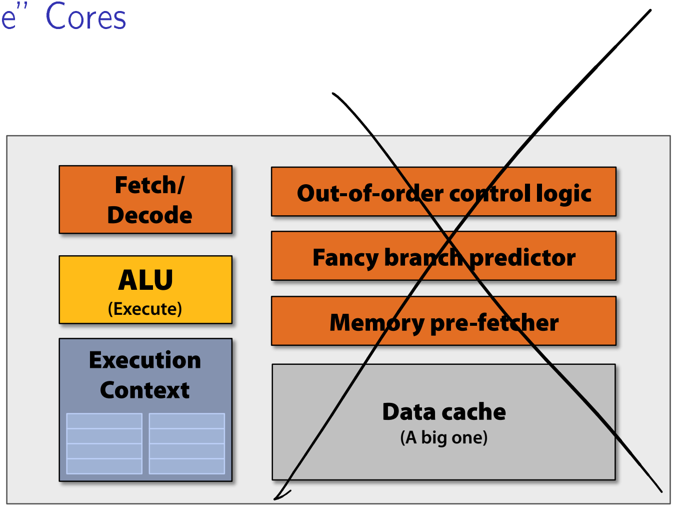
Annoncements:

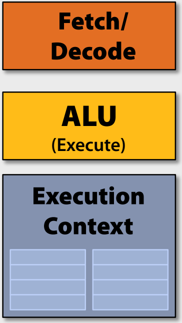
- Project submission logistics
- GPU-focused HW3: tonight / tomorrow

"CPU-style" Cores



[Fatahalian '08]

Slimming down

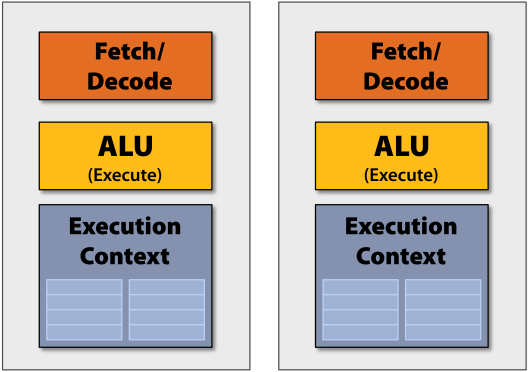


Idea #1:

Remove components that help a single instruction stream run fast

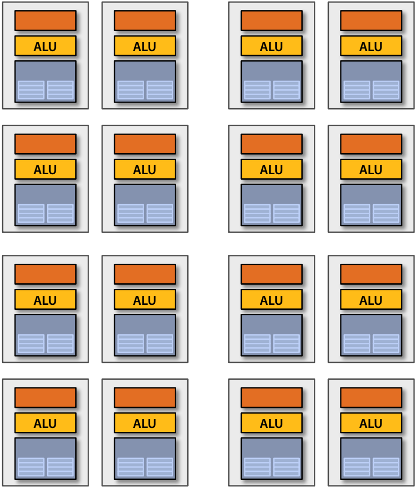
[Fatahalian '08]

More Space: Double the Number of Cores



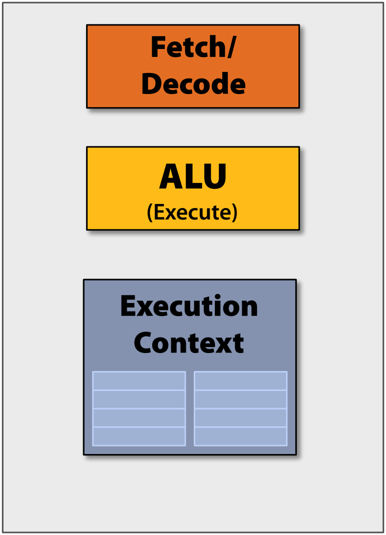
[Fatahalian '08]

Even more



[Fatahalian '08]

SIMD

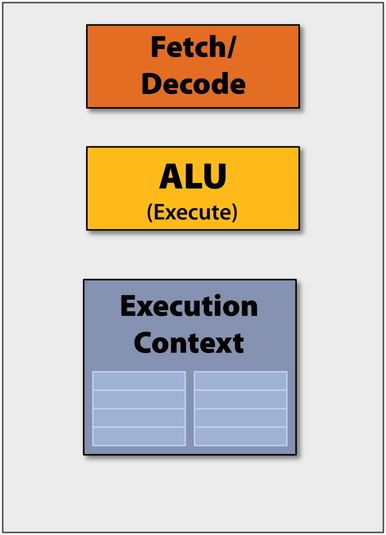


[Fatahalian '08]

Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

SIMD

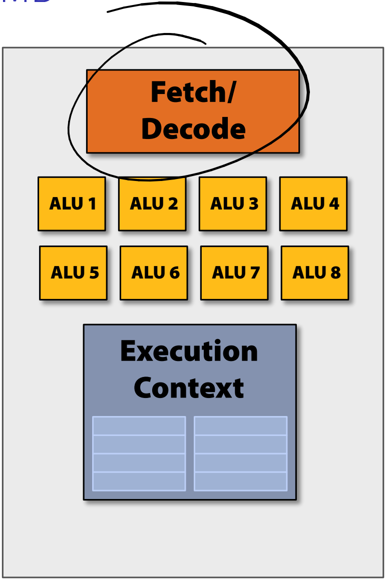


[Fatahalian '08]

Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

SIMD

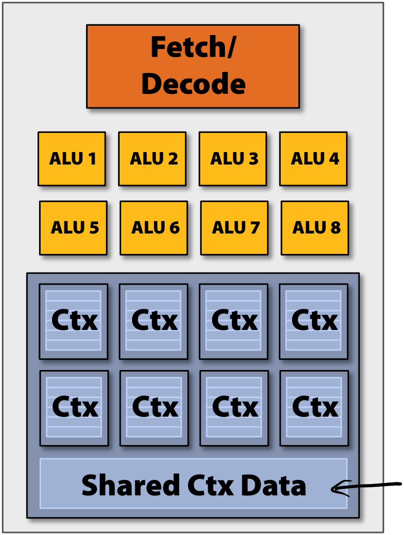


[Fatahalian ‘08]

Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

SIMD



scratch pad

[Fatahalian '08]

Idea #2: SIMD

Amortize cost/complexity of managing an instruction stream across many ALUs

Latency Hiding

- Latency (mem, pipe) hurts non-OOO cores
- Do *something* while waiting

What is the unit in which work gets scheduled on a GPU?

Vector

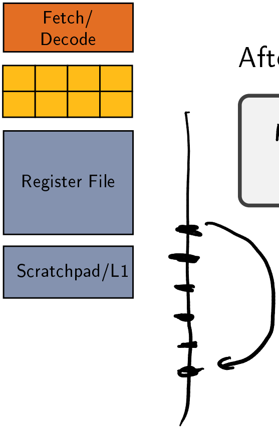
Nvidia: warp / wavefront

How can we keep busy?

- SMT
- ILP

Change in architectural picture?

Before:



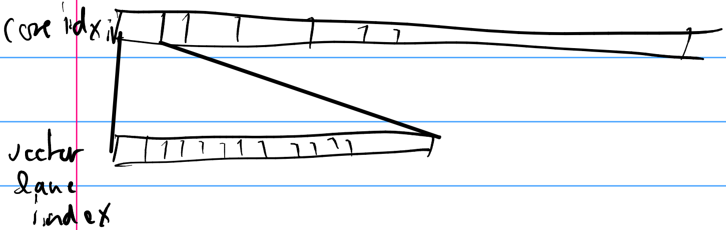
After:

more state space

GPUs: Core Architecture Ideas

Three core ideas:

- simpler / many of cores
- SIMD
- latency hiding through concurrency

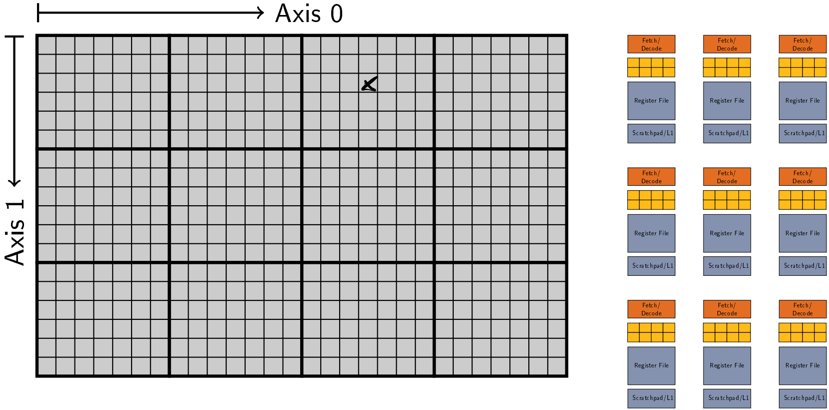


(core idx, vector lane idx)

