Algorithms on Multicores
The Advent of Multicores

Great moments in evolution
Moore’s Law

Clock speed flattening sharply

Transistor count still rising

- Clock Speed (MHz)
- Transistors (000)
Parallelism Everywhere
The Multicore Processor

Multiple processors - all on the same chip

Sun T2000 Niagara

Core/Processor
Why is Kunle Smiling?

T2000
Niagara
Traditional Scaling Process

- Speedup:
  - 1.8x
  - 3.6x
  - 7x

- User code:
  - Traditional
  - Uniprocessor

- Time: Moore’s law
Ideal Scaling Process

Unfortunately, not so simple…
Parallelization requires coordination which requires great care to scale…
Sequential Computation

memory

thread

object

object
Concurrent Computation

threads

memory

object

object

Art of Multiprocessor Programming
Asynchrony

- Sudden unpredictable delays
  - Cache misses (*short*)
  - Page faults (*long*)
  - Scheduling quantum used up (*really long*)
Parallel Primality Testing

• Challenge
  – Print primes from 1 to $10^{10}$

• Given
  – Ten-processor multicore machine
  – One thread per processor

• Goal
  – Get ten-fold speedup (or close)
Load Balancing

- Split the work evenly
- Each thread tests range of $10^9$
Procedure for Thread $i$

```c
void primePrint {
    int i = ThreadID.get(); // IDs in {0..9}
    for (j = i*10^9+1, j<(i+1)*10^9; j++) {
        if (isPrime(j))
            print(j);
    }
}
```
Issues

• Higher ranges have fewer primes
• Yet larger numbers harder to test
• Thread workloads
  – Uneven
  – Hard to predict
Issues

• Higher ranges have fewer primes
• Yet larger numbers harder to test
• Thread workloads
  – Uneven
  – Hard to predict
• Need *dynamic* load balancing
• Threads need to coordinate…
Shared Counter

each thread takes a number
Procedure for Thread $i$

```java
int counter = new Counter(1);

void primePrint {
    long j = 0;
    while (j < 10^{10}) {
        j = counter.getAndIncrement();
        if (isPrime(j))
            print(j);
    }
}
```
Counter counter = new Counter(1);

void primePrint {
    long j = 0;
    while (j < 10^{10}) {
        j = counter.getAndIncrement();
        if (isPrime(j))
            print(j);
    }
}
Where Things Reside

```c
void primePrint {
    int i = ThreadID.get(); // IDs in {0..9}
    for (j = i*10^9 + 1, j<(i+1)*10^9, j++) {
        if (isPrime(j))
            print(j);
    }
}
```
Procedure for Thread $i$

Counter counter = new Counter(1);

void primePrint {
    long j = 0;
    while (j < $10^{10}$) {
        j = counter.getAndIncrement();
        if (isPrime(j))
            print(j);
    }
}
Procedure for Thread $i$

Counter counter = new Counter(1);

void primePrint {
    long j = 0;
    while (j < $10^{10}$) {
        j = counter.getAndIncrement();
        if (isPrime(j))
            print(j);
    }
}
public class Counter {
    private long value;

    public long getAndIncrement() {
        long temp = value;
        value = temp + 1;
        return temp;
    }
}
Counter Implementation

```java
public class Counter {
    private long value;

    public long getAndIncrement() {
        long temp = value;
        value = temp + 1;
        return temp;
    }
}
```

OK for single thread, not for concurrent threads
Not so good…

Value…

<table>
<thead>
<tr>
<th>Value</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>read 1</td>
</tr>
<tr>
<td></td>
<td>write 2</td>
</tr>
<tr>
<td>2</td>
<td>read 2</td>
</tr>
<tr>
<td>3</td>
<td>write 3</td>
</tr>
<tr>
<td></td>
<td>write 2</td>
</tr>
</tbody>
</table>
Is this problem inherent?

