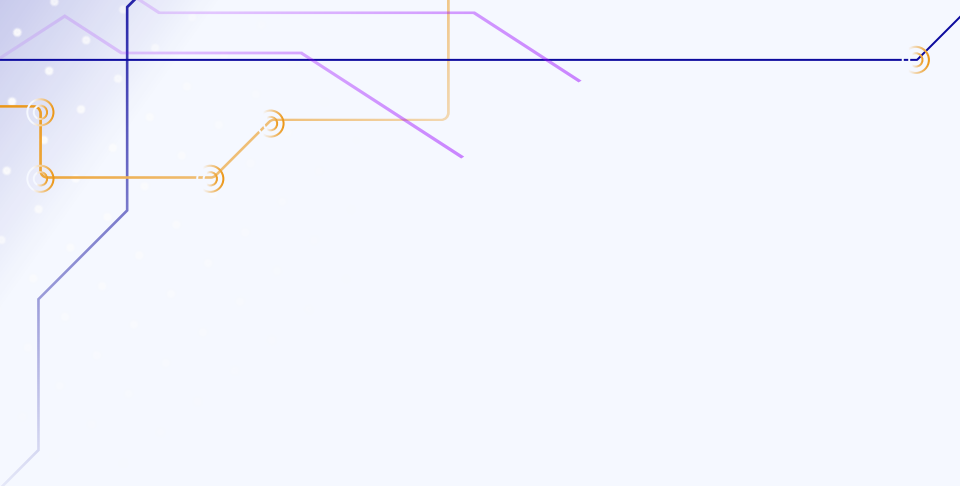


The top-left corner of the slide features a decorative graphic of circuit lines. It includes a horizontal blue line, a diagonal purple line, and an orange line with several small circular nodes connected by segments.

Execution on GPUs

The bottom-right corner contains another decorative graphic. It features a grid of small blue dots, a diagonal line of blue dots, and a wavy orange line with several small circular nodes. There are also some purple geometric shapes and lines in this area.

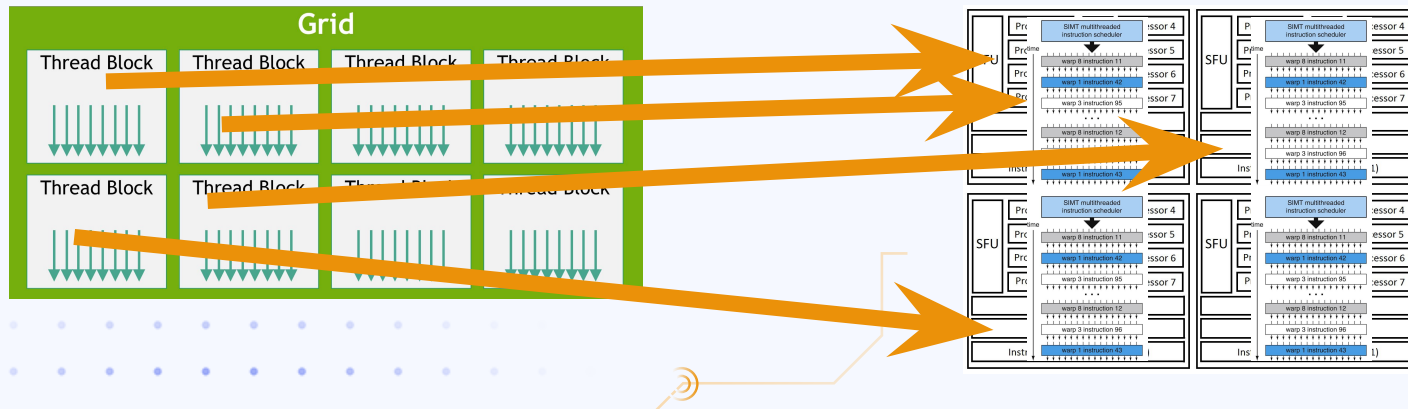
Each block to a
Multiprocessor
inside blocks are split
arps to enable

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

Chooses block size, # threads

Schedules each block to a Streaming Multiprocessor

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



A note about nVidia

They're not paying me

These slides use nVidia/CUDA terminology for practicality (matches dominant platform)

