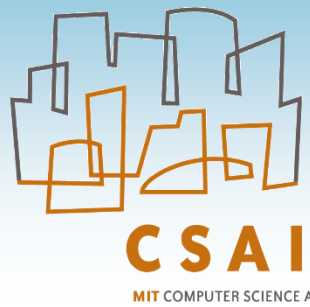


FROM

MODERN ALGORITHMS WORKSHOP

Parallel Algorithms

Prof. Charles E. Leiserson

Dr. Tao B. Schardl

September 19, 2018

Outline

- Introduction
- Cilk Model
- Detecting Nondeterminism
- What Is Parallelism?
- Scheduling Theory Primer
- *Lunch Break*
- Analysis of Parallel Loops
- Case Study: Matrix Multiplication
- Case Study: Jaccard Similarity
- Post-Moore Software

Software Properties

What software properties are more important than performance?

- Compatibility
- Correctness
- Clarity
- Debuggability
- Functionality
- Maintainability
- Modularity
- Portability
- Reliability
- Robustness
- Solution quality
- Usability

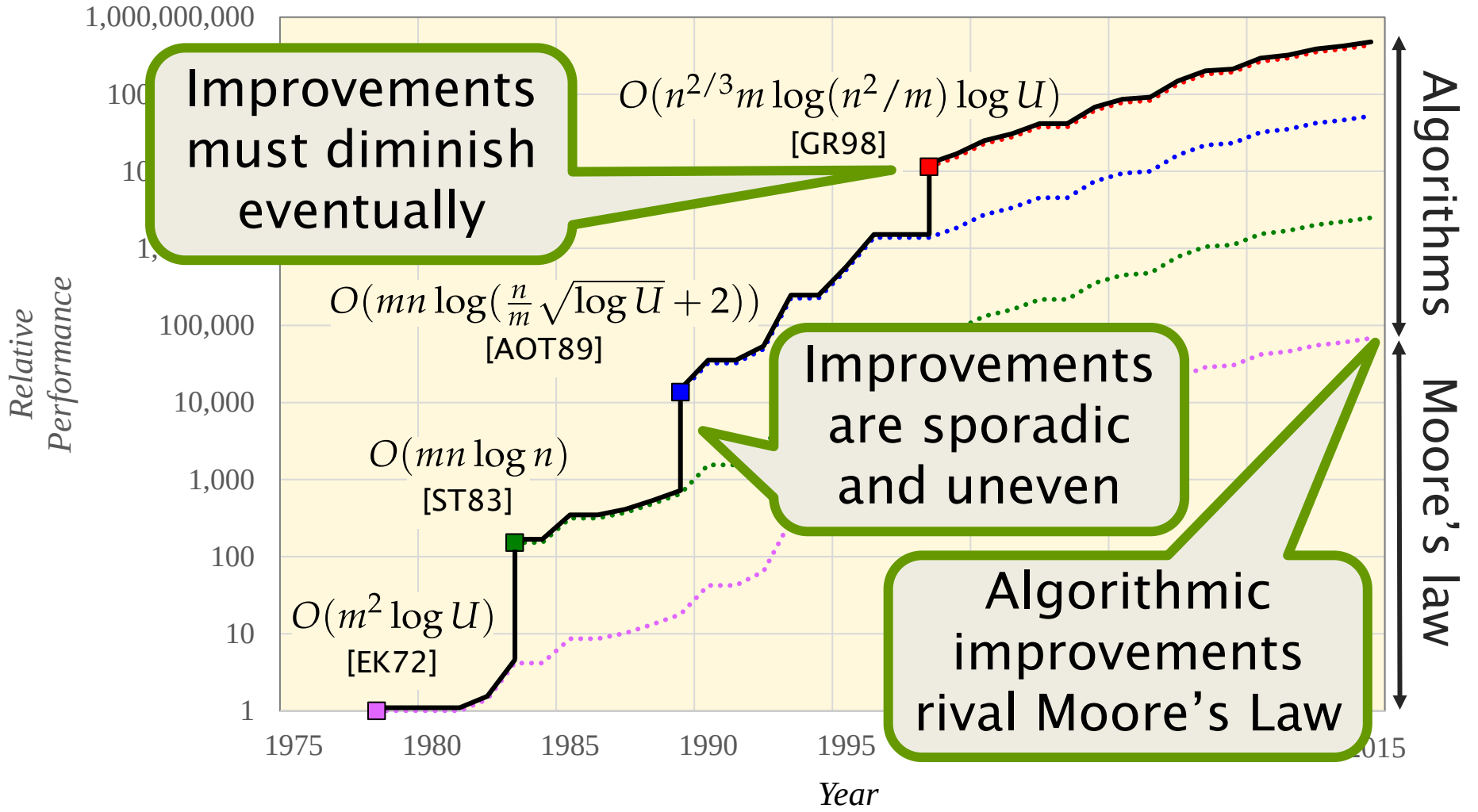
… and more.

