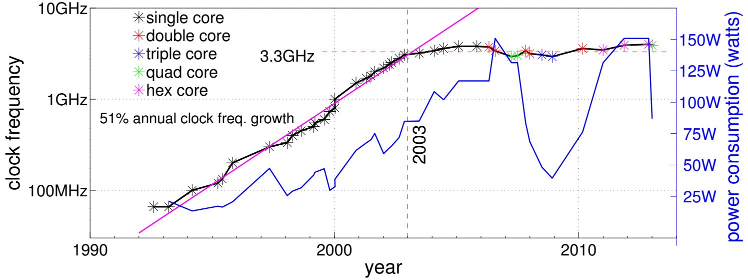
Parallel Computation

Mark Greenstreet and Ian M. Mitchell

CpSc 418 – January 3, 2018

- Why parallel programming matters
- Key aspects of parallel computing:
  - Architecture, performance, algorithms, paradigms.
- Course overview
- Our first parallel program

Why Parallel Computation Matters: **POWER**



Clock Speed and Power of Intel Processors vs. Year Released[a]

- 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.
- If the trend had continued, we would have $>$1000GHz CPUs today. ☺

[a]See http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation

Welcome to Parallel Computing

- Sequential computing is all the same
  - Architecture: IBM 360 = PDP 11 = Intel X86 = ARM = ...
  - Programming languages: Cobol = Fortran = C = C++ = Java = Python = ...
- Parallel computing doesn't have a dominant architecture or programming paradigm
  - Architecture: multi-core, super-scalar, multi-threading, shared-memory, clusters, GPUs, ...
  - Programming paradigms: PThreads, MPI, CUDA, Hadoop, ...
  - Platforms: multi-core mobile devices, multi-core desktops and laptops, GPU based-accelerators, data centers, clouds, scientific supercomputers.

Key Concepts of Parallel Computing

- On slide 3 we described the diversity of parallel computing.
  - There are unifying concepts.
- Scalable parallelism: divide your task in a way that scales with problem size.
  - If the problem is twice as large, ideally you'll have twice as many subtasks.
- Communication and task coordination (C&C) are expensive
  - The many parallel architectures can be understood by how they try to make C&C efficient.
  - The many parallel programming paradigms can be understood by how they support C&C.
  - When devising a parallel solution, look for ways to minimize the costs of C&C.

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.
  - GPUs: hundreds or few thousand simple cores that *all execute the same instruction sequence*.
  - Domain specific processors
    - 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.
- Know Big-O, and measure reality
  - Parallel computing makes inefficient algorithms worse because we care about large N.
  - Communication and coordination costs are critical
    - These costs involve the program, its run-time, the operating system, the network, contention with other tasks, and other factors.
    - Measure, identify the real issues, and model what matters.

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 Paradigms

CPSC 418 will take a multiparadigm approach:

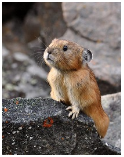
- There is no single dominant paradigm for parallel computing.
- We'll focus on two:
  - Erlang – message passing, functional programming
    - It's easy to illustrate many concepts
    - Message passing makes communication explicit, and communication is usually the critical design issue.
  - CUDA: your graphics card is a super-computer
    - It's a fairly restrictive model, but
    - It works very well for many problems from graphics, scientific computing, and machine learning.
- We'll mention other paradigms as the course goes on.
  - Our goal is to have your prepared so you can quickly pick-up new parallel programming paradigms – **especially** those that don't exist yet!

Course Overview

- Blah, blah, blah, ...
  - Every course has an instructor and usually there are TAs, a textbook, and a syllabus. This course has all that stuff. You can read all about it on some slides after the end of the lecture.
  - This course has its own, cool stuff:
    - PIKAs – pre/in-class activities
    - Early Bird bonuses
    - Plagiarism policy: please don't.
    - Bug Bounties
  - Mark can't hear (what?!)

PIKAs – Pre/In-Class Activities

- Worth 20% of points missed from HW and the midterm exam.
  - If your pre-final grade is 90%, you can raise that at most 2% through the pikas. Missing one or two isn't a big deal.
  - If your pre-final grade is 70%, you can raise that 6% from the pikas. This might move your letter grade up a notch (e.g. C+ to B−).
  - If your raw grade is 45%, you can raise that 11% from the pikas. So do the pikas – we hate turning in failing grades.
- The first will be posted by Jan. 5 and due before lecture on Jan. 8.
- Why **pika**?



- Pica is a serious eating disorder – if you suffer from an eating disorder, it's an important medical issue. Get help. Vancouver Coastal Health has excellent programs.
- A pika is a cute (but unfortunately endangered) mammal that is native to British Columbia.

