Modern CPU Performance Profiling

Hagen Paul Pfeifer

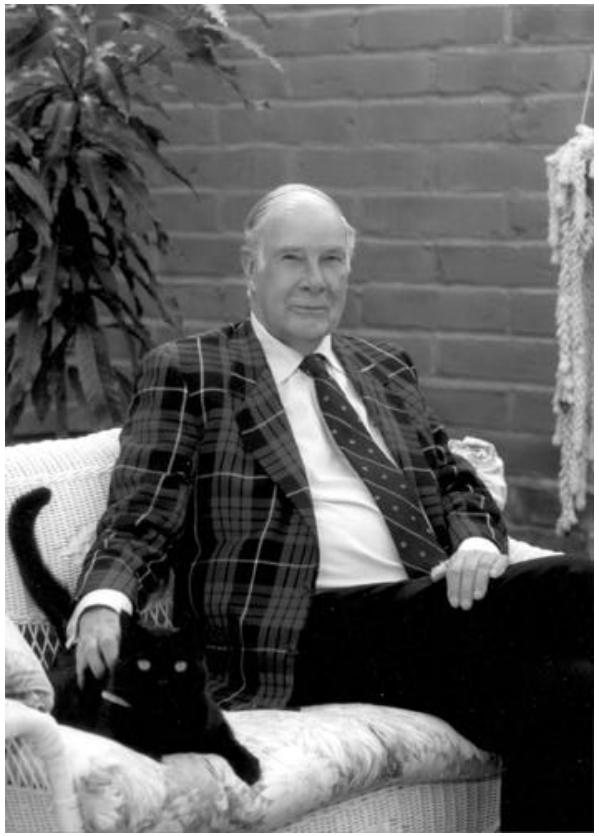
ProtocolLabs
Hofmannstraße 19
81379 München, Germany

