Arquiteturas de Computadores

Graphics Processing Unit

Fontes dos slides: Livro Patterson e Hennessy, Quantitative Approach e site do curso 15-740 e 18-740 Computer Architecture do Prof. Onur Mutlu

Desvio condicional

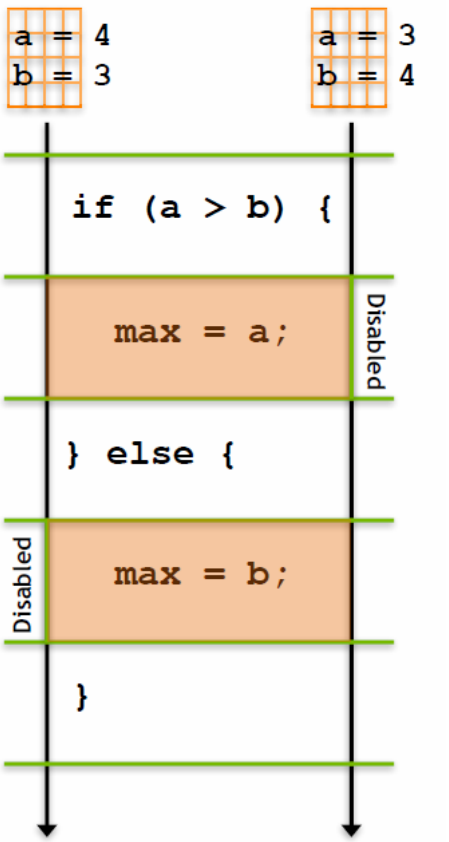
- Um warp contém 32 threads.
- Cada thread pode seguir seu próprio caminho.
- Hardware decodifica somente uma instrução para todo o warp
- Se threads em um warp devem ir para caminhos diferentes cada uma é executada separadamente até um ponto de reconvergência.
- Divergências devem ser raras ou rápidas no código
- Implementação é feita usando uma pilha de reconvergência
- Cada thread, após cada caminho pelo qual se passou ou não, vai para o ponto de reconvergência antes de seguir qualquer outro caminho

Desvio condicional

- Ponto de reconvergência:
  - Uma instrução que será executada caso o desvio tenha sido executado ou não
- Máscara de thread:
  - Um registrador que controla se a thread pode executar e possui 32 bits (um bit por thread do warp)
- Thread mascarada:
  - Uma thread cuja execução não é permitida pela máscara de thread

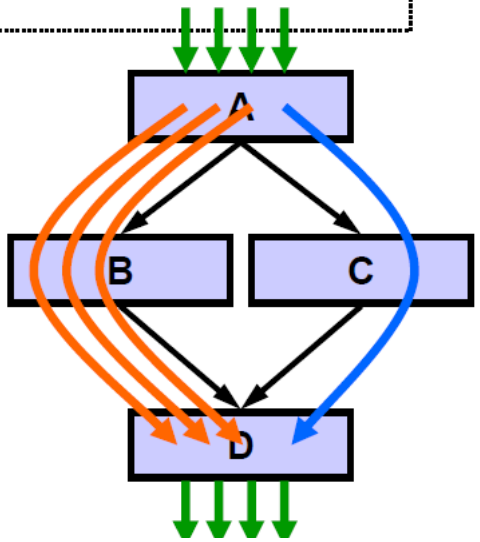
Desvio condicional

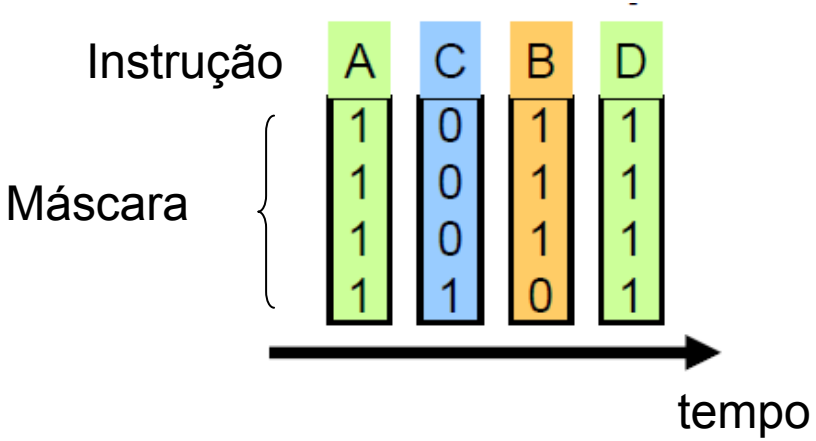
- Múltiplas threads executam a instrução lock-step
- As threads podem ser desabilitadas quando elas precisam executar instruções diferentes das outras do grupo
  - Mask and commit
- Divergência de desvio
  - Prejudica desempenho e eficiência



