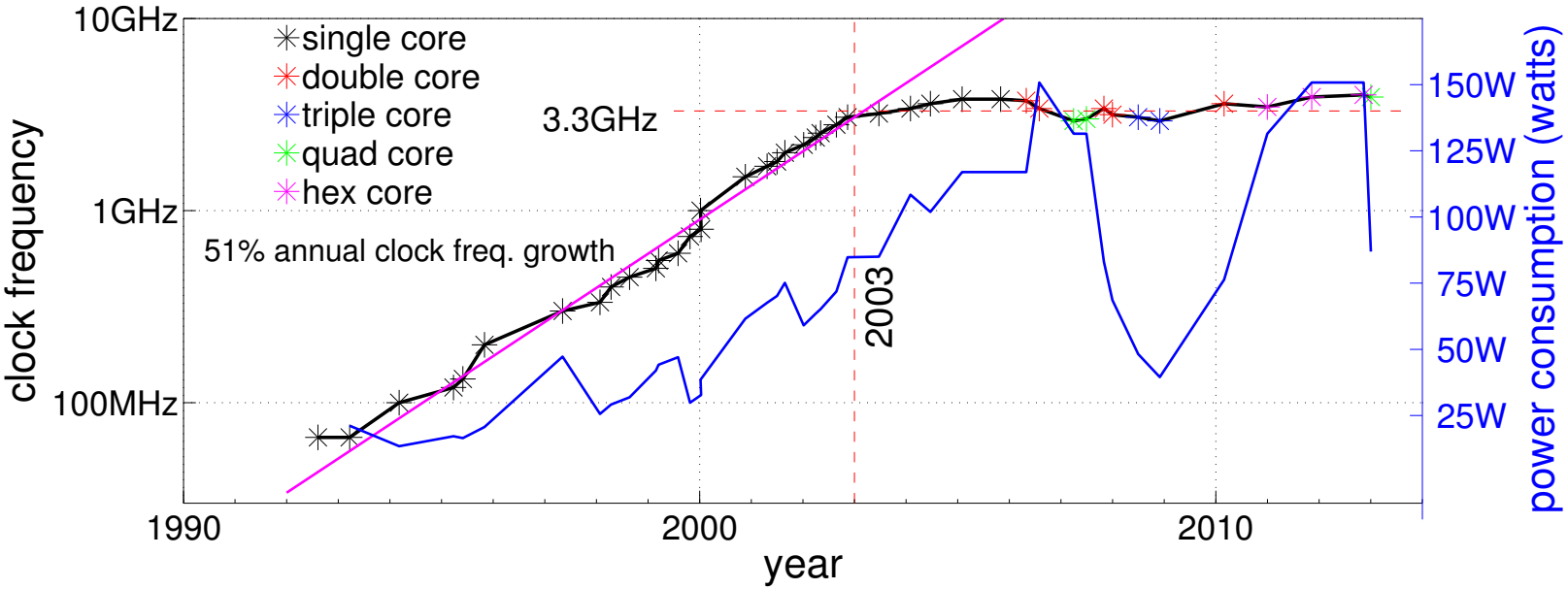
Parallel Computation

Mark Greenstreet

CpSc 418 – Jan. 4, 2017

Outline:

Why Parallel Computation Matters



Clock Speed and Power of Intel Processors vs. Year Released

[Wikipedia CPU-Power, 2011]

- In the good-old days, processor performance doubled roughly every 1.5 years.
- Single thread performance has seen small gains in the past 14 years.
  - Too bad. If it had, we would have 1000GHz CPUs today. ☺
- Need other ways to increase performance.

Why Sequential Performance Can't Improve (much)

Power

- CPUs with faster clocks use more energy per operation than slower ones.
- For mobile devices: high power limits battery life.
- For desktop computers and gaming consoles: cooling high-power chips requires expensive hardware.
- For large servers and clouds, the power bill is a large part of the operating cost.

More Barriers to Sequential Performance

- The memory bottleneck.
  - Accessing main memory (i.e. DRAM) takes hundreds of clock cycles.
  - But, we can get high bandwidth.
- Limited instruction-level-parallelism.
  - CPUs already execute instructions in parallel.
  - But, the amount of this ”free” parallelism is limited.
- Design complexity.
  - Designing a chip with 100 simple processors is **way** easier than designing a chip with one big processor.
- Reliability.
  - If a chip has 100 processors and one fails, there are still 99 good ones.
  - If a chip has 1 processor and it fails, then the chip is useless.
- See [Asanovic et al., 2006].

