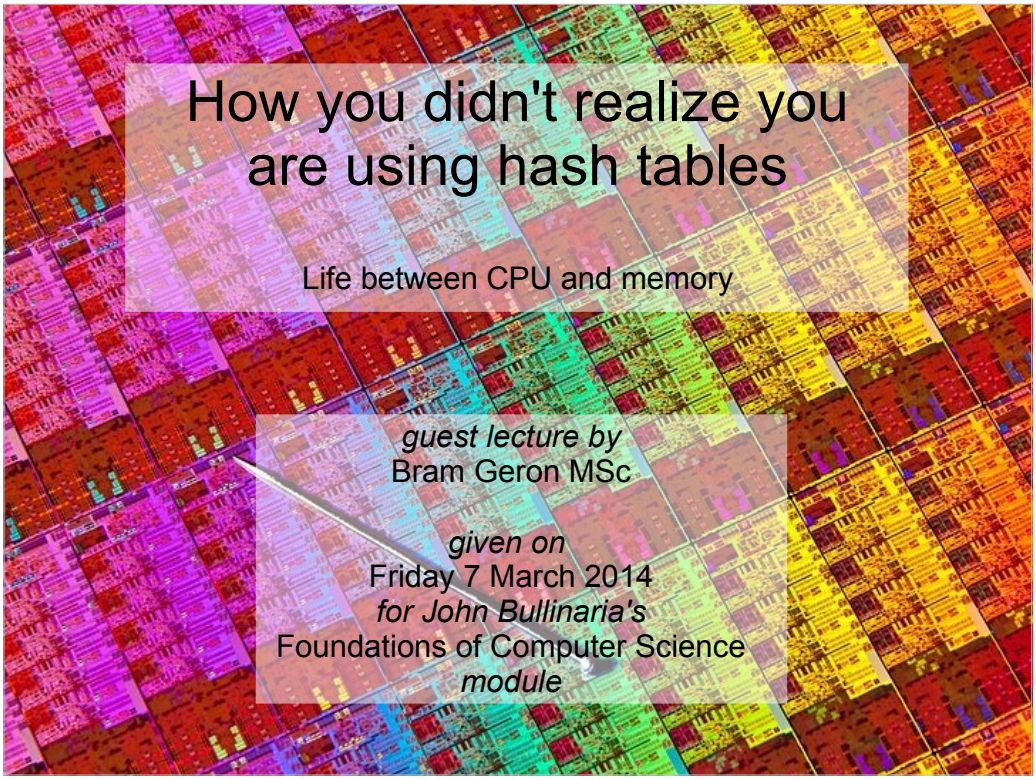
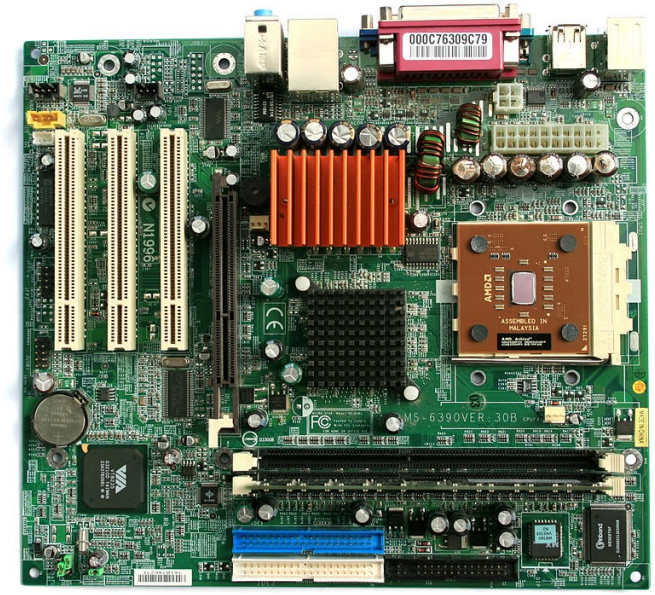


Can you figure out what the photo in the background is?

It is a Haswell wafer.

Computer architecture



Indicate CPU, RAM. CPU communicates with I/O devices. (Essentially.)

Me, myself & I

Bram Geron

- Graduated in 2013 from Eindhoven Uni. of Tech., NL
- Now: PhD student here

  Bridging the gap between practice and theory in programming languages

  Helping in Foundations, Functional Programming, Data Structures

What to expect from this lecture

- Nothing on exam
- No assignment

  You can thank John later.

- Some knowledge of computer architecture
- Understand how hash tables can be used
- Modified data structures can be useful
- Simple can be useful

Summing an array of numbers

```
int sum(int array[ ], size_t count) {
    int result = 0;
    for (int i = 0; i < count; i++) {
        result += array[i];
    }
    return result;
}
```

Registers:

- 0: *array*
- 1: *count*
- 2: *result*
- 3: *i*
- 4: *arrayi*

```
    set result to 0
    set i to 0
beginloop:
    compare count to i
    if smaller goto endloop
    load array[i] to arrayi
    increase result by arrayi
    increase i by 1
    goto beginloop
endloop:
    return from function
```

[Drawing of CPU, memory, outside world]

I'll use this example to illustrate what's happening inside a computer.

Run algorithm by hand on [3, 1, 4, 1, 5]
→ 14

Introduce CPU registers

Variables result, i, count, array would be stored in registers

The array itself would be stored in memory (though not the array *pointer*)

Instruction types

1. Load register from register
2. Load register from memory
3. Add registers
4. Compare registers

One of these takes 100× as long as the others.

This is due to chemical properties of memory.

