GPUs


WHAT GIVES PEOPLE FEELINGS OF POWER
MONEY
STATUS
Doing some weird math on GPU 20X faster than CPU

_AXPY loop (aX + Y)

modern ILP

load x	load y	inc i	
mul	load x	load y	inc i
add	mul	load x	load y
store	add	mul	inc i
branch	store	add	
branch	store		

naive sequential

load x
mul
load y
add
store
inc i
branch
load x
mul
load y
add
store
inc i
branch
load x
mul
load y

DLP approaches

vector processor

load x				
load x	mul			
load x	mul	load y		
	mul	load y	add	
		load y	add	store
			add	store
				store

SIMD unit

load x	load x	load x
mul	mul	mul
load y	load y	load y
add	add	add
store	store	store

require additions to ISA *and* hardware

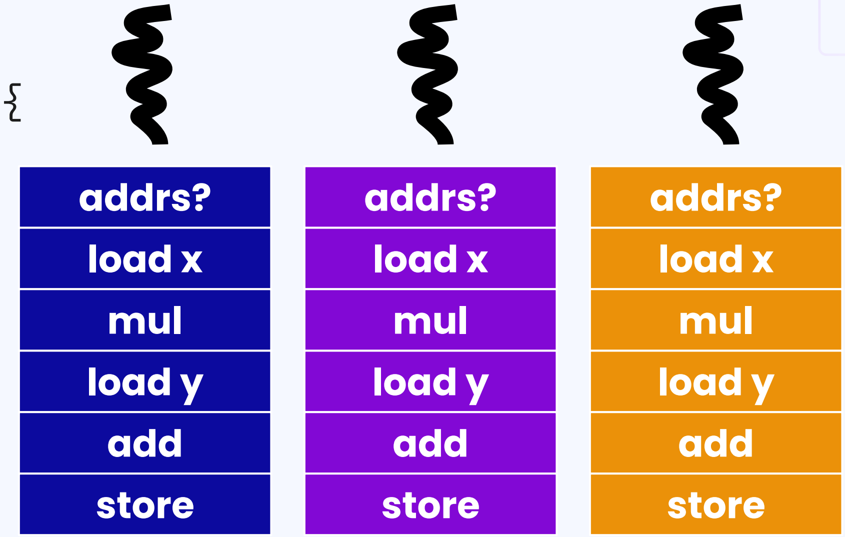
SPMD approach

Single *program*, Multiple data – spawn many versions of same program

Note that this is a *programming* model, not a hardware model!

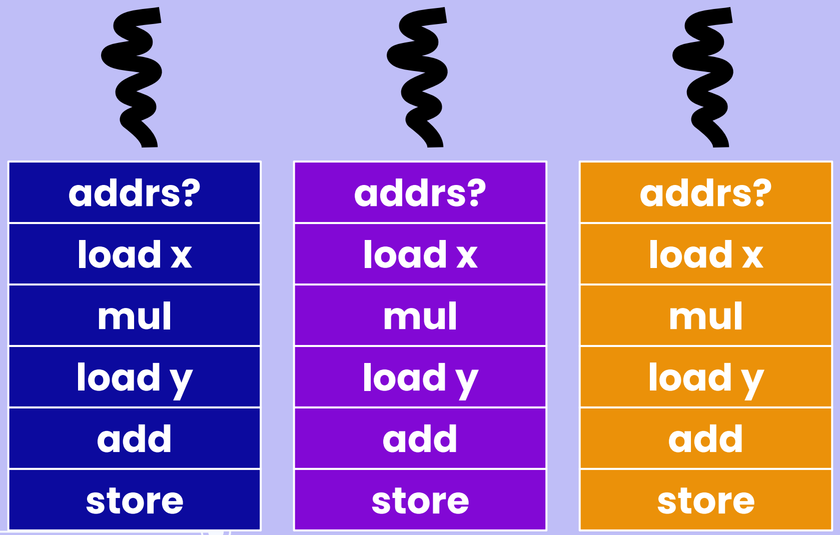
How threads are scheduled is invisible to programmer

```
# one thread for each index
void myThread(int n, float a, …) {
    int i = tid; // thread ID
    if (i < n)
        y[i] = a * x[i] + y[i];
}
```



? ? ?