Parallel Computers

- Mobile devices:
  - multi-core to get good performance on apps and reasonable battery life.
  - many dedicated “accelerators” for graphics, WiFi, networking, video, audio, . . .
- Desktop computers
  - multi-core for performance
  - separate GPU for graphics
- Commercial servers
  - multiple, multi-core processors with shared memory.
  - large clusters of machines connected by dedicated networks.

Outline

- Why Does Parallel Computation Matter?
- Course Overview
  - Topics
  - Syllabus
  - The instructor and TAs
  - The textbook(s)
  - Grades

Homework:	35%	roughly one HW every two weeks
Midterm:	25%	March 1, in class
Final:	40%	
Mini-Assignments:	see description on slide 19	
Bug Bounties:	see description on slide 20	

  - Plagiarism – please don't
  - Learning Objectives
- Our First Parallel Program

Topics

- Parallel Architectures
- Parallel Performance
- Parallel Algorithms
- Parallel Programming Frameworks

Parallel Architectures

- **There isn't one, standard, parallel architecture for everything.** We have:
  - Multi-core CPUs with a shared-memory programming model. Used for mobile device application processors, laptops, desktops, and many large data-base servers.
  - Networked clusters, typically running linux. Used for web-servers and data-mining. Scientific supercomputers are typically huge clusters with dedicated, high-performance networks.
  - Domain specific processors
    - GPUs, video codecs, WiFi interfaces, image and sound processing, crypto engines, network packet filtering, and so on.
- As a consequence, **there isn't one, standard, parallel programming paradigm**.

Parallel Performance

The incentive for parallel computing is to do things that wouldn't be practical on a single processor.

- Performance matters.
- We need good models:
  - Counting operations can be very misleading – "adding is free."
  - Communication and coordination are often the dominant costs.
- We need to measure actual execution times of real programs.
  - There isn't a unified framework for parallel program performance analysis that works well in practice.
  - It's important to measure actual execution time and identify where the bottlenecks are.
- Key concepts with performance:
  - Amdahl's law, linear speed up, overheads.

Parallel Algorithms

- We'll explore some old friends in a parallel context:
  - Sum of the elements of an array
  - matrix multiplication
  - dynamic programming.
- And we'll explore some uniquely parallel algorithms:
  - Bitonic sort
  - mutual exclusion
  - producer consumer

Parallel Programming Frameworks

- Erlang: functional, message passing parallelism
  - Avoids many of the common parallel programming errors: races and side-effects.
    You can write Erlang programs with such bugs, but it takes extra effort (esp. for the examples we consider).
  - Allows a simple presentation of many ideas.
  - But it's slow, for many applications, when compared with C or C++.
  - OTOH, it finds real use in large-scale distributed systems.
- CUDA: your graphics card is a super-computer
  - Excellent performance on the "right" kind of problem.
  - The data-parallel model is simple, and useful.

Syllabus

- January: Erlang

  Jan. 4– 9: Course overview, intro. to Erlang programming.
  Jan. 11–18: Parallel programming in Erlang, reduce and scan.
  Jan. 20–27: Parallel architectures
  Jan. 29–Feb. 6: Performance analysis

- February: Erlang, Midterm

  Feb. 8–17: Sorting
  Feb. 20–19: Midterm break.
  Feb. 27: Midterm Review
  Mar. 1: Midterm

- March: CUDA and other topics

  Mar. 3–10: Introduction to SIMD and CUDA.
  Mar. 13–24: More algorithms in CUDA (and a bit of Erlang)
  Mar. 27–Apr. 6: Map-Reduce, Mutual Exclusion, & More Fun.

- Note: I'll make adjustments to this schedule as we go.

Administrative Stuff – Who

- The instructors
  - **Mark Greenstreet,** mrg@cs.ubc.ca
    - ICCS 323, (604) 822-3065
    - Office hours: Tuesdays, 1pm – 2:30pm, ICCS 323
  - **Ian Mitchell,** mitchell@cs.ubc.ca
    - ICCS 217, (604) 822-2317
    - Office hours: Fridays, 12noon – 1pm, ICCS 217
- The TAs

  Devon Graham, drgraham@cs.ubc.ca
  Chenxi Liu, chenxil@cs.ubc.ca
  Carolyn Shen, shen.carolyn@gmail.com
  Brenda Xiong, krx.sky@gmail.com
- Course webpage: `http://www.ugrad.cs.ubc.ca/~cs418`.
- Online discussion group: on piazza.

