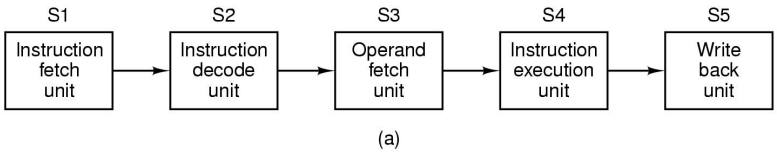
Estrutura do tema Avaliação de Desempenho (IA-32)

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 …

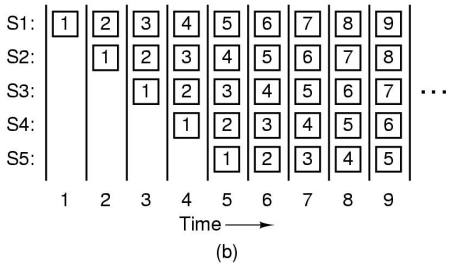
Otimização do desempenho (no *h/w*)

- no processador: com **paralelismo**
  - ao nível do processo (*multicore*/distribuídos/heterogéneos)
  - ao nível da instrução num *core* (***I****nstruction* ***L****evel* ***P****arallelism*)
    - na execução do código:
      - » paralelismo desfasado (*pipeline*)
      - » paralelismo "real" (VLIW, superescalaridade)
    - paralelismo só nos dados (processamento vetorial)
- no acesso à memória e com **hierarquia de memória**
  - na transferência de informação de/para a memória
    - com paralelismo desfasado (*interleaving*)
    - com paralelismo "real" (>largura do *bus*, mais canais)
  - *cache* dedicada/partilhada, acesso UMA/NUMA...

Exemplo de *pipeline*



(a)



(b)

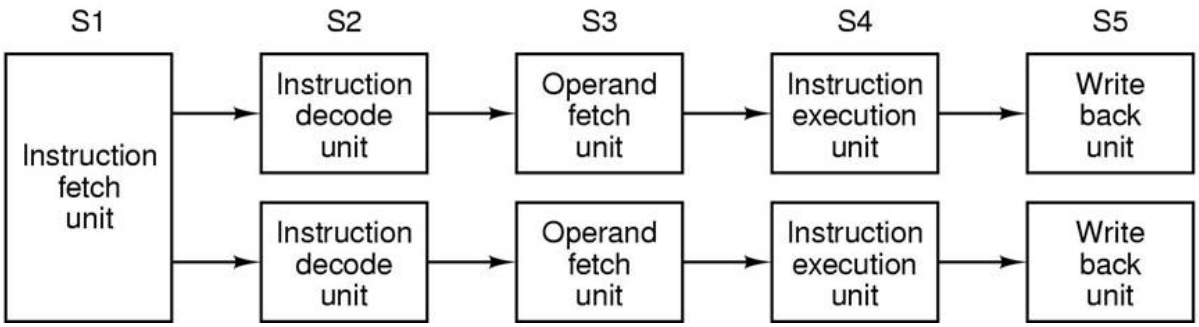
Objetivo

- **CPI = 1**

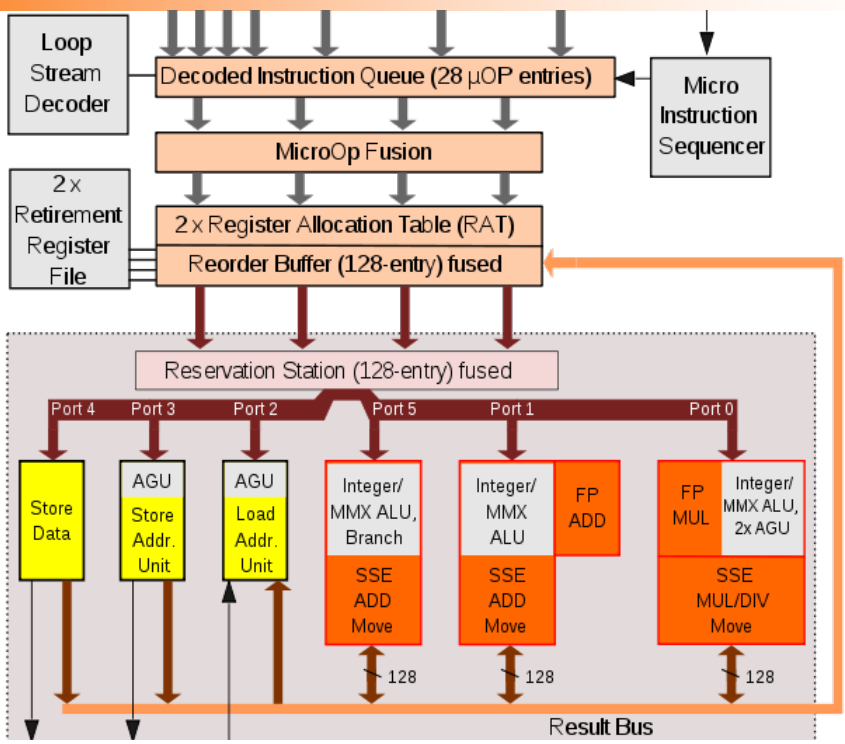
Problemas:

- 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

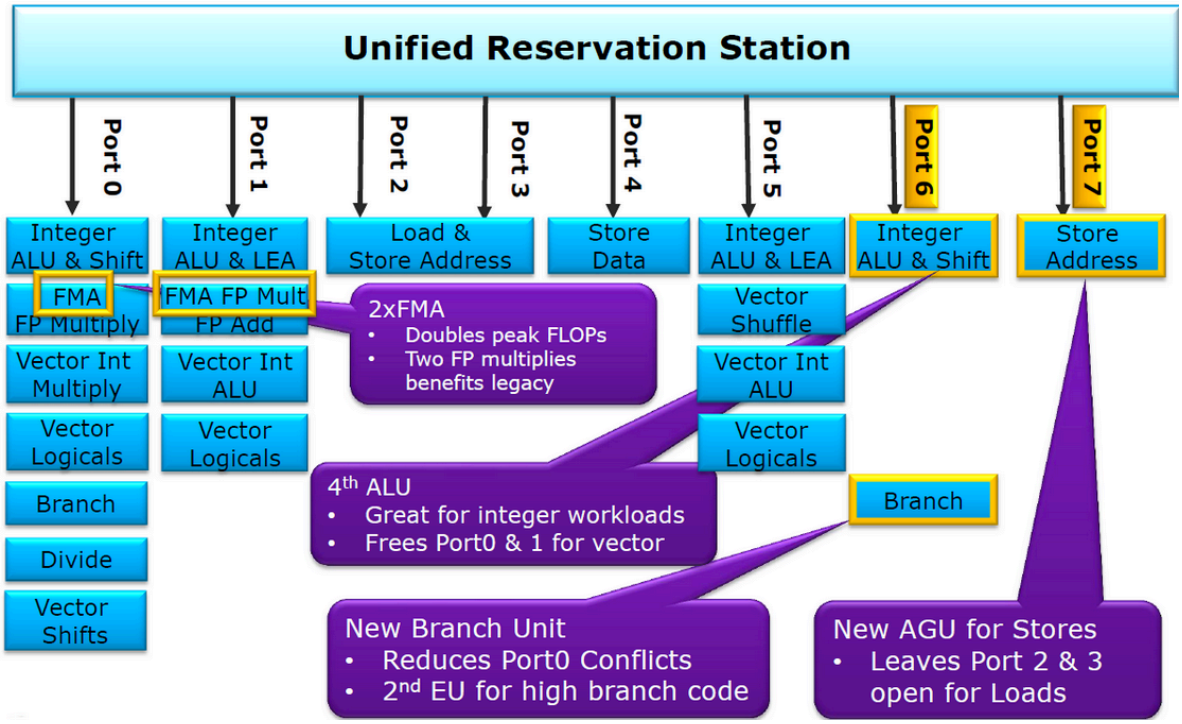
Exemplo de superescalaridade (2 vias)



