Lecture 26:

Course Wrap Up & Project Presentation Tips

Parallel Computer Architecture and Programming
CMU 15-418, Spring 2013

Announcements

- Please fill out course evaluations

- Parallelism competition
  - Monday May 13th, 8:30-11:30am
  - Doherty Hall A302
  - Project reports due at 6pm that afternoon

- Parallelism competition judges
  - Ron Babich, NVIDIA Research
  - Dillon Sharlet, Intel

Today

- Exam 2 discussion
- Parallelism competition hints/guidelines
- Wrap up: a few final comments about parallelism

Exam 2 discussion
(no slides)

Project presentation tips

Presentation format

- Each group has five minutes to talk
  - Plus 1-2 minutes for questions from judges
- Presentation format is up to you: slides, live demo, etc.
  - Start with name(s) of students, title of project
  - Both students in a 2-person group should speak
- Present off your own laptop, or make arrangements with staff

Your #1 priority should be to be clear, rather than be comprehensive

(your project writeup is the place for completeness)

Everything you say should be understandable by someone in this class.
If you don't think the audience will understand, leave it out or change.
(spend the time saying something we will understand)

This will be much harder than it seems.

Here are some tips to help.

Tip #1: consider your audience

- **Everyone in the audience knows about parallel programming**
    - **CS terminology/concepts need not defined**
- **Most of the audience knows little-to-nothing about the specific application domain or problem you are trying to solve**
    - **Application-specific terminology should be defined or avoided**
- **The judges are trying to figure out the "most interesting" thing that you found out or did (your job is to define most interesting for them)**

Tip #2: basic setup

- What is the computation being performed (or system built)?
  - What are the inputs, and the outputs?
- Why does this problem stand to benefit from optimization?
  - “Real-time performance could be achieved”
  - “Researchers could run many more trials, changing how science is done”
  - “It is 90% of the execution time in this particular system”
- What are the fundamental challenges to optimization?
  - What turned out to be the hardest part of the problem?
  - This may involve describing a few key characteristics of the workload (e.g., overcoming divergence, increasing arithmetic intensity)

Tip #3: pick a focus

- In this class, different projects should stress different results
- Some projects may wish to show a flashy demo and describe how it works (proof by "it works")
- Other projects may wish to show a sequence of graphs (path of progressive optimization) and describe the optimization that took system from performance A to B to C
- Other projects may wish to clearly contrast parallel CPU vs. parallel GPU performance for a workload

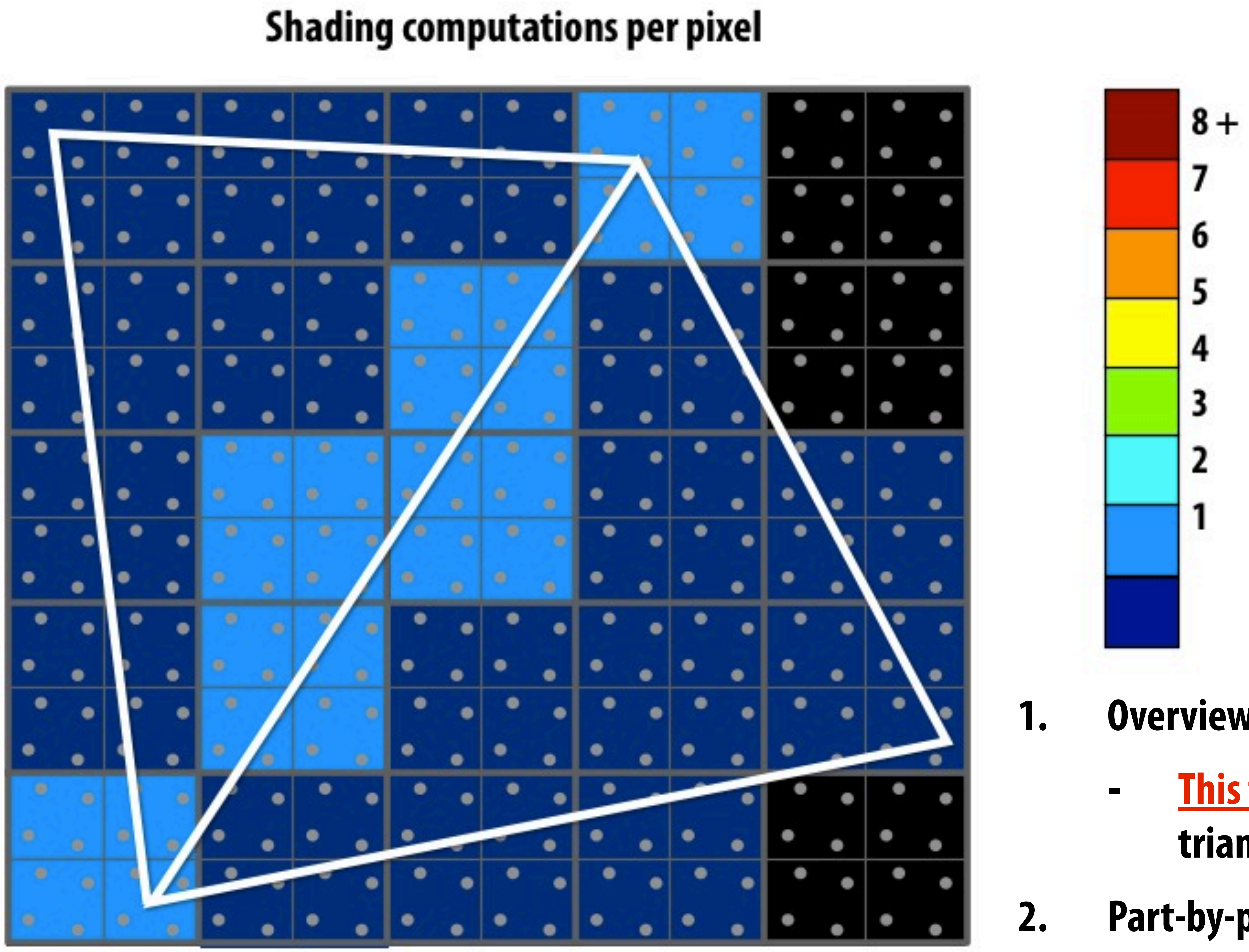
Your job is <u>not</u> to explain what you did, but to explain what you think we should know

