Big-$\mathcal{O}$ Ain't What it Used to Be

unix weapons school

Asymptotic notation review I

Asymptotic analysis gives us a means of speaking of arbitrarily large growth, independently of arbitrarily (but finitely) large costs not associated with problem size.

Notation	Name	Definition	Introduced
$f(n) \in \mathcal{O}(g(n))$	Big O	$\exists k > 0, \exists n_0, \forall n$ $n > n_0 \implies f(n) \leq g(n) * k$	Paul Bachmann (1894)
$f(n) \in o(g(n))$	Small O	$\forall k > 0, \exists n_0, \forall n$ $n > n_0 \implies \lvert f(n)\rvert \leq \lvert g(n)\rvert * k$	Edmund Landau (1909)
$f(n) \in \Theta(g(n))$	Big Theta	$\exists k_1 > 0, \exists k_2 > 0, \exists n_0, \forall n$ $n > n_0 \implies g(n) * k_1 \leq f(n)$, $f(n) \leq g(n) * k_2$	Donald Knuth (1976)
$f(n) \in \omega(g(n))$	Small Omega	$\forall k > 0, \exists n_0, \forall n$ $n > n_0 \implies \lvert f(n)\rvert \geq \lvert g(n)\rvert * k$	Donald Knuth (1976)
$f(n) \in \Omega(g(n))$	Big Omega	$\exists k > 0, \exists n_0, \forall n$ $n > n_0 \implies f(n) > g(n) * k$	Donald Knuth (1976)

Advances in (finite) computing technology can only reduce these ignored costs.

Wrap the earth with your register file, and still there will be numbers so large that their addition is $\Theta(n)$.

Asymptotic notation review II

Class (all $c_* > 1$)	**Name**	**Example**
$\mathcal{O}(1)$	Constant	Is word-sized unsigned int n a power of 2?
$\mathcal{O}(\lg_{c_1} \lg_{c_2} n)$	Double-log	Interpolative search on uniform distribution
$\mathcal{O}(\lg_c n)$	Logarithmic	Binary search
$\mathcal{O}(n \lg n) = \mathcal{O}(\lg n!)$	Linearithmic	FFT
$\mathcal{O}(n^c)$	Polynomial	Primality testing
$\mathcal{O}(c^n)$	Exponential	Brute-force Boolean equivalence
$\mathcal{O}(n!)$	Factorial	Unrestricted permutations of a poset
$\mathcal{O}(c_1^{c_2^n})$	Double-exp	Presburger arithmetic decision best case

Speaking of still faster growth rates[1] (hyper-exponential, $\mathcal{A}$) is mostly zoology.

[1] Check out "fast-growing hierarchies" and the LöbWainer hierarchy.

It's the constants, stupid

Algorithmic choices can dominate performance, especially at scale. By the definition of Big O, it should also be obvious that an asymptotically superior algorithm can be slower for small inputs[2].

That said, no one's going to think implementing a routing table with a linked list is a good idea.

Furthermore, asymptotic analysis speaks of performance as problem size grows. It doesn't speak of real-time. It doesn't speak of bounded memories. We rarely speak of piecewise asymptotics.

But, by all means, do ensure you're not doing linear searches on sorted data etc.

[2] We will see that small inputs can be surprisingly large.

What's hiding behind $\mathcal{O}$?

Naive square (nXn X nXn) matrix multiplication is $\Theta(n^3)$.

$$C = AB \implies C_{ij} = \sum_{m=1}^{k} A_{im} B_{mj} \tag{1}$$

Counting the explicit additions and multiplications, there are precisely $2n^3$ "operations".

Fused multiply-add

IEEE 754-2008 floating point support requires FMA, fused multiply-add. Let $rn()$ denote a rounding operation. Typically, a multiply-add chain requires two instructions, and rounds twice:

$$MAC(A, B, C) = rn(rn(A * B) + C) \tag{2}$$

Fused multiply-add rounds only once, preserving the fully precise product in an internal register:

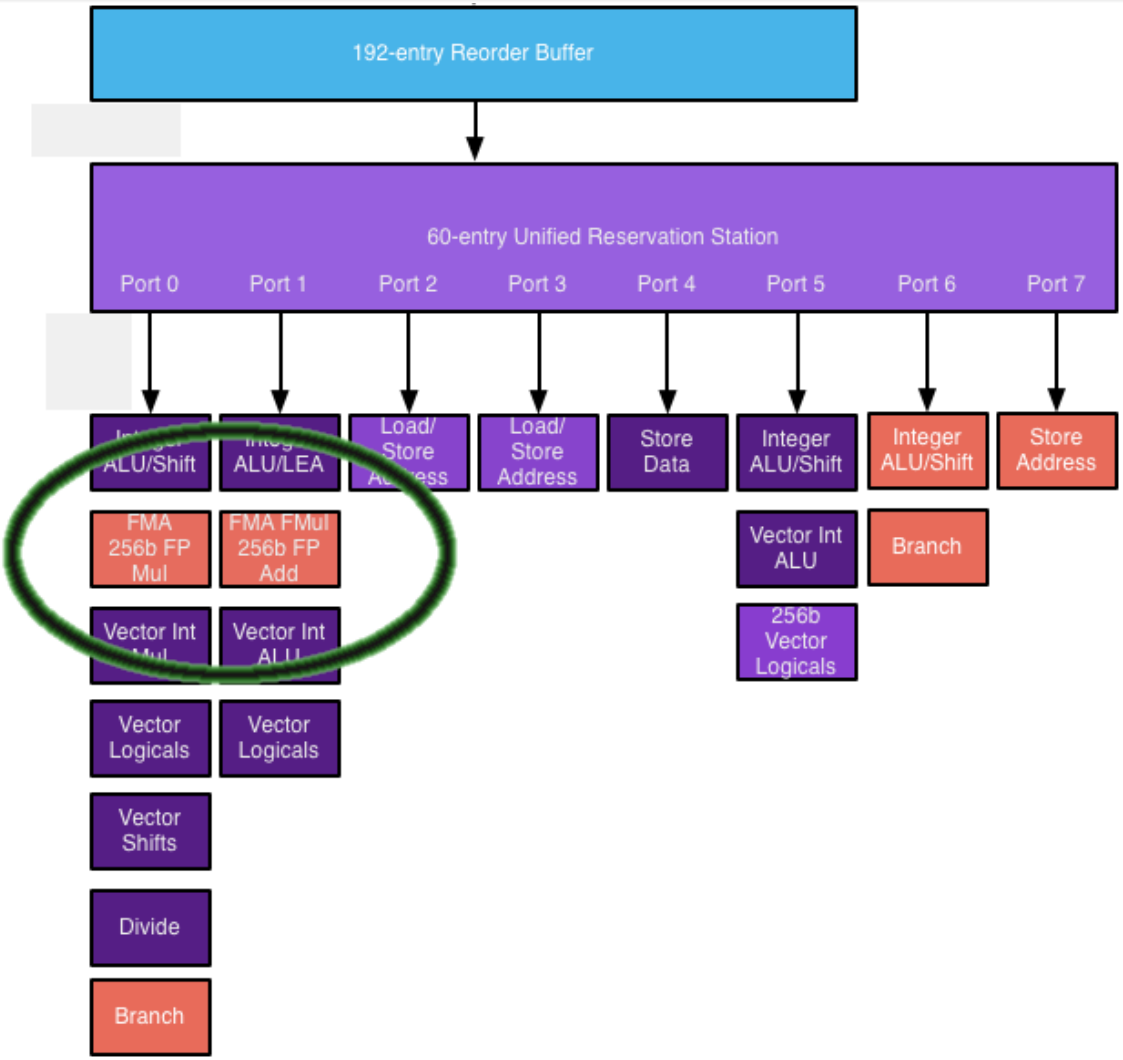
$$FMA(A, B, C) = rn(A * B + C) \tag{3}$$

