Pinned memory

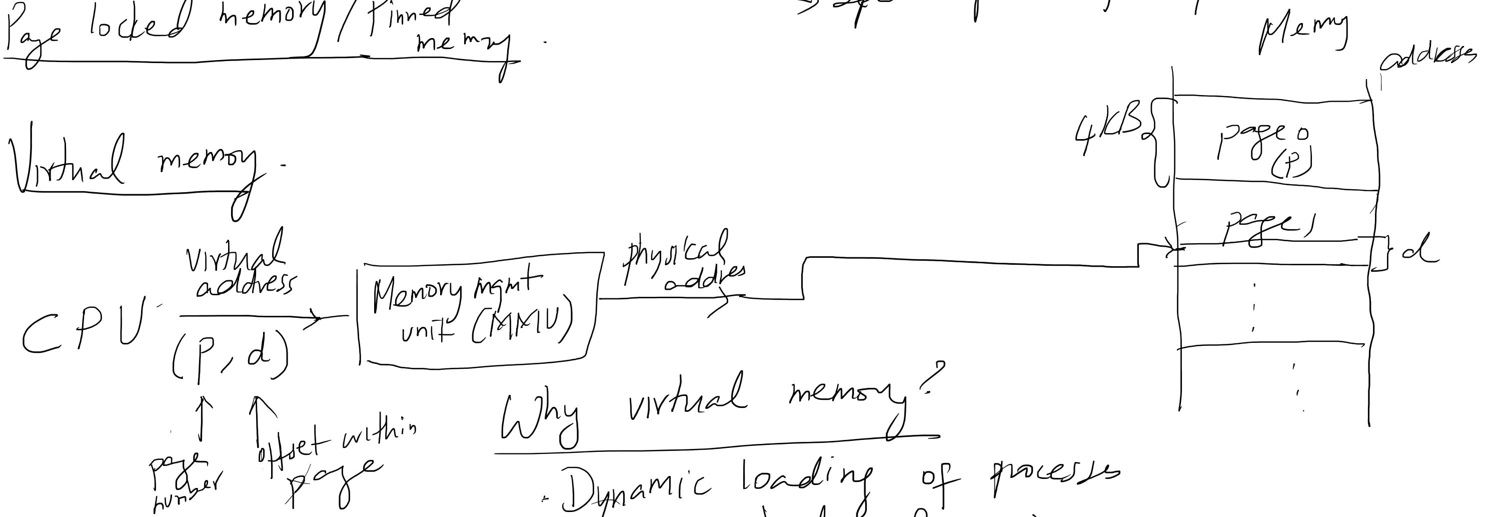
- Pinned or page-locked memory improves achieved bandwidth between host and device
  - Enables zero-copy transfers
- Use *cudaHostAlloc()* to request pinned pages from the Operating System
- Code example: *page_locked_memory*

bottleneck
CPU
GPU

↳ Speed up transfer of data?

. Page locked memory / Pinned memory .

. Virtual memory .

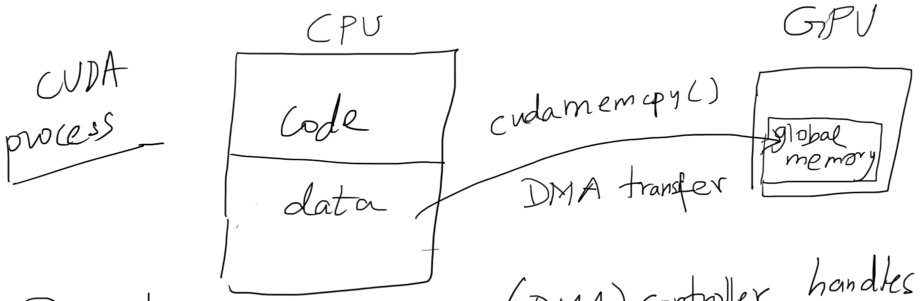


Why virtual memory?