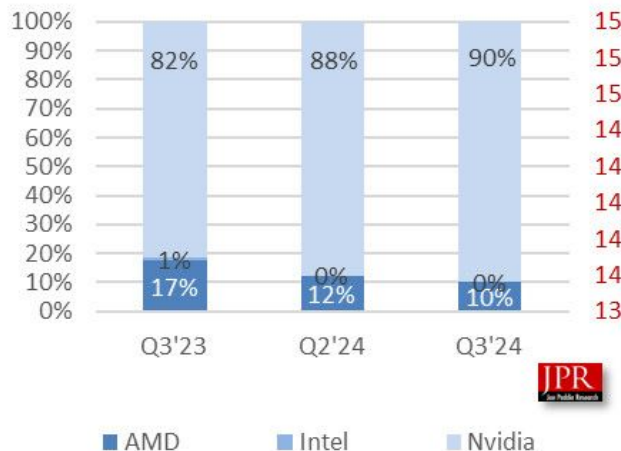
See H&P Fig. 4.25 for translations to their terms and OpenCL

Is this good for the consumer/society? I'm not an economist but I'm happy to hear your takes

source



Total AIB share and units



source

CUDA programming model

API for programming nVidia GPUs at the thread level

Allows for specifying host (CPU) code (to set up threads) + device (GPU) code (parallel **kernel**)

Host and device have separate memories

Note: this is not a graphics course, and not a GPU programming course (also GPUs are being used, developed, and marketed for more than graphics)

We're studying the details of highly parallel architectures that use the SPMD programming model

C Program
Sequential
Execution

Serial code

Parallel kernel
Kernel0<<<>>>()

Serial code

Parallel kernel
Kernel1<<<>>>()

Host

Device

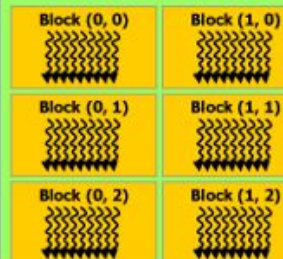
Grid 0



Host

Device

Grid 1



nVidia PTX ~~ISA~~

PTX = parallel thread execution

Full summary in P&H fig. B.4.3, documentation [here](#)

Suffixes may define operand data type, memory space

Arithmetic: add, sub, mul, etc

Special function: sqrt, sin, cos, etc

Logical: and, or, xor, etc

Memory: ld, st, tex (texture lookup), atom

Control: branch, call, ret, sync, exit

Why isn't it a "true" ISA?

- Actual machine instructions are proprietary/may differ by device – device has to translate PTX to its own instructions
- Virtual registers

CSR multiplication in CUDA

```
__host__  
// set up matrix in CSR format here: ...  
// 8 blocks, 256 threads per block to do multiplication  
csrMult<<<8, 256>>>(2048, Rp, C, V, x, y);  
  
// GPU side  
__device__  
void csrMult(int n, int* Rp, int* C, float* V, float* x, float* y) {  
    int r = blockIdx.x * blockDim.x + threadIdx.x;  
    if (r < n) {  
        int rBeg = Rp[r];  
        int rSize = Rp[r + 1] - Rp[r];  
        y[r] = multRow(rSize, C + rBeg, V + rBeg, x);  
    }  
}
```

Instead of loop, use these
provided variables as
per-thread "coordinates"
for data access



What do we need to watch out for when
programming using CUDA?