AMD's FMA4 (Bulldozer) implements a fully general SIMD FP FMA. Intel's FMA3 (Haswell, as part of AVX2; also in AMD's Piledriver) implements a destructive SIMD FP FMA[3]. The throughput and latency are equivalent to standard single SIMD FP adds and multiplies. NVIDIA's Fermi likewise introduced a full-throughput FMA.

There are now precisely n^3 "operations".

[3] AMD's XOP further implements an SIMD integer FMA.

Wide issue

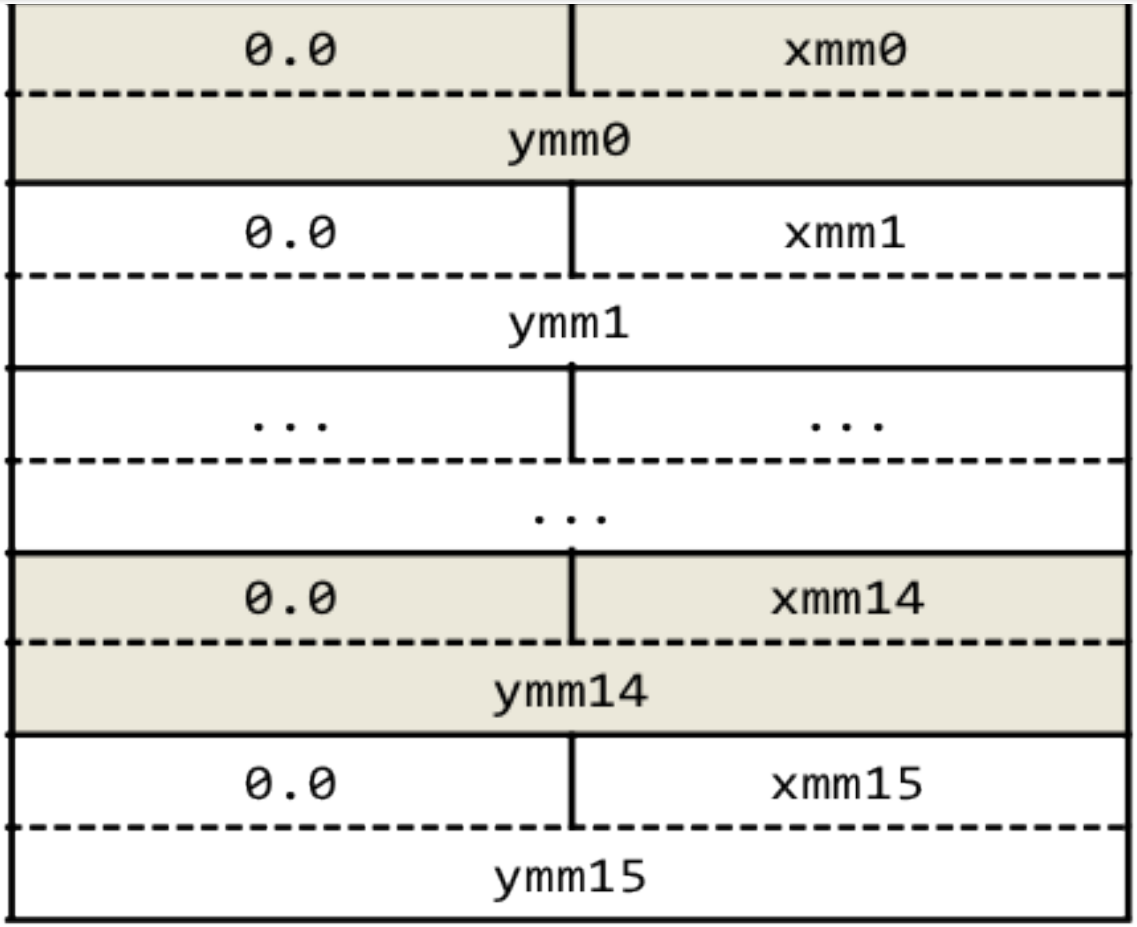


Haswell can issue and retire 2 VFMADD* instructions per cycle.

There are now precisely $n^3/2$ "operations"[4].

[4] Assuming that two operations are available every cycle.

