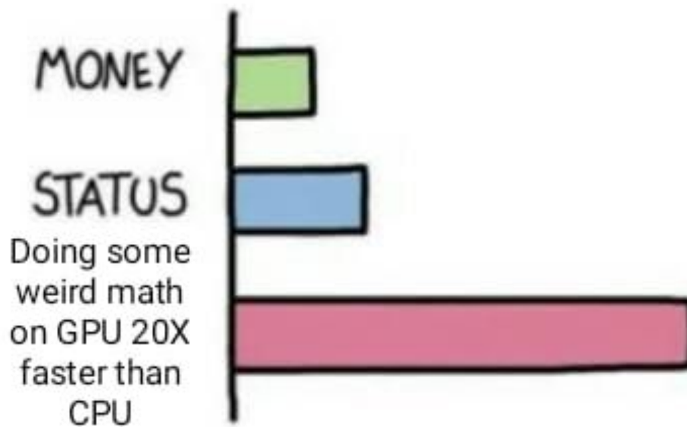
GPUs



Our friend the SAXPY loop

*Single-precision a*x plus y*

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

Basic (SISD) approach: compile sequentially w/ branches

Can be made more efficient with:

- Dynamic ILP: pipelining, OOO, speculation, superscalar
- Static ILP: loop unrolling, VLIW, etc

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

SIMD approach

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

Compile into vector or SIMD instructions

Requires additions to ISA *and* hardware

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

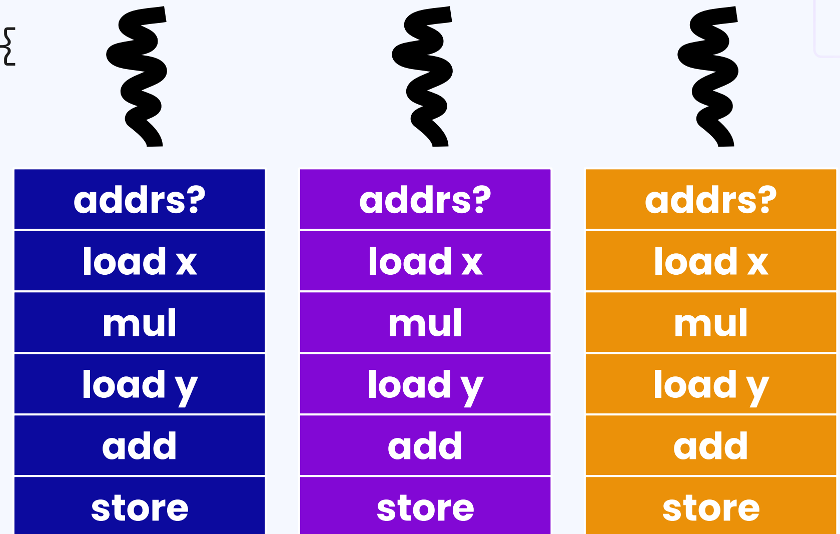
SPMD approach

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

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

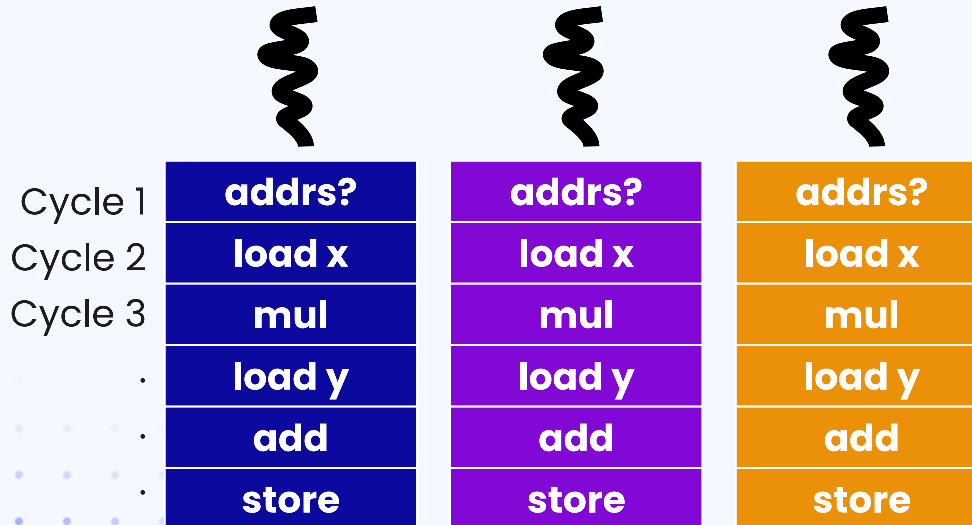


How can we run a SPMD program using SIMD ideas?