If we could only glue reads and writes together…
public class Counter {
    private long value;

    public long getAndIncrement() {
        temp = value;
        value = temp + 1;
        return temp;
    }
}

Challenge

public class Counter {
    private long value;

    public long getAndIncrement() {
        temp  = value;
        value = temp + 1;
        return temp;
    }
}
Challenge: 6.006 Algorithms and Data-Structures on Multicores

Concurrent Hash Table

Concurrent Heap

Concurrent AVL-Tree
6.816
Multicore Programming
Spring 2014
Algorithm most useful for cryptography: repeated squaring

\[
\begin{cases}
    1 & \text{if } b = 0, \\
    (a^{b/2})^2 & \text{if } b \text{ even}, > 0, \\
    a \cdot (a^{b-1}) & \text{if } b \text{ odd}
\end{cases}
\]

Values in parentheses are recursive calls.

Work modulo \( n \) (or modulo \( p \)) for modulus \( n \) (or \( p \)).

Time = \( O(\log(b) \cdot M(d)) \)

Where \( a \) and modulus \( n \) or \( p \) are \( d \)-bit integers.

\( M(d) \) = time to do (modular) multiplication of two \( d \)-bit values.

Can do for 2048-bit \( a, b, n \) in milliseconds!

Used everywhere in crypto...

Example: primality testing

Fermat's Little Theorem:

\[
p \text{ odd prime } \implies 2^{p-1} \equiv 1 \pmod{p}
\]

Converse:

\[
\text{is almost always true}
\]

In practice, to test a large (e.g., 1024-bit) random number \( p \) for primality, it suffices to test whether

\[
2^{p-1} \equiv 1 \pmod{p}
\]

(chance of error is negligible for practical purposes)
Example: RSA public-key cryptosystem

\( p, q \leftarrow \text{large random primes} \)
\( n = p \cdot q \)
\( e \leftarrow \text{public exponent} \) (details omitted, see text)
\( d \leftarrow \text{private exponent} \)

Encryption: \( C = M^e \pmod{n} \)
Decryption: \( M = C^d \pmod{n} \)

Lemons \( \Rightarrow \) Lemonade

Some problems seem to have no efficient algos (lemons)

For crypto, this is good - we force the adversary to try and solve them (lemonade!)

Factoring: given \( n = p \cdot q \), find \( p \) & \( q \).

No poly-time algorithm known ((inplace)

Protects RSA secret key \((p, q, d)\)

Best running time is
\[ \Theta \left( \exp \left( \frac{64}{9} \sqrt[3]{b^2} \ln b \right) \right) \]

for number \( n \) of \( b \) bits in length; exponential in cube root of length \( d \cdot n \), roughly.

(Record is 768 bits)

Take 6.857 - Crypto & Security
Erik's main research areas:
- computational geometry
- geometric folding algorithms
- self-assembly
- data structures
- graph algorithms
- recreational algorithms
- algorithmic sculpture

Geometric folding algorithms:
- design: algorithms to fold any polyhedral surface from a square of paper
  [Demaine, Demaine, Mitchell 2000; Demaine & Tachi 2011]
- bicolor paper \( \Rightarrow \) can 2-color faces
- OPEN: how to best optimize "scale factor"
- e.g. best \( n \times n \) checkerboard folding
  recently improved from \( n^{1/2} \) to \( n^{3/4} \)
- foldability: given a crease pattern, can you fold it flat?
- \( \text{NP-complete in general} \) [Berm&Hayes 1996]
- OPEN: \( m \times n \) map with creases specified as mountain/valley
  [Edmonds 1997]
- recently solved: \( 2 \times n \) [Demaine, Liu, Morgan 2012]
- hyperbolic paraboloid [Bauhaus 1929]
  doesn't exist!
  [Demaine, Demaine, Hart, Price, Tachi 2009]
- understanding circular creases
  [Demaine, Demaine, Lubiw 1998; Bern, Demaine, Eppstein, Hayes 1999]

Self-assembly: geometric model of computation
- glue: e.g. DNA strands, each pair has strength
- square tiles with glue on each side
- Brownian motion: tiles/constructions stick together if \( \sum \) glue strengths \( \geq \) temperature

