Lecture 12

Reduction

A few more CUDA issues

Sorting on GPU

Last time

- Coalescing
- Constant memory
- Texture memory
- OpenGL interoperability

Lab 5

Reduction

Sorting on the GPU

So what is lab 5 about?

Parallellize bitonic merge sort.

Start from a fairly parallel friendly implementation

Very easy to parallellize for small data sets (i.e. up to 512-1024)

Some more work to make it run with larger data

Not much use for shared memory in lab 4 and 5

Lab 6 is focused entirely on shared memory - but in OpenCL

Lecture questions

1) How can you efficiently compute the average of a dataset with CUDA?

2) In what way does bitonic sort fit the GPU better than many other sorting algorithms?

3) What is the reason to use pinned memory?

4) What problem does atomics solve?

Reduction

Parallelizing problems of limited parallel nature

Examples of reduction algorithms

Extracting small data from larger

- **Finding max or min**
- **Calculating median or average**
- **Histograms**

Common problems!

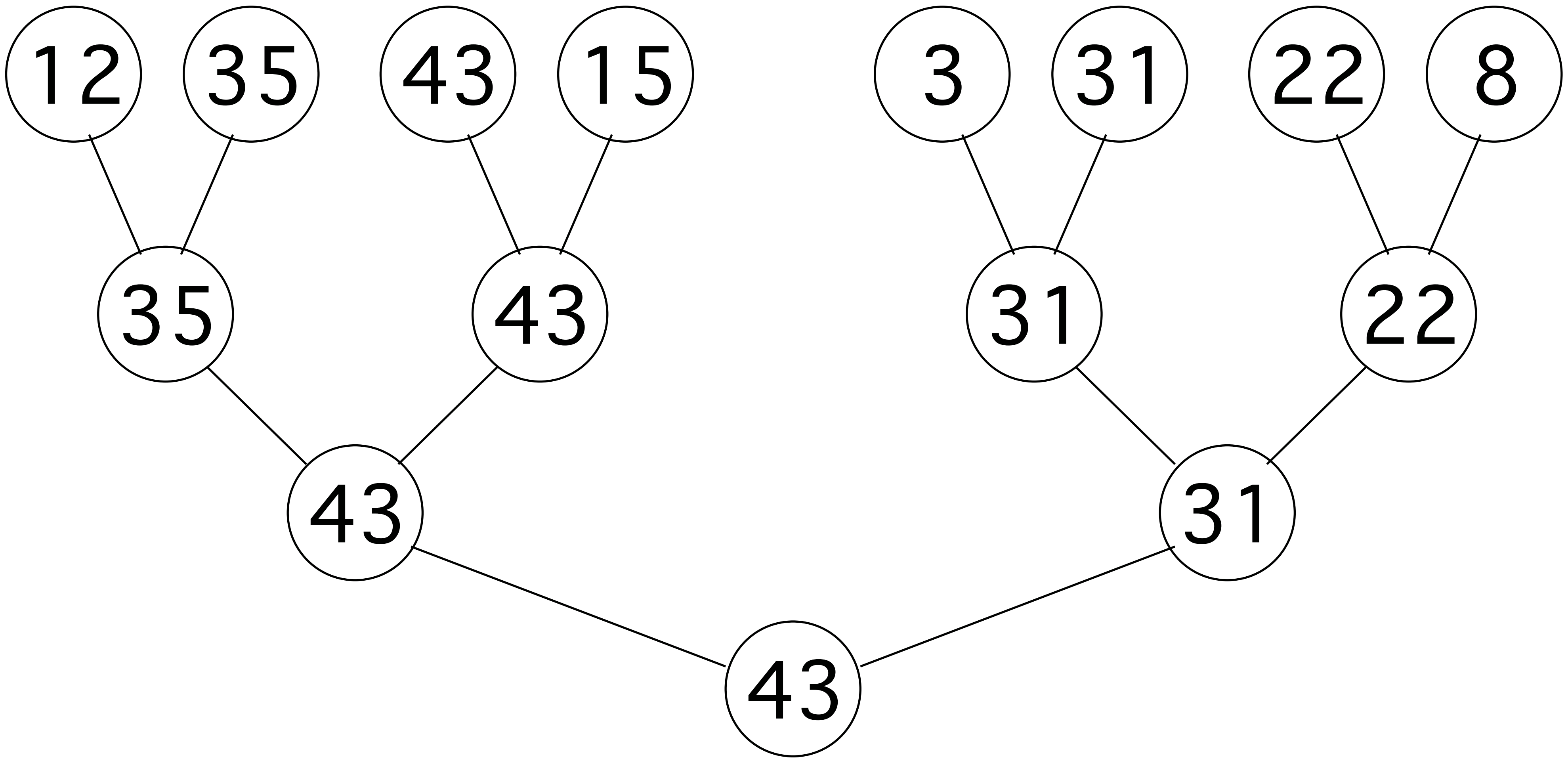
Sequentially trivial

Loop through data

Add/min/max, accumulate results

Fits badly in massive parallelism!