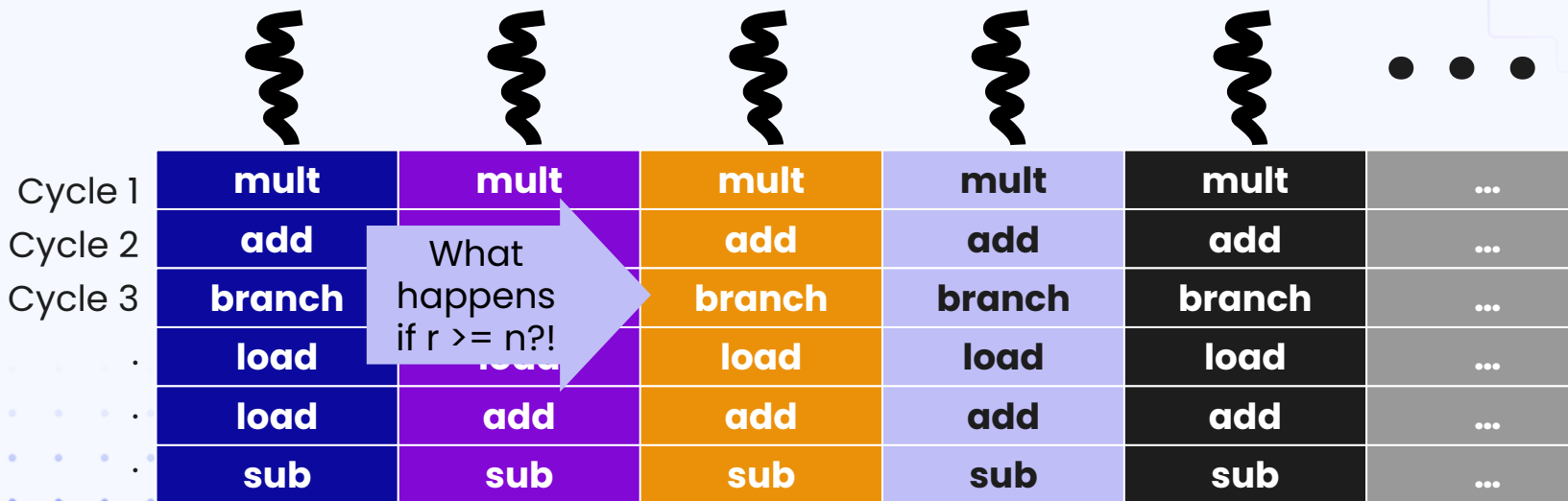
Hardware-aware SPMD

SPMD promises us the power of writing scalar programs and running them with high parallelism

It's (relatively) easy to write a SPMD program that runs, BUT:

Kernel execution

```
int r = blockIdx.x * blockDim.x + threadIdx.x;  
if (r < n) {  
    int rBeg = Rp[r];  
    int rSize = Rp[r + 1] - Rp[r];  
    y[r] = multRow(rSize, C + rBeg, V + rBeg, x);  
}
```



Active threads

```
if (threadIdx.x < 4) {  
    A;  
    B;  
} else {  
    X;  
    Y;  
}  
Z;
```