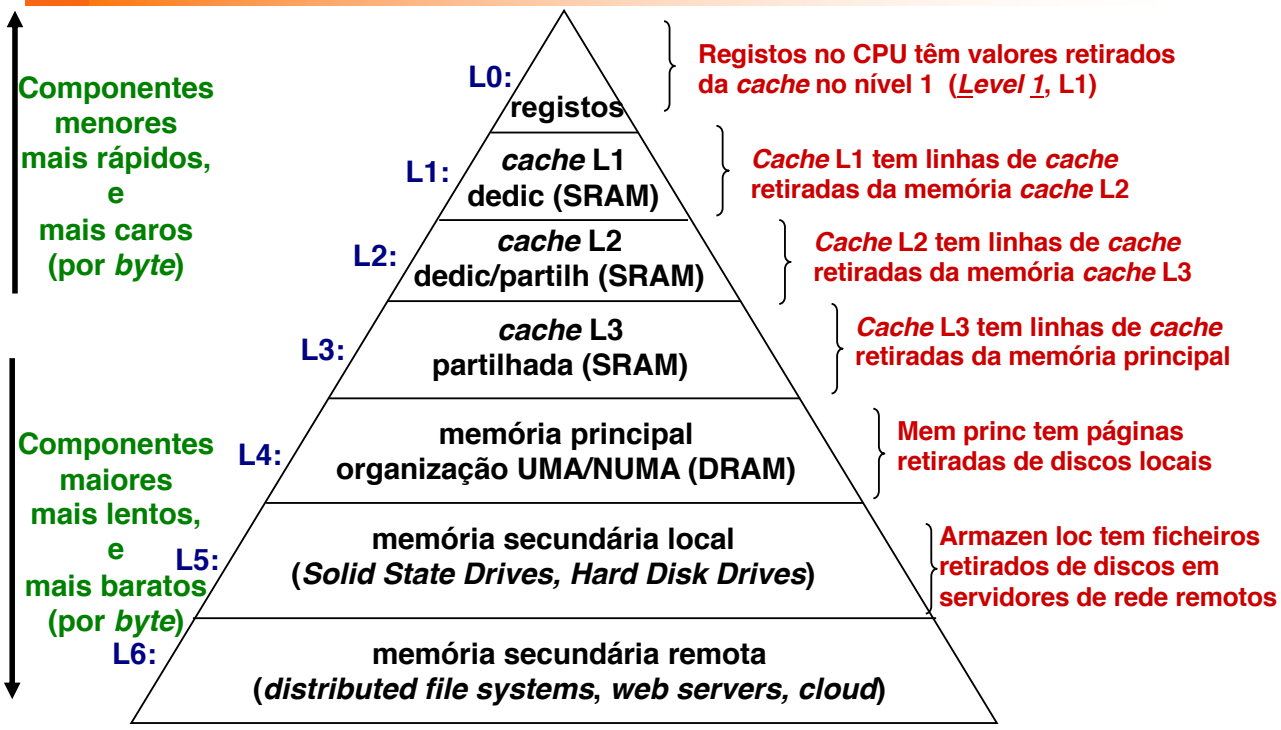
Paralelismo no processador
Exemplo 3 (superescalaridade 4-vias no Intel Nehalem)



Paralelismo no processador
Exemplo 4 (superescalaridade 8-vias no Intel Haswell)



Organização hierárquica da memória



Sucesso da hierarquia de memória: o princípio da localidade

Princípio da Localidade:

– programas tendem a re-usar dados e instruções próximos daqueles que foram recentemente usados ou referenciados por eles

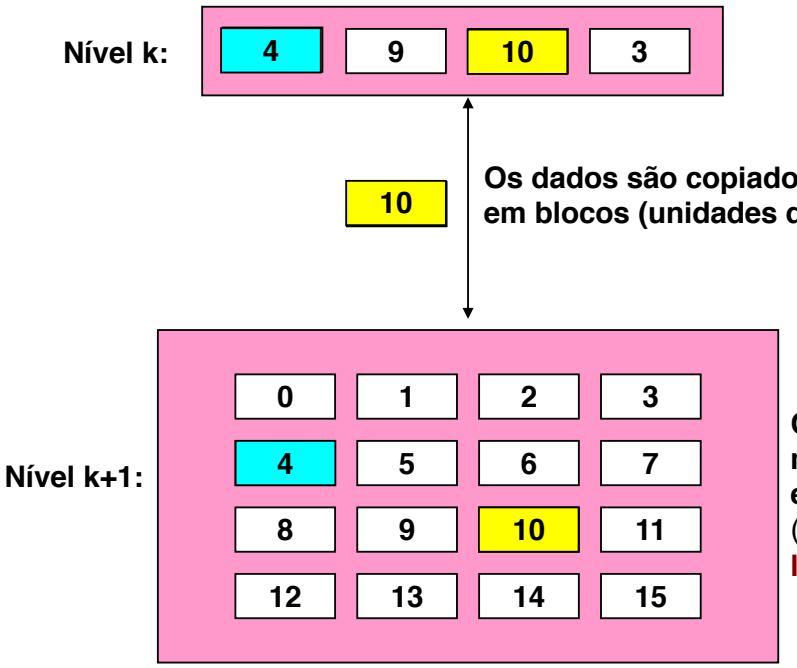
- **Localidade Espacial**: itens em localizações contíguas tendem a ser referenciados em tempos próximos
- **Localidade Temporal**: itens recentemente referenciados serão provavelmente referenciados no futuro próximo

