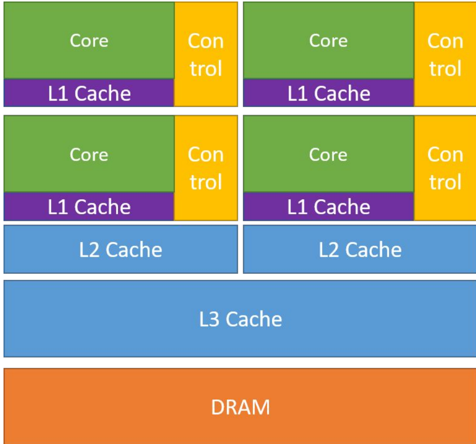
CUDA

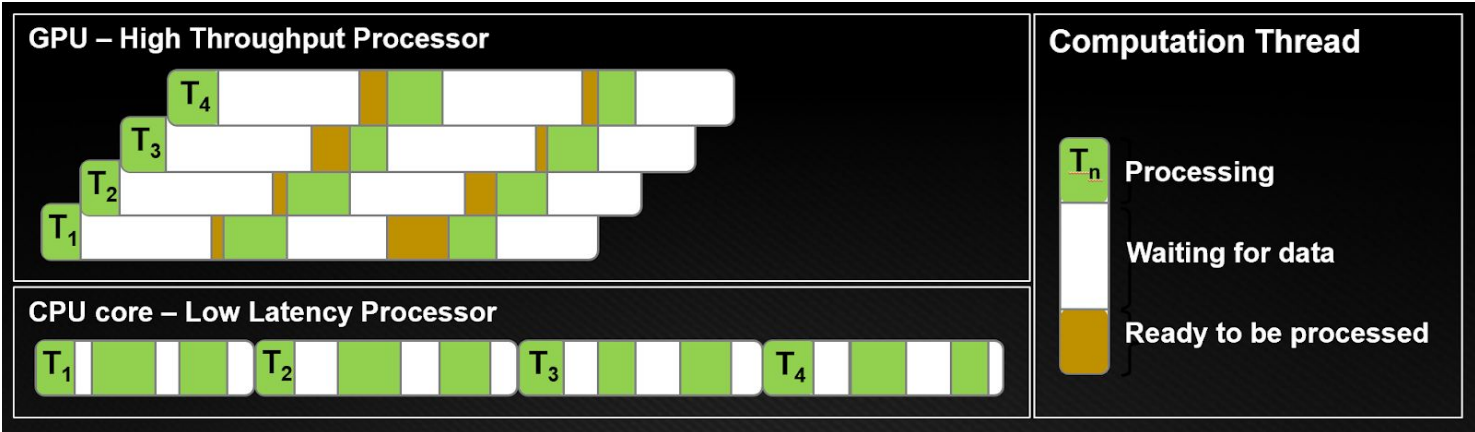
Técnicas de Programación Concurrente

GPU vs CPU



CPU

GPU



GPU vs CPU

CPU

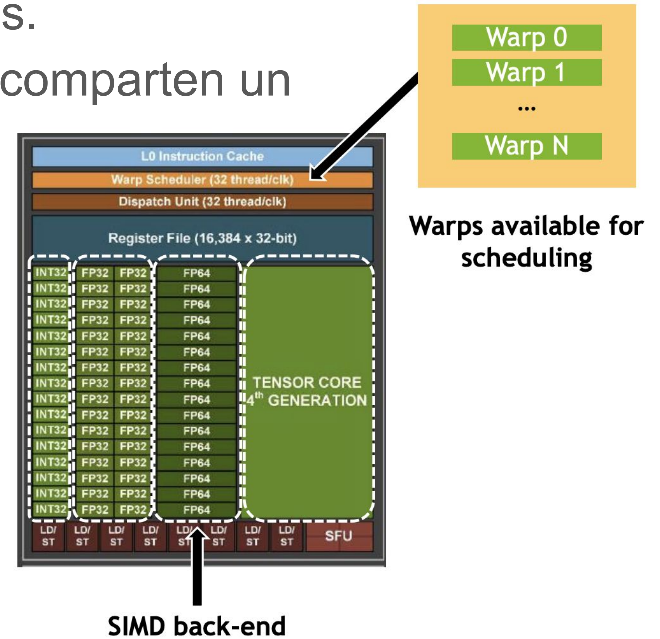
- Diseñadas para minimizar la latencia
- Caches grandes
- Lógica complicada (Pipeline, branch prediction, etc)
- Pocos hilos por núcleo (context switch caro)

GPU

- Diseñadas para maximizar el throughput
- Más transistores en ALUs
- Los tiempos de espera de la RAM se diluyen intercalando la ejecución de distintos hilos
- Context switch muy económico; hasta uno por clock cycle.