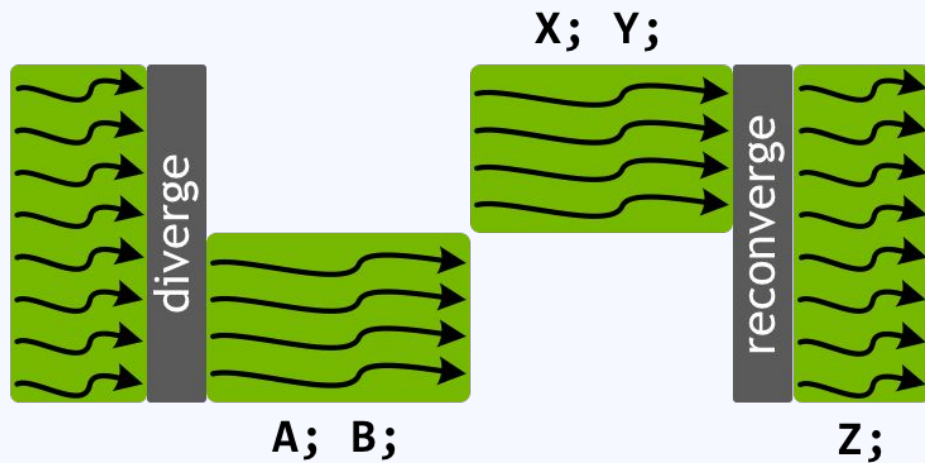


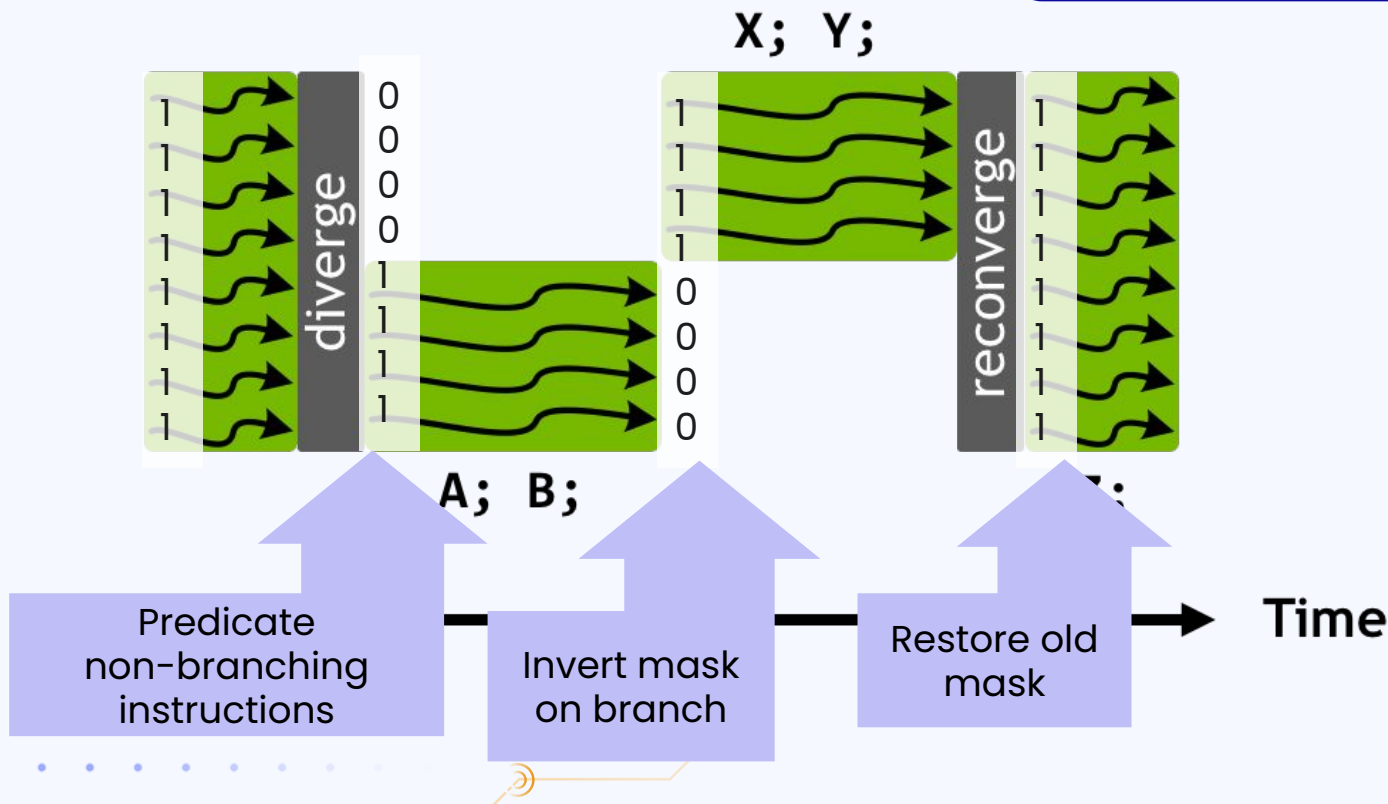
image source

How to keep track of which threads are active?
How to keep track of when to reconverge?