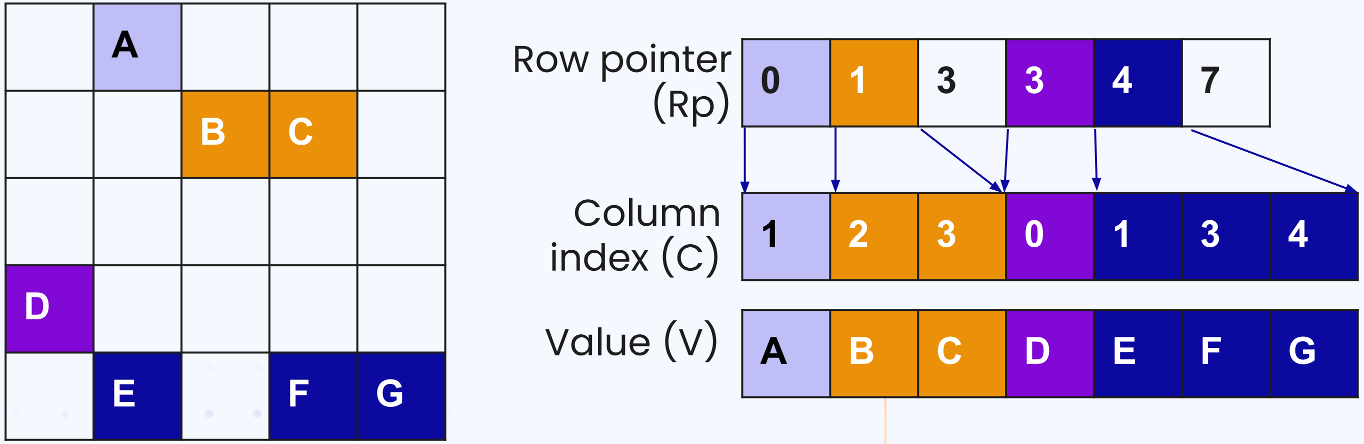
In what ways is SPMD a more flexible model than the SIMD processors we saw last week?



Sparse matrices

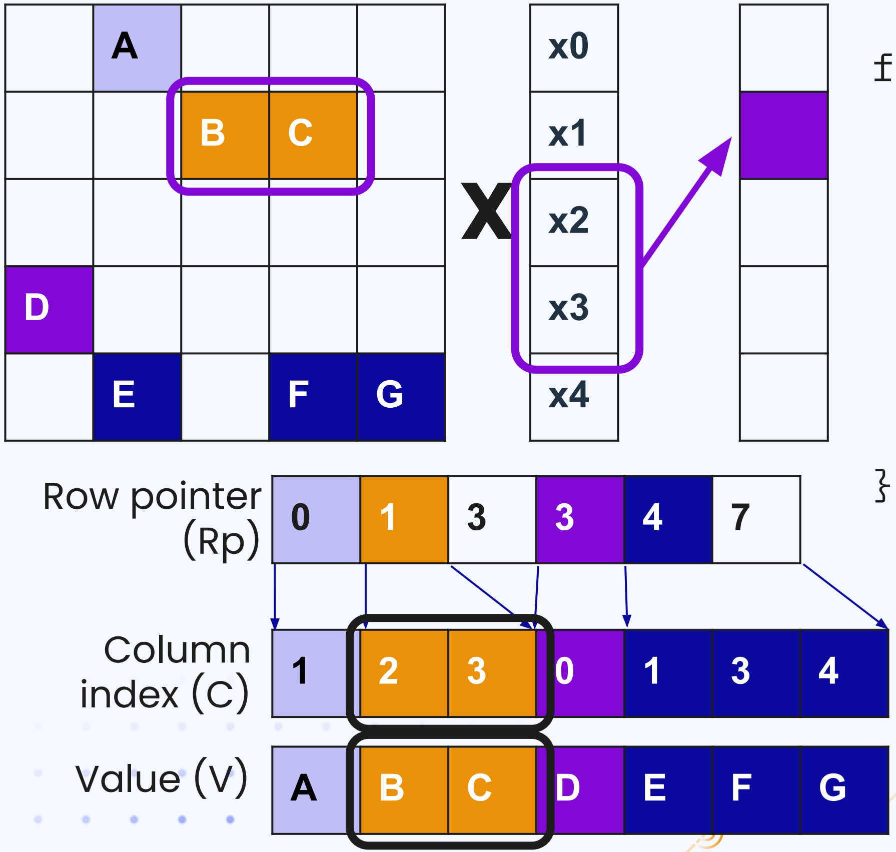
An *nxn* matrix is *sparse* if the number *m* of nonzero entries is a small fraction of the total

CSR (compressed sparse row) representation allows for easy access to nonzero elts



CSR can be computed from original matrix. Can the conversion be parallelized?

