A Tool-based Approach to Teaching Parallel and Concurrent Programming

Caitlin Sadowski

Joint work with:

* Tom Ball, Sebastian Burckhardt, Madan Musuvathi, Judith Bishop, and Shaz Qadeer (Microsoft Research)

* Stephen Toub (Microsoft)

* Ganesh Gopalakrishnan and Joeseph Mayo (University of Utah)

Me: Ph.D. Candidate at UC Santa Cruz

- Parallel and concurrent programming
  - Error detection tools
  - Mental models for debugging
  - PL-meets-HCI/education

- On the side
  - Real-time multiprocessor scheduling
  - Getting girls interested in CS through game programming

Talk Outline

- Seven principles of a parallel and concurrent programming curriculum
- Practical Parallel and Concurrent Programming (PPCP) course
- PPCP and the seven principles

Seven Principles

- Don’t discount the most popular model
- Start with abstractions
- Later, teach how to navigate abstractions
- No more matrix multiply
- Tool support is important
- Emphasize correctness
- Expose students to new research

1) Don't discount the most popular model

- Threads and shared memory all over
  - Want students to use parallelism
  - Need to be prepared for this model

  - Message passing also important

2) Start with Abstractions

- Parallel speedups for data parallel computations
  - Motivating
  - Independent loops

- DAG model

3) Later, teach how to navigate abstractions

- Need to look below abstractions to understand performance!
  - e.g. caching behaviour

- Still need high-level view
  - e.g. critical path

4) No more matrix multiply

- Appealing examples
- Visual & Relevant
    - Games
    - Graphics processing
    - Web-based analysis

5) Tool support is important

- Testing may not expose new concurrency errors
- Valuable skills for future
- Learn through experimentation

6) Emphasize correctness

- Multicore programming is hard
  - New bugs
  - Unpredictable bugs
  - Severe bugs

- What if one programmer does not understand the locking discipline?

7) Expose students to new research

- Cover the bases
  - What is “best” model?
  - Different problems, different paradigms

- Motivating for students!

Practical Parallel and Concurrent Programming

(PPCP)

The PPCP Course is ...

- **What**: 16 weeks (8 units) of material
  - Slides
  - Lecture notes
  - Quizzes, Labs, etc.
  - Sample programs and applications
  - Tests and tools

- **For Whom**: beginning graduates, senior undergraduates, a la carte
- **Where:** http://ppcp.codeplex.com

- **Dependencies**:
  - Visual Studio 2010 (includes .NET 4.0, C#, F#, TPL)

PPCP Currently

- Winter:

University of Washington
Computer Science & Engineering

- Now:

THE UNIVERSITY OF UTAH

http://eng.utah.edu/~cs5955

Please tell us about your experiences teaching with the Practical Parallel and Concurrent Programming (PPCP) course materials by completing the following survey. We value your insights and ideas in our efforts to produce quality teaching and learning materials. Thank you.

University:

Name:

What brought you to this website?

What level was the course you taught offered?

Select all that apply

- [ ] Lower-level undergraduate
- [ ] Upper-level undergraduate
- [ ] Undergraduate seminar
- [ ] Graduate Seminar
- [ ] Other

Did you teach an entire course using the material, or pick and choose units (a la carte)?

Select one of the following

- ○ Entire course
- ○ A la carte

Next

Mind your P's and C's

	Parallelism	Concurrency
Performance	Speedup	Responsiveness
Correctness	Atomicity, Determinism, Deadlock, Livelock, Linearizability, Data races, …	

