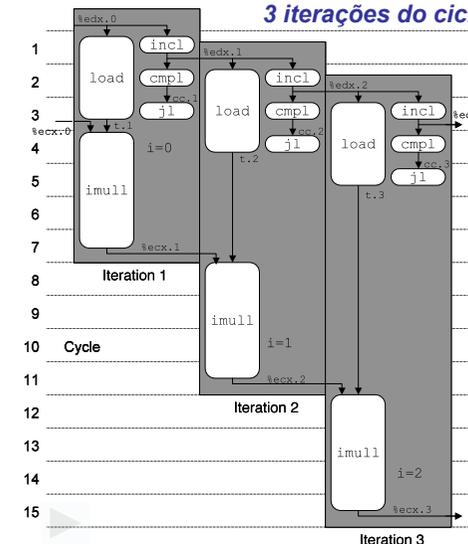
Análise visual da execução de instruções no P6: 1 iteração do ciclo de produtos em *combine*



```
load (%eax,%edx.0,4) → t.1
imull t.1,%ecx.0 → %ecx.1
incl %edx.0 → %edx.1
cpl %esi,%edx.1 → cc.1
jl -taken cc.1
```

- Operações**
 - a posição vertical dá uma indicação do tempo em que é executada
 - uma operação não pode iniciar-se sem os seus operandos
 - a altura traduz a latência
- Operandos**
 - os arcos apenas são representados para os operandos que são usados no contexto da *execution unit*

Análise visual da execução de instruções no P6: 3 iterações do ciclo de produtos em *combine*



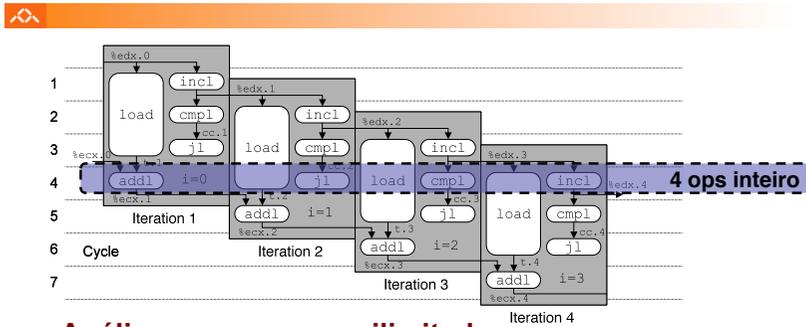
Análise com recursos ilimitados

- execução paralela e encadeada de operações na EU
- execução *out-of-order* e especulativa

Desempenho

- factor limitativo: latência da multipl. de inteiros
- CPE: 4.0

**Análise visual da execução de instruções no P6:
4 iterações do ciclo de somas em *combine***

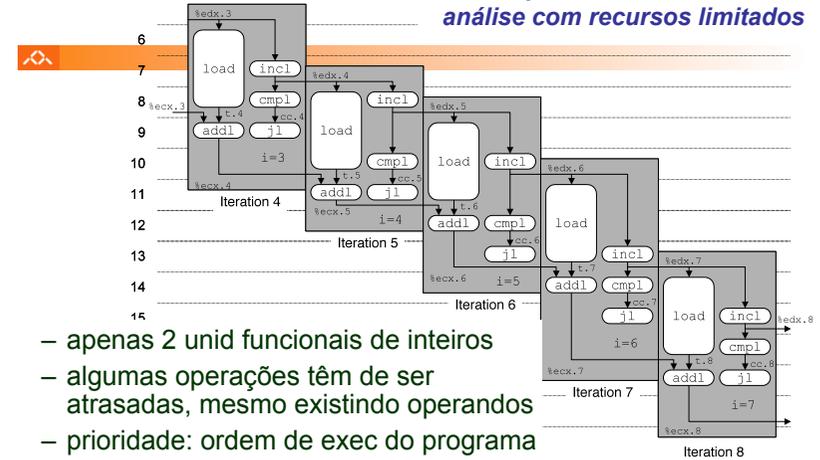


- **Análise com recursos ilimitados**

- **Desempenho**

- pode começar uma nova iteração em cada ciclo de *clock*
- valor teórico de CPE: 1.0
- requer a execução de 4 operações *c/* inteiros em paralelo

**As iterações do ciclo de somas:
análise com recursos limitados**



- apenas 2 unid funcionais de inteiros
- algumas operações têm de ser atrasadas, mesmo existindo operandos
- prioridade: ordem de exec do programa

- **Desempenho**

- CPE expectável: 2.0