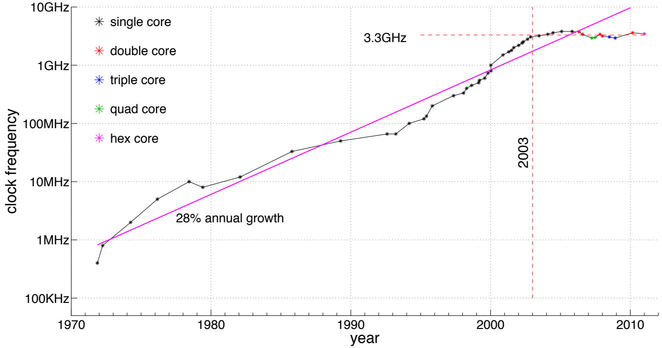
Parallel Computation

Mark Greenstreet

CpSc 448B – Sept. 8, 2011

Outline:

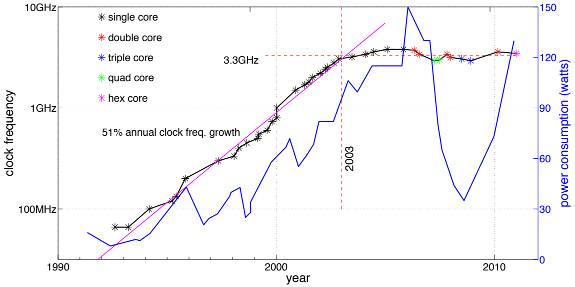
The Clock Plateau



Clock Speed of Intel Processors vs. Year Released[Intel, 2011]

- No significant changes since 2003.

Power is the Problem



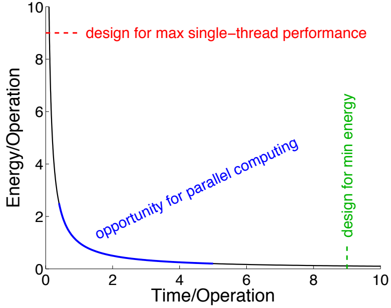
Clock Speed and Power of Intel Processors vs. Year Released[Wikipedia CPU-Power, 2011]

- Designs have been power-constrained since about 2003.
- 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].

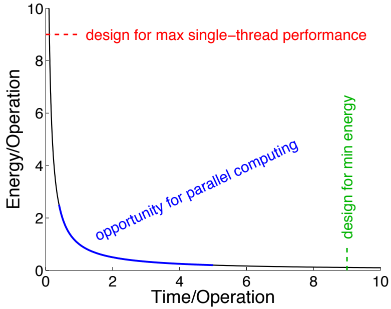
Processor Design Trade-Offs



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.

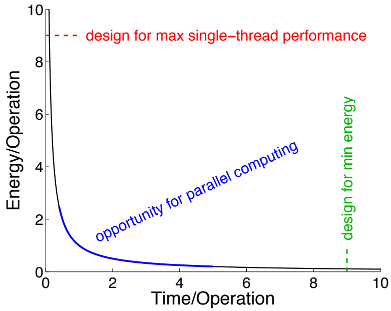
Processor Design Trade-Offs



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 desktop machines
- iPad2: dual-core ARM + dedicated processors for WiFi + codecs, etc.
- Tegra 3 (smart phone chip set): quad-core ARM + GPU + dedicated codecs, etc.
- GPUs
- Clouds
- Embedded computing: automotive, medical, appliances, all kinds of other gadgets.

Outline

Syllabus

- September
  - Parallel Programming in Erlang
  - Basic algorithms and parallel programming techniques
- October
  - Architectures for Parallel Computers
  - Performance Analysis: principles and measurements
  - Midterm: October 20, 2011 – in class
- November
  - Parallel Programming Paradigms:
    - message passing: MPI
    - threads: pthreads and/or Java threads
  - Applications of Parallel programming
- For the complete (but subject to revision) version see:

  http://www.ugrad.cs.ubc.ca/~cs448b/2011-1/syllabus.html

Administrative Stuff – Who

- The instructor
  - **Mark Greenstreet**, mrg@cs.ubc.ca
  - ICICS/CS 323, (604) 822-3065
  - Office hours: Mondays, 11am-12noon.
    - ⋆ 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 TA
  - **Celina Val**, vcelina@cs.ubc.ca
  - Office hours: Tuesdays, 2:00-3:15pm., location TBA.
- Course webpage: http://www.ugrad.cs.ubc.ca/~cs448b.
- Online discussion group: https://www.cs.ubc.ca/groups/2011/cpsc-448b