Textbook(s)

- For Erlang: *Learn You Some Erlang For Great Good*, Fred Hébert,
  - Free! On-line at `http://learnyousomeerlang.com`.
  - You can buy the dead-tree edition at the same web-site if you like.
- For CUDA: *Programming Massively Parallel Processors: A Hands-on Approach* (2^{nd} **or** 3^{rd} ed.), D.B. Kirk and W-M.W. Hwu.
  - Please get a copy by late February – I'll assign readings starting after the midterm. It's available at amazon.ca and many other places.
- I'll hand-out copies of some book chapters:
  - *Principles of Parallel Programming* (chap. 5), C. Lin & L. Snyder – for the reduce and scan algorithms.
  - *An Introduction to Parallel Programming* (chap. 2), P.S. Pacheco – for a survey of parallel architectures.
  - Probably a few journal, magazine, or conference papers.

Why so many texts?

- There isn't one, dominant parallel architecture or programming paradigm.
- The Lin & Snyder book is a great, paradigm independent introduction,
- But, I've found that descriptions of real programming frameworks lack the details that help you write real code.
- So, I'm using several texts, but
  - You only have to buy one! ☺

Grades

Homework:	35%	roughly one assignment every two weeks
Midterm:	25%	March 1, in class
Final:	40%	
Mini-Assignments:	see description on slide 19	
Bug Bounties:	see description on slide 20	

Homework

- Collaboration policy
  - You are welcome and encouraged to discuss the homework problems with other students in the class, with the TAs and me, and find relevant material in the text books, other book, on the web, etc.
  - You are expected to work out your own solutions and write your own code. Discussions as described above are to help understand the material. Your solutions must be your own.
  - You must properly cite your collaborators and any outside sources that you used. You don't need to cite material from class, the textbooks, or meeting with the TAs or instructor. See slide 22 for more on the plagiarism policy.
- Late policy
  - Each assignment has an "early bird" date before the main date. Turn in you assignment by the early-bird date to get a 5% bonus.
  - **No late homework accepted.**

Exams

- Midterm, in class, on March 1.
- Final exam will be scheduled by the registrar.
- Both exams are open book, open notes, open homework and solutions – open anything printed on paper.
  - You can bring a calculator.
  - No communication devices: laptops, tablets, cell-phones, etc.

Mini-Assignments

- Mini-assignments
  - Worth 20% of points missed from HW and exams.
    - If your raw grade is 90%, you can get at most 2% from the minis. Missing one or two isn't a big deal.
    - If your raw grade is 70%, you can get 6% from the minis. This can move your letter grade up a notch (e.g. C+ to B−).
    - If your raw grade is 45%, you can get up to 11% from the minis. Do the mini-assignments – I hate turning in failing grades.
  - The first is at

    http://www.ugrad.cs.ubc.ca/~cs418/2016-2/mini/1/mini1.pdf,

    and due Jan. 9.
  - **If you are on the course waitlist,** we will select from the students who submit acceptable solutions to **Mini Assignment 1** to fill any slots that open up.

Bug Bounties

- If I make a mistake when stating a homework problem, then the first person to report the error gets extra credit.
  - If the error would have prevented solving the problem, then the extra credit is the same as the value of the problem.
  - Smaller errors get extra credit in proportion to their severity.
- Likewise, bug bounties are awarded (as homework extra credit) for finding errors in mini-assignments, lecture slides, the course web-pages, code I provide, etc.
- The midterm and final have bug bounties awarded in midterm and final exam points respectively.
- If you find an error, report it.
  - Suspected errors in homework, lecture notes, and other course materials should be posted to piazza.
  - The first person to post a bug gets the bounty.
  - Bug-bounties reward you for looking at the HW when it first comes out, and not waiting until the day before it is due.

Grades: the big picture

$$
\begin{aligned}
\mathit{RawGrade} &= 0.35 * \mathit{HW} + 0.25 * \mathit{MidTerm} + 0.40 * \mathit{Final} \\
\mathit{MiniBonus} &= 0.20 * (1 - \min(\mathit{RawGrade}, 1)) * \mathit{Mini} \\
\mathit{BB} &= 0.35 * \mathit{BB}_{\mathit{HW}} + 0.25 * \mathit{BB}_{\mathit{MT}} + 0.40 * \mathit{BB}_{\mathit{FX}} \\
\mathit{CourseGrade} &= \min(\mathit{RawGrade} + \mathit{MiniBonus} + \mathit{BB}, 1) \times 100\%
\end{aligned}
$$