Tip #4: how to describe a system

- **Start with the nouns (the key boxes in a diagram)**
  - Major components (processors, memories, interconnects, etc.)
  - Major entities (particles, neighbor lists, pixels, pixel tiles, features, etc.)
  - What is <u>state</u> in the system?

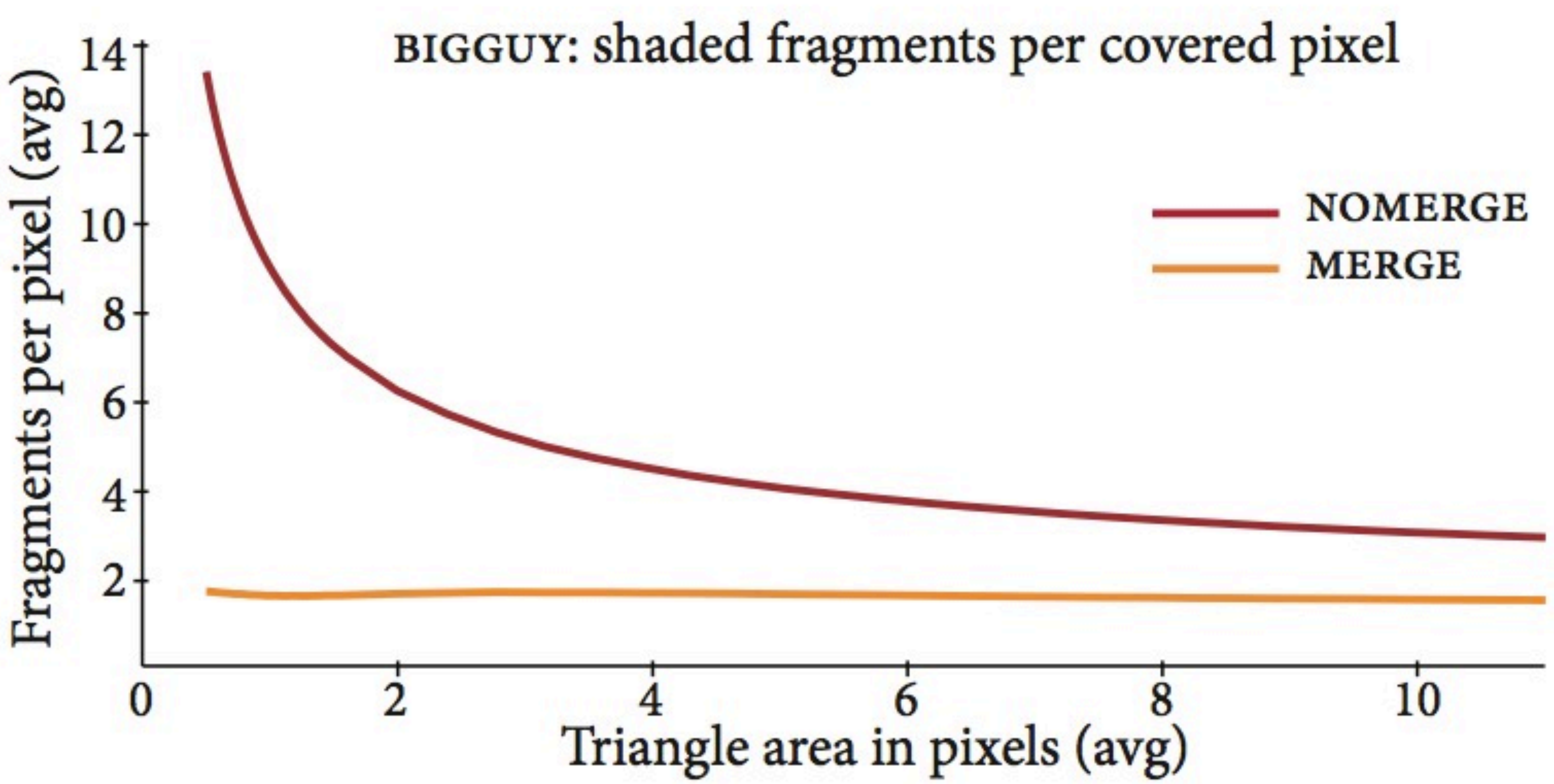
- **Then describe the verbs**
  - Operations that can be performed on the state (update particle positions, compute gradient of pixels, traverse graph, etc.)
  - Operations produce, consume, or transform entities