Time

Execution mask

Use single PC and keep track of which threads are using that PC

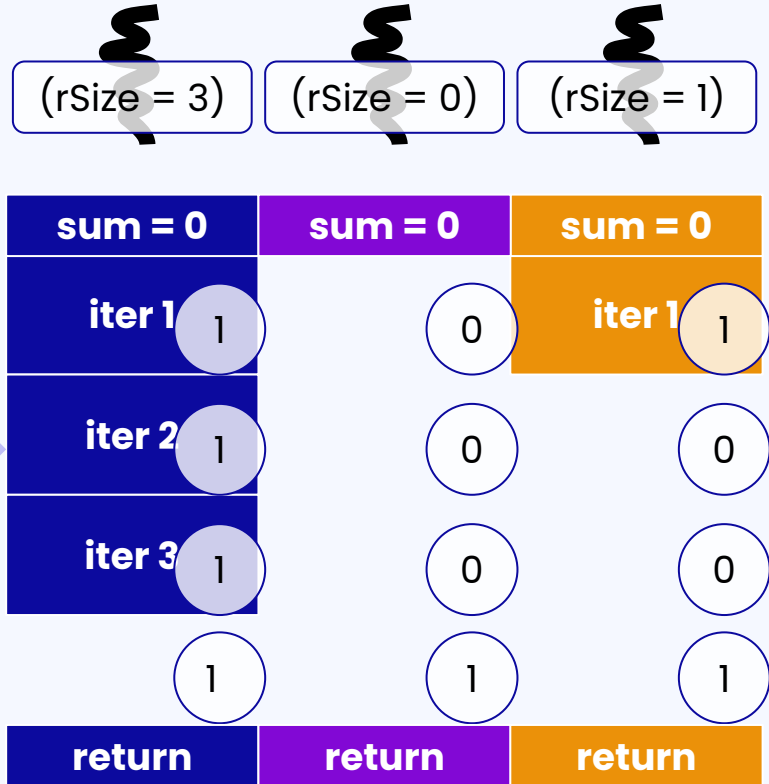


Execution mask & loops

```
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;
}
```