SIMT

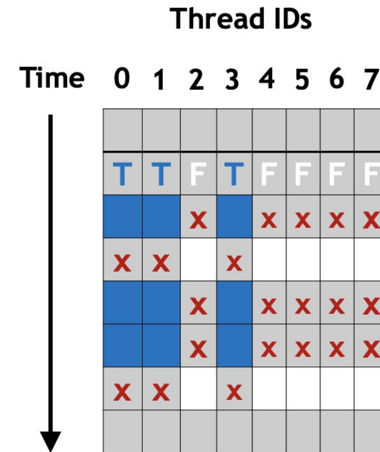
- La ejecución se schedulea en "warps" de 32 hilos.
- Cómo todos los hilos ejecutan el mismo código, comparten un único program counter (hasta Volta).
- Cada hilo trabaja en un "lane" SIMD.
- El contexto de ejecución de cada warp se mantiene en el chip para hacer interleaving



SIMT

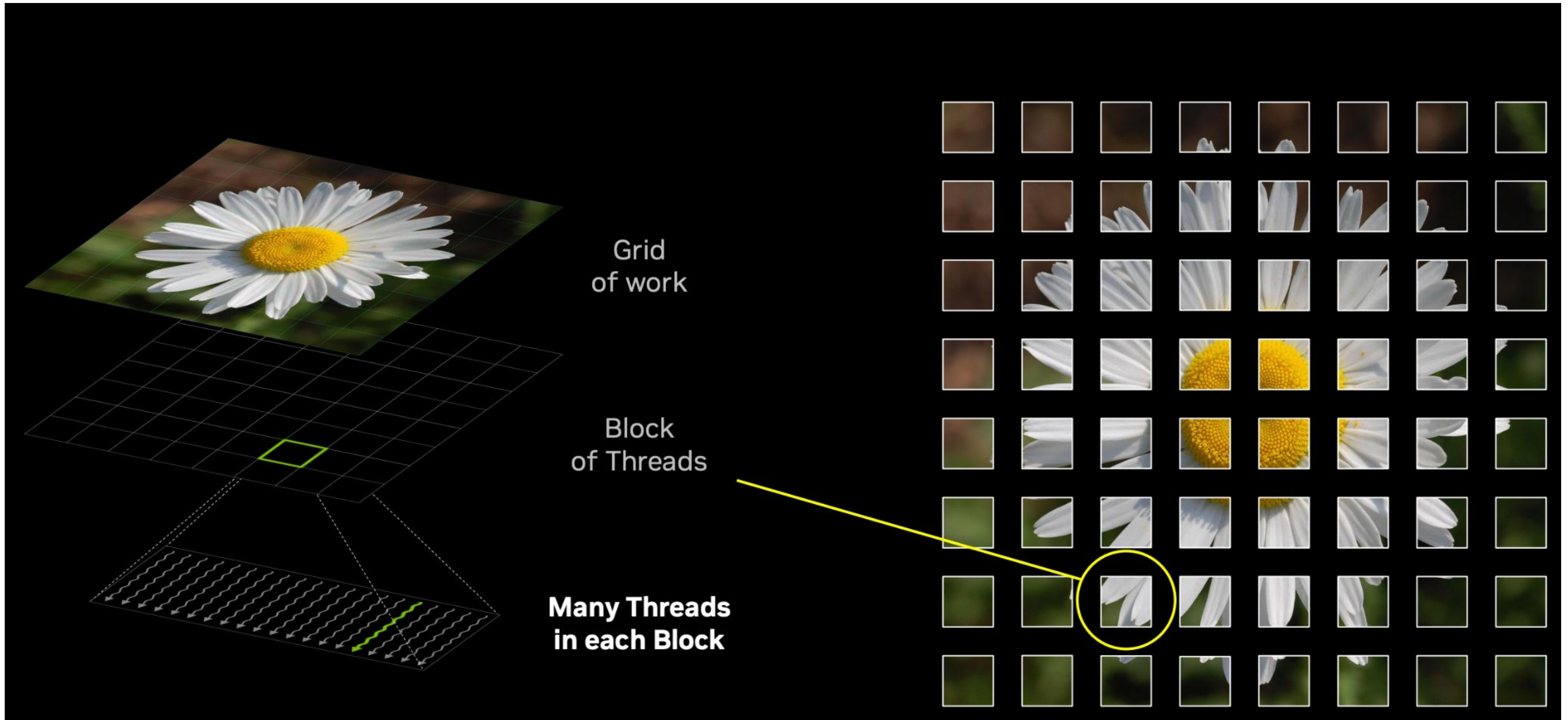
- Cuando el código del hilo tiene un salto condicional, el warp debe ejecutar todas las variantes aplicando una máscara de threads.