Plagiarism

- I have a very simple criterion for plagiarism:
  Submitting the work of another person, whether that be another student, something from a book, or something off the web and representing it as your own is plagiarism and constitutes academic misconduct.
- If the source is clearly cited, then it is not academic misconduct.
  If you tell me "This is copied word for word from Jane Foo's solution" that is not academic misconduct. It will be graded as one solution for two people and each will get half credit. I guess that you could try telling me how much credit each of you should get, but I've never had anyone try this before.
- I encourage you to discuss the homework problems with each other.
  If you're brainstorming with some friends and the key idea for a solution comes up, that's OK. In this case, add a note to your solution that lists who you collaborated with.
- More details at:
  - http://www.ugrad.cs.ubc.ca/~cs418/plagiarism.html
  - http://learningcommons.ubc.ca/guide-to-academic-integrity/

Learning Objectives (1/2)

- Parallel Algorithms
  - Familiar with parallel patterns such as reduce, scan, and tiling and can apply them to common parallel programming problems.
  - Can describe parallel algorithms for matrix operations, sorting, dynamic programming, and process coordination.
- Parallel Architectures
  - Can describe shared-memory, message-passing, and SIMD architectures.
  - Can describe a simple cache-coherence protocol.
  - Can identify how communication latency and bandwidth are limited by physical constraints in these architectures.
  - Can describe the difference between bandwidth and inverse latency, and how these impact parallel architectures.

Learning Objectives (2/2)

- Parallel Performance
  - Understands the concept of “speed-up”: can calculate it from simple execution models or measured execution times.
  - Can identify key bottlenecks for parallel program performance including communication latency and bandwidth, synchronization overhead, and intrinsically sequential code.
- Parallel Programming Frameworks
  - Can implement simple parallel programs in Erlang and CUDA.
  - Can describe the differences between these paradigms.
  - Can identify when one of these paradigms is particularly well-suited (or badly suited) for a particular application.

Lecture Outline

- Why Does Parallel Computation Matter?
- Course Overview
- Our First Parallel Program
  - Erlang quick start
  - Count 3s
  - Counting 3's in parallel
    - The root process
    - Spawning worker processes
    - The worker processes
    - Running the code

Erlang Intro – very abbreviated!

- Erlang is a functional language:
  - Variables are given values when declared, and the value never changes.
  - The main data structures are lists, `[Head | Tail]`, and tuples (covered later).
  - Extensive use of pattern matching.
- The source code for the examples in this lecture is available at:

  http://www.ugrad.cs.ubc.ca/~cs418/2016-2/lecture/01-06/code.html

Lists

- `[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]` is a list of 10 elements.
- If `L1` is a list, then `[0 | L1]` is the list obtained by prepending the element `0` to the list `L1`. In more detail:

```
1> L1 = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100].
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
2> L2 = [0 | L1].
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
3> L3 = [0 , L1].
[0, [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]]
```

- Of course, we traverse a list by using recursive functions:

Lists traversal example: sum

```
sum(List) ->
    if (length(List) == 0) -> 0;
       (length(List) > 0)  -> hd(List) + sum(tl(List))
    end.
```

- `length(L)` returns the number of elements in list `L`.
- `hd(L)` returns the first element of list `L` (the head), and throws an exception if `L` is the empty list.
  `hd([1, 2, 3]) = 1`. `hd([1]) = 1` as well.
- `tl(L)` returns the list of all elements after the first (the tail).
  `tl([1, 2, 3]) = [2, 3]`. `tl([1]) = []`.
- See `sum_wo_pm` ("sum without pattern matching") in simple.erl

Pattern Matching – first example

We can use Erlang's pattern matching instead of the `if` expression:

```
sum([]) -> 0;
sum([Head | Tail]) -> Head + sum(Tail).
```

- `sum([Head | Tail])` matches any non-empty list with `Head` being bound to the value of the first element of the list, and `Tail` begin bound to the list of all the other elements.
- More generally, we can use patterns to identify the different cases for a function.
- This can lead to very simple code where function definitions follow the structure of their arguments.
- See `sum` in simple.erl

Count 3's: a simple example

Given an array (or list) with `N` items, return the number of those elements that have the value `3`.

```
count3s([]) -> 0;
count3s([3 | Tail]) -> 1 + count3s(Tail);
count3s([_Other | Tail]) -> count3s(Tail).
```

- We'll need to put the code in an erlang module. See `count3s` in count3s.erl for the details.
- To generate a list of random integers, count3s.erl uses the function `rlist(N, M)` from course Erlang library that returns a list of `N` integers randomly chosen from `1..M`.