SIMD

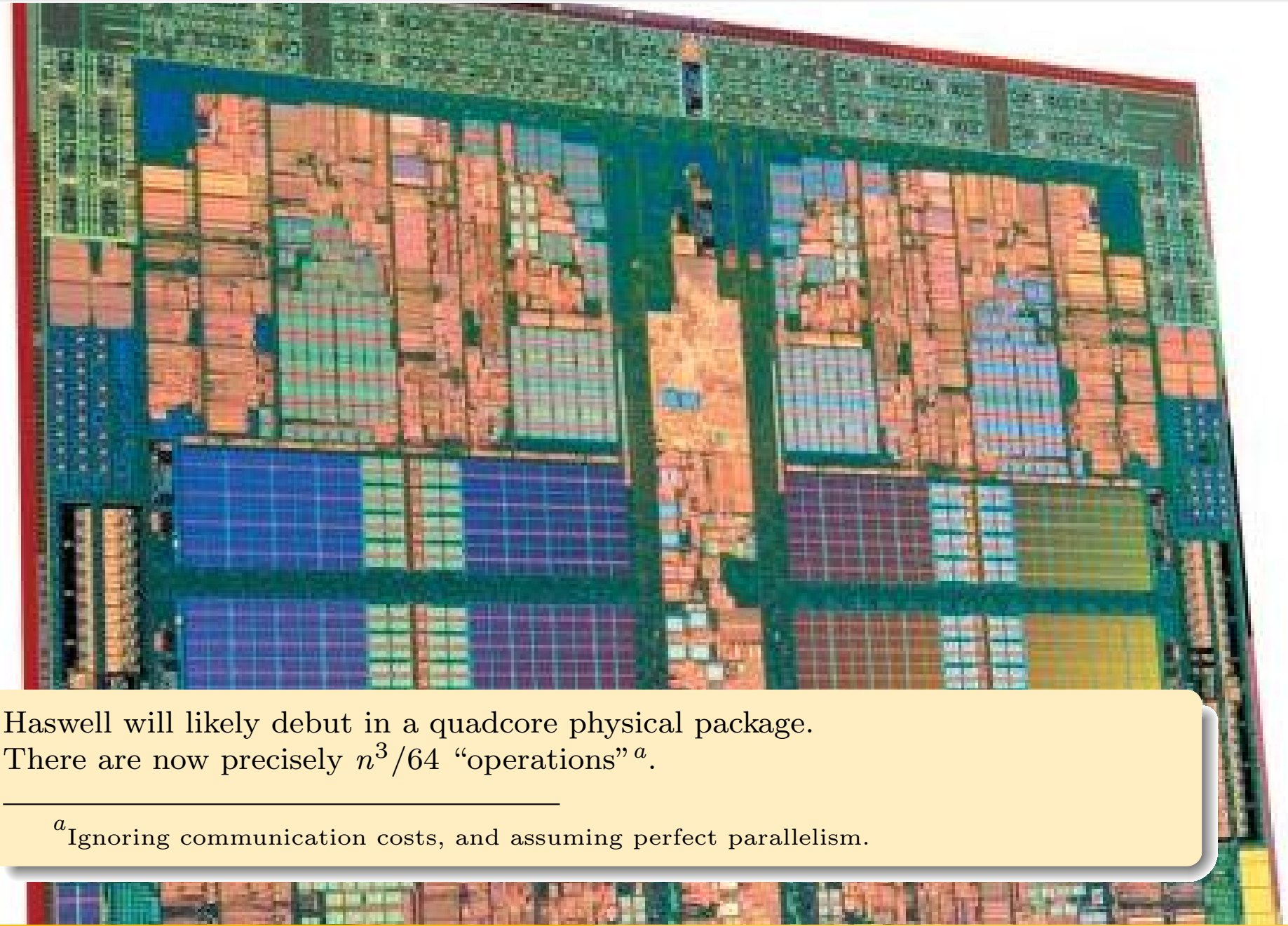


AVX uses the 16 256-bit YMM registers. There are 8 32-bit IEEE 754-2008 single-precision values in a 256-bit input.

There are now precisely $n^3/16$ "operations"[5].

[5] Assuming that values are usable in 256-bit chunks.

Multicore



Haswell will likely debut in a quadcore physical package.
There are now precisely $n^3/64$ "operations"[a].

[a] Ignoring communication costs, and assuming perfect parallelism.

Memory accesses

1. $\forall i, 0 \leq i \leq n$: Read row i from A into a
2. $\forall j, 0 \leq j \leq n$: Read column j from B into b, read C_{ij} into c
3. $\forall k, 0 \leq k \leq n$: Store $a_k * b_k + c$ into C_{ij}

1. n^2 loads from A
2. n^3 loads from B
3. n^2 loads from C
4. n^2 stores to C

Counting the loads and stores, there are precisely $n^3 + 3n^2$ memory accesses. There are now precisely $\frac{65n^3}{64} + 3n^2$ "operations".

Arithmetic intensity

$$\lim_{n \to \infty} \frac{\text{Arithmetic}}{\text{Memory}} = 2$$

Loads and stores

Hong and Kung proved in 1981 that any schedule of conventional matrix multiplication must transfer $\Omega(\frac{n^3}{\sqrt{Z}})$, $Z < \frac{n^2}{6}$ words between slow and fast memory. *Tiling* is the optimal strategy.

Of course, AVX's VMOVAPS moves 256 bits, or 8 32-bit single precision floating point words, at a time. And there's 4 cores. With two load/store pipes each. So that's $\Omega_{HK}/64$[6].

Of course, we're not going to be able to pack two VFMADDPS and two VMOVAPS instructions into every 16B/**c** I$ fetch[7].

