Parallelism in Haskell

Ben Sherman

February 26, 2014

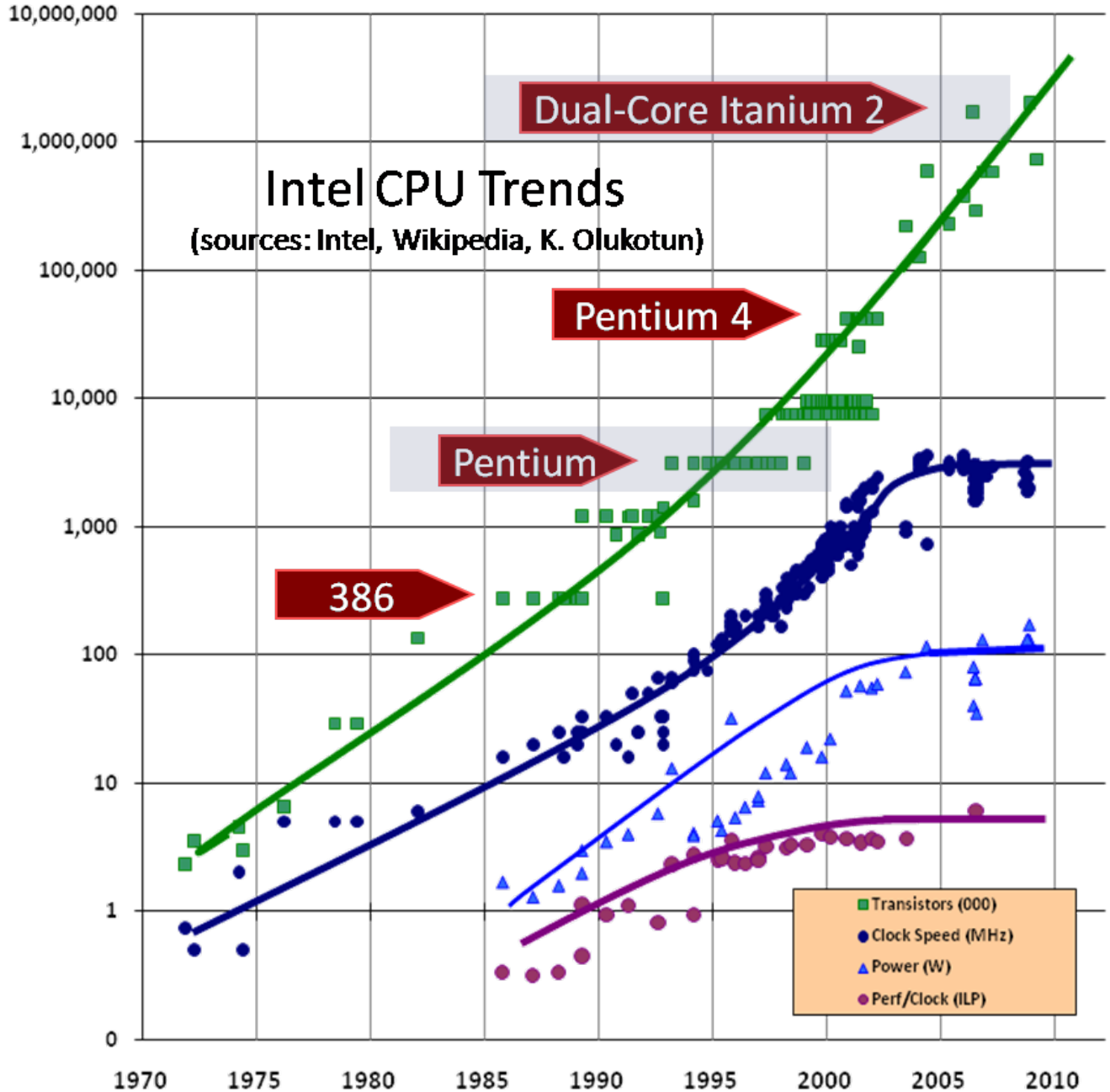
Historical trends

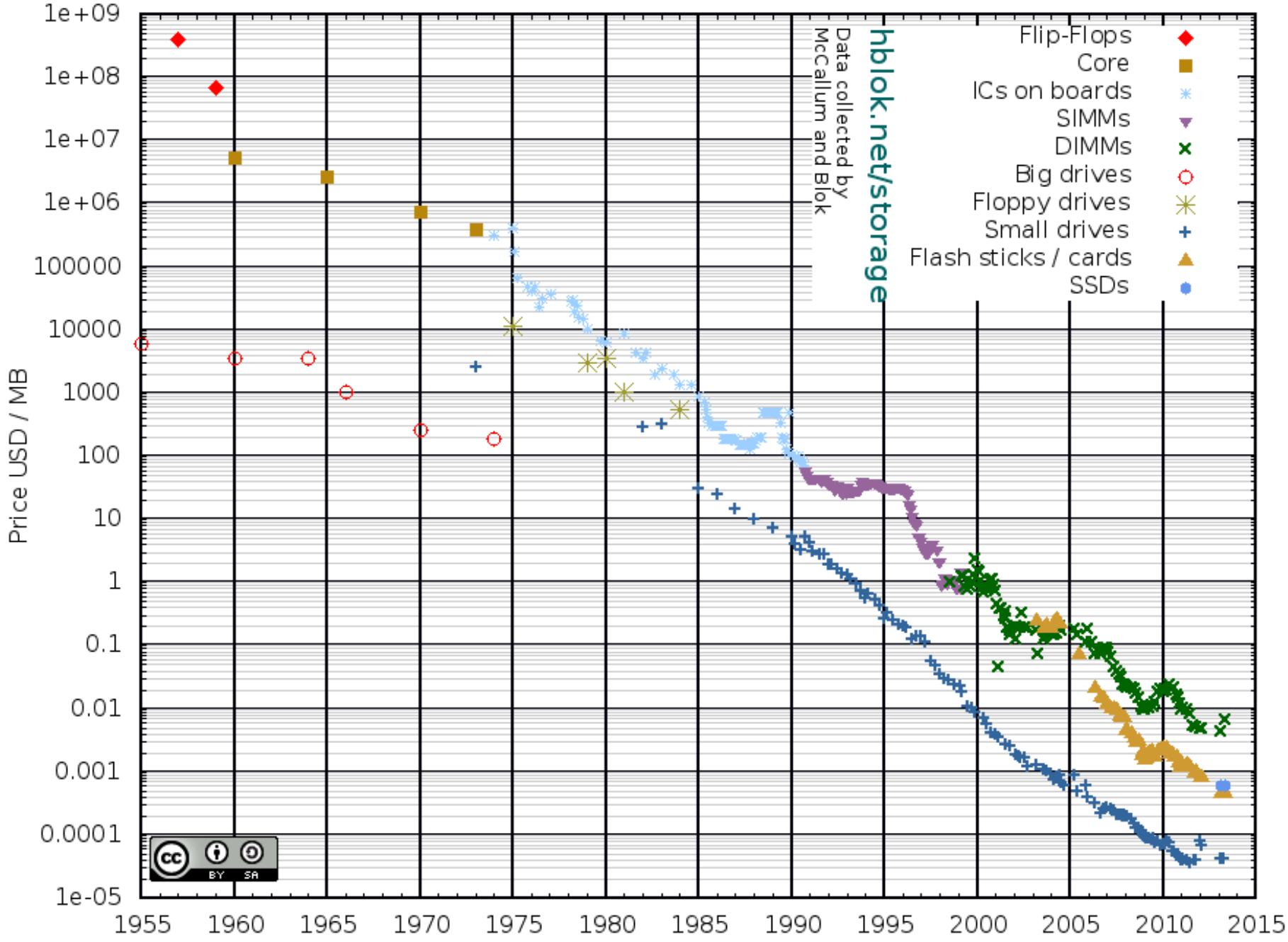
Moore's law:

The number of transistors on a chip (with cost held constant) doubles every 18 months.

Alive and kicking!

http://www.gotw.ca/publications/concurrency-ddj.htm




Historical Cost of Computer Memory and Storage
Price USD / MB
1e+09
1e+08
1e+07
1e+06
100000
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
1e-05
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
Data collected by McCallum and Blok
hblok.net/storage
Flip-Flops
Core
ICs on boards
SIMMs
DIMMs
Big drives
Floppy drives
Small drives
Flash sticks / cards
SSDs
BY SA

The need for parallelism

- Size of memory / hard disk storage is increasing exponentially
- Number of transistors on a chip is increasing super-exponentially
- CPU clock speed has stagnated since 2004
  - Hard physical barriers prevent increase in clock speed

Until 2004, we were getting *faster* processors. Now, we're only getting *more* processors.

To process the ever-increasing amount of information we have, we need to process in *parallel*.

Parallelism requires new thinking

- Not all procedures can be parallelized (e.g., access the last element of a linked list)
- Identifying what can be parallelized can be difficult
- Parallelism implies concurrency, which means we need shared memory and processor synchronization
- Must ensure *correctness* of the parallel algorithm

Imperative vs. Declarative

	Imperative	Declarative