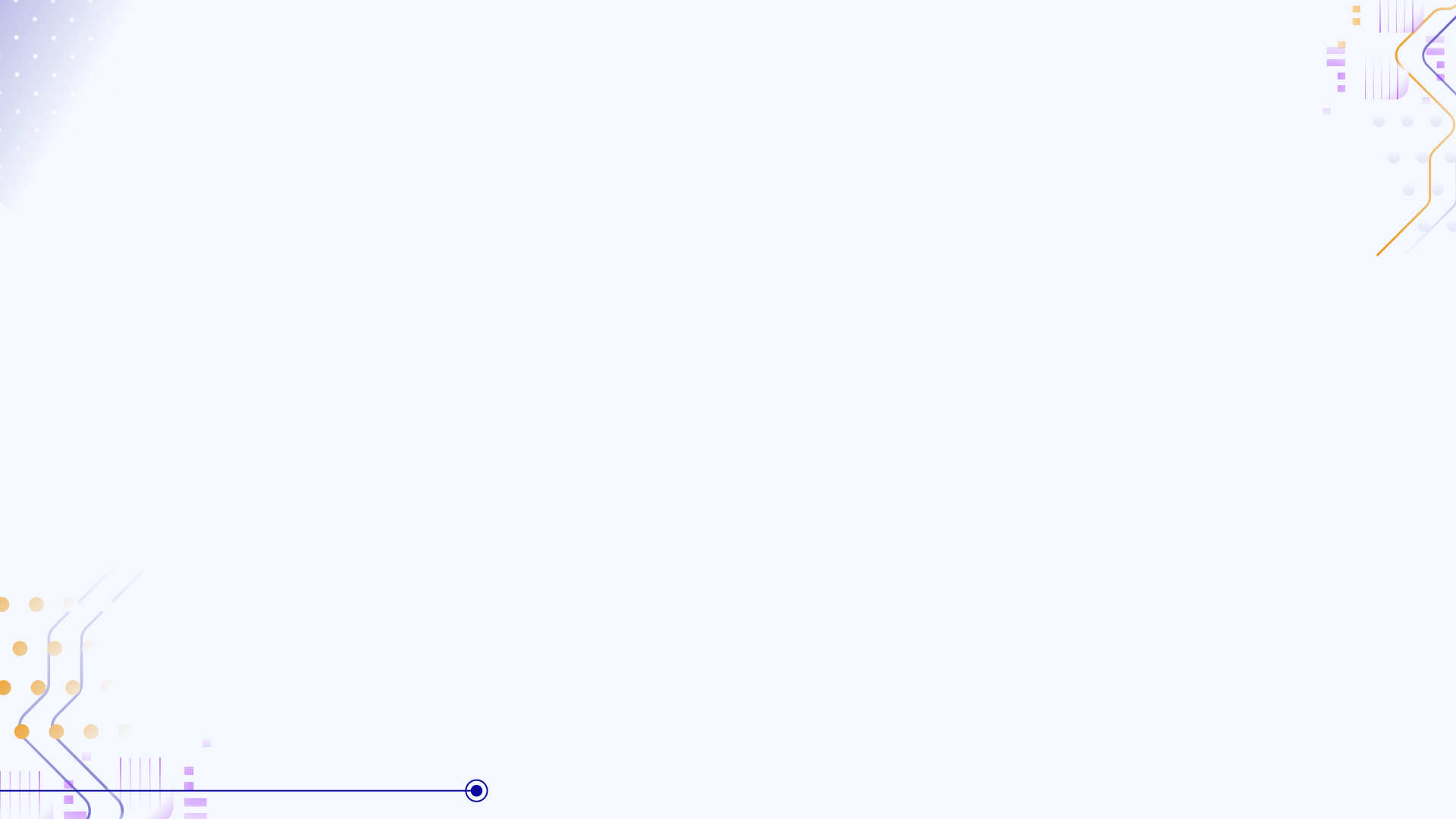
each iteration
computes branch

reconverge!





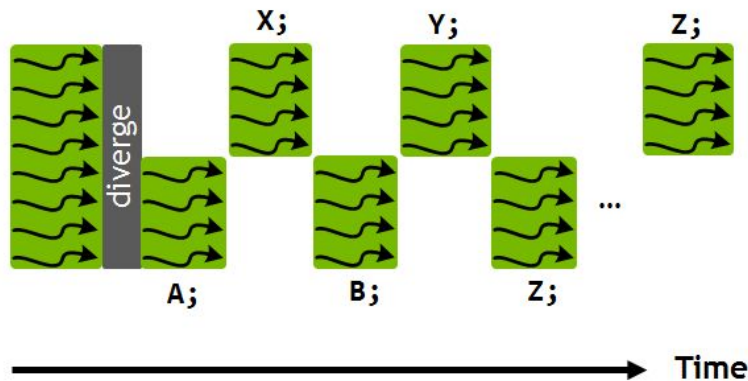
Where might execution mask usage become complicated?



nVidia Volta

Allows switching between
execution paths
...how?

```
if (threadIdx.x < 4) {  
    A;  
    B;  
} else {  
    X;  
    Y;  
}  
Z;
```

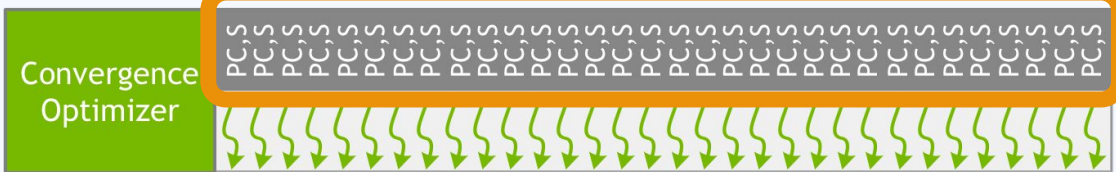


Pre-Volta

Program
Counter (PC)
and Stack (S)



32 thread warp **Volta**



32 thread warp with independent scheduling

image source

image source

Tensor cores

An accelerator on the accelerator
(see also Ray Tracing cores)

Circuits on GPUs optimized for AI
(matrix operation $D = A * B + C$)