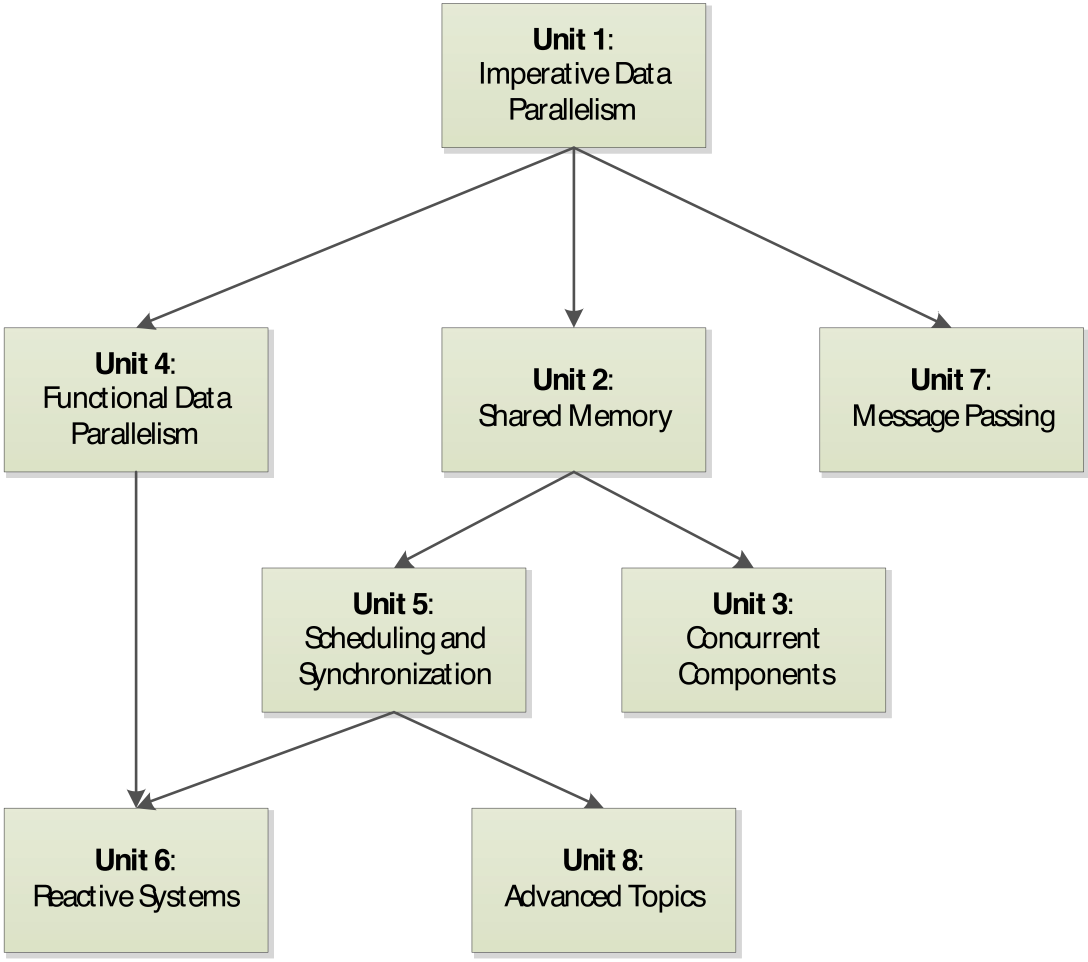
PPCP Units 1 – 4

- Unit 1: Imperative Data Parallel Programming
  - Data-intensive parallel programming (Parallel.For)
  - Concurrent Programming with Tasks
- Unit 2: Shared Memory
  - Data Races and Locks
  - Parallel Patterns
  - Cache Performance Issues
- Unit 3: Concurrent Components
  - Thread-Safety Concepts (Atomicity, Linearizability)
  - Modularity (Specification vs. Implementation)
- Unit 4: Functional Data Parallel Programming
  - Parallel Queries with PLINQ
  - Functional Parallel Programming with F#
  - GPU Programming with Accelerator