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**

A cache numa hierarquia de memória: introdução

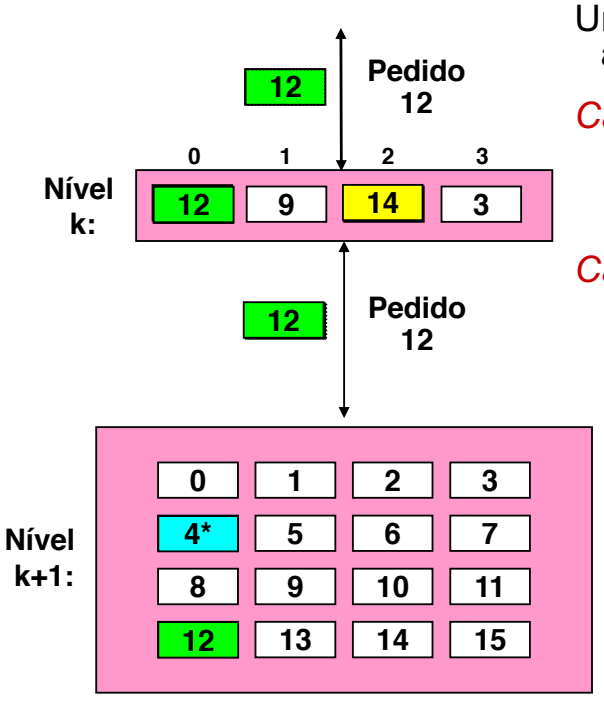


O componente no nível k, menor, +rápido e +caro, faz o *cache* de um sub-conj de blocos do nível k+1

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 (**páginas**, na memória virtual, **linhas** na cache)

A cache numa hierarquia de memória: conceitos



Um programa pede pelo objeto **`d`**, 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 mod 4`**

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 50-100 ciclos para aceder à memória principal

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 sum_array_rows(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;
}
```

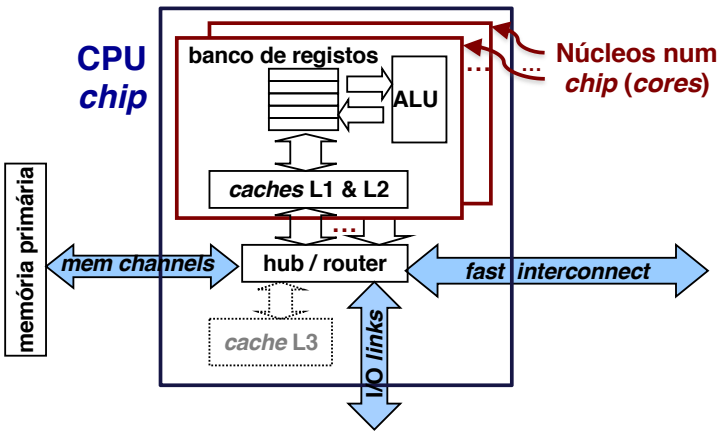
***Miss rate* = 1/4 = 25%**

```
int sum_array_cols(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* = até 100%**

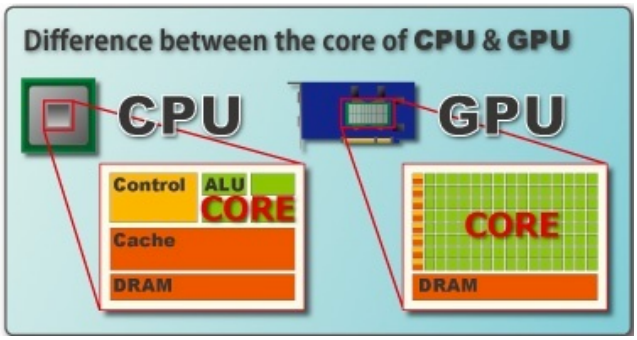
A cache em arquiteturas multicore



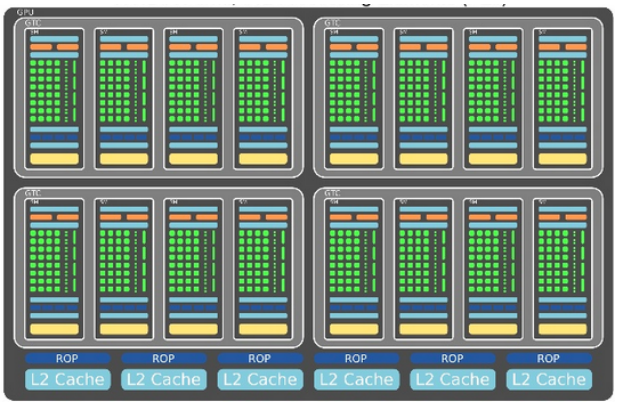
Notas:

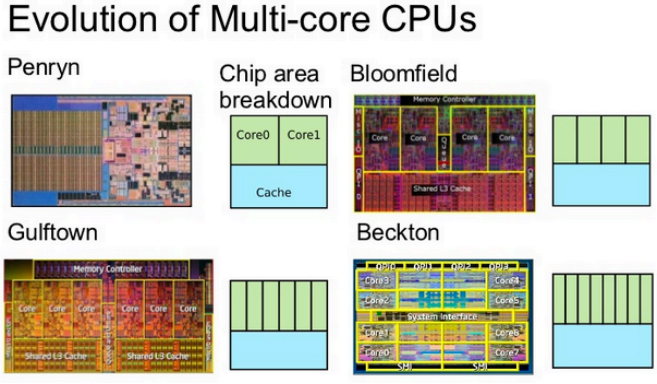
- as *caches* L1 de dados e de instruções são normalmente distintas
- as *caches* L2 em *multi-cores* podem ser partilhadas por outras *cores*
- muitos *cores* partilhando uma única memória traz complexidades:
  - manutenção da coerência da informação nas *caches*
  - encaminhamento e partilha dos circuitos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

