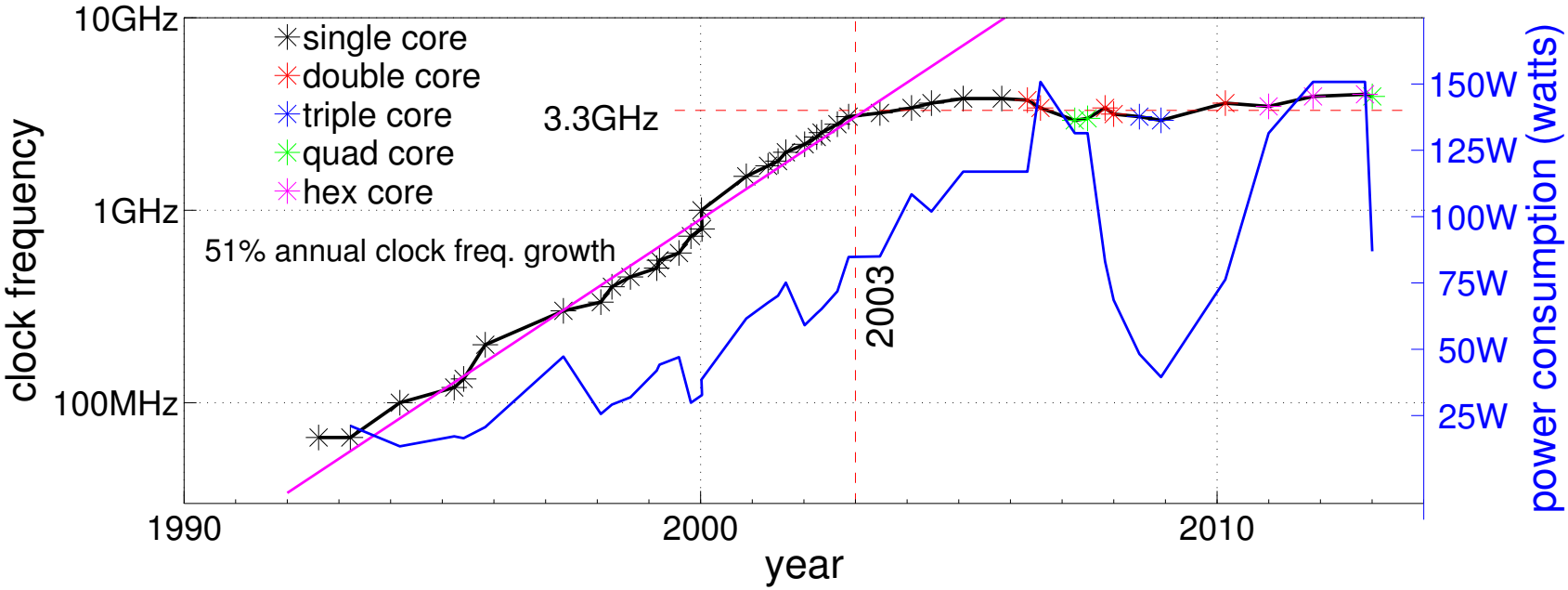
Parallel Computation

Mark Greenstreet

CpSc 418 – Sept. 5, 2013

Outline:

Clock Frequency and Power



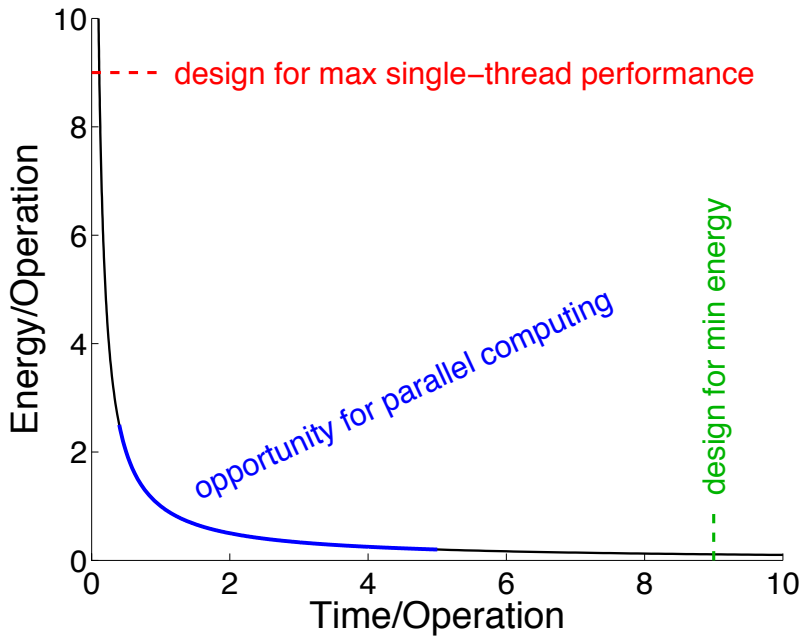
Clock Speed and Power of Intel Processors vs. Year Released[Wikipedia CPU-Power, 2011]

