Faster Computers

- The more we do with computers, the faster we’d like them to be.
- Fortunately, computer engineers have been working hard to satisfy the need for speed.
Moore’s Law
- Usually: Computer power doubles every 18 months (1959 until now!). Must stop eventually.

Fundamental Limits
- Speed is cycle time. Currently about a nanosecond.
- Information travels at the speed of light (about a foot per nanosecond).
- Need smaller chips to get shorter cycle times.
- At a super small scale, quantum randomness starts to dominate the signal (value of a bit).
- Fundamental limit hit in 2010-2020?
Many Hands...

- One way of getting faster and faster computers is to use more (not necessarily faster) computers to do the same work.
- Nice idea, but some tough problems arise.
- **Goal**: Give you a sense of the promise of parallel computing, some of the problems, and some solutions.

Lots of Processors

- Philip Emeagwali won the Gordon Bell Prize for supercomputing in 1989 by using Hillis’ 65,536 processor Connection Machine to do fluid dynamics calculations (3.1 billion ops per sec.)
- Single current Mac G5 (17 years, or about 11 doublings, 2048-fold increase, laster), 2.5 billion cycles per sec.
Google’s Super Computer

• Google won’t tell anyone how many computers they operate.

• Based on financial information released at the time of their IPO, some experts have made estimates.

• They might have 300,000 computers at their disposal.

• Might be more like 100,000. But, still. These computers are a lot more powerful than those on the Connection Machine.

Mythical Man Month

• So, more computers means faster calculations, right?

• Well, not exactly.

• Thought experiment: JK Rowling took about 2 years to write the latest Harry Potter book.

• Can her publisher hire 600 authors and get it in a day?

• You can’t just divide a task (computation) of length N by the number of workers (processors) to get the time to completion.

• Coordination, too.
Fence Painting

• How break up a big task among multiple workers?
• Painting a fence:
  1. open the paint cans
  2. prepare the surfaces
  3. apply the paint
  4. clean the brushes

Fence Painting 2

• Assign everyone a can, a sander, a section of fence, and cleaning materials. Everyone executes all 4 steps on their own part of the fence.
• First kind, “instruction parallelism” (different instructions executed in parallel on the same data).
• Second kind, “data parallelism” (different parts of the data treated in parallel).
Query Example

• Does anyone have a telephone with two or more “5”s in it?
• Think of your telephone number. (If you have more than one, choose one.)
• If it has at least two 5s, raise your hand.
• I’ll see if anyone’s hand is up.
• How about three “5”s?

Python Comparison

• Sequential approach
  • List $l$ of numbers.
  • Loop to see if any has enough 5s.
  • $O(n)$ if $n$ numbers.

```python
def checkAll():
    for x in l:
        if enoughFives(x):
            return True
    return False
```