Uses mixed-precision to speed up
math, reduce memory demands

Claim huge (8x-32x per
generation) performance gains
over SM matrix multiply

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 FP16 FP16 FP16 or FP32

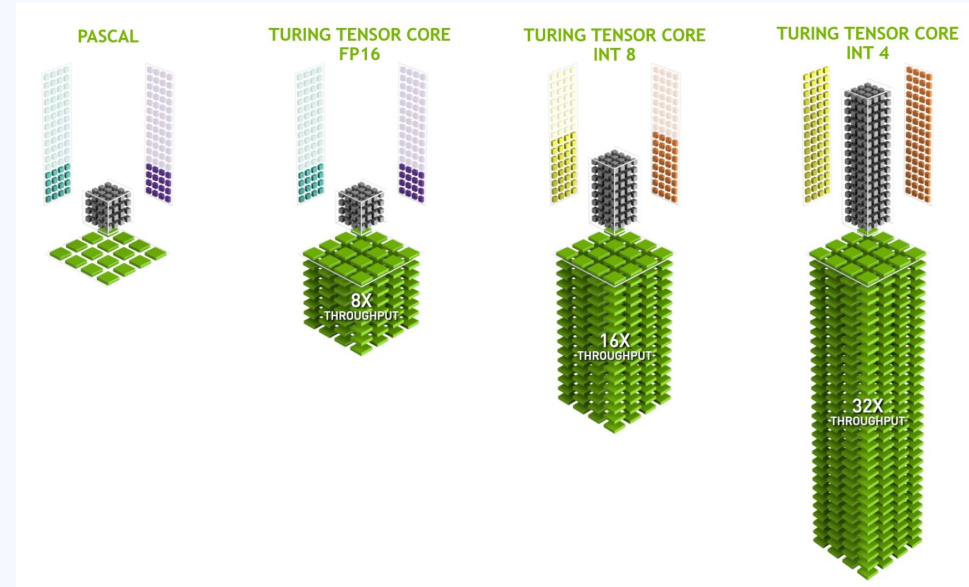
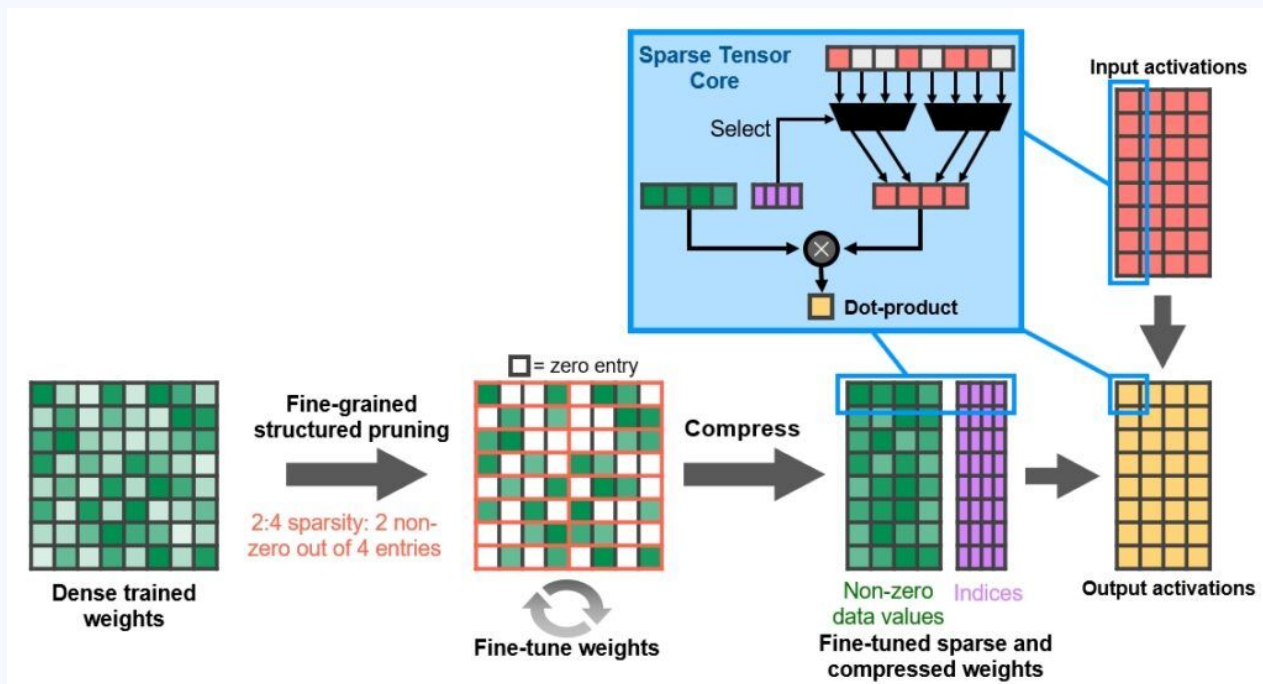


image source

Sparse acceleration

source



source

GA100 SM in Ampere



GPU memory model

Local memory for the state

Resides in device memory, not on chip... why?

Shared memory to communicate across threads

How?

```
__shared__ bool isDone;
...
if (threadIdx.x == 0)
    if (ecc) isDone = true;
...
__syncthreads();
if (isDone) {
    ...
}
```