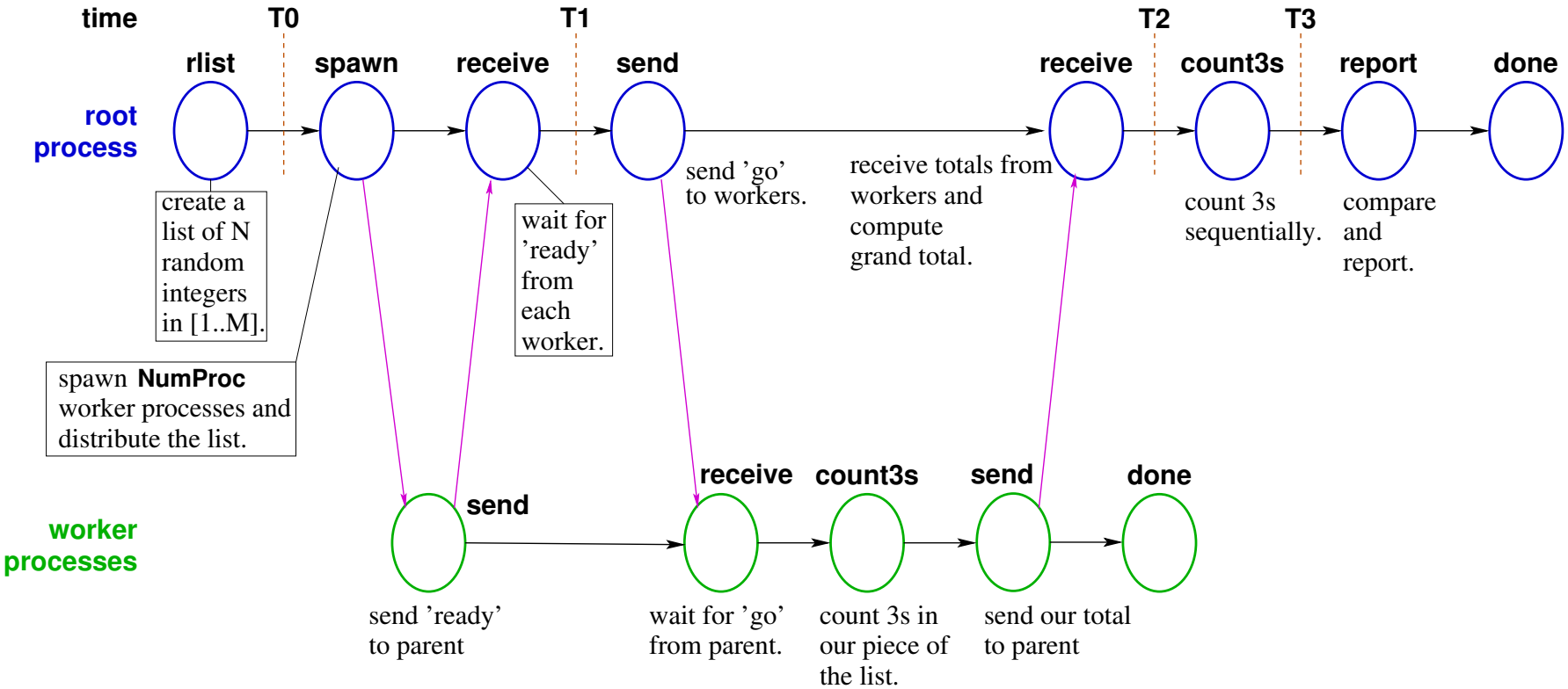
Running Erlang

```
bash-3.2$ erl
Erlang/OTP 18 [erts-7.0] [source] ...

Eshell V7.0 (abort with ^G)

1> c(count3s).
{ok,count3s}
2> L20 = count3s:rlist(20,5).
[3,4,5,3,2,3,5,4,3,3,1,2,4,1,3,2,3,3,1,3]
3> count3s:count3s(L20).
9
4> count3s:count3s(count3s:rlist(1000000,10)).
99961
5> q().
ok
6> bash-3.2$
```

A Parallel Version



The code is in

http://www.ugrad.cs.ubc.ca/~cs418/2016-2/lecture/01-04/src/count3s.erl

Preview of the next month

January 6: Introduction to Erlang Programming

Reading: *Learn You Some Erlang*, the first eight sections – Introduction through Recursion. Feel free to skip the stuff on bit syntax and binary comprehensions.