Evolução das arquiteturas multicore

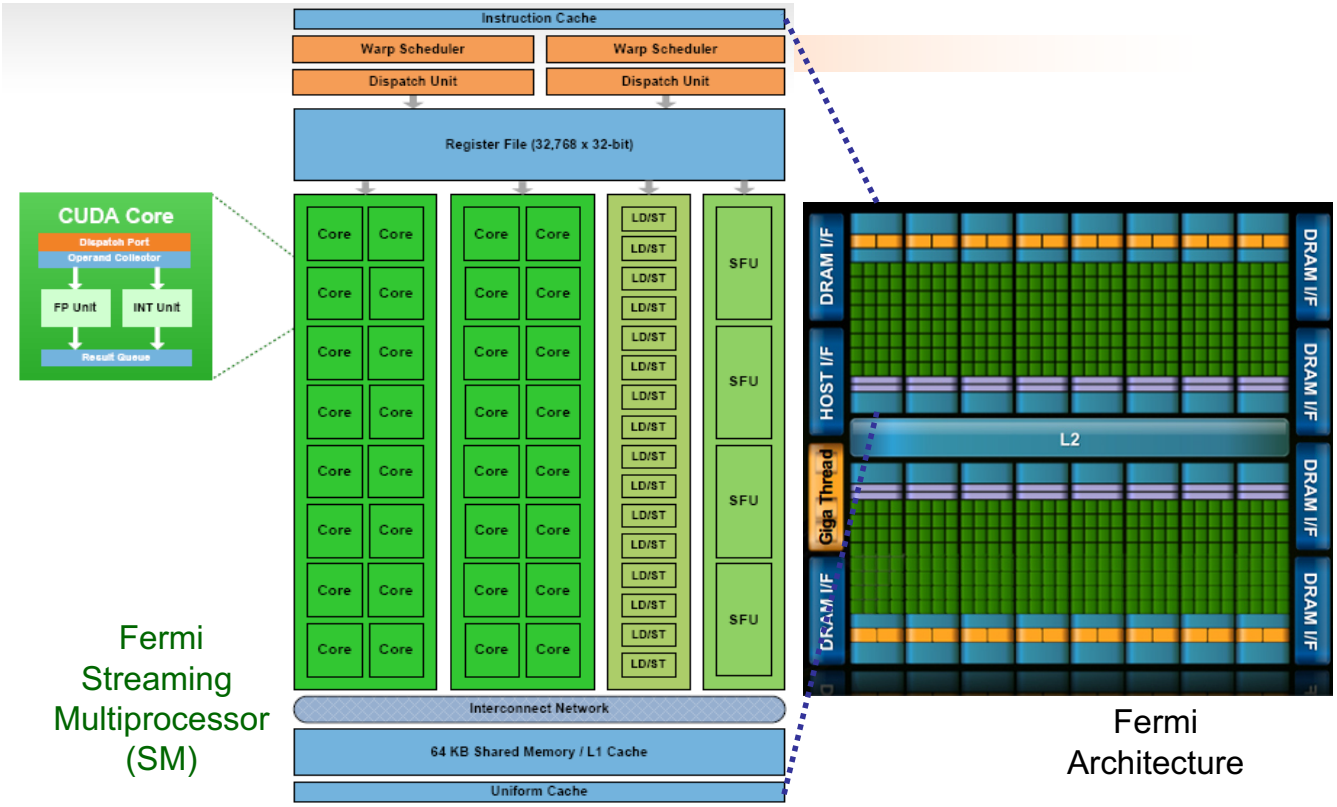


The "multicore" GPU





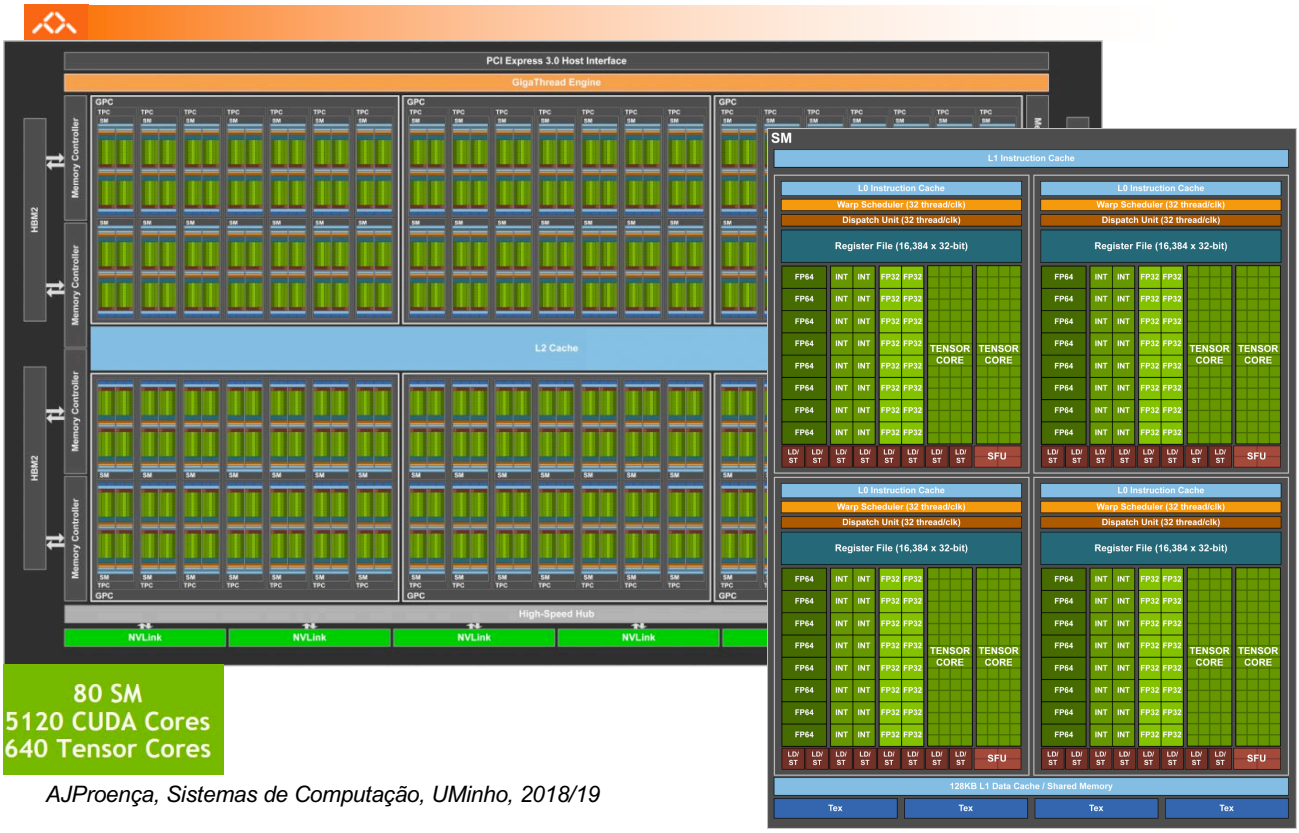
A arquitetura dos GPUs Fermi da NVidia



Fermi Streaming Multiprocessor (SM)