http://www.protocollabs.com
hagen.pfeifer@protocollabs.com

Introduction

„ The purpose of computing is insight, not numbers.“

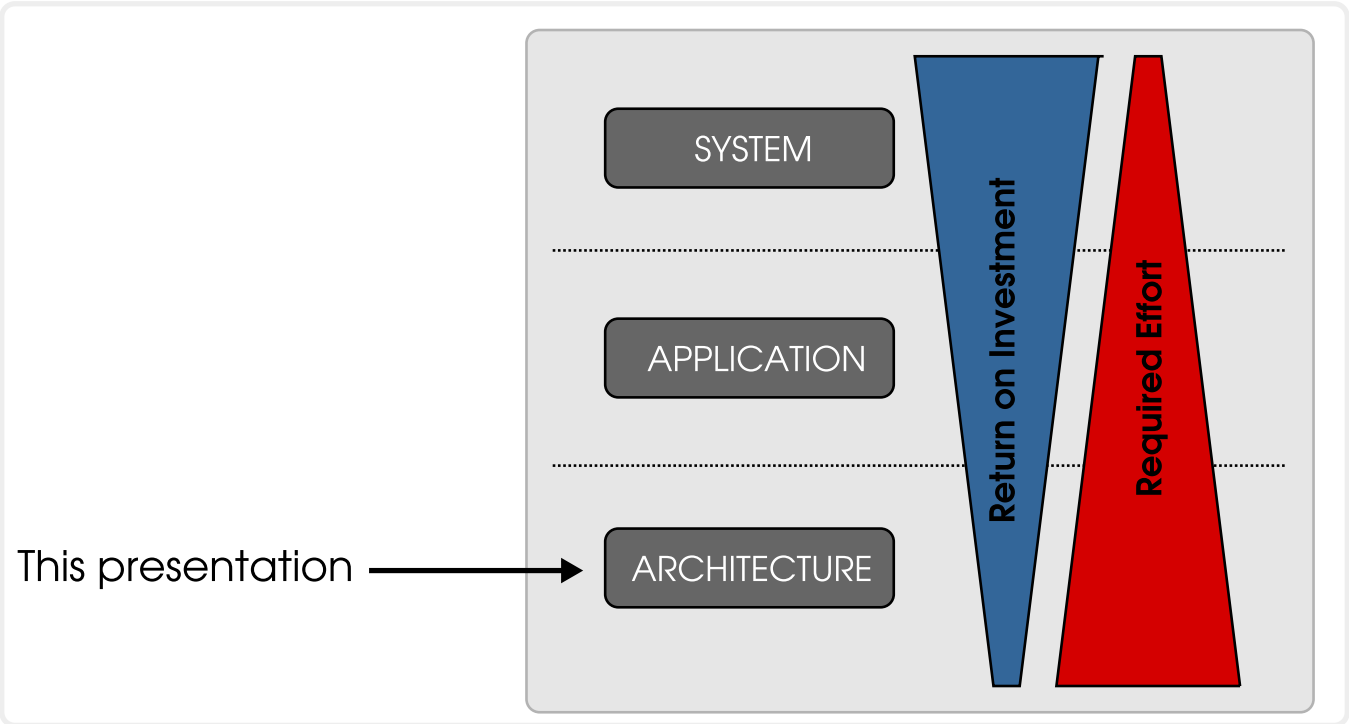
— Richard W. Hamming



Introduction

- No tool can be better then the user – this is especially true for performance analysis!
- Depending on the level of optimization you need a fairly good understanding of
  - How the compiler work
  - Processor specific instructions
  - How instructions are scheduled and the costs of instructions
  - About memory hierarchy: CPU caches, DRAM and system bus
  - IO Subsystem knowledge: network interface adapter, harddisk, ...
  - Kernel knowledge (including system calls, memory management, scheduling, IO system, ...)

Introduction



Introduction

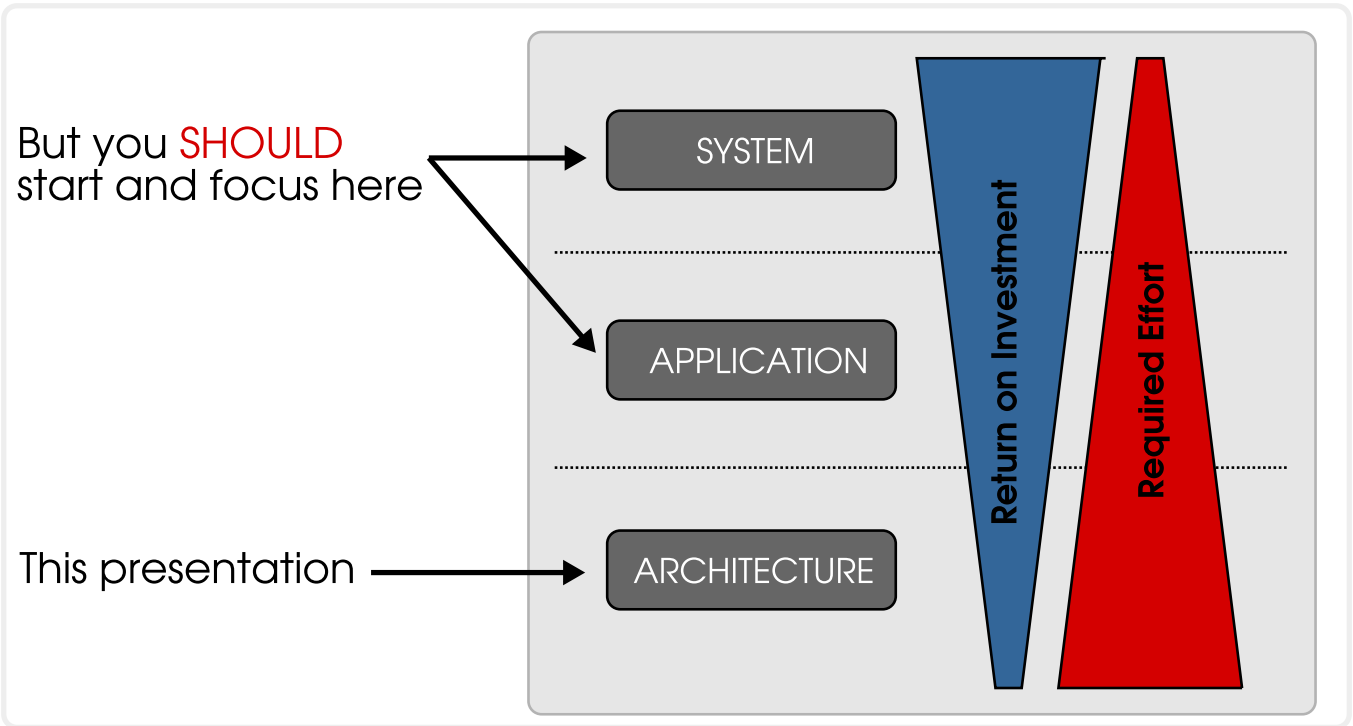


Table of Contents

A Brief History of Time

Evolution of Latency and Bandwidth

	VAX 11 (1977)	Sandy Bridge	Improvement
Clock Speed	5 MHz[1]	3000 MHz	500x
Memory Size	2 MB	8000 MB	4000x
Memory Bandwidth	13 MB/s	5000 MB/s	385x
Memory Latency	225 ns	70 ns	3x

[1] 200 ns cycle time

Latency and Bandwidth

Latency and Bandwidth

- Bandwidth
  - How wide is your pipeline

Latency and Bandwidth

- Bandwidth
  - How wide is your pipeline
- Latency
  - How long is your pipeline

Latency and Bandwidth

- Bandwidth
  - How wide is your pipeline
- Latency
  - How long is your pipeline

You can solve bandwidth problems – but you can't solve latency problems!