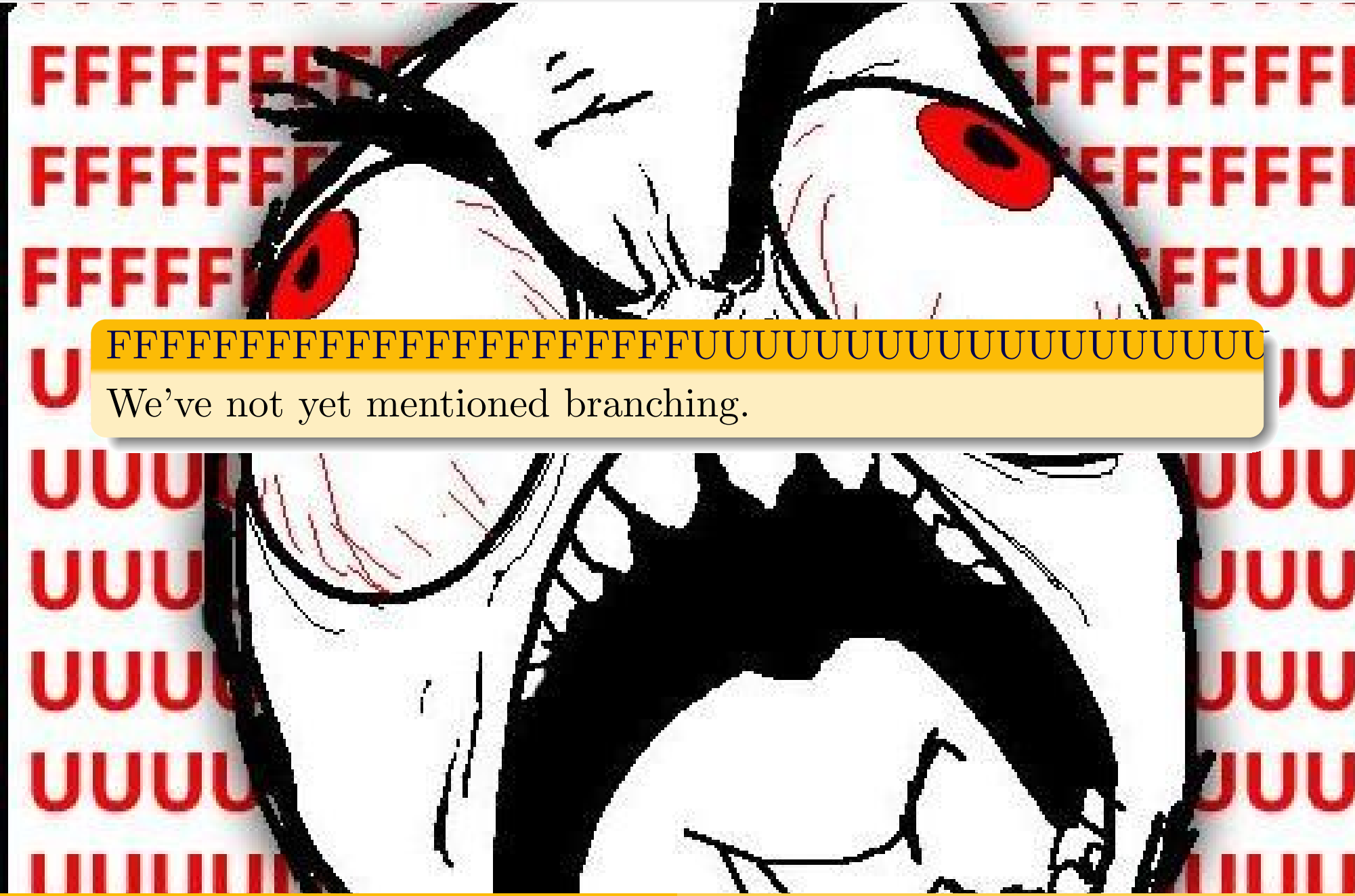
How does this interact with register banking? Multilevel caching? TLBs? Page cache? Prefetching? DRAM banking? Multilevel disk? Logical cores? Other physical cores? NUMA?

---

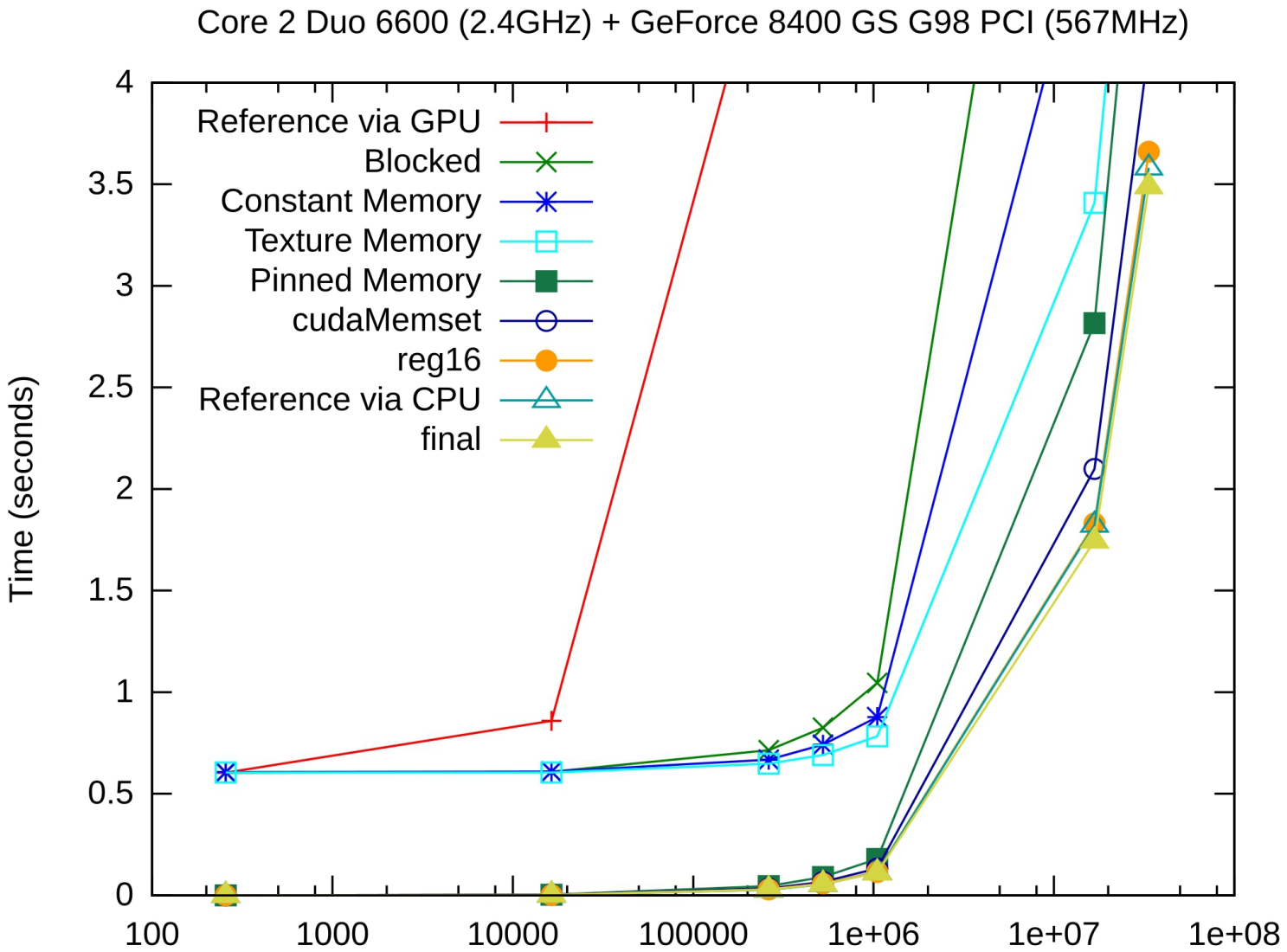
[6] Assuming that the values are located in aligned, contiguous 256-bite chunks in memory. Wait... we can use VMOVUPS if they're unsuitably aligned.