Photo from `http://pixdaus.com/northern-pika-photographer-sandro-animals-pika-yakutia/items/view/281604/`.

Early Bird Bonuses

- Late policy: Late homework will not be accepted.
- Early policy: Early homework gets a 5% bonus.
  - Typically, the early-bird deadline is two days before the hard deadline.
- No worms.

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/

Collaboration is good

- Talk to your friends.
- Talk with your enemies, if this class brings peace between you, all the better.
- Figure things out together, but write your own code, write your own solutions, and clearly state who you worked with.
- We will give full credit if you work out your own solution after discussing the problem with others.
- If you and a friend work out one solution, turn in two copies of it, **and clearly state that you did so**,
  - This is not academic misconduct. No prosecution.
  - Of course, we'll only give credit for one solution; so, you'll each get half.

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 pikas, 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.

Mark Can't Hear (very well)

- When working with the whole class, Mark will often move to the person who's asking or answering a question so he can hear (and see) them better.
  - Besides, the exercise is good for him.
- Devon Graham is a TA who will repeat questions and comments when Mark doesn't hear them.
  - Thanks, Devon.
- Don't be shy. Mark wants you to ask questions and make remarks. He's had the hearing trouble for about 30 years. If Mark doesn't hear you, it's not your fault.

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.

Count 3s: a simple example

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

- In Erlang, we'll use lists
  - `[]` is the empty list.
  - `[3]` is a list with one-element, where that element has the value `3`.
  - `[1, 2, 3, 4, 5]` is a list of five elements.
  - `[1 | List2]` is the list whose first element has the value `1` and whose tail is the list `List2`.
- We can write functions using pattern matching

```
count3s([]) -> 0; % The empty list has 0 threes

% A list whose head is 3 has one more 3 than its tail
count3s([3 | Tail]) -> 1 + count3s(Tail);

% A list whose head is not 3 has the same number of 3s as its tail
count3s([_Other | Tail]) -> count3s(Tail).
```

Counting Threes – In Parallel

Design:

- Use P worker processes.
  - Each worker generates a list with $\frac{N}{P}$ elements.
- A master process sends a request to each worker asking it to count the 3s in its piece of the list and send its subtotal back to the master.
- The master adds up the subtotals from each worker and reports the grand total.
- The code is in

  http://www.ugrad.cs.ubc.ca/~cs418/2017-2/lecture/src/course_intro.e

The Results (part 1)

	lulu		thetis	
P	Time	Speed-up	Time	Speed-up
1	12.69	1.00	15.40ms	1.00
2	7.97	1.59	9.82ms	1.57
4	6.26	2.03	5.98ms	2.58
8	3.28	3.87	3.11ms	4.95
16	2.93	4.33	2.04ms	7.55
32	2.91	4.36	1.66ms	9.28
64	2.80	4.53	1.62ms	9.51
128	2.78	4.56	2.24ms	6.87
256	2.83	4.84	1.64ms	9.39

Approach:

- This slide: Set *N*, the total number of data values to something "large" (i.e. 10^6), sweep *P* to find optimal value.
- Next slide: Use optimal value for *P* and sweep *N* to show trends.
- Notes: lulu.ugrad.cs.ubc.ca has four, two-way multithreaded cores; thetis.ugrad.cs.ubc.ca has sixteen, two-way multithreaded cores.

The Results (part 2)

TBD

	lulu		thetis	
N	Time	Speed-up	Time	Speed-up
1,000	???ms	???	???ms	???
10,000	???ms	???	???ms	???
100,000	???ms	???	???ms	???
1,00,000	???ms	???	???ms	???
10,000,000	???ms	???	???ms	???
100,000,000	???ms	???	???ms	???

- Speed-up increases with the problem size.
- To utilize more processors (e.g. thetis) we need a bigger problem size to approach peak speed-up.
- Speed-up can be **greater** than the number of cores!
  This is a consequence of “multithreading”.

Variations

- Start with all the data on the “master” process – send each worker its chunk of the list (or array) in which to count 3s.
  - This is a common “newbie” error motivated by a desire to make the code “look sequential” (i.e. familiar, comfortable, and safe).
  - In practice, this makes the parallel version slower than the sequential version.
  - Sending the data take more time than counting the threes.
  - Communication is very often the bottleneck for parallel computing. See slide 4.
- Combine the sub-totals using a tree structure.
  - This is a great idea!
  - It allows communication to be done in parallel.
  - This version achieve near-peak speed-up for smaller problem sizes than the simple method described on slide 19
  - We will explore this approach in more detail starting with the Jan. 10 lecture.