In one word...	doing	being
languages	C, ...	Haskell, ...
fundamental unit	statement	expression

Parallelism, imperatively

- We must rearrange statements to run in parallel
  - Fundamentally changes our code
  - Therefore, we must always ask, “is this still correct?”
- Must worry about synchronization and atomic access to shared memory

Parallelism, declaratively

- Our code describes what our answer *is*, not how to compute it
- Therefore, it is the compiler's job, not ours, to ensure correctness of the parallel algorithm, and to handle synchronization
- We just give "hints" to the compiler (which do not affect the result) to say where it may be beneficial to evaluate things in parallel

Simon Peyton Jones' presentation

`http://research.microsoft.com/~simonpj/papers/parallel/Parallel-Haskell.pdf`

Lazy evaluation in Haskell

- ghci's $:\mathrm{sprint}$
- Debug.Trace (a.k.a., unsafePerformIO + seq)

Weak head normal form

`http://www.haskell.org/haskellwiki/Weak_head_normal_form`

The *parallel* library

Parallel folds

```
λ> foldr (+) 0 [1,2,3,4,5]
```

n queens

http://en.wikipedia.org/wiki/Eight_queens_puzzle

n queens goals

- return a data structure representing all possible positions of queens
- allow this data structure to be computed in parallel
- Bonus:
  - Allow the user to compute only *parts* of the answer, in parallel, without wasting computation (e.g., all positions where the first two queens are at [1,4])
  - If n queens has already been computed, we should be able to compute $n + 1$ queens quicker than we would previously