Tip #5: explain every figure or graph



1. Overview
   - This figures shows the effect of rasterizing two triangles in renderer
2. Part-by-part explanation:
   - Pixels are the boxes, they are colored according to the number triangles overlapping them ...
   - The sample points are given by the dots
3. Point: as you can see pixels along the boundary ...

Tip #5: explain every figure or graph



1. <u>In this graph</u>, the X-axis is ______.
2. The Y-axis is _______.
3. <u>If you look at</u> the left side ...
4. So the trend that you see means ...

Common error: only explaining the result (the point) of the graph

Tip #6: provide evidence that your code is fast
(or your system is efficient… in other words, that you did a good job)

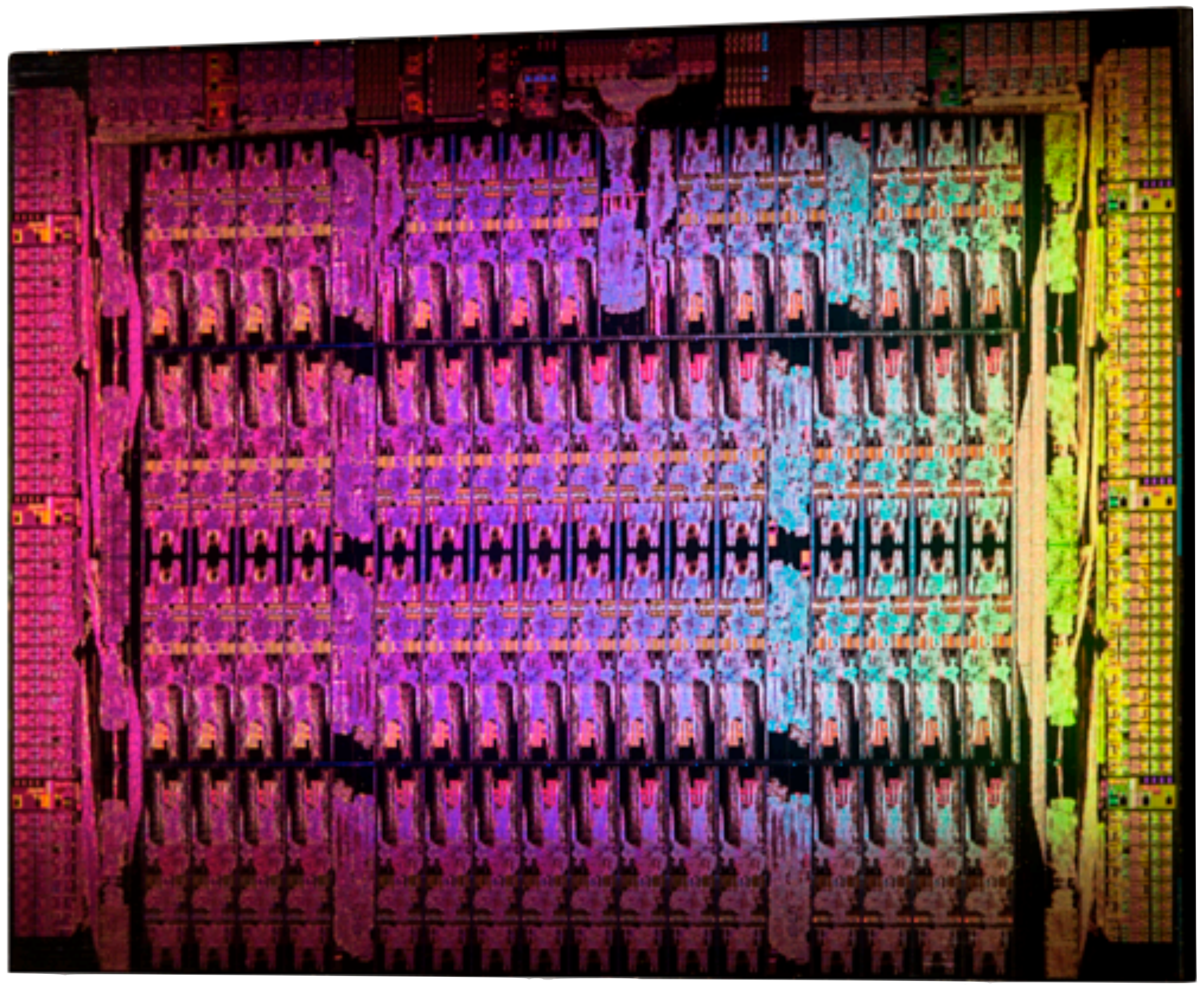
- **Compare against published results**
  - **"Our code is 10% faster than this publication"**
- **Determine a fraction of peak**
  - **"We achieve 80% of peak performance on this machine"**
- **Be truthful about comparisons between a CUDA implementation utilizing an entire GPU and single threaded, non-SIMD C program on a CPU (I'd rather not hear the conclusion: "GPU is 100x faster than the CPU".)**

Tip #7: practice the presentation

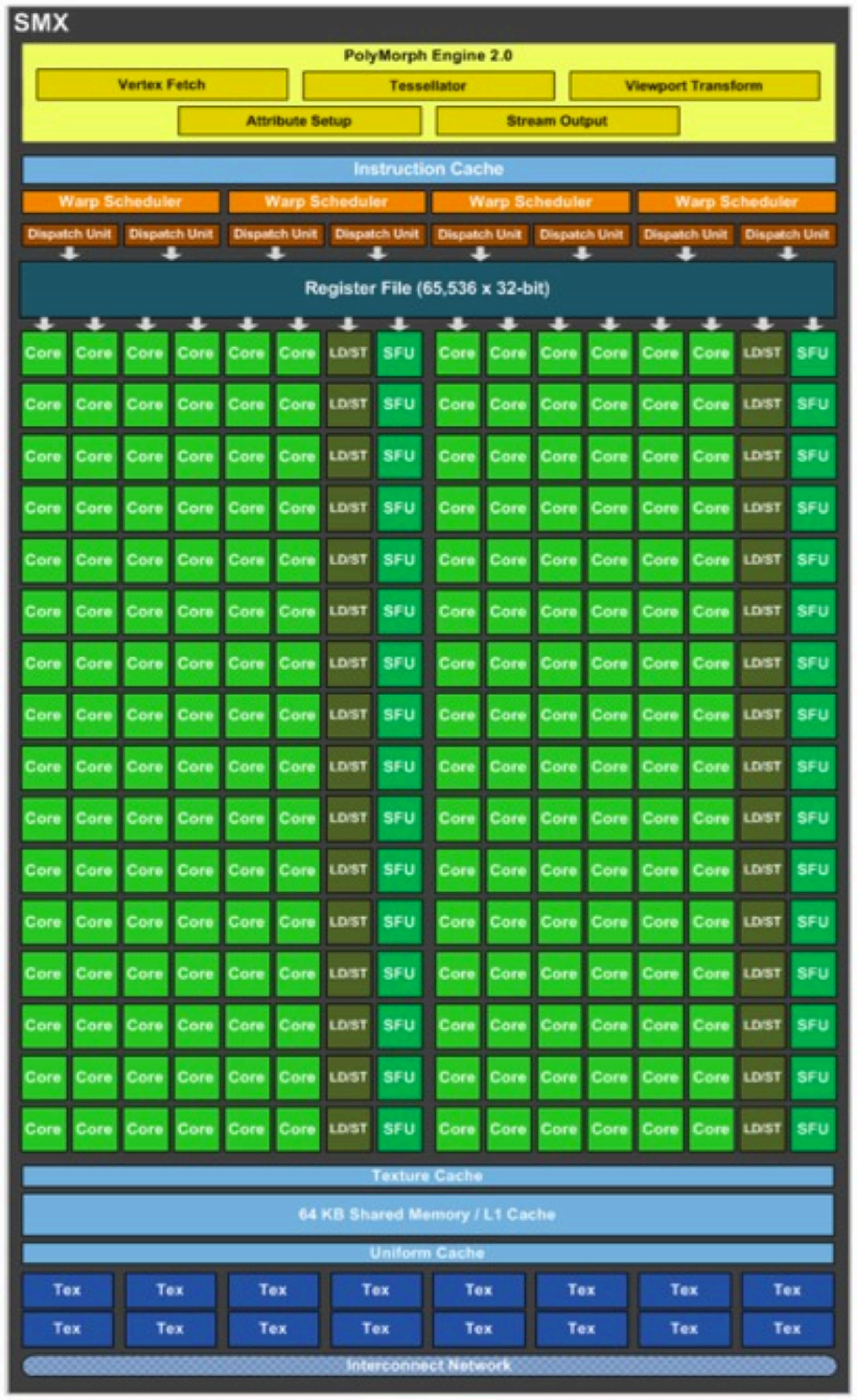
- Given the time constraints, you'll need to be smooth to say everything you want to say
- To be smooth you'll have to practice
- I hope you rehearse your demo or presentation a few times the night before (in front of a friend or two that's not in 418)