[7] VEX-encoded VMOVAPS tends to run ~5 bytes.

Argh

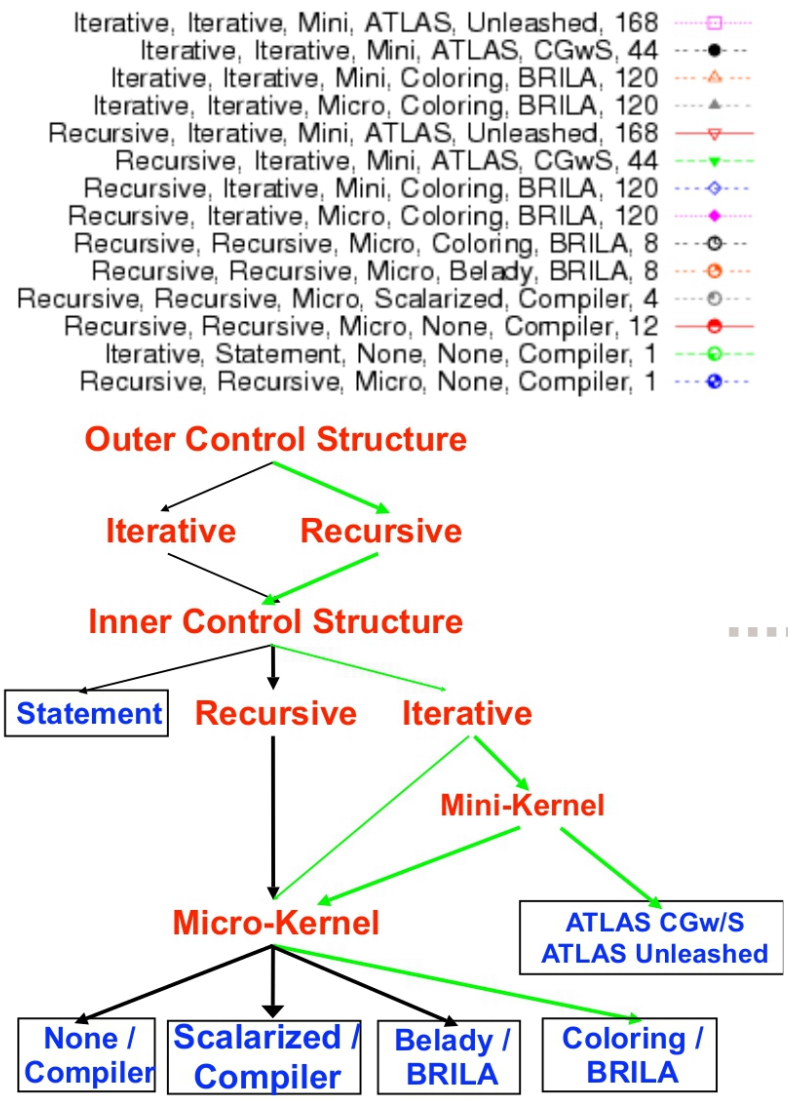

FFFFFFFFFFFFFFFFFFFFFFFUUUUUUUUUUUUUUUUUUUU
We've not yet mentioned branching.