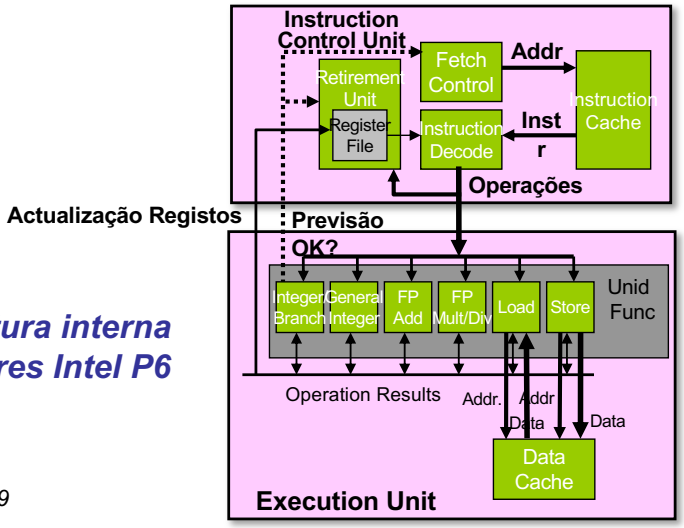
Fermi Architecture

Fermi→Kepler→Maxwell→Pascal→Volta: arquitetura do Tesla GV100



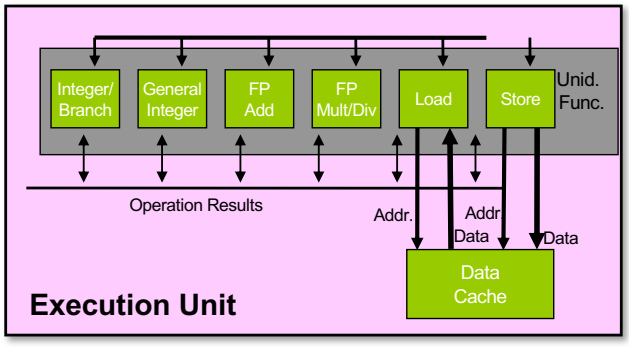
Evolução das microarquiteturas de CPUs da Intel