PPCP Units 5-8

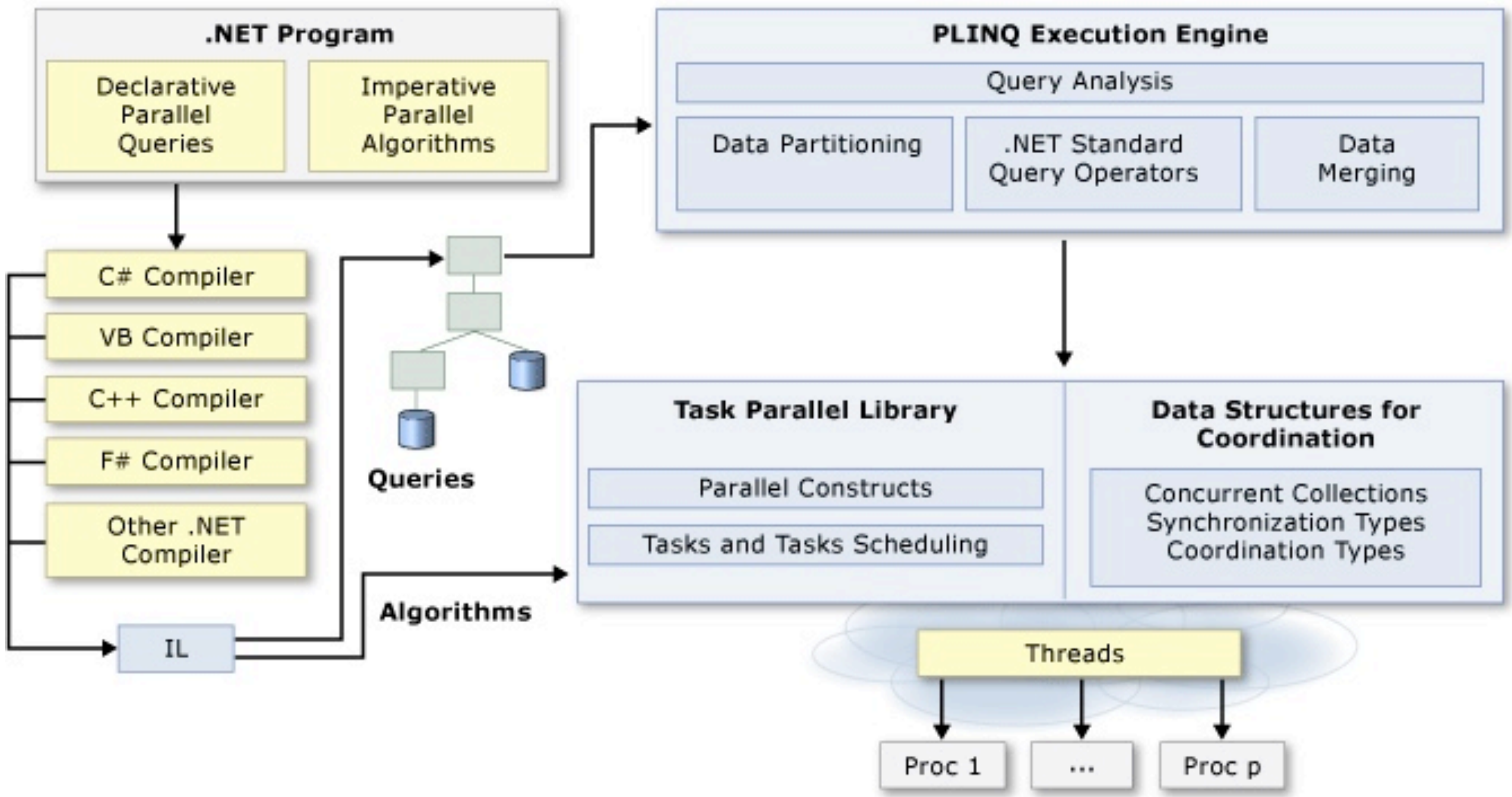
- Unit 5: Scheduling and Synchronization
  - From {tasks, DAGs} to {threads, processors}
  - Work-stealing
- Unit 6: Interactive/Reactive Systems
  - Asynchronicity
  - Event-based programming
- Unit 7: Message Passing
  - Conventional MPI-style programming
- Unit 8: Advanced Topics
  - Memory models, lock-free data structures, optimistic concurrency, Revisions

8 Units: A lot of flexibility



1) Don't discount the most popular model

- Breadth, but embedded in .NET



2) Abstraction-first….