CS4803 Spring 2010 Lab 3—All $\mathcal{O}(n^3)$



ATLAS Unleashed—All $\mathcal{O}(n^3)$

Image source: Richard Vuduc's CSE6230



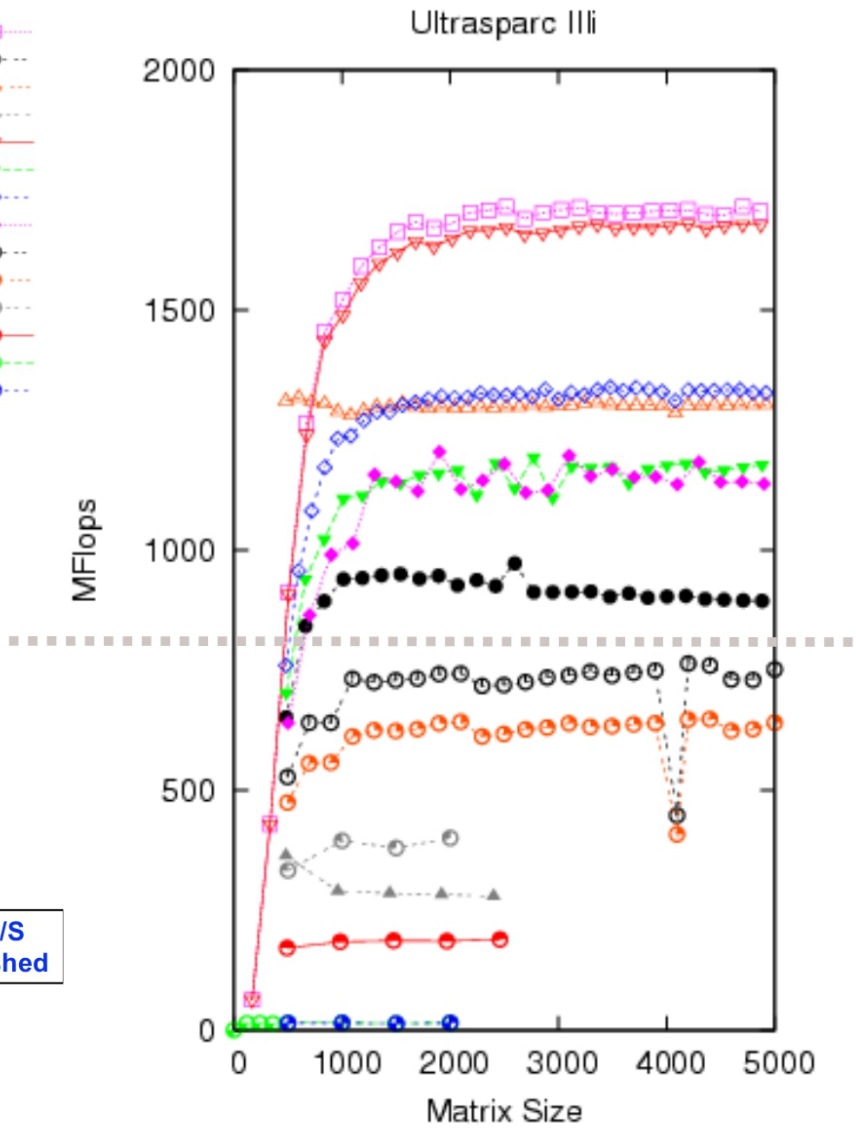
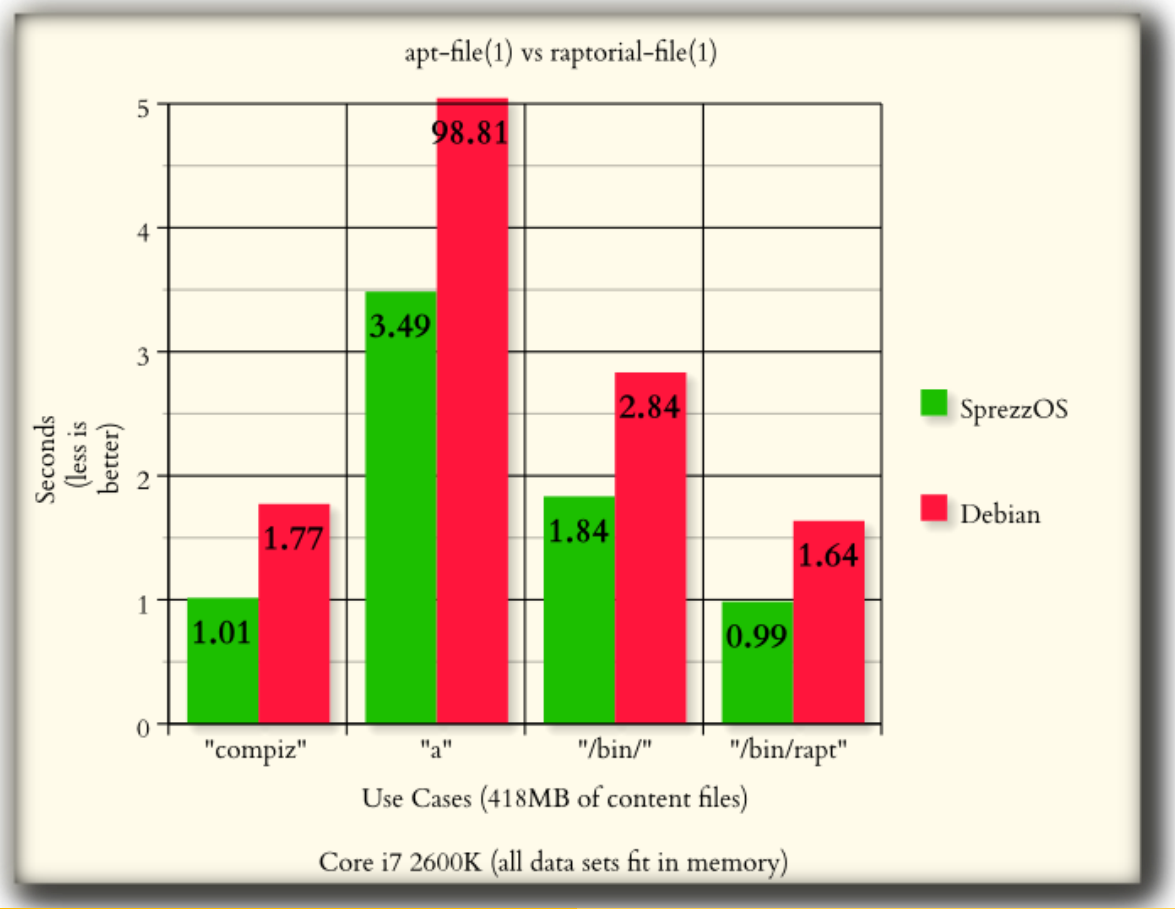
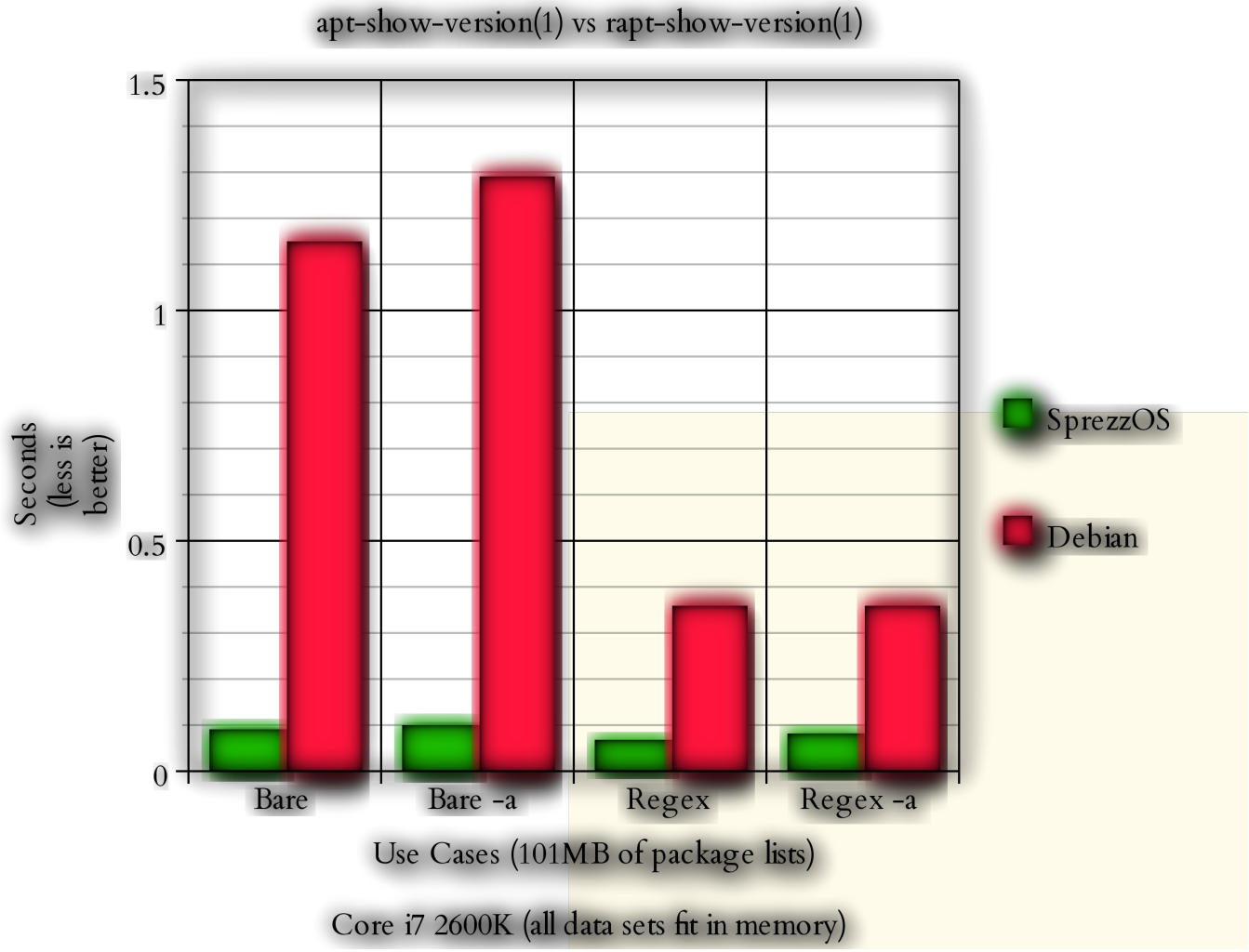