Tratamemto para desvio condicional

```
A;
if (some condition) {
    B;
} else {
    C;
}
D;
```





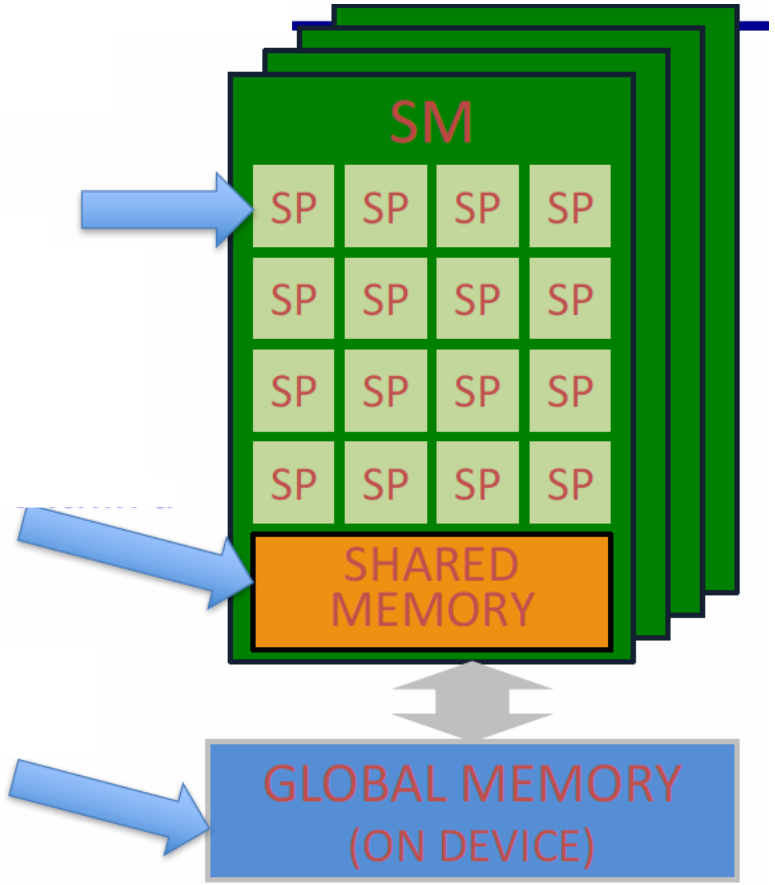
Exemplo em PTX

```
if (X[i] != 0)
   X[i] = X[i] - Y[i];
else X[i] = Z[i];
```

```
        ld.global.f64RD0, [X+R8] ; RD0 = X[i]
        setp.neq.s32 P1, RD0, #0 ; P1 is predicate register 1
        @!P1, bra ELSE1, *Push   ; Push old mask, set new mask bits
                                 ; if P1 false, go to ELSE1
        ld.global.f64RD2, [Y+R8] ; RD2 = Y[i]
        sub.f64 RD0, RD0, RD2    ; Difference in RD0
        st.global.f64[X+R8], RD0 ; X[i] = RD0
        @P1, bra ENDIF1, *Comp   ; complement mask bits
                                 ; if P1 true, go to ENDIF1
ELSE1:  ld.global.f64 RD0, [Z+R8] ; RD0 = Z[i]
        st.global.f64 [X+R8], RD0 ; X[i] = RD0
ENDIF1: <next instruction>, *Pop ; pop to restore old mask
```

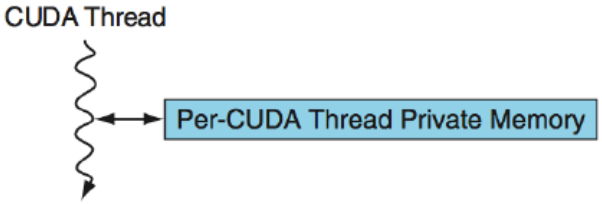
Estruturas de memória da GPU NVIDIA

- Cada faixa SIMD tem sua seção privada de off-chip DRAM
  - “Memória privada”
  - Contém pilha, registradores de spilling, e variáveis privadas
- Cada processador SIMD multithreaded tem sua memória local
  - Compartilhada pelas faixas/threads SIMD em um bloco
- Memória compartilhada pelos processadores SIMD é memória da GPU
  - Host pode ler e escrever na memória da GPU