If programmers are willing to sacrifice performance for these properties, why study performance?

Performance is the currency of computing. You can often "buy" needed properties with performance.

Study: Improvements to Max Flow

What performance improvements can algorithms offer? Improvements to **max flow** provide an example.



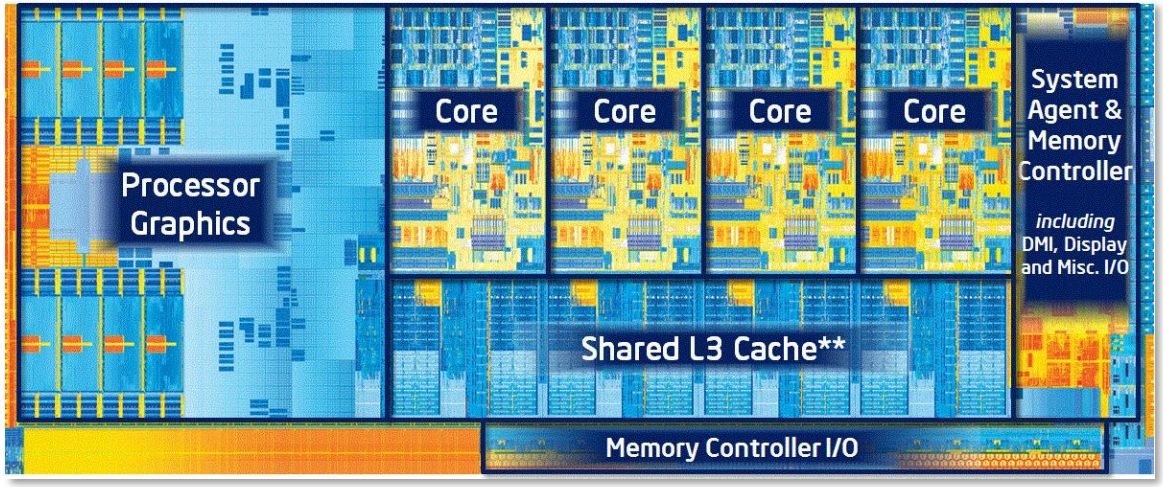
Future Algorithmic Improvements

Key ways algorithmic performance engineering will speed up computing:

- Attacking new problem domains.
- Addressing scalability concerns.
- Tailoring algorithms to take advantage of modern hardware.

How much can one get from tailoring algorithms?

2012 Intel Ivy Bridge



Matrix-Multiplication Python Benchmark

```
import sys, random
from time import *

n = 4096

A = [[random.random() for row in xrange(n)] for col in xrange(n)]
B = [[random.random() for row in xrange(n)] for col in xrange(n)]
C = [[0 for row in xrange(n)] for col in xrange(n)]

start = time()
for i in xrange(n):
    for j in xrange(n):
        for k in xrange(n):
            C[i][j] += A[i][k] * B[k][j]
end = time()

print '%0.6f' % (end - start)
```

Joint study with Bradley Kuszmaul of CSAIL based on an idea originally due to Saman Amarasinghe of CSAIL.

Quiz: 4k x 4k Matrix-Multiplication

How long does this code take to execute?

