ECE 454
Computer Systems Programming

Introduction

Ding Yuan
ECE Dept., University of Toronto
http://www.eecg.toronto.edu/~yuan

Content of this lecture

- Administration (personnel, policy, agenda, etc.)
  - Boring stuff
    - You can go to sleep now

- Why ECE 454?
  - Fun stuff
    - I will wake you up
Who am I

• Ding YUAN (call me Ding)
• Research: operating system, software reliability and performance
• Brief BIO:
  • Ph.D. University of Illinois (UIUC), 2012
  • Microsoft Research 2008
  • Technique invented are requested by many large companies
  • Contributed patches to many open-source projects:
    • Linux kernel
    • Hadoop
    • HBase
    • Cassandra
    • Hive
    • ...
Personnel

- Instructor:
  - Ding Yuan (yuan@eecg.toronto.edu)
  - Office hour: Wednesday after the lecture
  - Office: Sandford Fleming 2002E
  - Homepage: http://www.eecg.toronto.edu/~yuan

- Teaching Assistants:
  - David Lion, Yongle Zhang, Xu Zhao
  - TA available in computer lab GB251 in the first hour of the practice session

Recommended Textbook

- Textbook is not required
- The relevant contents will be covered in the slides
- Google & Wikipedia can tell you all
- I will post some online resources in Piazza

- Randal E. Bryant and David R. O’Hallaron,
Communications

- Class web site available from instructor's home page
  - http://www.eecg.toronto.edu/~yuan/teaching/ece454/
  - Provides slides, agenda, grading policy, etc.
  - All information regarding to the labs

- Piazza (See course homepage) used for discussion
  - Q/A & discussion with peers, TAs, prof
  - Bonus marks:
    - each instructor endorsed answer on Piazza will get 0.5 bonus marks
    - 1 mark for in-class participation
    - maximum: 2 marks

- UofT Portal is only used for Grades

Policies: Grading

- Exams (65%)
  - Midterm (25%)
  - Final (40%)
  - All exams are close book/close notes.

- Homework (35%)
  - 5 homeworks (varying % each)
  - 10% penalty per day submitted late
Policies: Assignments

- **Work groups**
  - You can work in groups of two for all labs (or individually)
  - You can change groups for each assignment (if you want)
  - No extensions for group changes mid-assignment
  - Don't put assignment code on public Google or github repositories!

- **Handins**
  - Electronic hand-ins only
  - Follow the submit procedure (as specified in lab handout)

Policies: Cheating

- Cheating is a serious offence, will be punished harshly
  - 0 grade for assignment, potential for official letter in file.

- What is cheating?
  - Using someone else's solution to finish your assignment to avoid having to understand/learn
  - Sharing code with a non-group-member
  - Copying or retyping

- What is NOT cheating?
  - Helping others use systems or tools.
  - Helping others with high-level design issues.
  - Helping others debug their code.

- We do use cheater-beaters
  - Automatically compares your solutions with others
How NotTo pass ECE454

• Do not come to lecture
  • It's nice out, the slides are online, and material in the book anyway
  • TRUTH: Lecture material is the basis for exams
  • It is much more efficient to learn through discussion

• Copy other people’s project
  • It is cheating!
  • How can you answer the questions in midterm or final exams?

How NotTo pass ECE454 (2)

• Do not ask questions in lecture, office hours, or piazza
  • It's scary, I don't want to embarrass myself
  • TRUTH: asking questions is the best way to clarify lecture material at the time it is being presented
    • “There is no such things as stupid question…”

• Wait until the last couple of days to start a project
  • The project cannot be done in the last few days
Facilities

- Official lab time: Thursdays 12-3 p.m. or 3-6 p.m.
  - Both GB243 and GB 251
  - Optional: you don’t have to attend

- Identical workstations:
  - GB243: ug132-ug180
  - SF2102: ug201-ug225
  - GB251: ug226-ug249
  - Develop and measure on any of these
  - Try to measure on an unloaded machine!

- Similar workstations:
  - SF2204: ug51-ug75
  - Can use for development, but don't measure on these!

- try your UG-machine accesss ASAP!
Before we start

- Any questions?

Why ECE 454?
Why Take this Course?