SIMT

Single instruction, multiple threads

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



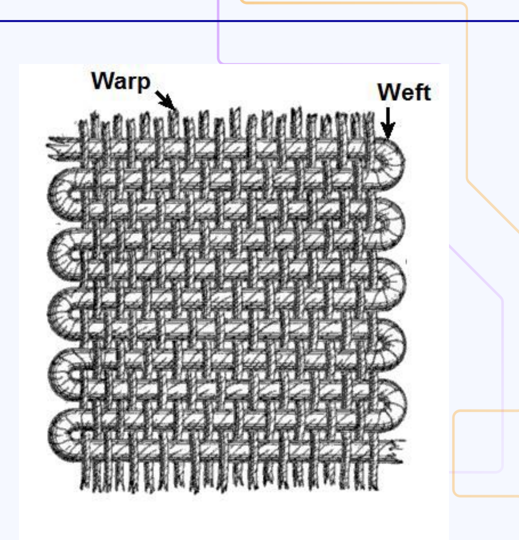


image source

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

Recap:

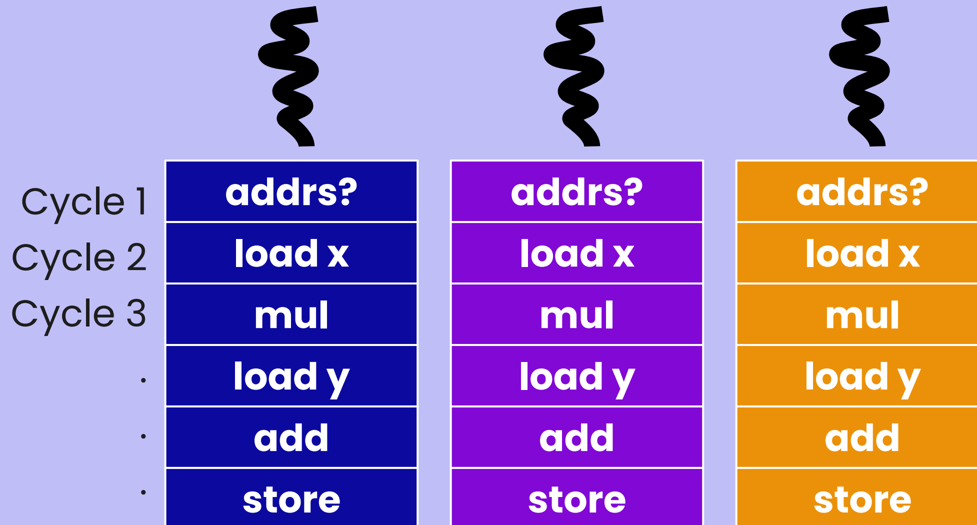
Flynn's taxonomy: describes **hardware computation model**

- *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

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

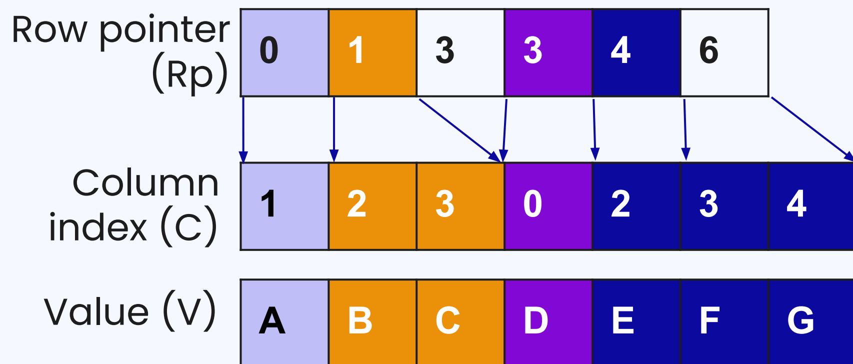


Sparse matrices

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

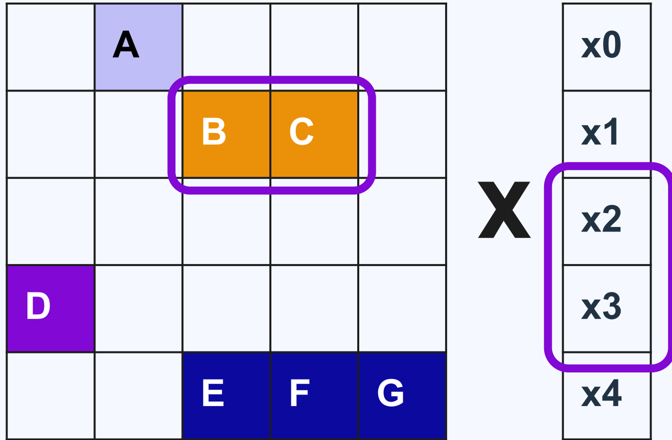
CSR (compressed sparse row) representation allows for easy access to nonzero elts