- Designs have been power-limitted since about 2003.
  Otherwise, you would be able to buy a 240GHz processor today!
- Once power was taken into account, lower power processors have been produced.
- Multi-core now is the dominant CPU paradigm.

More Problems

- The memory bottleneck.
- Limited instruction-level-parallelism.
- Design complexity.
- Reliability.
- See [Asanovic et al., 2006].

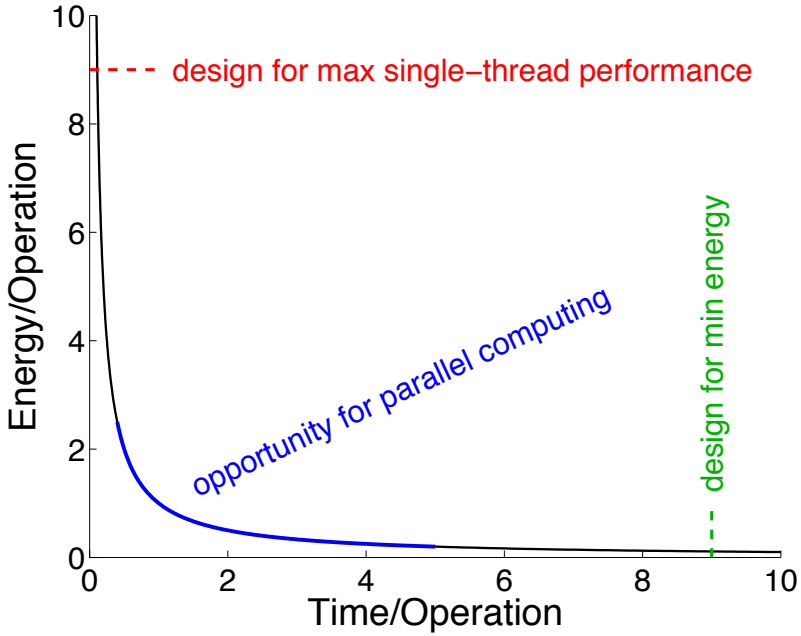
Processor Design Trade-Offs



Energy vs. Time Trade-Offs (idealized).

- If single-thread performance is the primary concern,
  then CPU designers push against the power-limit for cooling the chip.

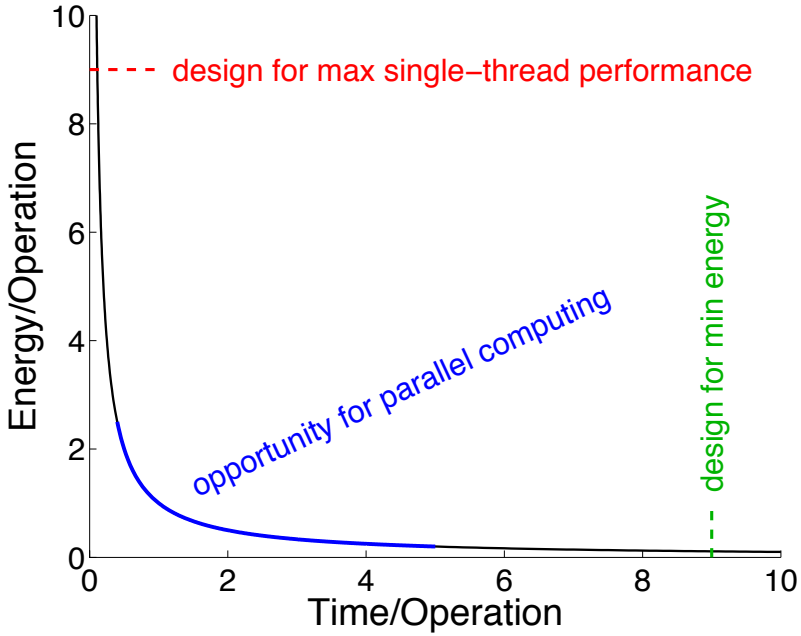
Processor Design Trade-Offs



Energy vs. Time Trade-Offs (idealized).