Source: http://norvig.com/21-days.html

execute typical instruction 1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory 0.5 nanosec
branch misprediction 5 nanosec
fetch from L2 cache memory 7 nanosec
Mutex lock/unlock 25 nanosec
fetch from main memory 100 nanosec
send 2K bytes over 1Gbps network 20,000 nanosec
read 1MB sequentially from memory 250,000 nanosec
fetch from new disk location (seek) 8,000,000 nanosec
read 1MB sequentially from disk 20,000,000 nanosec
send packet US to Europe and back 150 milliseconds = 150,000,000 nanosec

1969: introduction of buffer storage: 16 kB

IBM System/360 Model 85

[extend drawing on blackboard]

The processor memory gap

[picture removed for copyright reasons]

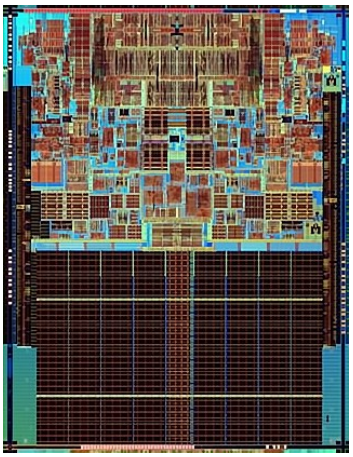
This picture shows that from 1980 to 2000, CPU speeds increased ~1000×, while RAM speed only increased ~5×.

See
http://ieeexplore.ieee.org/ielx1/40/12709/00592312.pdf?tp=&arnumber=592312&isnumber=12709
page 2

From: Patterson et al, A Case for Intelligent RAM, in: IEEE Micro 17:2, pp 34.

But! We can make small amounts of very fast memory.

Cache levels



Model 85:

16 kB buffer storage in 16 lines

Intel Haswell (4th gen i7):

Level 1 cache: 64 kB in 1024 lines
Level 2 cache: 256 kB
Level 3 cache: 8 MB
(Level 4 cache: 128 MB)

Picture: Intel Conroe die, probably in Core 2 Duo

Different associativity is one of the reasons for the speed difference between L1 and L2 cache. L1 is typically 4-way ass., and L2 8-way.

Fully associative cache

(like buffer storage, but automatic)

slot	tag			value
0	**100**	00	→	**3**
	100	01	→	**1**
	100	02	→	**4**
	100	03	→	**1**
	100	...		
	100	99	→	**7**
1	**152**	00	→	**6**
	152	01	→	**4**
	152	...		
	152	99	→	**3**
2	**279**	00	→	**5**
	279	01	→	**8**
	279	...		
	279	99	→	**1**
	...			
9	**051**	00	→	**2**
	051	01	→	**1**
	051	...		
	051	99	→	**9**

Model 85 had something like this.

This doesn't scale.

In Model 85: 16 buffer lines

Perfect for scanning arrays though!

Direct-mapped cache

slot	tag			value
0	**100**	00	→	**3**
	100	01	→	**1**
	100	02	→	**4**
	100	03	→	**1**
	100	...		
	100	99	→	**7**
1	**051**	00	→	**2**
	051	01	→	**1**
	051	...		
	051	99	→	**9**
2	**152**	00	→	**6**
	152	01	→	**4**
	152	...		
	152	99	→	**3**
	...			
9	**279**	00	→	**5**
	279	01	→	**8**
	279	...		
	279	99	→	**1**

Now we can have loads and loads of entries. But: conflicts.

Typically in current L1: 1024 cache lines

2-way associative cache

slot	tag			value
0-0	**10**0	00	→	**3**
	100	01	→	**1**
	100	02	→	**4**
	100	03	→	**1**
	100	...		
	100	99	→	**7**
1-0	**05**1	00	→	**2**
	051	01	→	**1**
	051	...		
	051	99	→	**9**
2-0	**15**2	00	→	**6**
	152	01	→	**4**
	152	...		
	152	99	→	**3**
	...			
9-0	**27**9	00	→	**5**
	279	01	→	**8**
	279	...		
	279	99	→	**1**

slot	tag			value
0-1	**51**0	00	→	**1**
	510	01	→	**2**
	510	02	→	**8**
	510	03	→	**8**
	510	...		
	510	99	→	**9**
1-1	**48**1	00	→	**0**
	481	01	→	**5**
	481	...		
	481	99	→	**7**
2-1	**76**2	00	→	**3**
	762	01	→	**4**
	762	...		
	762	99	→	**8**
	...			
9-1	**23**9	00	→	**6**
	239	01	→	**3**
	239	...		
	239	99	→	**7**

This is better.

Can you recognize the hash table here? What is the hash function on address 12345?

Caches, caches, caches

Typically:

- Level 1 cache: 4-way associative
- Level 2 cache: 8-way associative, more entries
- Level 3 cache: ???, way more entries

Where is the hash table?

Lessons to be learned

If things seem jolly rotten
There's something you've forgotten
And maybe that's to push a different button

- Hash tables are useful
- Sometimes adaptations are useful too
  This also holds for sorting, lists, trees, graphs, …, although the solution isn't always obvious.
- Time-memory tradeoffs are frequent
- There's more to computing than a CPU and RAM

Thanks for your attention!

That's all, folks.

Questions?

Ask me anything.

Acknowledgements

- Haswell wafer photo CC BY 2.0 from James086 and Intel Free Press
- Motherboard photo by Jonathan Zander (photography.jznet.org), CC BY-SA 3.0
- Conroe die: supposedly public domain, https://commons.wikimedia.org/wiki/File:Conroe_die.png, uploaded by Xoqolatl
- CPU/memory graph from DOI 10.1109/40.592312 (“A case for Intelligent RAM” by Patterson et al), who reference J.L. Hennessy and D.A. Patterson, Computer Organization and Design, 2nd ed., Morgan Kaufmann Publishers, San Francisco, 1997.