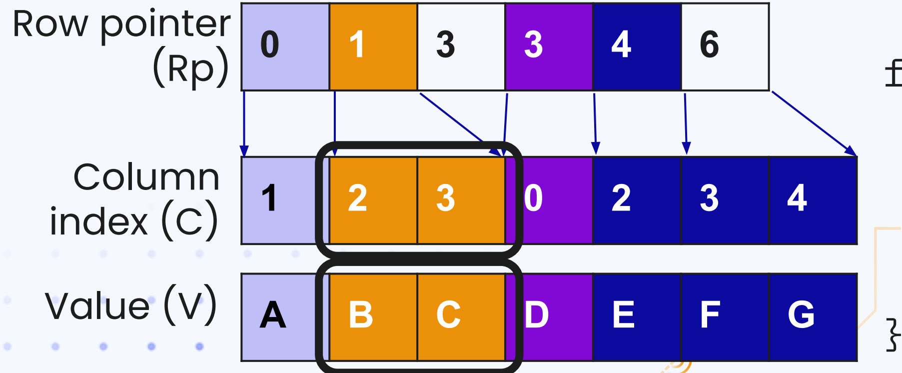
	A			
		B	C	
D				
		E	F	G



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

Multiplication in CSR





What about this loop?

```
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;
}
```

Loop can be turned into threads!

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

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: outer loop

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

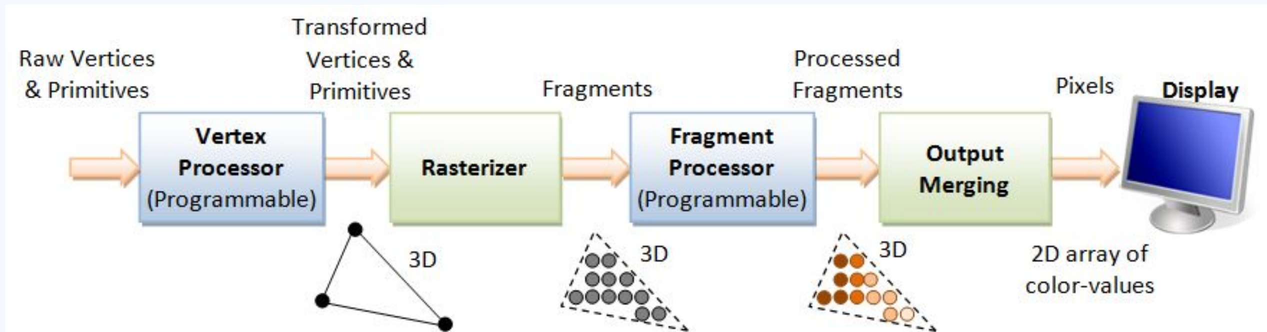
What's a GPU?

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)

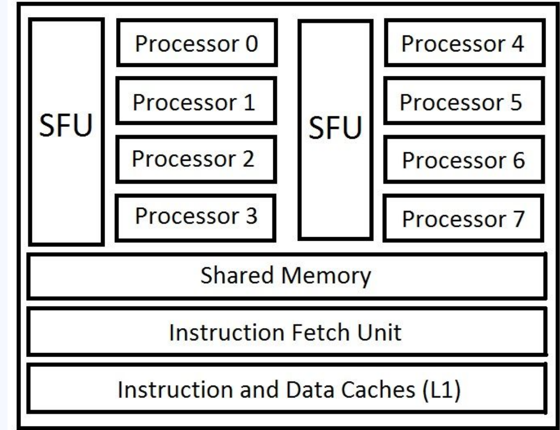
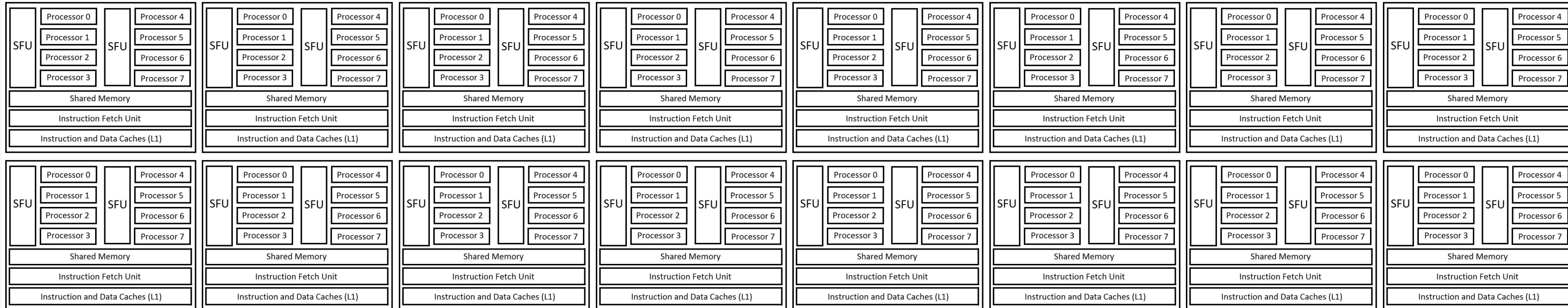