- If minimizing energy consumption (e.g. maximizing battery life) is the primary concern,
  then CPU designers aim for the minimum performance that completes the required computation in time.

Processor Design Trade-Offs



Energy vs. Time Trade-Offs (idealized).

- Parallel computing allows us to use more, slow processors to get the same task done using less time and less energy than a sequential version.

Trading Energy and Time

Computations take less energy when they are done slower.

- Simple model: $\mathit{Energy} \cdot \mathit{Time} = \text{Constant}$
- Consider a task that requires energy E and time T.
  - If we can break the task into two equal pieces, each requiring energy $E/2$ and time $T/2$,
  - Then, we could do the problem in parallel using total energy E and time $T/2$.
  - Or, we could do each of the smaller tasks on a processor running at half the speed as the original.
    - Now, the energy for each task is $E/4$.
    - The total energy is $E/2$.
    - And the time is T.
- We can save time and/or energy by using parallel computation.

Parallel Computing Solves Other Problems Too!

- The memory bottleneck
  - It is much easier to increase memory bandwidth than it is to decrease memory latency.
  - Many slow CPUs can have references in progress at the same time to high-latency, high-bandwidth memory. A single CPU would stall.
- Limited instruction-level-parallelism.
  - Exploit explicit parallelism instead.
  - BUT, the programmer has to write parallel code.
- Design complexity.
  - It's much easier to design a CPU chip with many copies of the same, simple CPU than with one big, complicated CPU.
- Reliability.
  - Many core designs have built-in redundancy.
  - E.g., nVidia Fermi GPUs have 512 shaders, but only expose 448 or 480 to the programmer.

Parallel Computing is Everywhere

- Multicore desktops and laptops machines
- Samsung Galaxy S4: Quad-core ARM processor+ GPU + dedicated processors for modems, codecs, etc.
- GPU: hundreds to thousands of programmable shaders.
- Clouds
- Supercomputers
- Embedded computing: automotive, medical, appliances, all kinds of other gadgets.

Outline

- Why Does Parallel Computation Matter?
- Course Overview
  - Syllabus
  - The instructor, TA, text, ...
  - Plagiarism
  - Bug bounties
- Our First Parallel Program

Syllabus

- September
  - Parallel Programming in Erlang
  - Basic algorithms and parallel programming techniques
- October
  - Architectures for Parallel Computers
  - Performance Analysis: principles, models, and measurements
  - Midterm: October 22, 2013 – in class
- November
  - Parallel Programming Paradigms:
    - message passing: MPI
    - threads: pthreads and/or Java threads
  - More parallel algorithms
    - mutual exclusion and other synchronization mechanisms
    - sorting, dynamic programming, ...
- The more detailed syllabus (but subject to revision):

  http://www.ugrad.cs.ubc.ca/~cs418/2013-1/syllabus.html

Administrative Stuff – Who

- The instructor
  - **Mark Greenstreet,** mrg@cs.ubc.ca
  - ICICS/CS 323, (604) 822-3065
  - Office hours: Mondays, 12noon – 1pm, ICICS/CS 323
    - Office hours will change if the proposed time doesn't work for many students in the class,
    - or if I end up with another meeting scheduled at that time.
    - You can always send me e-mail to make an appointment.
- The TAs

  RJ Sumi, rjsumi@cs.ubc.ca
  Jake Lever, jake.lever@alumni.ubc.ca
  Office Hours: Thursdays, 10am – 11am, Demco 150
- Course webpage: `http://www.ugrad.cs.ubc.ca/~cs418`.
- Online discussion group: on piazza (how do I give that a url?)

Administrative Stuff – What

- The book: "Principles of Parallel Programming"
  Calvin Lin and Lawrence Snyder
- web: http://www.ugrad.cs.ubc.ca/~cs418
- Grades:

Homework:	35%	roughly one assignment every two weeks
Midterm:	25%	October 22, in class
Final:	40%	

- Quizzes and Pre-lecture mini-assignments
  - Worth 20% of point missed from HW and exams.
  - There will be 5-10 such mini-assignments.
  - The first one is for Sept. 10. See

http://www.ugrad.cs.ubc.ca/~cs418/2013-1/mini/sept10/q.pdf

- Homework late policy:
  - Homework N due one week after homework $N + 1$ assigned.
  - No late homework accepted.