Image source: Dirty South Supercomputing and Waffles

In the pure systems space, $\mathcal{O}$ makes still less sense.
What's $\mathcal{O}$ of multithreaded file lexing?



Nonetheless, optimization can be very fruitful.



Analysis-driven optimization I

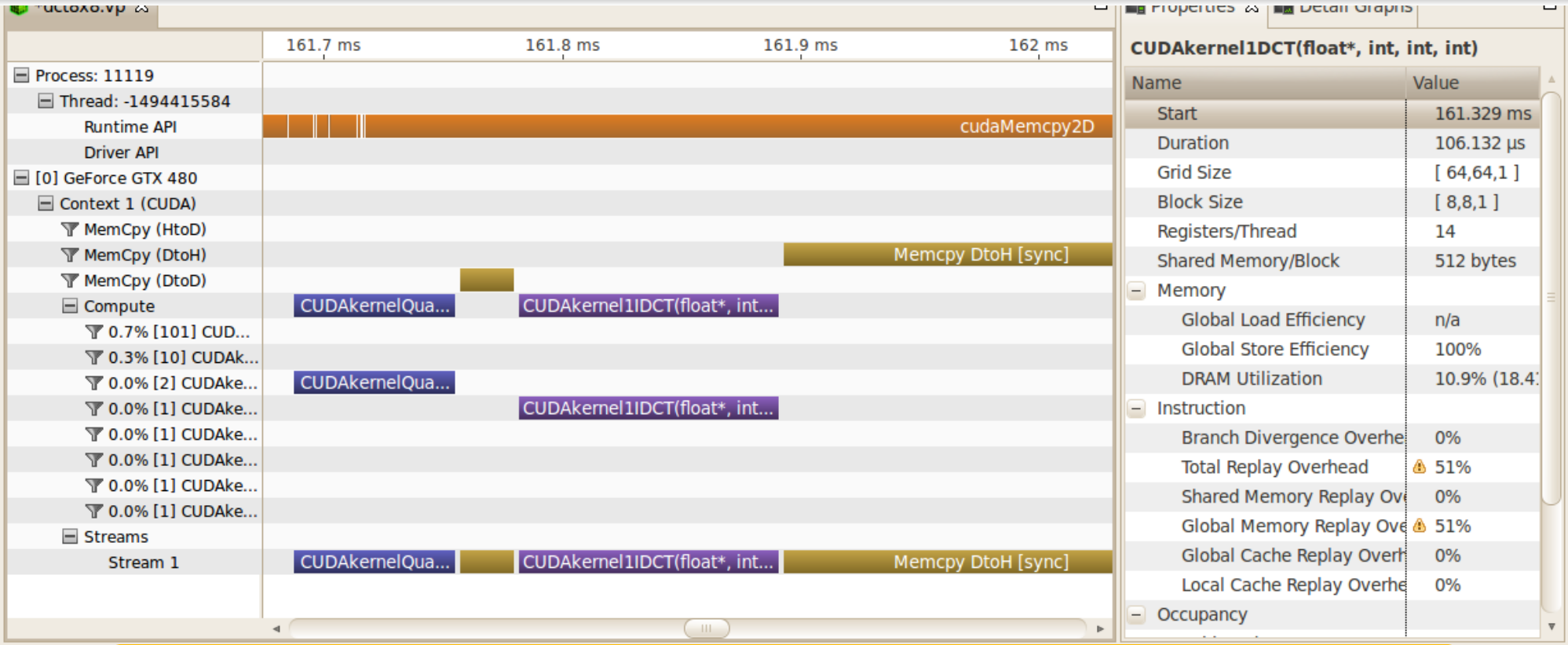
```
 samples    pcnt   kernel function
 _______   _____   _______________

12022.00 - 44.9% : read_hpet
 1766.00 -  6.6% : _spin_lock_irqsave
 1269.00 -  4.7% : acpi_os_read_port
 1209.00 -  4.5% : hpet_next_event
  410.00 -  1.5% : schedule
  280.00 -  1.0% : do_sys_poll
  273.00 -  1.0% : sched_clock_local
  265.00 -  1.0% : fget_light
  241.00 -  0.9% : native_read_tsc
  237.00 -  0.9% : _spin_lock
  231.00 -  0.9% : _spin_unlock_irqrestore
```

High-level ("Macroanalysis")

Coarse tools and algorithmic reasoning, e.g.:

- Ensure sufficient task-level parallelism
- Ensure cores aren't overutilized
- Profiler-driven hotspot location
- High-level memory and I/O flow

Analysis-driven optimization II

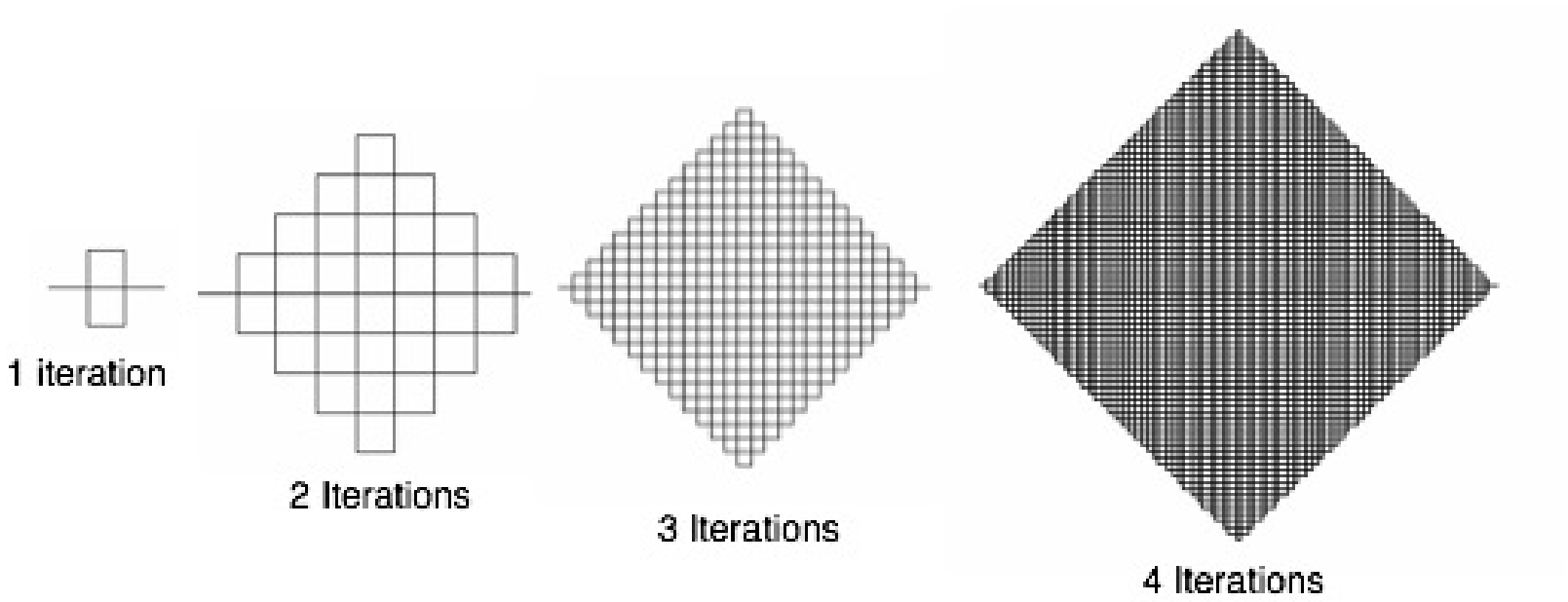


Low-level ("Microanalysis")

Performance counters and throughput-based reasoning, e.g.:

- Vectorize
- Tune for caches
- Watch for µarchitectural stalls

Complexity finds a way



A fractal having Hausdorff dimension 2,
the space-filling Peano curve is used in cache-oblivious algorithms.

Mastering the complex modern design space requires deftly switching between theoretical analysis, guided experiment, and pure exploration.

This class is about doing so.

Toledo 1999(?) (2004) → UWS
Vudve CSE6230 Fall 2009

Proof of

Each phase P_2 transfers Z words
$c(i,j)$ is live in p_2 if

dx ∆ ∃k we compute $a(i,k) * b(k,j)$
dx ∆ and some partial sum of $c(i,j)$ is either in cache or written to main memory (why

at most $2*Z$ live $c(i,j)$ in phase
two Z

at most $2*Z$ distinct A in cache
B in cache
(either in beginning or moved)

A_p : set of A in cache B_p : set of B

S_{p+} = rows of A with $\sqrt{Z}$ or more in

S_{p-} = rows w/ fewer

$|S_{p+}| \leq 2*\sqrt{Z}$

For rows in $|S_{p+}|$; $a(i,:)*B$ touches each B once
mults $\leq |S_{p-}| * (2*M) = 4Z^{3/2}$

Insert 2/3

for rows in Sp- consider what
c(i,j) = row x col
$m_u(r) \leq$ no live x (max row l_m) $\leq 2Z$
Total m_u = n^3
$m_u(t)$/phase $\leq 6Z^{3/2}$
phases $\geq \lceil \frac{n^3}{6Z^{3/2}} \rceil$
work mult $\geq Z\left(\frac{n^3}{6Z^{3/2}} - 1\right)$
$= \frac{n^3}{6\sqrt{Z}} - Z$
$\Omega\left(\frac{n^3}{\sqrt{Z}}\right)$ $Z < \frac{n^2}{6}$

UWS #1

Busy Beaver

Noncomputable function

grows faster than any computable functi

BB(n) game!

design a transition function that maxim

1's left on tape, having n sta

for every n finitely many machine

have finitely many running sim

BB(n) > any computable fun

Rado 1962

Appl of Kolmogorov complexity

Recommended reading

- Hong and Kung. "I/O complexity: The red-blue pebble game" (1981).
- Yotov et al. "An experimental comparison of cache-oblivious and cache-conscious programs" (2007).
- Irony et al. "Communication Lower Bounds for Distributed-Memory Matrix Mul" (2004).
- Goto et al. "Anatomy of High-Performance Matrix Multiplication" (2008).
- Kalyanasundaram et al. "Improved Simulation of NTMs" (2011).
- François Le Gall. "Faster Algorithms for Rectangular Matrix Multiplication" (2012).
- Eric Quinnell. "Floating-Point Fused Multiply-Add Architectures" (2007).
- Tom Leighton. "Better Master Theorems for Divide-and-Conquer Recurrences" (1996).
- *Intel Instruction Set Extensions Programming Reference*