A arquitetura interna dos processadores Intel P6



Algumas potencialidades do Intel P6

- **Execução paralela de várias instruções**
  - 2 **integer** (1 pode ser **branch**)
  - 1 **FP Add**
  - 1 **FP Multiply ou Divide**
  - 1 **load**
  - 1 **store**

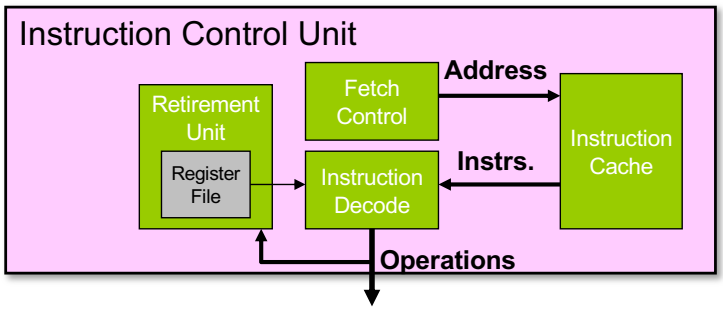


- Algumas instruções requerem > 1 ciclo, mas podem ser encadeadas

Instrução	Latência	Ciclos/Emissão
Load / Store	3	1
Integer Multiply	4	1
Integer Divide	36	36
Double/Single FP Add	3	1
Double/Single FP Multiply	5	2
Double/Single FP Divide	38	38

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++
  cmpl   %esi,%edx                 # i:length
  jl     .L24                      # if < goto Loop
```

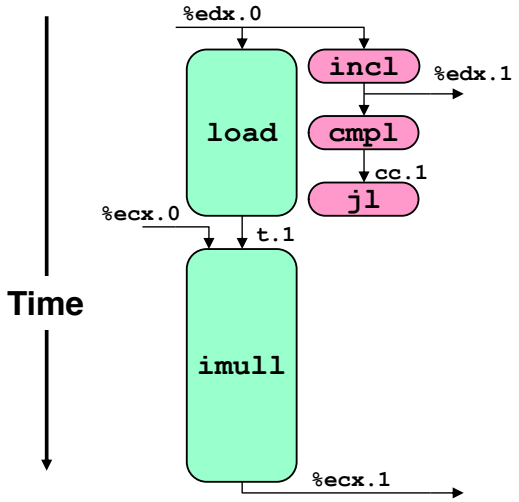
- **Tradução da 1ª iteração**