- Start at high abstraction level (Unit 1)
  - Example: Parallel.For loops

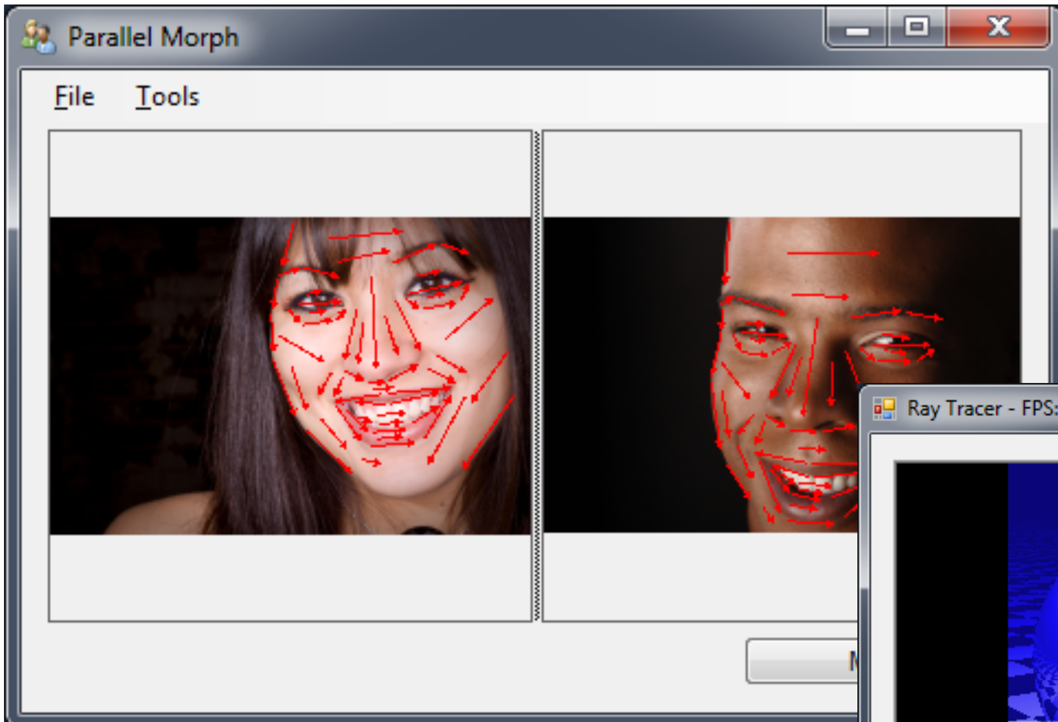
- Introduce patterns, not primitives (Unit 2)
  - Example: Producer-Consumer pattern

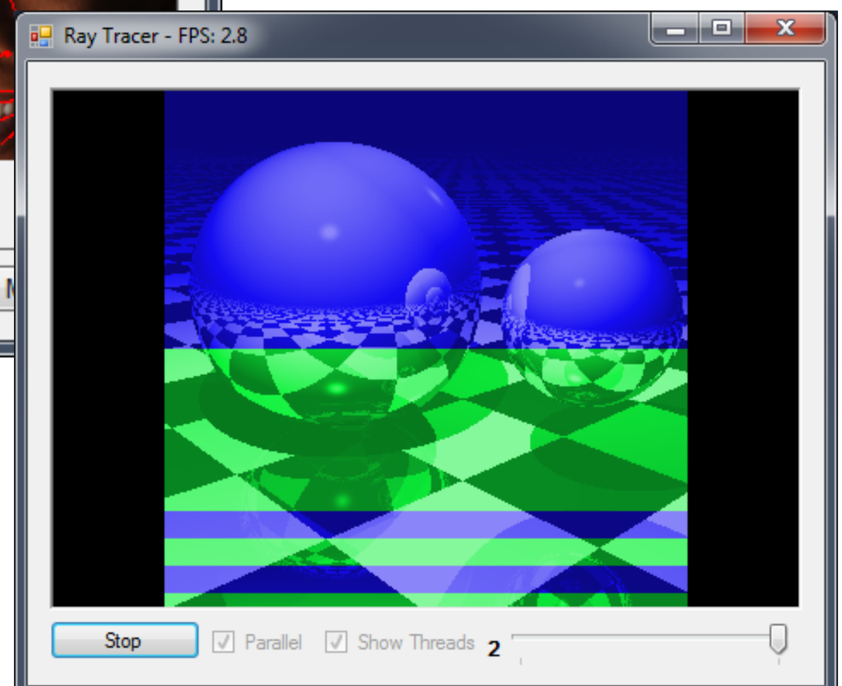
3) Abstraction-first…. then open them up

- Unit 2: Discuss data locality, cache coherence, false sharing, lock overheads, etc.

- Unit 5: the actual primitives
  - Example: threads, building a thread-safe buffer