Tree-based approach



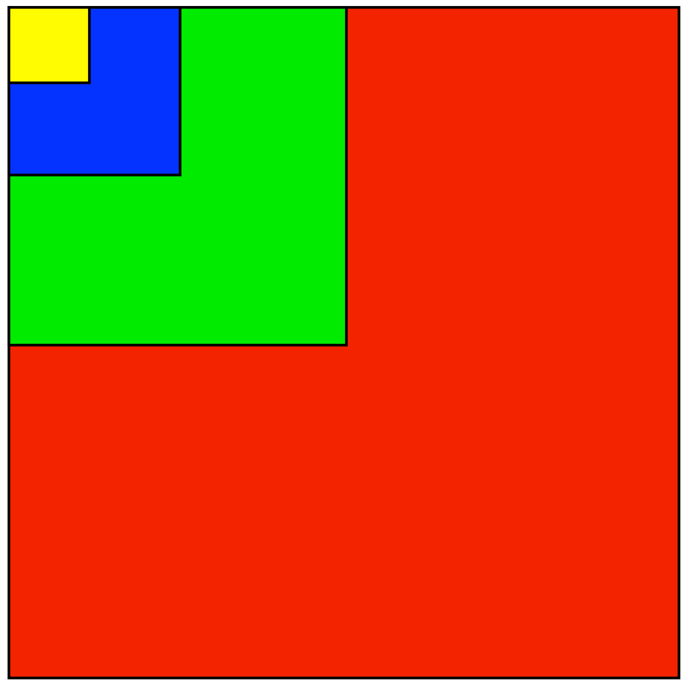
In 2D, typically 4-to-1 per level

Pyramid hierarchy

47	2	3	57	5	12	7	8
10	20	6	13	14	15	16	17
19	11	21	22	23	68	25	26
38	29	64	31	32	33	35	34
37	28	39	49	53	42	41	52
46	1	48	40	61	51	44	43
55	71	4	58	69	62	50	60
30	65	66	67	24	59	70	56