```
.L24:
  imull  (%eax,%edx,4),%ecx

  incl   %edx
  cmpl   %esi,%edx
  jl     .L24
```

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

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

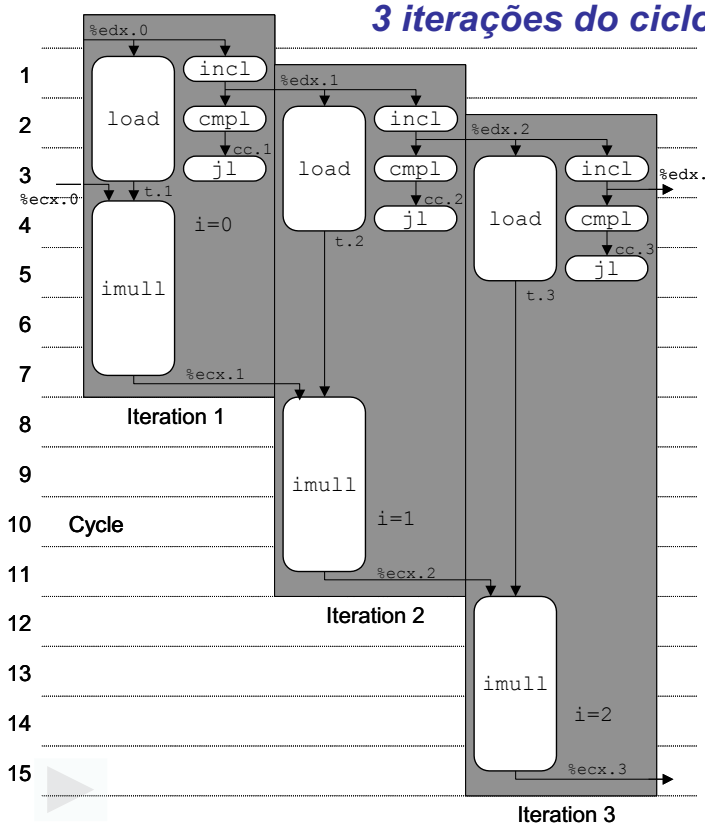


Instrução	Latência	Ciclos/Emissão
Load / Store	3	1
Integer Multiply	4	1
Integer Divide	36	36
Double/Single FP Add	3	1
Double/Single FP Multiply	5	2
Double/Single FP Divide	38	38

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

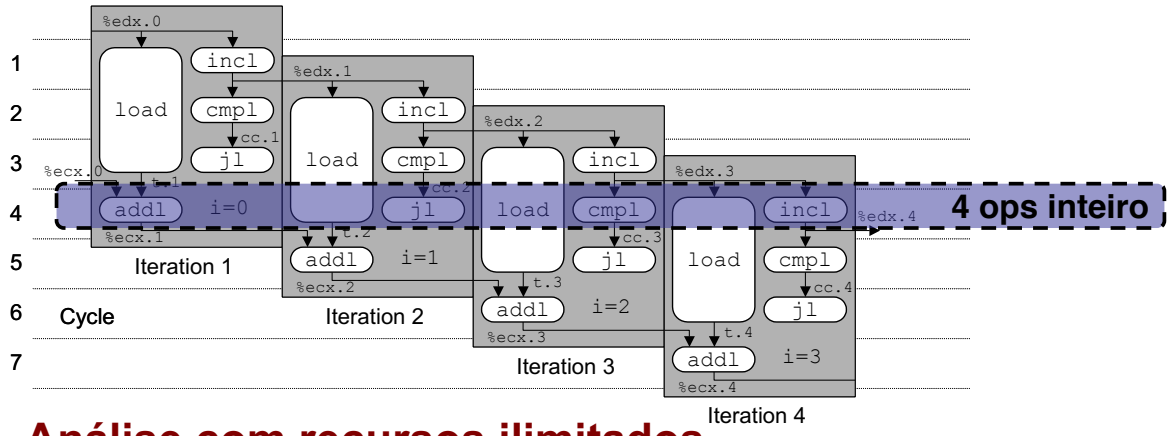
- **Operações**
  - a posição vertical dá uma indicação do tempo em que é executada
    - uma operação não pode iniciar-se sem os seus operandos
  - a altura traduz a latência
- **Operandos**
  - os arcos apenas são representados para os operandos que são usados no contexto da *execution unit*

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



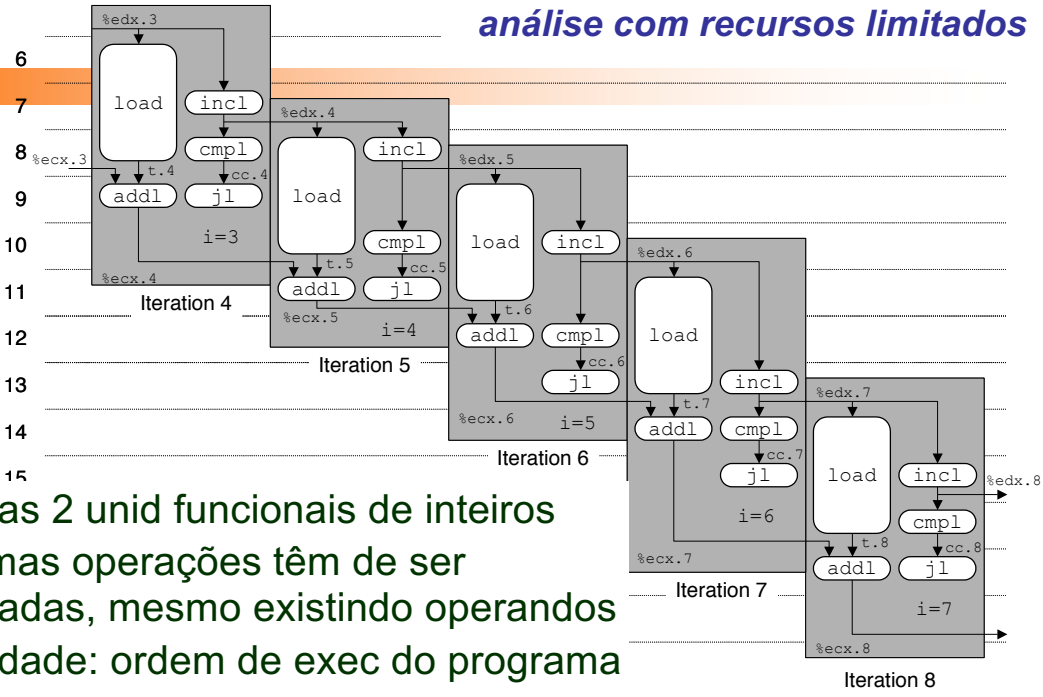
- **Análise com recursos ilimitados**
  - execução paralela e encadeada de operações na EU
  - execução *out-of-order* e especulativa
- **Desempenho**
  - factor limitativo: latência da multipl. de inteiros
  - CPE: 4.0

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