4) No more matrix multiply: Parallel extensions samples



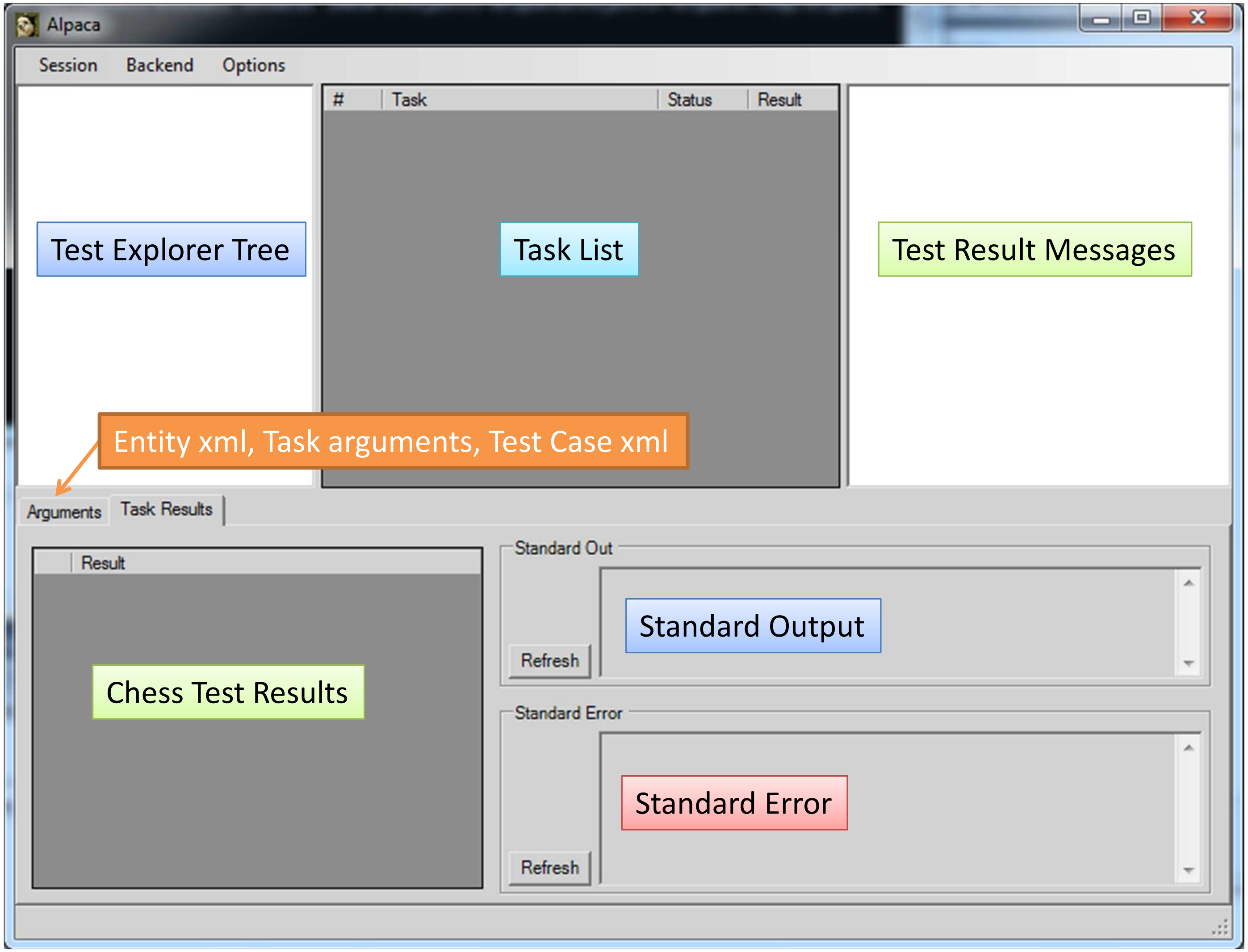


5) Tool-based approach to correctness & performance

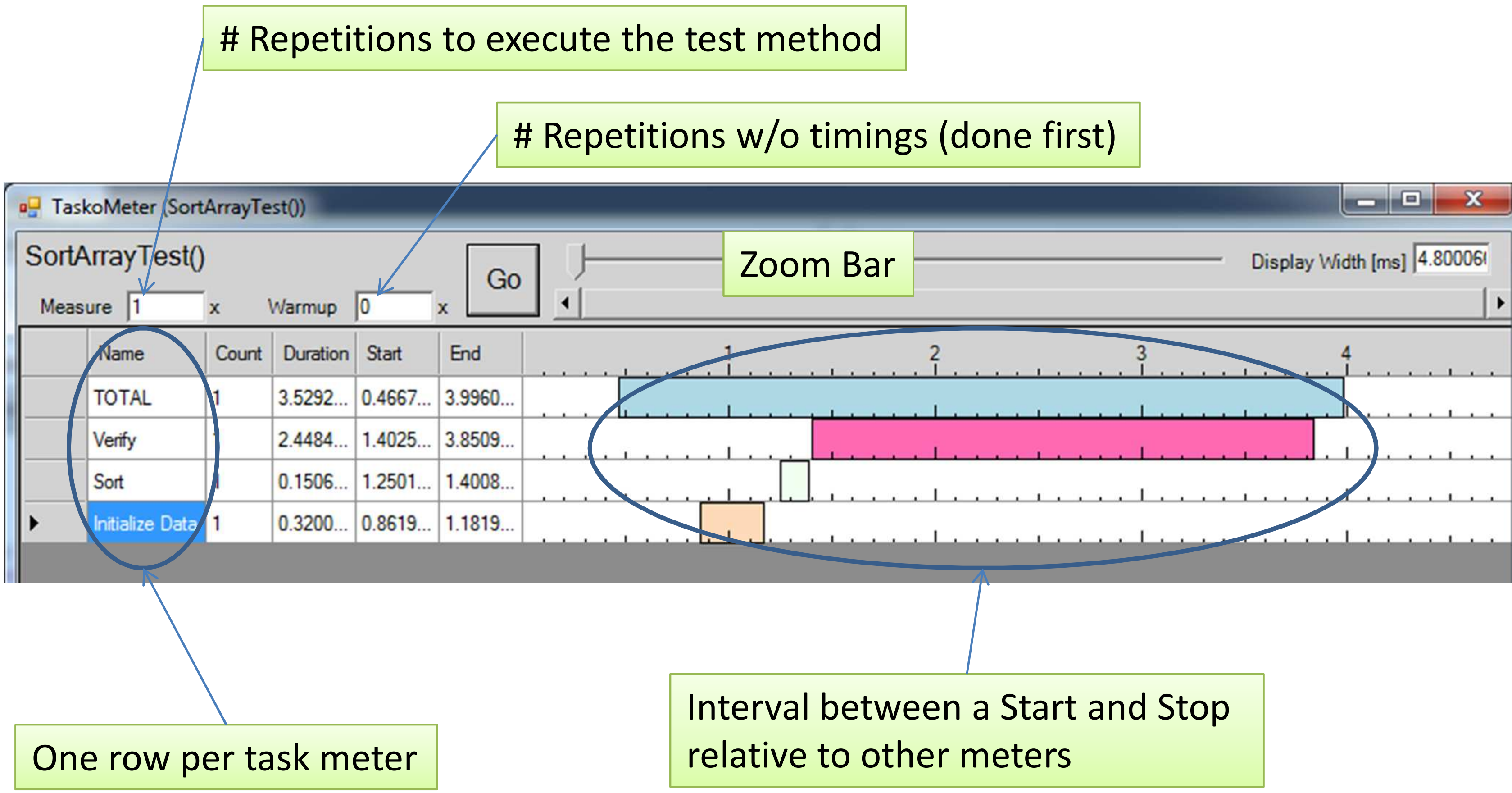
- Building understanding of correctness conditions through experimentation
  - Stateless Model Checking (with CHESS)
  - Concurrency Error Detection

- Emphasize unit testing, and performance testing
  - Alpaca tool
  - Taskometer

Alpaca (A lovely parallelism and concurrency analyzer)



Taskometer


Repetitions to execute the test method
Repetitions w/o timings (done first)
TaskoMeter (SortArrayTest())
SortArrayTest()
Go
Zoom Bar
Display Width [ms] 4.80006
Measure 1 x
Warmup 0 x
Name
Count
Duration
Start
End
1
2
3
4
TOTAL 1 3.5292... 0.4667... 3.9960...
Verify 2.4484... 1.4025... 3.8509...
Sort 0.1506... 1.2501... 1.4008...
Initialize Data 1 0.3200... 0.8619... 1.1819...
One row per task meter
Interval between a Start and Stop relative to other meters

Attribute-based testing

- [UnitTestMethod]
  - simply run this method normally, and report failed assertions or uncaught exceptions.
- [DataRaceTestMethod]
  - Run a few schedules (using CHESS tool) and detect data races.
- [ScheduleTestMethod]
  - Run all possible schedules of this method (with at most two preemptions) using the CHESS tool.
- [PerformanceTestMethod]
  - Like UnitTestMethod, but collect & graphically display execution timeline (showing intervals of interest)

6) Emphasize correctness

- Tool support
  - Checking for concurrency bugs

7) Expose students to new research

- CHESS
  - stateless model checking
- Code Contracts
  - lightweight specifications
- Accelerator
  - GPU data parallelism
- Reactive Extensions (Rx)
  - Asynchronous & event-based
- Revisions

Questions?

- http://ppcp.codeplex.com/

-