Preview of the next lecture

January 5: Introduction to Erlang

- Read *Learn You Some Erlang*: “Introduction” through “Functionally Solving Problems” – You scan skip “Errors and Exceptions” (until next week).
- Erlang is **functional**
- Erlang uses **message passing**
- Abstraction of programming patterns using higher-order functions.
- There is a pre-class activity
  - You need to complete an Erlang template file.
  - Must be submitted through handin by 1pm on January 5.
- There will be in-class programming activities:
  - Bring your laptop or share with someone who brings theirs.
- Peaking ahead:
  - The Piazza course will be created later today. Expected url: piazza.
  - The first PIKA will go out on Jan. 5. Due on Jan. 8 at 1pm.
  - Reading for Jan. 8, *Learn You Some Erlang*: “A short visit to common data structures” through “More on Multiprocessing”.
  - The first homework will go out on Jan. 8. Due on Jan. 17 at 11:59pm (Early-Bird deadline: Jan. 15 at 11:59pm).

Review Questions

- Name one, or a few, key reasons that parallel programming is moving into mainstream applications.
- What is a pika?
- How does the impact of your pika score on your final grade depend on how you did on the other parts of the class?
- What are bug-bounties?
- What is the count 3s 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

- Course Organization
- Getting Started with Erlang
- Slide Index

Course Organization

- Topics: see slide 4
- Syllabus
- The instructor and TAs
- The textbook(s)
- Grades
  - **February 28** Midterm – in class
- Plagiarism: see slide 12
- Learning Objectives: see slide 16

Syllabus

- January: Erlang & an intro. to everything

  Jan. 3– 8: Course overview, intro. to Erlang programming.
  Jan. 10–17: Parallel programming in Erlang, reduce and scan.
  Jan. 19–26: Parallel architectures
  Jan. 28–Feb. 5: Performance analysis

- February: Erlang, Midterm

  Feb. 7–16: Sorting
  Feb. 19–23: Midterm break.
  Feb. 26: Midterm Review
  February 28: Midterm

- March: CUDA and other topics
- Note: We'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: TBA
- The TAs

Devon Graham,	drgraham@cs.ubc.ca
Kristian Jensen,	l2x1b@ugrad.cs.ubc.ca
Sitao Lu,	p3l0b@ugrad.cs.ubc.ca
Jocelyn Minns,	jminns@cs.ubc.ca
Mason Yang,	o7p0b@ugrad.cs.ubc.ca

- 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* (2nd **or** 3rd 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 on-line from the UBC library or your can buy the dead-tree version at amazon.ca or 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 (roughly every two weeks): 35%
- Midterm exam (February 28 in class): 25%
- Final exam (Date, time and location to be determined): 40%
- PIKA and bug bounties

$$\begin{aligned}
\mathit{PreFinalGrade} &= 0.35 * \mathit{HW} + 0.25 * \mathit{MidTerm} \\
\mathit{PikaBonus} &= 0.20 * (0.60 - \min(0.60, \mathit{PreFinalGrade})) * \mathit{Pika} \\
\mathit{FinalGrade} &= 0.40 * \mathit{Final} \\
\mathit{BB} &= 0.35 * \mathit{BB}_{\mathit{HW}} + 0.25 * \mathit{BB}_{\mathit{MT}} + 0.40 * \mathit{BB}_{\mathit{Final}} \\
\mathit{CourseGrade} &= \min\left(1.00, \begin{array}{c} \mathit{PreFinalGrade} + \mathit{FinalGrade} \\ + \mathit{PikaBonus} + \mathit{BB} \end{array}\right)
\end{aligned}$$

- To pass the course you must obtain a 50% overall score and pass the final exam.
- The instructors reserve the right to adjust the grading scheme when necessary to accurately reflect student learning outcomes and/or departmental expectations.

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 12 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 February 28.
- 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.

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, lulu, thetis, lin01, . . . , lin25, . . . ,

  all machines above are in the ugrad.cs.ubc.ca domain, 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.

```
thetis % erl
Erlang/OTP 20 [erts-9.0] [source] ...
Eshell V9.0 (abort with ^G)

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

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

Index – Main Lecture

- Why Parallel Computation Matters
- Topics covered
  - Topics covered
  - Parallel Architectures
  - Key Concepts of Parallel Computing
  - Parallel Architectures
  - Parallel Performance
  - Parallel Algorithms
  - Parallel Programming Paradigms
- Course Overview
  - PIKAs – Pre/In-Class Activities
  - Early Bird Bonuses
  - Plagiarism
  - Bug Bounties
  - Mark Can't Hear (very well)
  - Learning Objectives
- Count 3s: a simple example
- Preview & Review

Index – Supplementary Material

- Course Organization
  - Syllabus
  - Administrative Stuff – Who
  - Textbook(s)
  - Grades
  - Homework
  - Exams
- Erlang Resources
- Index