→

47	57	15	17
38	64	68	35
46	49	61	52
71	67	69	70

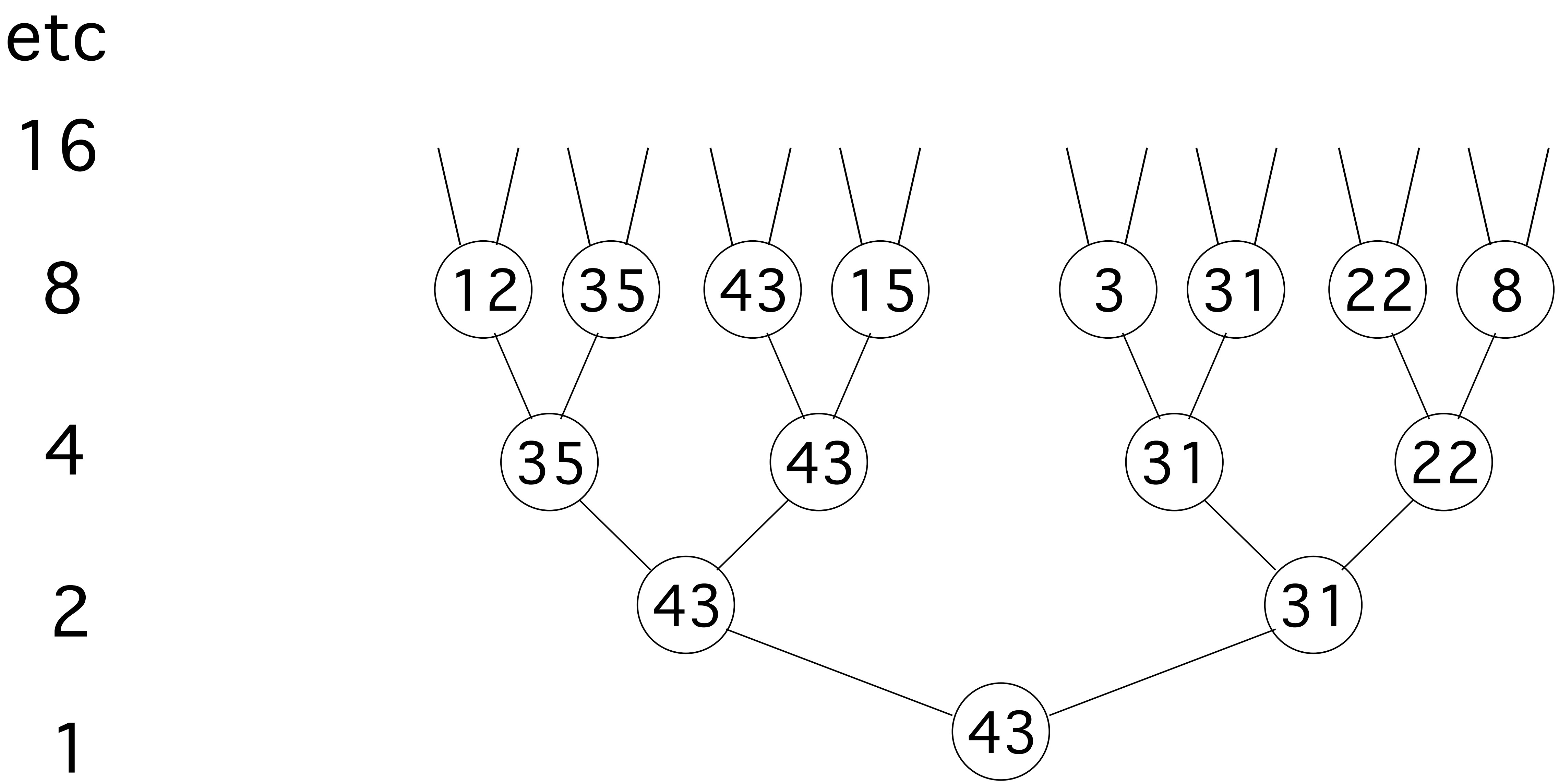


Tree-based approach

Each level parallel! Can be split onto large numbers of threads

but

the parallelism is reduced for each level, and the results need to be reorganized to a smaller number of threads!


etc
16
8
4
2
1
12
35
43
15
3
31
22
8
35
43
31
22
43
31
43

Multiple kernel runs for varying size!

For n = k downto 0 do

Launch 2^n kernels

Multiple levels can be merged into one - but not all of them!

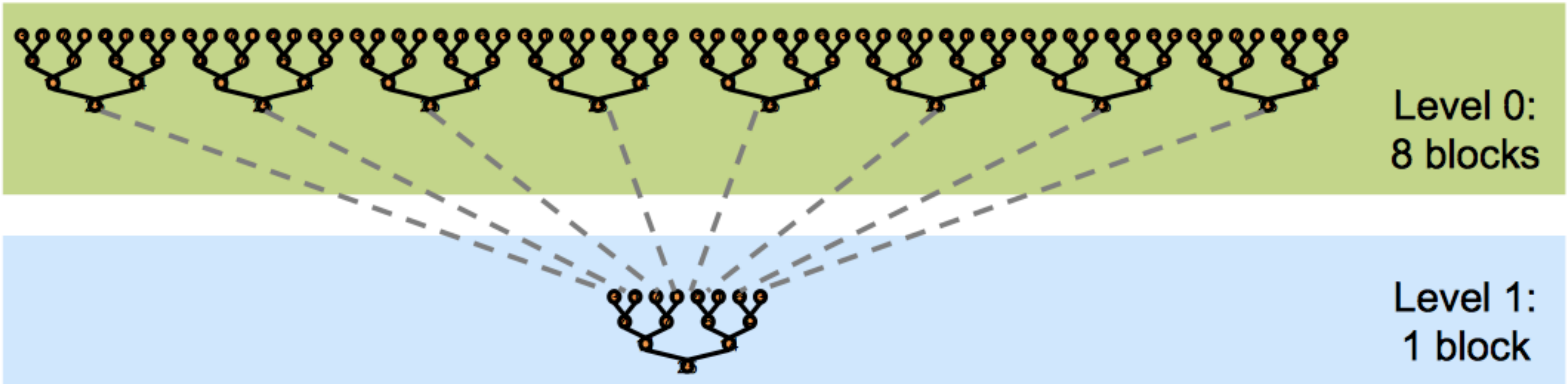
Important note: You can not synchronize between blocks!

Why?

- **Complex hardware**
- **Risk for deadlock between blocks that are not simultaneously active**

(Picture by Mark Harris, NVidia)

Multiple levels per kernel run for avoiding overhead



(Picture by Mark Harris, NVidia)

Many important optimizations:

- Avoid "if" statements, divergent branches
- Avoid bank conflicts in shared memory
- Loop unrolling to avoid loop overhead (classic old-style optimization!)

Huge speed difference reported by Harris

	Time (2^{22} ints)	Bandwidth	Step Speedup	Cumulative Speedup
Kernel 1: interleaved addressing with divergent branching	8.054 ms	2.083 GB/s		
Kernel 2: interleaved addressing with bank conflicts	3.456 ms	4.854 GB/s	2.33x	2.33x
Kernel 3: sequential addressing	1.722 ms	9.741 GB/s	2.01x	4.68x
Kernel 4: first add during global load	0.965 ms	17.377 GB/s	1.78x	8.34x
Kernel 5: unroll last warp	0.536 ms	31.289 GB/s	1.8x	15.01x
Kernel 6: completely unrolled	0.381 ms	43.996 GB/s	1.41x	21.16x
Kernel 7: multiple elements per thread	0.268 ms	62.671 GB/s	1.42x	30.04x

Conclusions:

- **Multiple kernel runs for varying problem size**
- **Multiple kernel runs for synchronizing blocks**
- **Optimizing matters! Not only shared memory and coalescing!**