- La máxima eficiencia se logra cuando todos los threads del warp convergen.



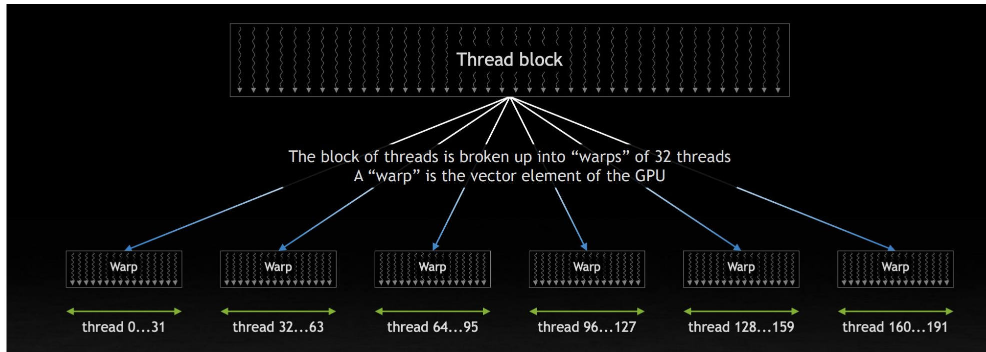

```
if (true) {
  instruction 1
  instruction 2
  instruction 3
} else {
  instruction 4
  instruction 5
}
```

Programming Model - Thread hierarchy

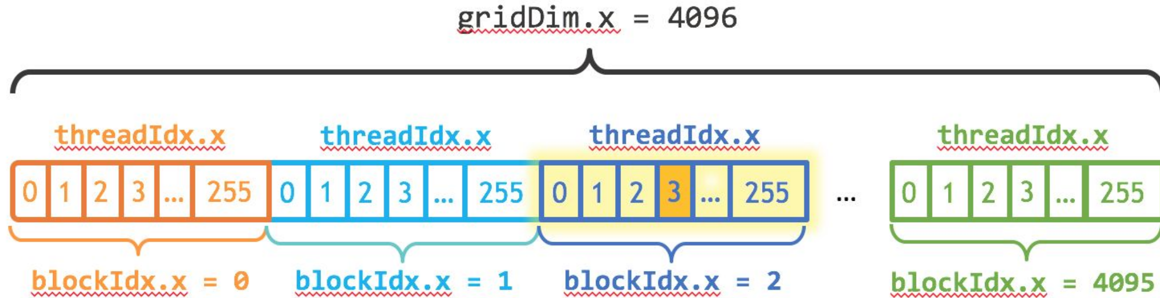


Programming Model - Thread hierarchy

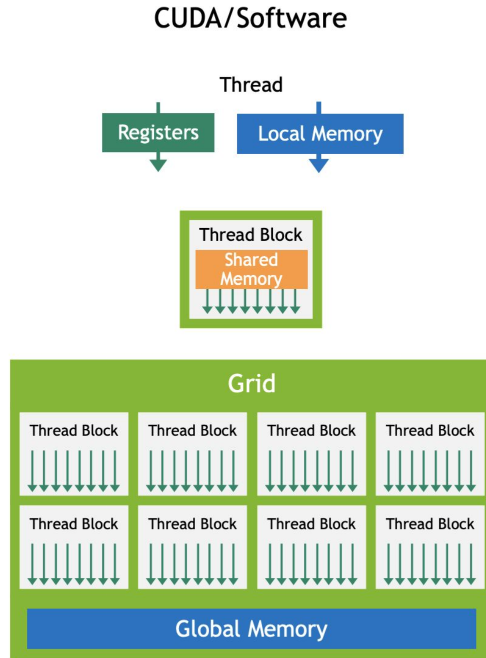
- El programa se divide en un **grid** de bloques independientes; cada uno con N threads.
- Los bloques se mantienen en el mismo SM el ejecutarse.
- Su ejecución se subdivide en warps (SIMT)
- Óptima performance cuando la cantidad de threads en un block es múltiplo de 32.