Administrative Stuff – What

- The book: "Principles of Parallel Programming"
  Calvin Lin and Lawrence Snyder
- web: http://www.ugrad.cs.ubc.ca/~cs448b
- Grades:

Homework:	35%	roughly one assignment every two weeks
Midterm:	25%	October 20, in class
Final:	40%	

- Should programming be done as pairs?
  Class vote.
- Homework late policy:
  - Homework N due on same day as homework $N + 1$ assigned.
    Late homework penalty of 10%/day. No credit after one week.
  - OR, Homework N due one week after homework $N + 1$ assigned.
    No late homework accepted.
  - Class vote.

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/~cs448b/plagiarism.html
  - http://learningcommons.ubc.ca/guide-to-academic-integrity/

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 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 the course discussion group.
  - The first person to post a bug gets the bounty.

Outline

Count 3's in Java

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

Erlang:

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

Erlang Intro – very abbreviated!

- Erlang is a functional language:
  - The main data structures are lists, `[Head | Tail]`, and tuples (covered later).
  - Extensive use of pattern matching.
- `count3s([]) -> 0;`
  An empty list, `[]`, contains `0` threes.
- `count3s([3 | Tail]) -> 1 + count3s(Tail);`
  A list whose first element is a `3` contains one more three than the rest of the list.
- `count3s([_Other | Tail]) -> count3s(Tail);`
  A list whose first element is not a `3` contains the same number of threes as the rest of the list.

Pattern Matching

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

- If a constant (e.g. `[]` or `3`) appears in a pattern, then it matches any expression that evaluates to that value.
- If an unbound variable (e.g. `Tail`) appears in a pattern, then it matches any expression. This binds the value of the expression to the variable, and the variable becomes bound.
- If a bound variable appears in a pattern, then it matches any expression that evaluates to the same value as the one bound to the variable.

Pattern Matching

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

- In Erlang, variable names begin with upper-case letters. Identifiers starting with lower-case letters are constants called 'atoms'. An identifier that starts with an underscore indicates that the value will not be used and avoids compiler warnings about variables whose values are never used.
- Examples:
    - `[3 | Tail]` matches any non-empty list whose first element is `3`. The rest of the list is bound to `Tail`.
    - `[_Other | Tail]` matches any non-empty list. The first element of the list is discarded, and the rest of the list is bound to `Tail`.
- If multiple patterns match an expression, then the first match is selected.

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 **3.7x 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/~cs448b/2011-1/lecture/09.08/code.html

More Erlang Stuff

- Tutorial
  http://www.erlang.org/doc/getting_started/users_guide.html
- Erlang Language Manual
  http://www.erlang.org/doc/reference_manual/users_guide.html
- Erlang Library Documentation
  http://www.erlang.org/doc/man_index.html
- The book: *Programming Erlang: Software for a Concurrent World*, Joe Armstrong, 2007,
  http://pragprog.com/book/jaerlang/programming-erlang

Preview of the next few weeks

September 13: Introduction to Erlang

Homework:	Homework 1 goes out – getting started with Erlang
Reading:	Lin & Snyder, chapter 1

September 15: Quantifying Performance

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

September 20: Matrix multiplication – algorithms

Homework:	Homework 2 goes out – parallel programming with Erlang
Reading:	Lin & Snyder, chapter 3, pp. 68–77

September 22: Matrix Multiplication – performance

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

September 27: Superscalars and compilers

Homework:	Homework 1 due
Reading:	The MIPS R10000 Superscalar Microprocessor (Yeager)

September 29: Shared-memory multi-processors

Reading:	Lin & Snyder Chapter 2, pp. 30–44.

Bibliography

Krst 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 2011.
accessed 26 July 2011.

List of CPU power dissipation.
`http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation`, April 2011.

accessed 26 July 2011.