- Become a superstar programmer
- Most engineering jobs involve programming
- Superstar programmers are increasingly in demand
- *A superstar programmer is worth 1000x normal* – Bill Gates

**Google Offers Staff Engineer $3.5 Million To Turn Down Facebook Offer**

---

Why Take this Course?

- Better understanding of software/hardware interaction
- Important whether you are a software or hardware type
- Considering a programming job or grad school

- Jobs and Entrepreneurial Opportunities
  - Computing is at the heart of most interesting ventures
Start a Company in your 20's!

What Good Programmers Care About

1) Readability
2) Debugability
3) Reliability
4) Maintainability
5) Scalability
6) Efficiency

Productivity (choice of language, practice)
Performance (systems understanding)

ECE 454
Let’s be more concrete

- Suppose you’re building
  - The “homepage” feature

```c
void display_homepage (user)
  {
    friendlist = get_friendlist (user);
    foreach (friend in friendlist)
    {
      update = get_update_status (friend);
      display (update);
    }
  }
```

☞ How can I double the speed?
☞ Easy: TAKE ECE 454!!!

Pre 2005

- To improve the performance, just buy a new computer
Recent Intel Processors

<table>
<thead>
<tr>
<th>Year</th>
<th>Processor</th>
<th>Chip # Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>Pentium</td>
<td>3.1 million</td>
</tr>
<tr>
<td>1999</td>
<td>Pentium II</td>
<td>7.5 million</td>
</tr>
<tr>
<td>1997</td>
<td>Pentium III</td>
<td>28 million</td>
</tr>
<tr>
<td>1993</td>
<td>Pentium IV</td>
<td>42 million</td>
</tr>
</tbody>
</table>

"Moore’s Law"

Gordon Moore, 1965

The number of transistors per chip seems to be doubling every 18 months!

Increases in Transistors vs. Clock Freq.

Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten
A Multicore Present and Future

2x cores every 1-2yrs: 1000 cores by 2020!? 

Only One Sequential Program to Run?

```cpp
void display_homepage (user) {
    friendlist = get_friendlist (user);
    foreach (friend in friendlist) {
        update = get_update_status (friend);
        display (update);
    }
}
```

one core idle

15 cores idle!
Improving Execution Time

Single Program:

need parallel threads to reduce execution time

```c
void display_homepage (user) {
  friendlist = get_friendlist (user);
  foreach (friend in friendlist) {
    pthread_create(fetch_and_display, friend);
  }
}

void fetch_and_display (friend) {
  update = get_update_status (friend);
  display (update);
}
```

fetch_and_display fetch_and_display fetch_and_display fetch_and_display

alya ambal amla amal
Punch line: We Must Parallelize All Software!

\textit{You will learn it in ECE 454}

\[\text{But...}\]

- So far we only discussed CPU
- But is it true that faster CPU $\rightarrow$ faster program?
  - \textit{The same program may run slower on a faster CPU. Why?}

```c
void display_homepage (user) {
    friendlist = get_friendlist (user);
    foreach (friend in friendlist) {
        update = get_update_status (friend);
        display (update);
    }
}
```
Storage hierarchy

- Your program needs to access data. That takes time!

![Storage hierarchy diagram]

Numbers everyone should know

- L1 cache reference 0.5 ns* (L1 cache size: < 10 KB)
- Branch mispredict 5 ns
- L2 cache reference 7 ns (L2 cache size: hundreds KB)
- Mutex lock/unlock 100 ns
- Main memory reference 100 ns (mem size: GBs)
- Send 2K bytes over 1 Gbps network 20,000 ns
- Read 1 MB sequentially from memory 250,000 ns
- Round trip within same datacenter 500,000 ns
- Flash drive read 40,000 ns
- Disk seek 10,000,000 ns (10 milliseconds)
- Read 1 MB sequentially from network 10,000,000 ns
- Read 1 MB sequentially from disk 30,000,000 ns
- Send packet Cal.->Netherlands->Cal. 150,000,000 ns

*1 ns = 1/1,000,000,000 second

For a 2.7 GHz CPU (my laptop), 1 cycle = 0.37 ns
Performance optimization is about finding the bottleneck

* If you can avoid unnecessary disk I/O
  * --> your program could be 100,000 times faster
  * Have you heard of Facebook's memcached?