image source



Warp size is 32 (2^5) threads

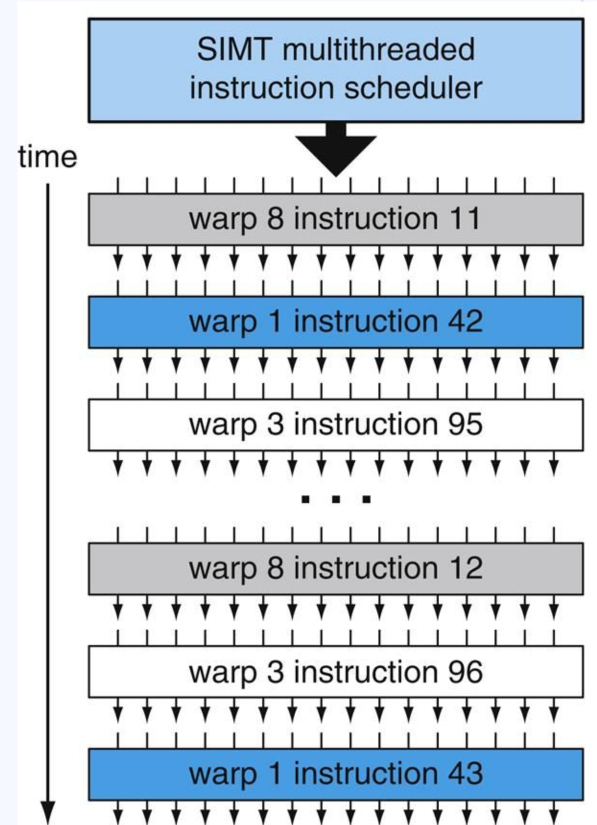
We want to run 8192 (2^{13}) threads

Do we need 256 SMs?

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

Terminology + CUDA

Thread block: threads running on the same SM

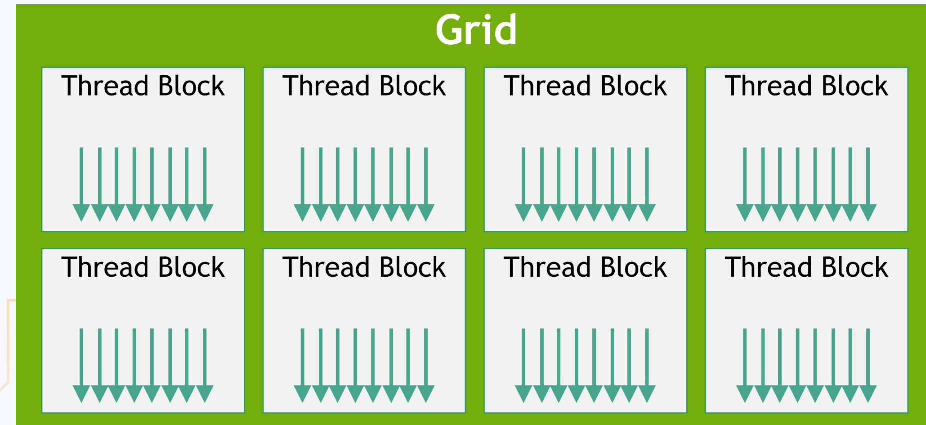
ex: 512 threads in a block = 16 warps to schedule

Grid: multiple blocks combined to enable computation

Blocks can be scheduled to different SMs

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

image source



SAXPY example rewritten in CUDA

```
// CPU side to invoke 8192 threads
__host__
// 32 blocks, 256 threads per block
myThread<<<32, 256>>>(8129, 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];
}
```