. Dynamic loading of processes
↳ demand paging

Address space protection

Overhead due to address translation

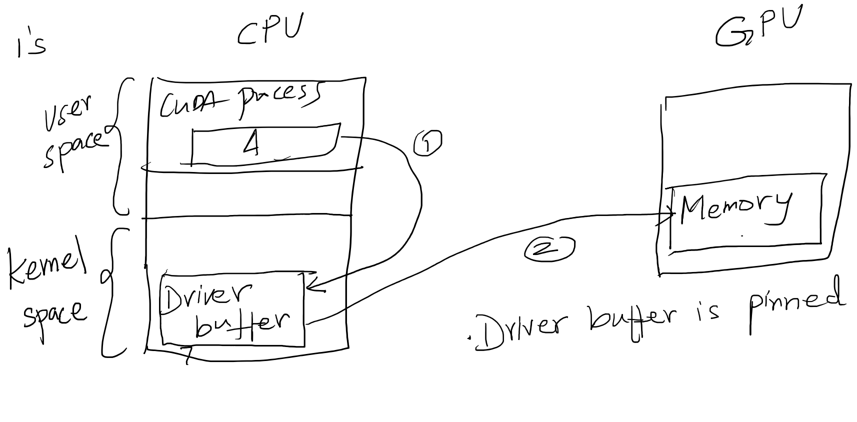


Direct memory access (DMA) controller handles data transfer

- src address (host)
- dest address (device)
- # of bytes to transfer

→ DMA happens in background
→ Frees up CPU for other jobs.

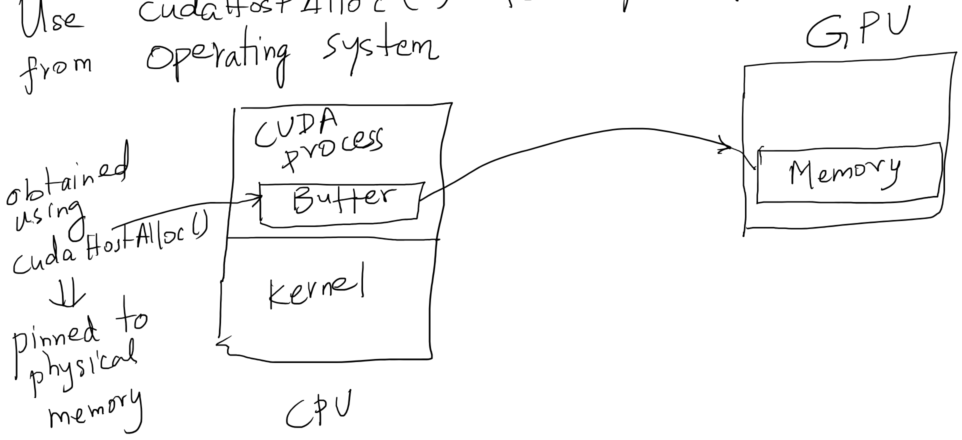
. DMA transfer is a two step process



. Driver buffer is pinned to memory

Zero-copy transfer:

Use cudaHostAlloc() to request pinned memory from operating system



Upto 2x speedup of CPU ⟷ GPU data transfer

CUDA streams

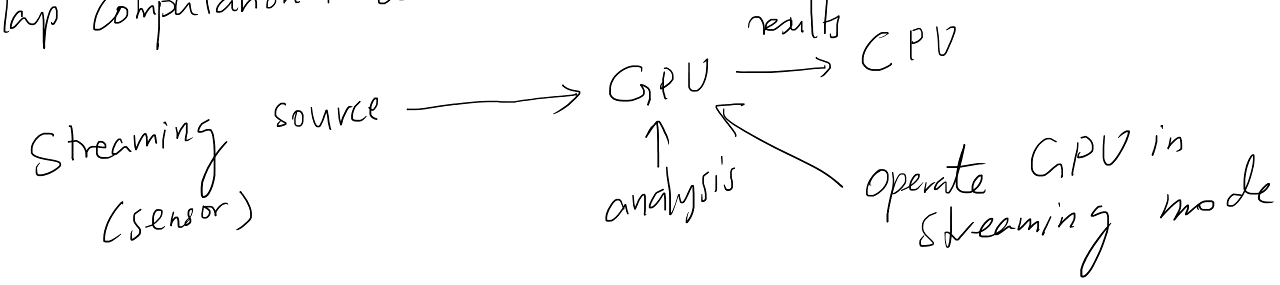
Code example: *streams*

So far: data is transferred in full before computation begins

CPU → GPU

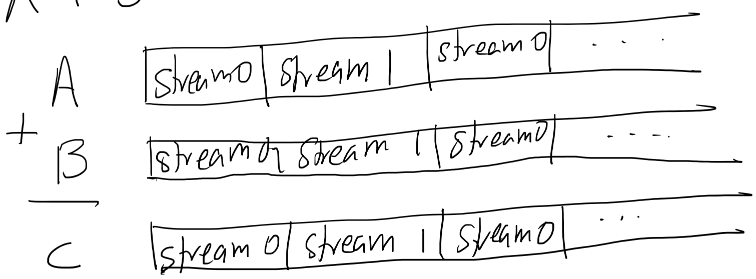
Streaming model:

1) Transfer data in smaller chunks
2) GPU can start processing chunk
3) Continue to transfer additional chunks to GPU
4) Overlap computation + communication

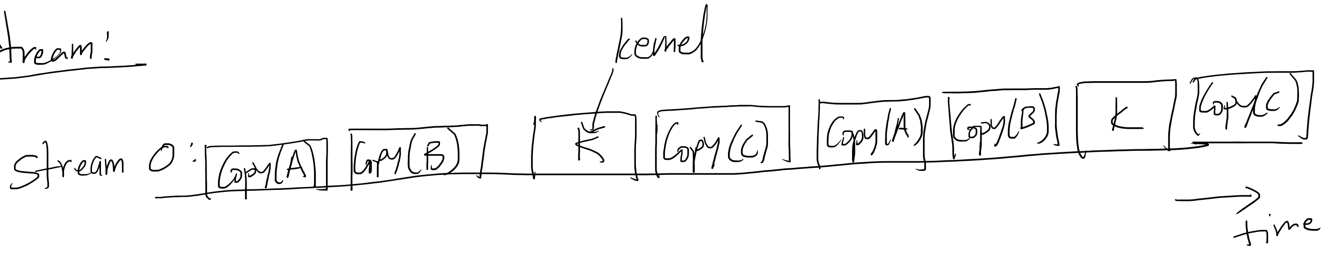


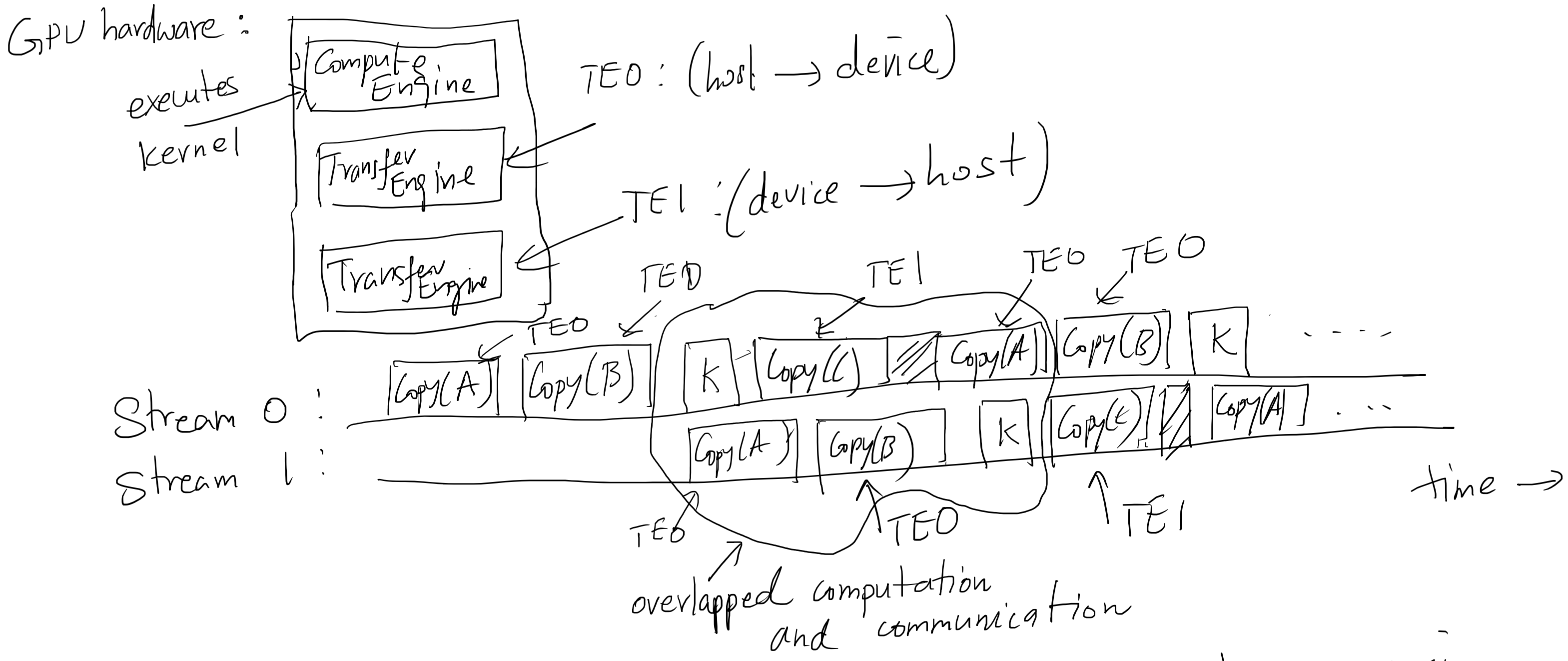
Vector addition:

$C = A + B$



Single stream:





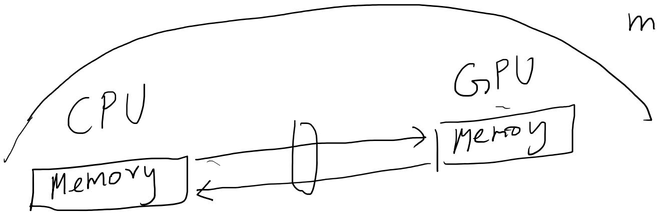
- All operations are done in async fashion using pinned memory.

Unified memory

https://devblogs.nvidia.com/unified-memory-cuda-beginners/

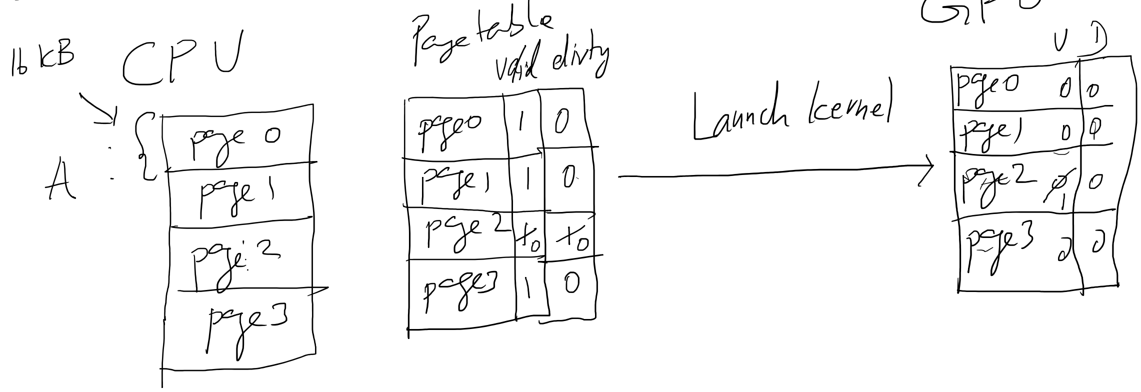
https://devblogs.nvidia.com/maximizing-unified-memory-performance-cuda/

Code example: *unified_mem*



Data transfer between the memories is managed by the CUDA runtime

Page migration between CPU and GPU:



Page table

	valid	dirty
page0	1	0
page1	1	0
page 2	~~1~~ 0	~~1~~ 0
page3	1	0

	V	D
page0	0	0
page1	0	0
page2	~~0~~ 1	0
page3	0	0

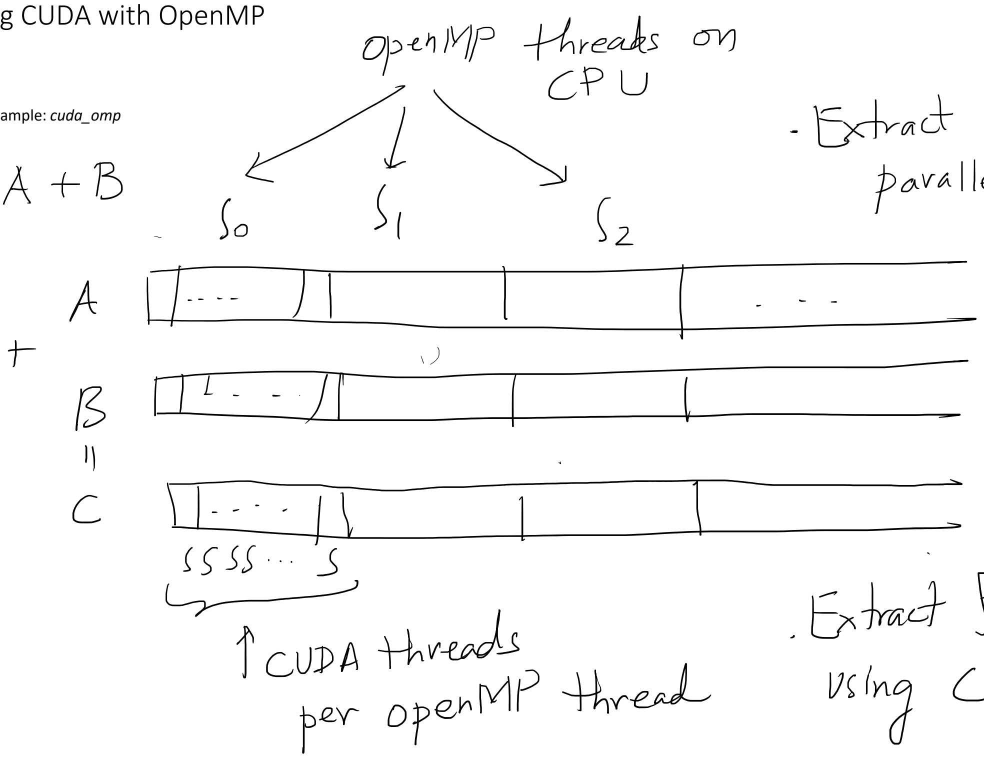
Suppose kernel accesses data in page 2:

1) Page fault
2) Page 2 migrated from CPU to GPU
3) Page tables updated on GPU and CPU

Using CUDA with OpenMP

Code example: *cuda_omp*

$C = A + B$



- Extract coarse-grained parallelism using OpenMP

- Extract fine-grained parallelism using CUDA

Concurrent kernel execution

Code example: *concurrent_kernels*

SIMD fashion → single kernel runs at any given time

MIMD fashion → launchs concurrent kernels on the GPU

⇓

different kernels

CUDA BLAS examples

Sunday, May 31, 2020 12:04 PM

https://docs.nvidia.com/cuda/cublas/index.html

Single precision AX plus Y (SAXPY)

Vector dot product (dot)

Single precision general matrix-matrix multiplication (sgemm)

Single precision general matrix-vector multiplication (sgemv)