* If you allocate your memory in a smart way
  * --> your data can fit entirely in cache
  * and your program could be another 100 times faster
  * You will learn this in lab assignments

Back to the Facebook example

```c
void display_homepage (user) {
  friendlist = get_friendlist (user);
  foreach (friend in friendlist) {
    pthread_create(fetch_and_display, friend);
  }
}

void fetch_and_display (friend) {
  update = get_update_status (friend);
  display (update);
}
```

☞ Challenge: the data rows are too BIG!

100 Petabytes = 200,000 x my laptop
Back to the Facebook example

```cpp
void display_homepage (user) {
    friendlist = get_friendlist (user);
    updates = MULTI_GET ("updates", friendlist);
    display (updates);
}
```

Optimization 1: parallelization
Opt. 2: Store in memory instead of hard disk

Course Content
Course Breakdown

- Module 1: Code Measurement and Optimization
- Module 2: Memory Management and Optimization
- Module 3A: Multi-core parallelization
- Module 3B: Multi-machine parallelization

1) Code Measurement and Optimization

- Topics
  - Finding the bottleneck!
  - code optimization principles
  - measuring time on a computer and profiling
  - Understanding and using an optimizing compiler

- Assignments
  - HW1: Compiler optimization and program profiling
    - basic performance profiling, finding the bottleneck.

- **Topics**
  - Memory hierarchy
  - Caches and Locality
  - Virtual Memory
  
  Note: all involve aspects of software, hardware, and OS

- **Assignments**
  - HW2: Optimizing Memory Performance
    - profiling, measurement, locality enhancements for cache performance
  - HW3: Writing your own memory allocator package
    - understanding dynamic memory allocation (malloc)

## 3) Parallelization

- **Topics**
  - A: Parallel/multicore architectures (high-level understanding)
    - Threads and threaded programming
    - Synchronization and performance
  - B: Parallel on multiple machines
    - Big data & cloud computing

- **Assignments**
  - HW4: Threads and Synchronization Methods
    - Understanding synchronization and performance
  - HW5: Parallelizing a program
    - Parallelizing and optimizing a program for multicore performance
A big picture

Topic 1: code optimization

Topic 2: mem. management

Topic 3A: multi-core parallelization

Topic 3B: parallelization using the cloud

Homework Schedule

- HW1: 2 weeks 5%
- HW2: 2 weeks 7%
- HW3: 4 weeks 9%
- HW4: 1.5 weeks 7%
- HW5: 2 weeks 7%
- 35% total
The bigger picture

- Optimization is not the ONLY goal!

1) Readability
2) Debugability
3) Reliability
4) Maintainability
5) Scalability
6) Efficiency

More important than performance!!!!

Premature optimization is the root of all evil!

– Donald Knuth

Example 1

- Premature optimization causing bugs
  - `cp /proc/cpuinfo`
  - Created an empty file!!! (Demo)

```c
bool copy_reg (. ) { 
  if (src.st_size != 0) { ← Premature optimization!!!
    /* Copy the file content */
  }
  else {
    /* skip the copy if the file size = 0 */
  }
}
```
Example 2

• Optimization might reduce readability

```c
int count (unsigned x) {
  int sum = x;
  while (x != 0) {
    x = x >> 1;
    sum = sum - x;
  }
  return sum;
}
```

```c
int count (unsigned x) {
  int sum, i;
  sum = x;
  for (i = 1; i < 31; i++) {
    x = rotate1(x, 1);
    sum = sum + x;
  }
  return -sum;
}
```

They're both to count the number of ‘1’ bits in ‘x’. How could someone else is to maintain this code?

/*
 * When I wrote this, only God and
 * I understood what I was doing.
 * Now, only God knows
 */
But how do I know if my optimization is “premature”?

- Hard to answer…
- “Make it work; Make it right; Make it Fast” --- Butler Lampson
  - Purpose of my program?
    -- e.g., will it have a long lifetime or it’s a one-time thing (e.g., hackathon or ACM programming contest)
  - Am I optimizing for the bottleneck?
    -- e.g., if the program is doing a lot of I/O, there is no point to optimize for “count the number of bits in an integer”
  - Am I optimizing for the common case or special case?
    -- e.g., the “cp” bug was optimizing for a special case…
  - What’s the price I pay? e.g., reduced readability, increase program size, etc.

- Again, “Premature optimization is the root of all evils”
  - If you are only going to remember one thing from ECE 454, this is it!

- And let the fun begin!