$x = vec \% 17$

$y = vec // 17$

‘SIMT’

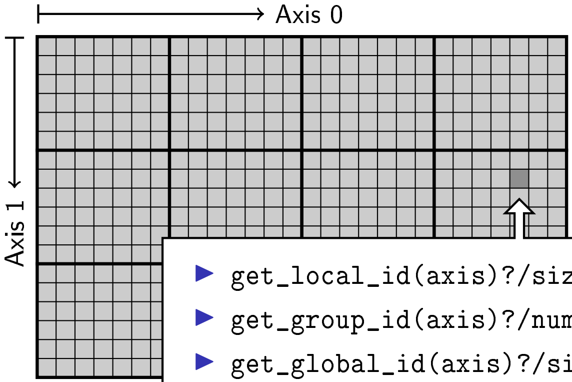

Axis 0
Axis 1
Fetch/ Decode
Fetch/ Decode
Fetch/ Decode
Register File
Register File
Register File
Scratchpad/L1
Scratchpad/L1
Scratchpad/L1
Fetch/ Decode
Fetch/ Decode
Fetch/ Decode
Register File
Register File
Register File
Scratchpad/L1
Scratchpad/L1
Scratchpad/L1
Fetch/ Decode
Fetch/ Decode
Fetch/ Decode
Register File
Register File
Register File
Scratchpad/L1
Scratchpad/L1
Scratchpad/L1

Wrangling the Grid

blockIdx.xyz

grpid 0: 3

1: 1



lid 0: 4

1: 1

threadIdx.xyz

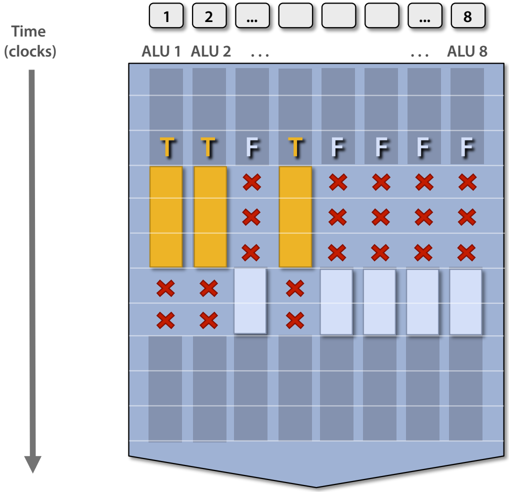
- get_local_id(axis)?/size(axis)?
- get_group_id(axis)?/num_groups(axis)?
- get_global_id(axis)?/size(axis)?

axis=0,1,2,...

Demo CL code

Demo: machabstr/Hello GPU

‘SIMT’ and Branches



```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

[Fatahalian ‘08]

GPU Abstraction: Core Model Ideas

How do these aspects show up in the model?

- View concrete counts as an implementation detail
    - SIMD lane
    - Core
    - Scheduling slot
- Program as if there are infinitely many of them
- Hardware division is expensive
  Make *n*D grids part of the model to avoid it
- Design the model to expose *extremely* fine-grain concurrency
  (e.g. between loop iterations!)
- Draw from the same pool of concurrency to hide latency

GPU Program 'Scopes'

Hardware	CL adjective	OpenCL	CUDA
SIMD lane	private	Work Item	Thread
SIMD Vector	—	Subgroup	Warp
Core	local	Workgroup	Thread Block
Processor	global	`NDRange`	Grid

GPU: Communication

What forms of communication exist at each scope?

- SIMD lanes: Vector shuffles
- Scratchpad + barrier, atomics + mem fences
- across workgroups: atomics + memfences

Can we just do locking like we might do on a CPU?

no: indep. fw progress required

GPU Programming Model: Commentary

- “Vector” / “Warp” / “Wavefront”
  - Important hardware granularity
  - Poorly/very implicitly represented
- What is the impact of reconvergence?