Course wrap up

For the foreseeable future, the primary way to obtain higher performance computing hardware is through increased parallelism and hardware specialization.



Intel Xeon Phi, 62 cores, 16-wide SIMD



NVIDIA Kepler Core (SMX)
32 wide SIMD, up to 2048 CUDA/core threads

Heterogeneous SoC

Today's software is surprisingly inefficient compared to the capability of modern machines

A lot of performance is currently left on the table (increasingly so as machines get more complex, and parallel processing capability grows)

Extracting this performance stands to provide a notable impact on many compute-intensive fields (or enable new applications of computing!)

Given current software programming systems and tools, understanding how a parallel machine works is important to achieving high performance.

A major challenge going forward is making it simpler for programmers to extract performance on these complex machines.

Efficiency matters more, not less!

Key issues

Identifying parallelism

(or conversely, identifying dependencies)

Scheduling parallelism

1. Achieving good workload balance

2. Overcoming communication constraints:

Bandwidth limits, dealing with latency, synchronization

Exploiting locality in data or execution = managing state!

We addressed these issues at many scales and in many contexts

Single chip, multi-core CPU
Multi-core GPU
CPU+GPU connected via bus
Large scale, multi-node supercomputer
Collection of machines in cloud

Other classes I teach

- 15-869 Visual Computing Systems (Fall 2013)
  - Parallel systems for "visual computing" applications (graphics, vision, computational photography).
  - Hoping for a mix of students
    - 15-418 gives you systems background
    - At least one of 15-462, 15-463, 15-385 gives you some applications background

Example topics: OpenGL pipeline implementation, camera imaging pipeline, systems for searching through big visual data

Beyond assignments and exams

- Come talk to me (or other professors) next fall about participating in research!
- Consider a senior thesis!
- Pitch a seed idea to Project Olympus
- Get involved with organizations like Hackathon or ScottyLabs

Kayvon’s final claim

It is far more difficult (and creative) to deliver on a “once in a few years” final project in 15-418 than it is take an extra class.

Ditto for pursuits like senior theses, independent study, starting a company...

Kayvon's final claim

The world rewards initiative.

The world rewards risk takers.

(Force yourself to get comfortable with risk taking and the possibility of occasional failure while at CMU.)

Many people are really smart.

Many people can work really hard.

Far fewer people develop the confidence and creativity to lead.

Thanks for being a great class!

Good luck on projects. Expectations are high.

See you a week from Monday!