# Programming Tips

Chris Rossbach and Calvin Lin

# Good Programming Habits

Important for sequential programming

More important for parallel programming

Why?

# Correctness

Why is it more difficult to produce correct parallel programs than correct serial programs?

What new complications arise?

- Asynchrony, race conditions, etc
- Explosion of state space
- Symptoms
  - Difficult to reproduce results
  - A larger number of edge cases
  - Issues of scale

# Performance

Why is it more difficult to produce efficient parallel programs than efficient serial programs?

What new complications arise?

- 90/10 rule does not apply (remember Amdahl's Law)

- Many more performance tradeoffs to consider
  - Communication vs. computation
  - Memory vs. parallelism
  - Parallelism vs. load balance
  - Overhead vs. granularity

- When have you succeeded?

# Good Programming Habits

Important for sequential programming

More important for parallel programming

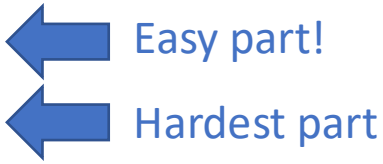
Why?

- It’s not that difficult to write parallel programs.
- It’s difficult to write parallel programs that are correct and efficient

# Lifetime of a Bug

Before we proceed, let's consider the lifetime of a bug

1. Create bug
2. Find bug
3. Fix bug
4. Test again



# Tip: Develop Incrementally

"Writing Solid Code" by Steve Maguire

At Microsoft, their projects were always behind schedule, even when they had their best programmers on a project.

They found that their projects always got behind in the "test and debug" step

1. Create bug
2. Find bug
3. Fix bug
4. Test again



Hardest part

# Tip: Develop Incrementally (cont)

Bad approach

Write a huge chunk of code, then test and debug it

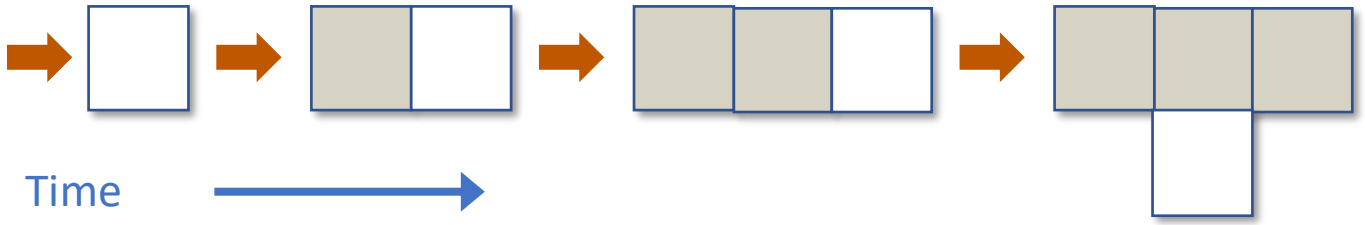



Harder to find new bugs

# Tip: Develop Incrementally (cont)

Good approach

Get small chunks of code working and tested before moving on to the next small chunk



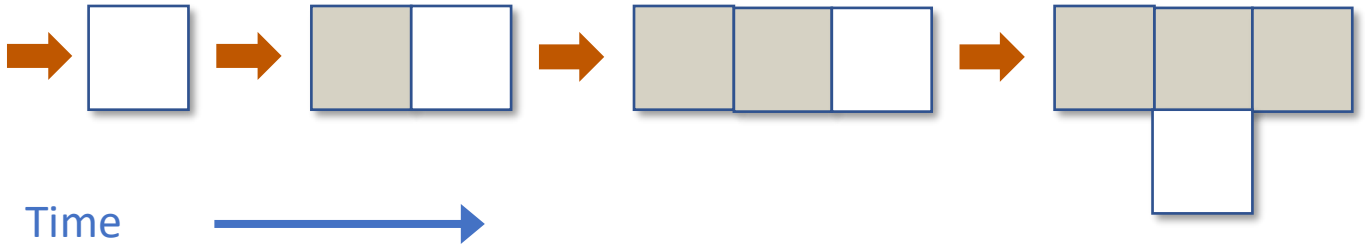

Advantage?

Easier to find new bugs

How does this idea translate to parallel programs?

# Tip: Develop Incrementally (cont)

Good approach




Work incrementally by separating parallel structure from sequential code

- Spawning of threads
- Communication and synchronization
- Partitioning the data

# Tip: Write Test Cases First

Also known as Test-Driven Development

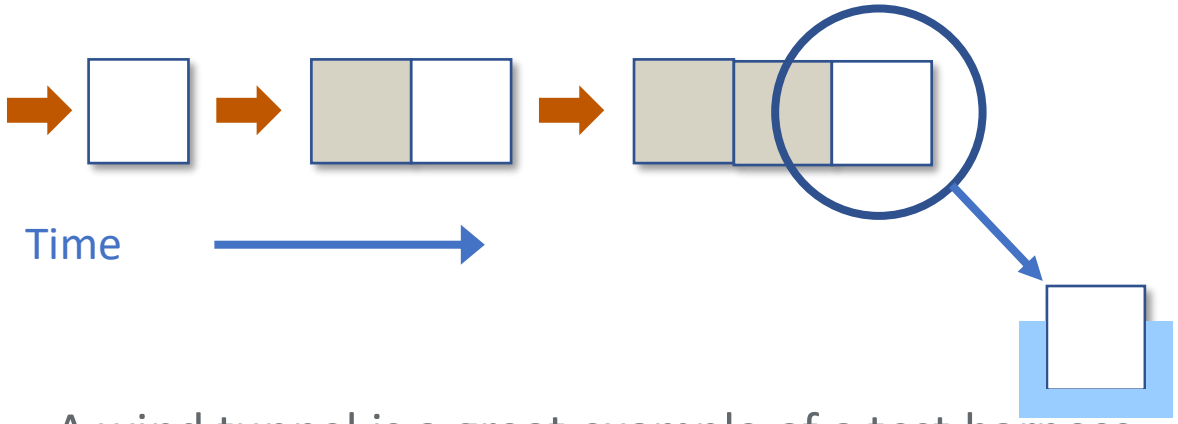
Advantages?

- Easier to think of corner cases
- Force programmer to think of entire solution
- Can help programmer understand the problem and the solution
- Serves as an informal functional specification of the code
- Almost complete removes the Find the Bug step

# Tip: Be Willing to Write Extra Code

Create tools for helping you understand your code and manage the complexity

Create a test harness for a module




A wind tunnel is a great example of a test harness

Instead of testing airplanes in the wild, put it in an environment that is simpler and that can be systematically controlled

# Tip: Be Willing to Write Extra Code (cont)

Other uses of extra code

- Instrument the code to help you understand what's happening
- Force the program to execute in a deterministic manner
- Systematically change inputs, number of threads, etc.
- Help you visualize the data (or parts of the data)

# Conclusion

These ideas are common sense

But they’re not always exercised

We encourage you to try these on your next programming assignment