Multiplication in CSR



Rp[r + 1] - Rp[r]

C + Rp[r]

V + Rp[r]

```
float multRow(int rSize, int* Cr,
              float* Vr, float* x) {
    float sum = 0;
    for (int i = 0; i < rSize; i++) {
        sum += Vr[i] * x[Cr[i]];
    }
    return sum;
}
```

```
for (int r = 0; r < n; r++) {
    int rBeg = Rp[tid];
    int rSize = Rp[tid + 1] - Rp[tid];
    result[tid] = multRow(rSize, C[rBeg],
                          V[rBeg], x);
}
```

can turn this (**outer loop!!**) into threads!

? ? ?

How can we run a SPMD program using SIMD ideas?

SIMT

Single instruction, multiple threads

- Each thread uses the same program memory, PC (but its own registers, FUs, stack pointer)
- *Warp* (Nvidia term) of threads executes in lockstep

Cycle 1	addrs?	addrs?	addrs?
Cycle 2	load x	load x	load x
Cycle 3	mul	mul	mul
.	load y	load y	load y
.	add	add	add
.	store	store	store

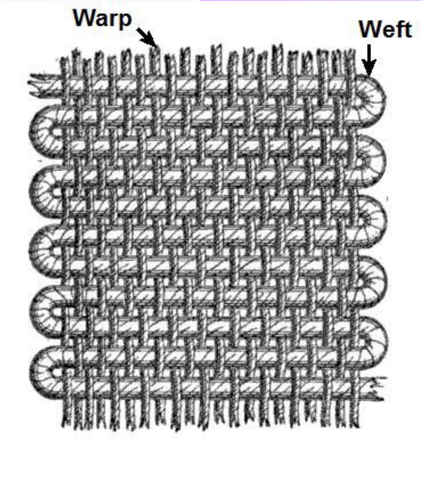


image source

S...P?I?...M?...D?T?!? What???

Recap:

Flynn's taxonomy: describes **hardware computation model**

- *SISD*: single instruction, single data (traditional uniprocessor)
- *SIMD*: single instruction, multiple data (ISA/hardware for DLP)
- *MIMD*: multiple instruction, multiple data (multiprocessor)

SPMD: single program, multiple data (describes **programming model**)

- *SIMT*: single instruction, multiple threads; SIMD-style computation (**hardware**) to implement SPMD operation

Large data

What do we do if n = 8192??

```
for (int i = 0; i < n; i++) {
    y[i] = a * x[i] + y[i];
}
```

```
void myThread(int n, float a, …) {
    int i = tid; // thread ID
    if (i < n)
        y[i] = a * x[i] + y[i];
}
```

SISD approach: ¯_(ツ)_/¯

Vector approach: vector inner loop, scalar outer loop

SPMD approach: get a GPU
← aka: spawn 8192 of those guys on beefy hardware

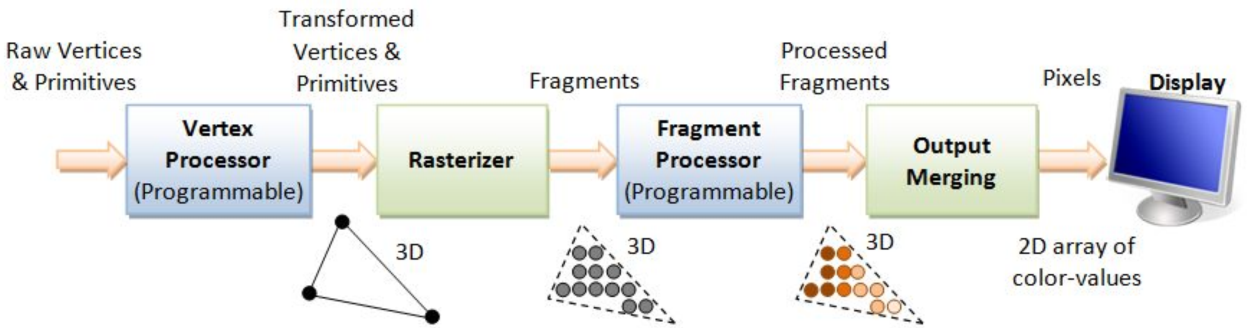
What's a GPU?