Machine: Amazon AWS c4.8xlarge

- Dual-socket Intel Xeon E5-2666 v3 (Haswell)
- 18 cores, 2.9 GHz, 60 GiB DRAM

Python matrix-multiplication kernel

```
n = 4096
start = time()
for i in xrange(n):
    for j in xrange(n):
        for k in xrange(n):
            C[i][j] += A[i][k] * B[k][j]
end = time()
```

a. 6 milliseconds
b. 6 seconds
c. 6 minutes
d. 6 hours
e. 6 days

Algorithmic Tailoring of Matrix Multiplication

Version	Implementation	Running time (s)	Relative speedup	Absolute Speedup	GFLOPS	Percent of peak
1	Python	21041.67	1.00	1	0.006	0.001
2	Java	2387.32	8.81	9	0.058	0.007
3	C	1155.77	2.07	18	0.118	0.014
4	+ interchange loops	177.68	6.50	118	0.774	0.093
5	+ optimization flags	54.63	3.25	385	2.516	0.301
6	Parallel loops	3.04	17.97	6,921	45.211	5.408
7	+ tiling	1.79	1.70	11,772	76.782	9.184
8	Parallel divide-and-conquer	1.30	1.38	16,197	105.722	12.646
9	+ compiler vectorization	0.70	1.87	30,272	196.341	23.486
10	+ AVX intrinsics	0.39	1.76	53,292	352.408	41.677

Machine: Amazon AWS c4.8xlarge

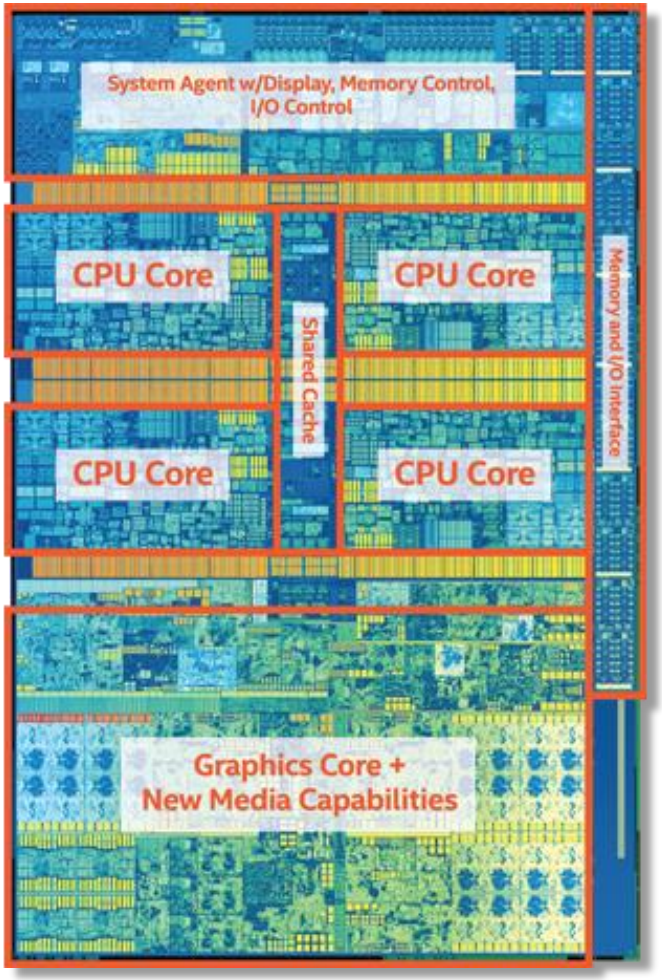
- Dual-socket Intel Xeon E5-2666 v3 (Haswell)
- 18 cores, 2.9 GHz, 60 GiB DRAM

Performance Engineering Is Still Hard

A modern multicore desktop processor contains parallel-processing cores, vector units, caches, prefetchers, GPU's, hyperthreading, dynamic frequency scaling, etc., etc.

How can we write software to utilize modern hardware efficiently?

Today's focus: Parallel-processing cores



2017 Intel 7th-generation desktop processor