• Parallel approach
  • Each processor checks one number.
  • Results collected.
  • $O(\log n)$ if $n$ processors.
  • Essentially constant!

```python
reduce(lambda x, y: x or y, [enoughFives(x) for x in l])```
Map and Reduce

• Even this simple example illustrates the two most significant data parallel operations.
• Map: Apply a given function to all data elements. (Example: enoughFives.)
• Reduce: Summarize the results. (Example: “or” them all together.)

MapReduce

• Well known in functional languages, recently implemented on a massive scale at Google.
• Automatically breaks data up across large numbers of machines, executes the “map” function on the machines in parallel, and combines results using “reduce”.
• (Also, separate “combiner” used to partially reduce local copies of the output.)
### Zeitgeist: 2M Queries/day

<table>
<thead>
<tr>
<th>Rank</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>st. patrick’s day</td>
</tr>
<tr>
<td>2.</td>
<td>channing tatum</td>
</tr>
<tr>
<td>3.</td>
<td>v for vendetta</td>
</tr>
<tr>
<td>4.</td>
<td>ncaa</td>
</tr>
<tr>
<td>5.</td>
<td>tara rose mcavoy</td>
</tr>
<tr>
<td>6.</td>
<td>spencer tunick</td>
</tr>
<tr>
<td>7.</td>
<td>natalie portman</td>
</tr>
<tr>
<td>8.</td>
<td>oblivion</td>
</tr>
<tr>
<td>9.</td>
<td>terrell owens</td>
</tr>
<tr>
<td>10.</td>
<td>ides of march</td>
</tr>
<tr>
<td>11.</td>
<td>scientology</td>
</tr>
<tr>
<td>12.</td>
<td>she’s the man</td>
</tr>
<tr>
<td>13.</td>
<td>march madness</td>
</tr>
<tr>
<td>14.</td>
<td>will ferrell</td>
</tr>
<tr>
<td>15.</td>
<td>shannon elizabeth</td>
</tr>
</tbody>
</table>

Gaining Search Queries: Week Ending March 20, 2006

---

### Preparing the Index

- 20 Terabytes (20,000 Gigabytes).
- 5-10 MapReduce operations.
- Handles machine reliability issues, greatly simplifies how algorithms are expressed.
Reduce in Parallel

- Example: Add the first digit of all your phone numbers together.
  - Think of your number.
  - Raise your hand.
  - If your hand is raised, pair yourself with someone else (nearby) with a raised hand.
  - Add your numbers together.
  - One of you put your hand down.
  - Repeat until only one person’s hand is up.

Analysis of Parallel Reduce

- First round, $n$ hands are up.
- Second round, $n/2$ hands are up.
- ...
- Last round, 1 hand is up.

How many rounds?
Making Queries Fast

• How can you search an index of billions of webpages in less than a second?
• Data parallel: Split up the index among thousands of computers.
• Send same query to all of them.
• Summarize the results for the users.
• Embarrassingly parallel.

SETI @ Home

• Radio SETI uses radio telescopes to listen for narrow-bandwidth radio signals from space. Such signals are not known to occur naturally, so a detection would provide evidence of extraterrestrial technology.

• Previous radio SETI projects have used special-purpose supercomputers, located at the telescope, to do the bulk of the data analysis. In 1995, David Gedye proposed doing radio SETI using a virtual supercomputer composed of large numbers of Internet-connected computers, and he organized the SETI@home project to explore this idea. SETI@home was originally launched in May 1999.

• With over 5.2 million participants worldwide, the project is the grid computing project with the most participants to date.

• No aliens have been identified at this time.
Other Good Examples

More examples for which parallelism helps:

- Computer graphics (processor per light ray)
- Weather modeling (processor per local area)
- Independent computations like mailing out phone bills (processor per parallel batch of customers).

Chain Following

Encuentra el hilo para volar la cometa/Find the correct thread to fly the kite

Doesn’t sound parallel, but it is.
Kite String Sequence

- Follow the chain from 14 to either 0, 1, or 2.

Sequential Code

```python
kiteString = [0, 1, 2, 11, 1, 10, 8, 2, 7, 5, 0, 4, 9, 6, 3]
i = 14
while i > 2:
    i = kiteString[i]
print i
```

- Initially 14, stops when 0, 1, or 2.
- How many times through the loop?
- Proportional to length of the chain (n, say).

• $i$ holds the position of where we've traced the string to so far.
Parallel Idea

- One processor for each position along the string.
- Initially, know where you point to in one hop.
- In one step, can figure out where you point to in 2 hops.
- If this information is shared, in one more step, can figure out where you point to in 4 hops.
- Then 8, then 16...

Trace All Strings

<table>
<thead>
<tr>
<th>index position</th>
<th>threads</th>
<th>free ends</th>
<th>kite</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>3 4 5</td>
<td>6 7 8</td>
<td>9 10 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kiteString</th>
<th>one hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 11 1 10 8 2 7 5 0 4 9 6 3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>two hops: shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 4 1 0 7 2 2 10 0 1 5 8 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>four hops: double shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 1 1 0 2 2 2 0 0 1 0 2 1</td>
</tr>
</tbody>
</table>
Python Version

```python
kiteString = [0,1,2,11,1,10,8,2,7,5,0,4,9,6,3]
twoHops = [kiteString[i] for i in kiteString]
fourHops = [twoHops[i] for i in twoHops]
fourHops[14]
```
```
>>> 1
```

- In two parallel operations, compute where the string takes us in four hops.
- In general, if strings have length $n$, only $\lg n$ updates needed.
- In practice, parallel lookups are tricky, though.

Next Time

- Hillis Chapter 6.