Programming Model - Thread hierarchy

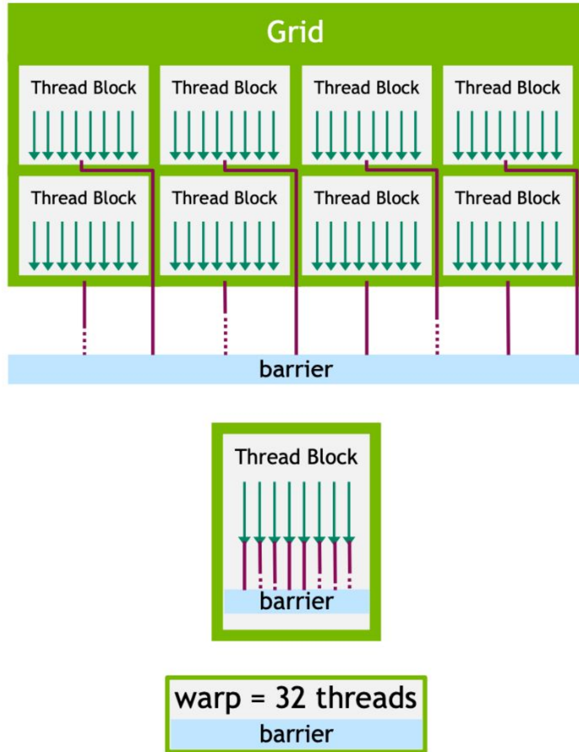


Programing Model - Shared memory



- Per-thread **registers**.
  - Lowest possible latency.
- Per-thread **local** memory.
  - Private storage.
  - Slowest access.
- Per-block **shared** memory.
  - Visible by all threads in a block.
  - Can be used to exchange data between threads in a thread block.
  - Very fast access.
- **Global** memory.
  - Visible by all threads in a grid.
  - Slowest access.

Programming Model - Synchronization



- **Grid** boundary.
  - Kernel completion.
  - `grid_group::sync()` via Cooperative Groups API
    - Requires the kernel to be launched via the `cudaLaunchCooperativeKernel()` API
    - **Slow!** Avoid unless necessary.
- **Thread-block** boundary.
  - `__syncthreads()`
  - `thread_block::sync()` via Cooperative Groups API
  - **Fast!** The most common synchronization level.
- **Warp** or **sub-warp** boundary.
  - `__syncwarp()`
  - `coalesced_group::sync()` via Cooperative Groups API
  - **Very fast!**

Típico programa

1. Declare and allocate host and device memory.
2. Initialize host data.
3. Transfer data from the host to the device.
4. Execute one or more kernels (functions running on GPU). Wait until they finish
5. Transfer results from the device to the host.

Ejemplos

https://colab.research.google.com/github/concurrentes-fiuba/ejemplos-concurrentes/blob/main/practicas/2-vectorizacion/CUDA.ipynb

- Thread Indexing 1D, 2D, 3D
- Procesamiento de imagen

Referencias

- https://developer.nvidia.com/blog/cuda-refresher-reviewing-the-origins-of-gpu-computing/
- https://developer.nvidia.com/blog/easy-introduction-cuda-c-and-c/
- https://colab.research.google.com/github/NVDLI/notebooks/blob/master/even-easier-cuda/An_Even_Easier_Introduction_to_CUDA.ipynb#scrollTo=V1C6GK_MO5er
- https://docs.nvidia.com/cuda/cuda-c-programming-guide/
- Introduction to CUDA Programming and Performance Optimization https://www.nvidia.com/en-us/on-demand/session/gtc24-s62191/
- How CUDA Programming works https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41487/