- can build \( n \times n \) square using \( O(\log n) \) tiles
  [Rothenmund & Winfree 2008]
  or using \( O(1) \) tiles \& \( O(\log n) \) “stages”
  algorithmic steps by the bioengineer?
  [Demaine, Demaine, Fekete, Ishaque, Raffinot, Schweller, Souvaine 2007]
- can replicate \( \omega \) copies of given unknown
  shape using \( O(1) \) tiles \& \( O(1) \) stages
  [Abel, Benbernou, Damian, Demaine, Demaine, Flatland, Kominers, Schweller 2010]
Data structures: [6.851, next semester & videos]
- integer data structures: store \( n \) integers in \([0, 1, \ldots, u-1]\) subject to insert, delete, predecessor, successor (on word RAM)
- hashing does exact search in \( O(1) \)
- AVL trees do all in \( O(\log u) \)
- \( O(\log \log u) / \text{op.} \) [van Emde Boas]
- \( O(\sqrt{\log u}) / \text{op.} \) [fusion trees: Fredman & Willard]
- \( O(\sqrt[4]{\log u}) / \text{op.} \) [min of above]

- cache-efficient data structures:
  - memory transfers happen in blocks

\[
\text{CACHE} \quad \downarrow \quad \text{DISK/MAIN MEMORY}
\]

- searching takes \( \Theta(\log_B N) \) transfers (vs. \( \log n \))
- sorting takes \( \Theta(\frac{N}{B} \log_c \frac{N}{B}) \) transfers

- possible even if you don’t know \( B \) & \( C \)!
Almost planar graphs: [6.889, videos online]
- Dijkstra in $O(n)$ time
  [Henzinger, Klein, Rao, Subramanian 1997]
- Bellman-Ford in $O(n \log n / \log \log n)$ time
  [Mozes & Wolff-Nilson 2010]

- many problems
  NP-hard, even on planar graphs
- but can find a solution within
  $1+\varepsilon$ factor of
  optimal, for any $\varepsilon$
- run BFS from
  any root vertex $r$
- delete every $k$ layers
  (guess initial offset)
- for many problems, solution messed up
  by only $1 + \frac{1}{k}$ factor ($\Rightarrow k = \frac{1}{\varepsilon}$)
- connected components of remaining graph
  have $\leq k$ layers $\sim$ can solve via DP
  typically in $\sim 2^k \cdot n$ time

[Baker 1994 & others]
Recreational algorithms:
- many algorithms & complexities of games [some in SP. 268 & our book Games, Puzzles, & Computation (2009)]
- $n \times n \times n$ Rubik’s Cube diameter is $O(n^{2.5} \log n)$ [Demaine, Demaine, Eisenstat, Lubiw, Winslow 2011]
- Tetris is NP-complete [Breukelaar, Demaine, Hohenberger, Hoogeboom, Kusters, Liben-Nowell 2004]
- Super Mario Bros., Legend of Zelda, Donkey Kong Country, Pokémon NP-complete [Alampis, Demaine, Guo 2012]
- balloon twisting any polyhedron [Demaine, Demaine, Hart 2008]
- algorithmic magic tricks
  - coin flipping [Bembenek, Demaine, Demaine, Rossman 2008]
  - picture hanging [Demaine, Demaine, Minsky, Mitchell, Rivest, Pătraşcu 2012]
Algorithms classes at MIT:  (post-6.006)
- #1: 6.046: Intermediate Algorithms (more adv. algorithms & analysis, less coding)
- 6.047: Computational Biology (genomes, phylogeny, etc.)
- 6.854: Advanced Algorithms (intense survey of whole field)
- 6.850: Geometric Computing (working with points, lines, polygons, meshes, ...) 
- 6.849: Geometric Folding Algorithms (origami, robot arms, protein folding, ...)
- 6.851: Advanced Data Structures (sublogarithmic performance)
- 6.852: Distributed Algorithms (reaching consensus in a network with faults)
- 6.853: Algorithmic Game Theory (Nash equilibria, auction mechanism design, ...)
- 6.855: Network Optimization (optimization in graph: beyond shortest paths)
- 6.856: Randomized Algorithms (how randomness makes algs. simpler & faster)
- 6.857: Network and Computer Security (cryptography)
- 6.816: Multicore Programming
Other theory classes:
- 6.045: Automata, Computability, & Complexity
- 6.840: Theory of Computing
- 6.841: Advanced Complexity Theory
- 6.842: Randomness & Computation
- 6.845: Quantum Complexity Theory
- 6.440: Essential Coding Theory
- 6.441: Information Theory