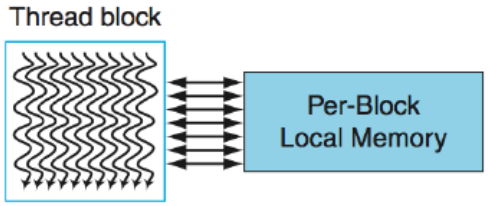


Memória da GPU para programação CUDA

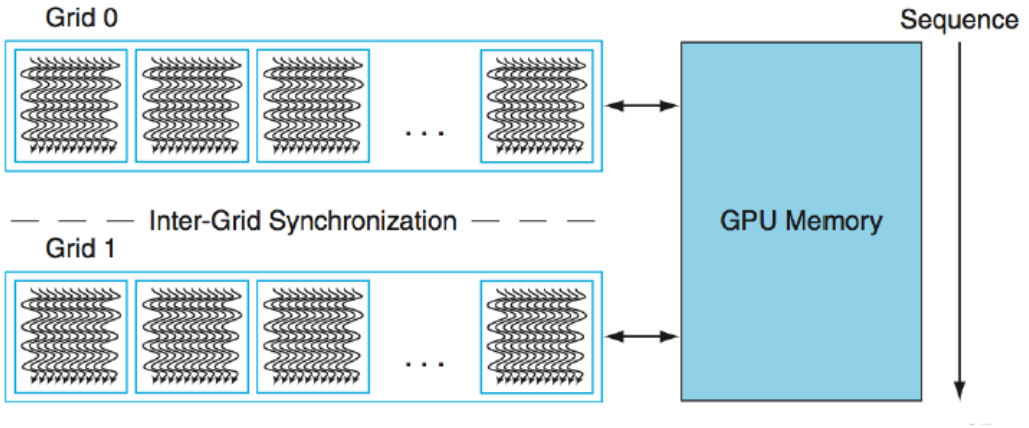
Variáveis locais



Explicitamente gerenciada utilizando a palavra **shared**

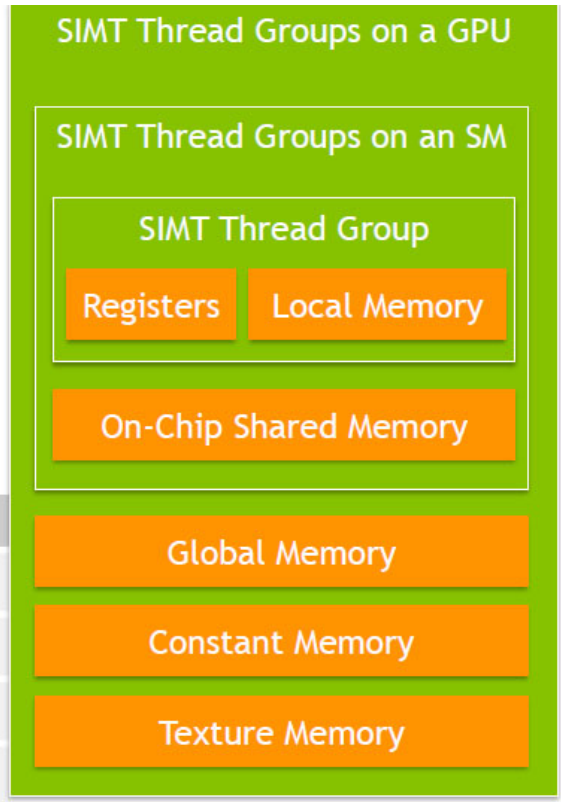


cudaMalloc



Memória GPU

- Mais complicada
- Diferentes escopos de uso
- Tamanhos e desempenho diferentes



MEMORY	ON/OFF CHIP	CACHED	ACCESS	SCOPE	LIFETIME
Register	On	n/a	R/W	1 thread	Thread
Local	Off	†	R/W	1 thread	Thread
Shared	On	n/a	R/W	All threads in block	Block
Global	Off	†	R/W	All threads + host	Host allocation
Constant	Off	Yes	R	All threads + host	Host allocation
Texture	Off	Yes	R	All threads + host	Host allocation