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.

Grades: the big picture

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

Mini-assignments:

- If your raw grade is 90%, you can get at most 2% from the mini-assignments.

  You can afford to skip them if you're doing well and want to spend your on other courses.
- If your raw grade is 70%, you can get at most 6% from the mini-assignments.

  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 mini-assignments. Do the mini-assignments – I hate turning in failing grades.

Grades: the big picture

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

- I'll probably toss in some extra credit marks into the regular HW – these tend to be "unreasonable" problems. They are intended to be fun challenges for those who are otherwise blowing the course away and would enjoy learning more.
- Bug-bounties are for everyone. The reward you for looking at the HW when it first comes out, and not waiting until the day before it is due. ☺

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/

Lecture Outline

- Why Does Parallel Computation Matter?
- Course Overview
- Our First Parallel Program
  - Erlang quick start
  - Count 3's
  - Sequential Erlang version
  - First parallel version
  - Second parallel version

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/~cs418/2013-1/lecture/09.05/code.html

Simple Example: factorial

In Java:

```
int fac(int n) {
    int p = 1;
    while(n > 0) {
        p = p*n;
        n = n-1;
    }
    return(p);
}
```

- But, this isn't functional: the values of `n` and `p` change with each iteration of the while loop.
- How can we get iteration without modifying the values of variables?

Simple Example: factorial

Old

```
int fac(int n) {
   int p = 1;
   while(n > 0) {
      p = p*n;
      n = n-1;
   }
   return(p);
}
```

New)

```
int fac(int n) {
   if(n == 0) return(1);
   else return(n*fac(n-1));
}
```

- How can we get iteration without modifying the values of variables?
- Replace iteration with recursion.

Simple Example: factorial

Old (Java)

```
int fac(int n) {
   if(n == 0) return(1);
   else return(n*fac(n-1));
}
```

New (Erlang)

```
fac(N) ->
   if N == 0 -> 1;
      N > 0  -> N*fac(N-1)
   end.
```

- Erlang function definitions have the form:

```
functionName(ParameterList) -> expression .
```

- Erlang variable names start with upper-case letters.
- An `if`-expressions has form:

```
if cond1 -> expr1;
   cond2 -> expr2;
   ...
   condn -> exprn
end
```

  - The value of the `if`-expression is the value of the first expression whose corresponding condiion is true.
  - If no condition is satisfied, then an exception is thrown.

Simple Example: factorial

Old

```
fac(N) ->
    if N == 0 -> 1;
       N > 0  -> N*fac(N-1)
    end.
```

New)

```
fac(0) -> 1;
fac(N) -> N*fac(N-1).
```

- A function definition can consist of multiple “patterns”.
- The first pattern that can be matched to the actual arguments is the one that gets used.
- We’ll cover more with patterns next week.

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]) = []`.

Lists traversal example: sum

```
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.

Count 3's: a simple parallel programming example

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

Java:

```
int count3s(int[] data) {
   int count = 0;
   for(int i = 0; i < data.hi; i++)
      if(data[i] == 3)
         count++;
   return(count);
}
```

Erlang:

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

Let's try it!

```
-module count3s.
-export [count3s/1, rlist/1, rlist/2].

% count3s: return the number of 3's in a list.
count3s([]) -> 0;
count3s([3 | Tail]) -> 1 + count3s(Tail);
count3s([_Other | Tail]) -> count3s(Tail).

% rlist: return a list of N random digits, each selected from 1..M
rlist(0,_M) -> [];
rlist(N,M) -> [random:uniform(M) | rlist(N-1,M)].

% list of N random digits selected from 1..10
rlist(N) -> rlist(N, 10).
```

Running Erlang

```
bash-3.2$ erl Erlang R14B03 (erts-5.8.4) [source]
  [smp:8:8] [rq:8] [async-threads:0] [hipe]
  [kernel-poll:false]

Eshell V5.8.4 (abort with ^G)
1> c(count3s).
{ok,count3s}
2> L20 = count3s:rlist(20,5).
[1,3,4,5,3,2,3,5,4,3,3,1,2,4,1,3,2,3,3,1]
3> count3s:count3s(L20).
8
4> count3s:count3s(count3s:rlist(1000000,10))).
99961
5> q().
ok
7> bash-3.2$
```

First Parallel Version

```
-module count3s_p1.
-export [count3s/1, count3s/2, childProc/2].