See CUBLAS documentation on BBLearn

BLAS : Basic linear algebra subroutines

Vector processing on the CPU

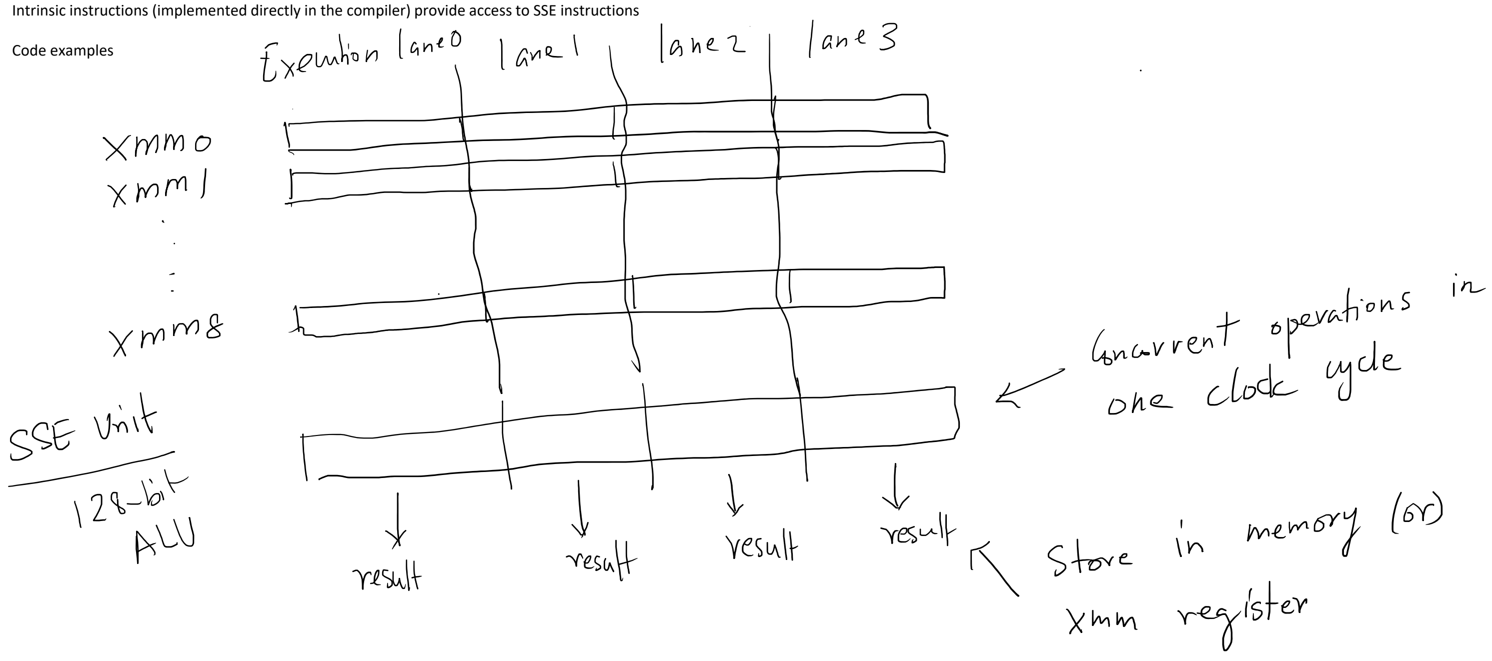
- *Streaming SIMD Extensions* (*SSE*) is a single instruction, multiple data (SIMD) instruction set extension to the x86 architectureop

- SSE2:
  - ○ Eight new 128-bit registers known as XMM0 through XMM7
  - ○ XMM registers can be configured to hold
    - Four 32-bit single-precision floating-point numbers or
    - Two 64-bit double-precision floating-point numbers or
    - Two 64-bit integers or
    - Four 32-bit integers or
    - Eight 16-bit short integers or
    - Sixteen 8-bit bytes or characters
  - ○ The ISA supports both scalar and packed scalar (vector) instructions
    - Memory-to-register/register-to-memory/register-to-register data movement
    - Arithmetic operations (add. Subtract, multiply, divide)
    - Bit-wise logical operations
    - Compare and shuffle
    - …

- More recent developments: *Advanced Vector Extensions* (*AVX*)
  - AVX uses sixteen YMM registers, each of width 256 bits, to perform SIMD operations
    - Eight 32-bit single-precision floating point numbers or
    - Four 64-bit double-precision floating point numbers
  - AVX-512 uses 32 512-bit registers (ZMM0-ZMM31) for SIMD operations

- Intrinsic instructions (implemented directly in the compiler) provide access to SSE instructions

- Code examples


Execution lane0
lane 1
lane 2
lane 3
xmm0
xmm1
xmm8
SSE Unit
128-bit ALU
result
result
result
result
Concurrent operations in one clock cycle
Store in memory (or) xmm register