January 9: Processes and Messages

Reading: *Learn You Some Erlang*, Higher Order Functions and The Hitchhiker's Guide... through More on Multprocessing

Homework: **Homework 1 goes out (due Jan. 18)** – Erlang programming

Mini-Assignment: Mini-Assignment 1 due **10:00am**

Mini-Assignment 2 goes out (due Jan. 13)

January 11: Reduce

Reading: *Learn You Some Erlang*, Errors and Exceptions through A Short Visit to Common Data Structures

---

January 13: Scan

Reading: Lin & Snyder, chapter 5, pp. 112–125

Mini-Assignment: Mini-Assignment 2 due **10:00am**

January 16: Generalized Reduce and Scan

Homework: Homework 1 deadline for early-bird bonus (11:59pm)

Homework 2 goes out (due Feb. 1) – Reduce and Scan

January 18: Reduce and Scan Examples

Homework: Homework 1 due **11:59pm**

January 20–27: Parallel Architecture

January 29–February 6: Parallel Performance

Review Questions

- Name one, or a few, key reasons that parallel programming is moving into mainstream applications.
- How does the impact of your mini assignment total on your final grade depend on how you did on the other parts of the class?
- What are bug-bounties?
- What is the count 3's problem?
- How did we measure running times to compute speed up?
  - Why did one approach show a speed-up greater than the number of cores used?
  - Why did the other approach show that the parallel version was slower than the sequential one?

Supplementary Material

- Erlang Resources
- Bibliography
- Table of Contents – at the end!!!

Erlang Resources

- Learn You Some Erlang

  http://learnyousomeerlang.com

  An on-line book that gives a very good introduction to Erlang. It has great answers to the "Why is Erlang this way?" kinds of questions, and it gives realistic assessments of both the strengths and limitations of Erlang.

- Erlang Examples:

  http://www.ugrad.cs.ubc.ca/~cs418/2012-1/lecture/09-08.pdf

  My lecture notes that walk through the main features of Erlang with examples for each. Try it with an Erlang interpreter running in another window so you can try the examples and make up your own as you go. This will cover everything you'll need to make it through all (or most) of what we'll do in class, but it doesn't explain how to think in Erlang as well as "Learn You Some Erlang" or Armstrong's Erlang book (next slide).

More Erlang Resources

- The erlang.org tutorial
  http://www.erlang.org/doc/getting_started/users_guide.html
  Somewhere between my "Erlang Examples" and "Learn You Some Erlang."
- Erlang Language Manual
  http://www.erlang.org/doc/reference_manual/users_guide.html
  My go-to place when looking up details of Erlang operators, etc.
- On-line API documentation:
  http://www.erlang.org/erldoc.
- The book: *Programming Erlang: Software for a Concurrent World*, Joe Armstrong, 2007,
  http://pragprog.com/book/jaerlang/programming-erlang
  Very well written, with lots of great examples. More than you'll need for this class, but great if you find yourself using Erlang for a big project.
- More resources listed at http://www.erlang.org/doc.html.

Getting Erlang

- You can run Erlang by giving the command `erl` on any departmental machine. For example:
  - Linux: bowen, thetis, lin01, . . . , lin25, . . . ,

  all machines above are .ugrad.cs.ubc.ca, e.g. bowen.ugrad.cs.ubc.ca, etc.
- You can install Erlang on your computer
  - Erlang solutions provides packages for Windows, OSX, and the most common linux distros
    https://www.erlang-solutions.com/resources/download.html
  - Note: some linux distros come with Erlang pre-installed, but it might be an old version. You should probably install from the link above.

Starting Erlang

- Start the Erlang interpreter.

```
theis % erl
Erlang/OTP 18 [erts-7.0] [source] ...

Eshell V7.0 (abort with ^G)

1> 2+3.
5
2>
```

- The Erlang interpreter evaluates expressions that you type.
- Expressions end with a "." (period).

Bibliography

Krste Asanovic, Ras Bodik, et al.

The landscape of parallel computing research: A view from Berkeley.

Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Science Department, University of California, Berkeley, December 2006.

http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf.

Microprocessor quick reference guide.

http://www.intel.com/pressroom/kits/quickrefyr.htm, June 2013.

accessed 29 August 2013.

List of CPU power dissipation.

http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation, April 2011.

accessed 26 July 2011.

Table Of Contents (1/2)

Table Of Contents (2/2)