% count3s: return the number of 3's in a list.
count3s(L0, _N0, 1, _MyPid) ->          % 1 processor
   count3s:count3s(L0);                 % just do it.
count3s(L0, N0, NProcs, MyPid) ->       % > 1 processor.
   % spawn a process to handle the first N/NProcs elements of L.
   % make a recursive call with NProcs-1 to handle the rest.
   N1 = N0 div NProcs,
   N2 = N0 - N1,
   {L1, L2} = lists:split(N1, L0),
   spawn(count3s_p1, childProc, [L1, MyPid]),
   C2 = count3s(L2, N2, NProcs-1, MyPid),
   receive % get a value from a child process, and add it to C2.
      {count3s, C1} -> C1 + C2
   end.
```

First Parallel Version (cont.)

```
% versions of count3s that fill in default arguments.
count3s(L, NProcs) ->
    count3s(L, length(L), NProcs, self()).
count3s(L) ->
    count3s(L, erlang:system_info(schedulers)).

childProc(L, ParentPid) ->
    ParentPid ! {count3s, count3s:count3s(L)}.
```

- Time to run sequential version on list with 1,000,000 elements:
  13.8ms.
- Time to run parallel version on list with 1,000,000 elements:
  51.3ms.
- Parallel programming achieves a **71% slow down!**
  - Why?
    Because Erlang copies the arguments for child processes.
    ☹

Second Parallel Version

[1] Create `NProcs` worker processes.
[2] Each worker process generates `Nelements/Nprocs` random values.
[3] The root process asks each worker to send back the
number of 3's in it's portion of the values.
[4] The root process computes the sum of these values.

- I measured the time to count the 3's as the time for steps [3] and [4] above.
- Time to run the second parallel version on a list with 1,000,000 elements:
  2.5ms.
- Parallel programming achieves a **5.5x speed up.**
  - Pretty good for a quad-core machine!
  - $>$ 4x speed-up do to multi-threading.
  - To get the code, go to

`http://www.ugrad.cs.ubc.ca/~cs418/2013-1/lecture/09.05/code.html`

Erlang Resources

- 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 interpretter 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 the Erlang book (below).

- 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.

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.
- Erlang Library Documentation

  http://www.erlang.org/doc/man_index.html

  Describes `lists`, `io`, `math`, and so much more.
- 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.

Getting Erlang

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

  all machines above are .ugrad.cs.ubc.ca, e.g. bowen.ugrad.cs.ubc.ca, etc.
- Or, download it for your own computer.
- See http://www.erlang.org/download.html

Starting Erlang

- Start the Erlang interpretter.

```
gambier % erl
Erlang R14B (erts-5.8.1) [source]
  [smp:64:64][rq:64][async-threads:0]
  [kernel-poll:false]

Eshell V5.8.1 (abort with ^G)
4> 2+3.
5
5>
```

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

Preview of the next few weeks

September 10: Sequential programming with Erlang

Homework: Homework 1 goes out
Mini-Assignment: Mini-Assignment due before class
Reading: Lin & Snyder, chapter 1

September 12: Parallel programming with Erlang

Mini-Assignment: Mini-Assignment due before class

September 17: Quantifying Performance

Reading: Lin & Snyder, chapter 3, pp. 61–68

September 19: Matrix multiplication – algorithms

Homework: Homework 2 goes out – parallel programming with Erlang
Reading: Lin & Snyder, chapter 3, pp. 68–77
Homework: Homework 1 deadline for early-bird bonus

September 24: Matrix Multiplication – performance

Reading: Lin & Snyder, chapter 3, pp. 77–85
Homework: Homework 1 due

September 26: Superscalars and compilers

Reading: The MIPS R10000 Superscalar Microprocessor (Yeager)

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?
- Why don't functional languages have `while` or `for` loops?
- What do they use instead?
- What is the count 3's problem?
- Why was the first parallel version slower than the sequential version?
- Why was the speed-up factor for the second parallel version larger than the number of processor cores used?

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.