Graphics Processing Unit

Accelerator that originally helped CPU render 3d graphics

Predecessors: arcade game circuits, VGA controllers

Huge parallelism, programmability: attractive for largescale computations



3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

image source

What's a GPU? (CS1952y answer)

A heck of a lot of SIMT processors, arranged in groups

Nvidia terminology: Streaming Multiprocessors (SMs)

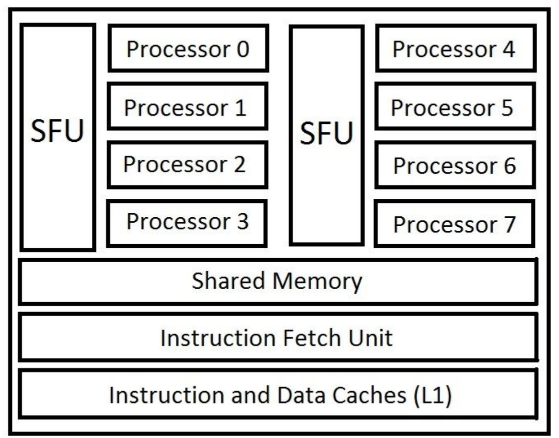
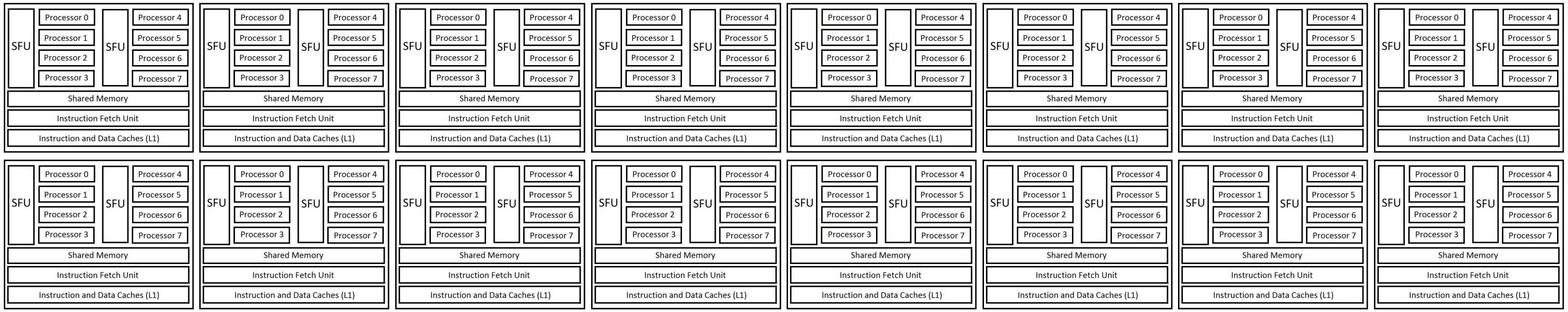


image source



? ? ?

Warp size is 32 (2^5) threads

We want to run 8192 (2^{13}) threads

Do we need 256 SMs?

Programming model + terminology

CUDA (Compute Unified Device Architecture): API for programming Nvidia GPUs at the thread level

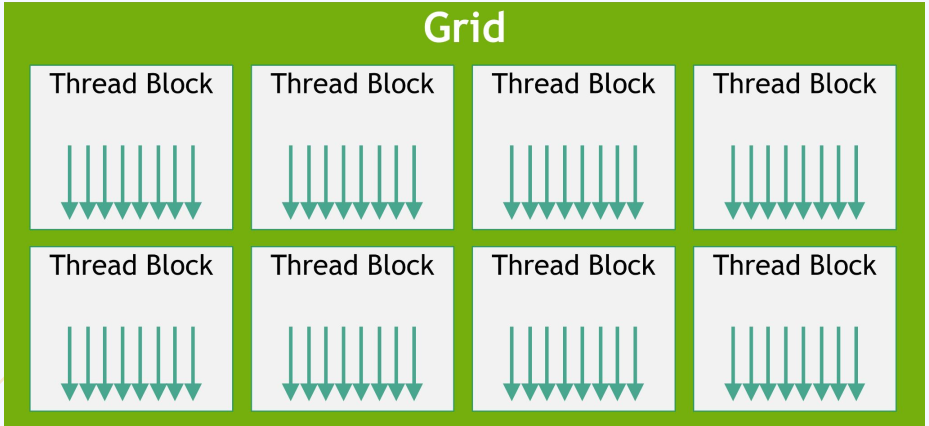
Programmers don't worry about warps (controlled by hardware)

Thread block: threads running on the same SM (scheduled by hardware)

Exposed to programmer – can make assumptions about eg shared memory within a block

Grid: entirety of thread workload

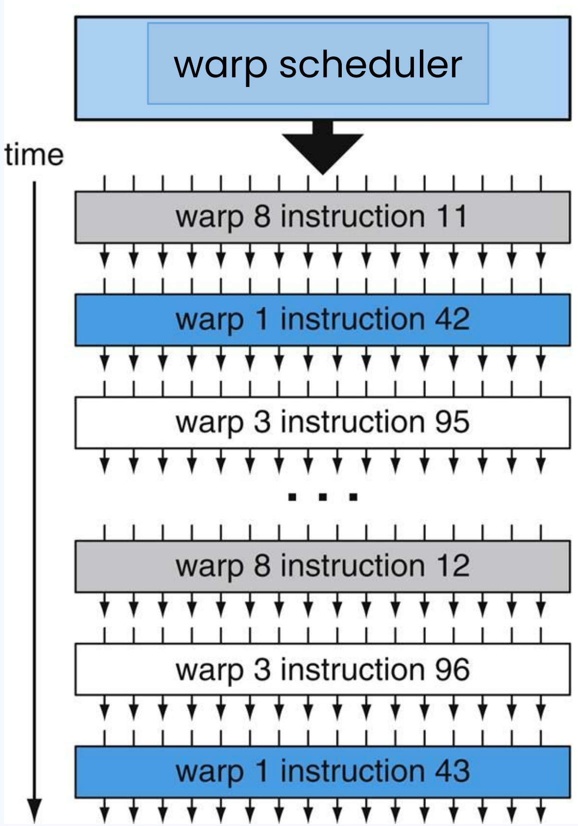
so how are warps managed within a block?



Finegrained multithreading, again

(Truly more like multiwarping)

Why not just run all of warp 1, then 2, then 3?



P&H fig. B.4.2

Recap

Programmer

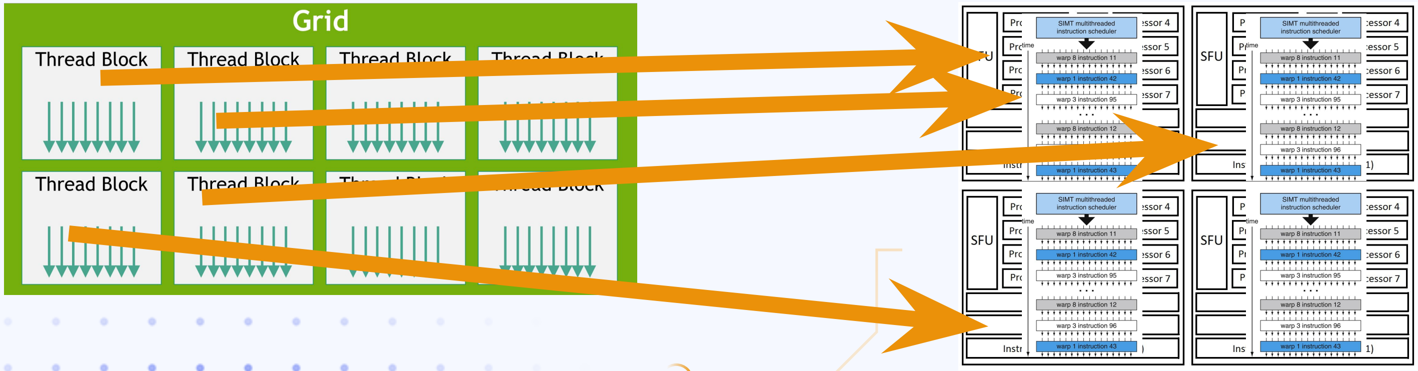
Chooses block size, # threads

Writes one program to run on many, many threads (SPMD)

Hardware

Schedules each block to an SM

Threads inside blocks are split up into warps to enable lockstep execution



SAXPY example rewritten in CUDA

```
// CPU side to invoke 8192 threads to run 8000 computations
__host__
// 32 blocks, 256 threads per block
myThread<<<32, 256>>>(8000, 2.0, x, y);

// GPU side
__device__
void myThread(int n, float a, float* x, float* y) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a*x[i] + y[i];
}
```