How can we cope with latency?

How can we cope with latency?

How can we cope with latency?

- SIMD - Single Instruction Multiple Data
  - Intel's MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, ...
  - AMD's 3DNow
  - IBM's AltiVec
  - ARM's NEON
- Graphics Processing Units (GPU)

How can we cope with latency?

How can we cope with latency?

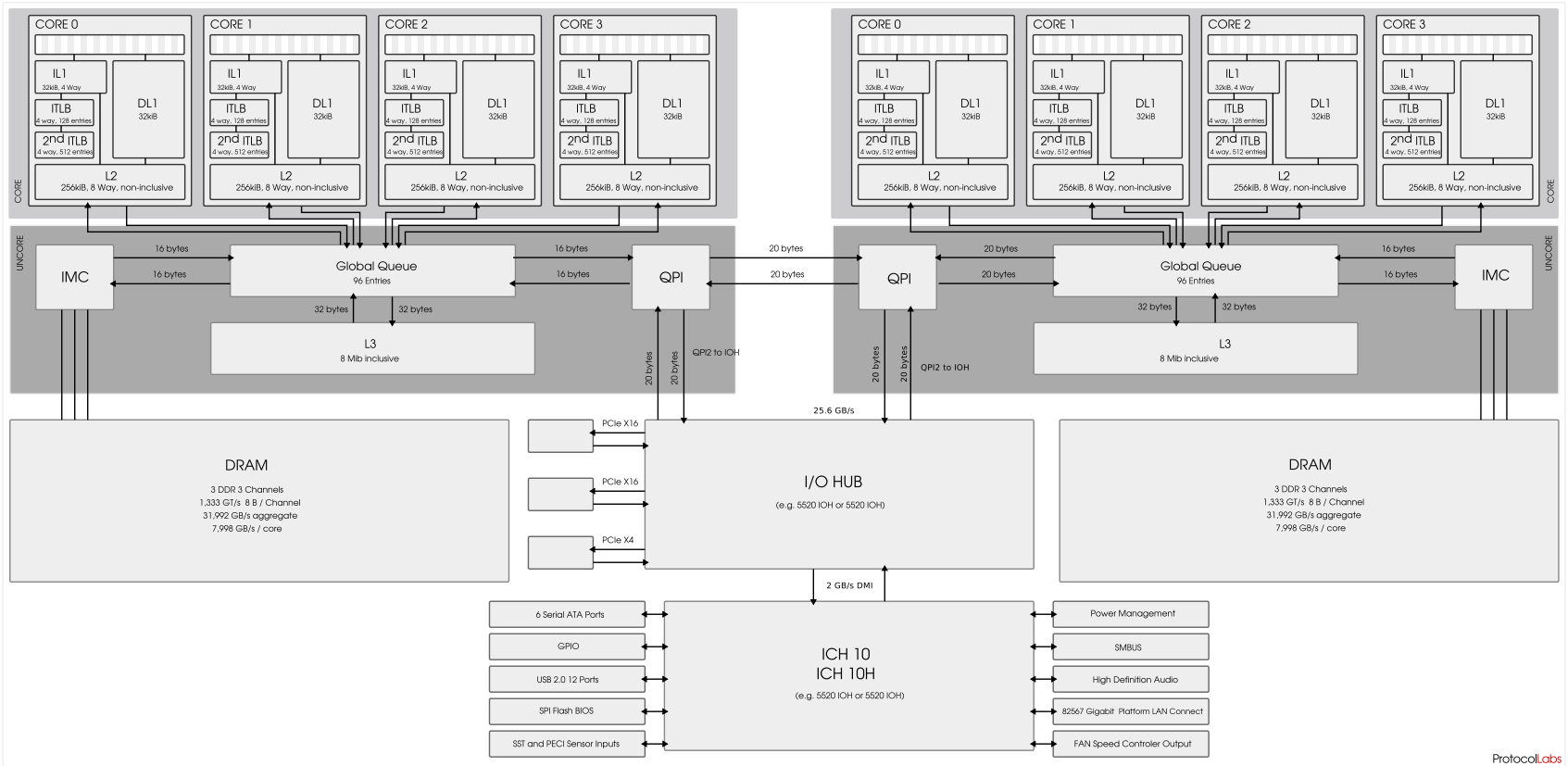
- Parallelize - process concurrently where ever possible:
  - Pipelining
  - Instruction Decoding
  - Hyperthreading
  - SMP/CMP Systems
- Cache[2] as much as possible
- Increase memory hierarchy
  - Uops cache
  - L1 cache
  - L2 cache
  - L3 cache
  - HD flash memory cache
  - HD

[2] SRAM is expensive; currently 95% of a core is already for the cache; in the end: you cannot increase SRAM to ∞

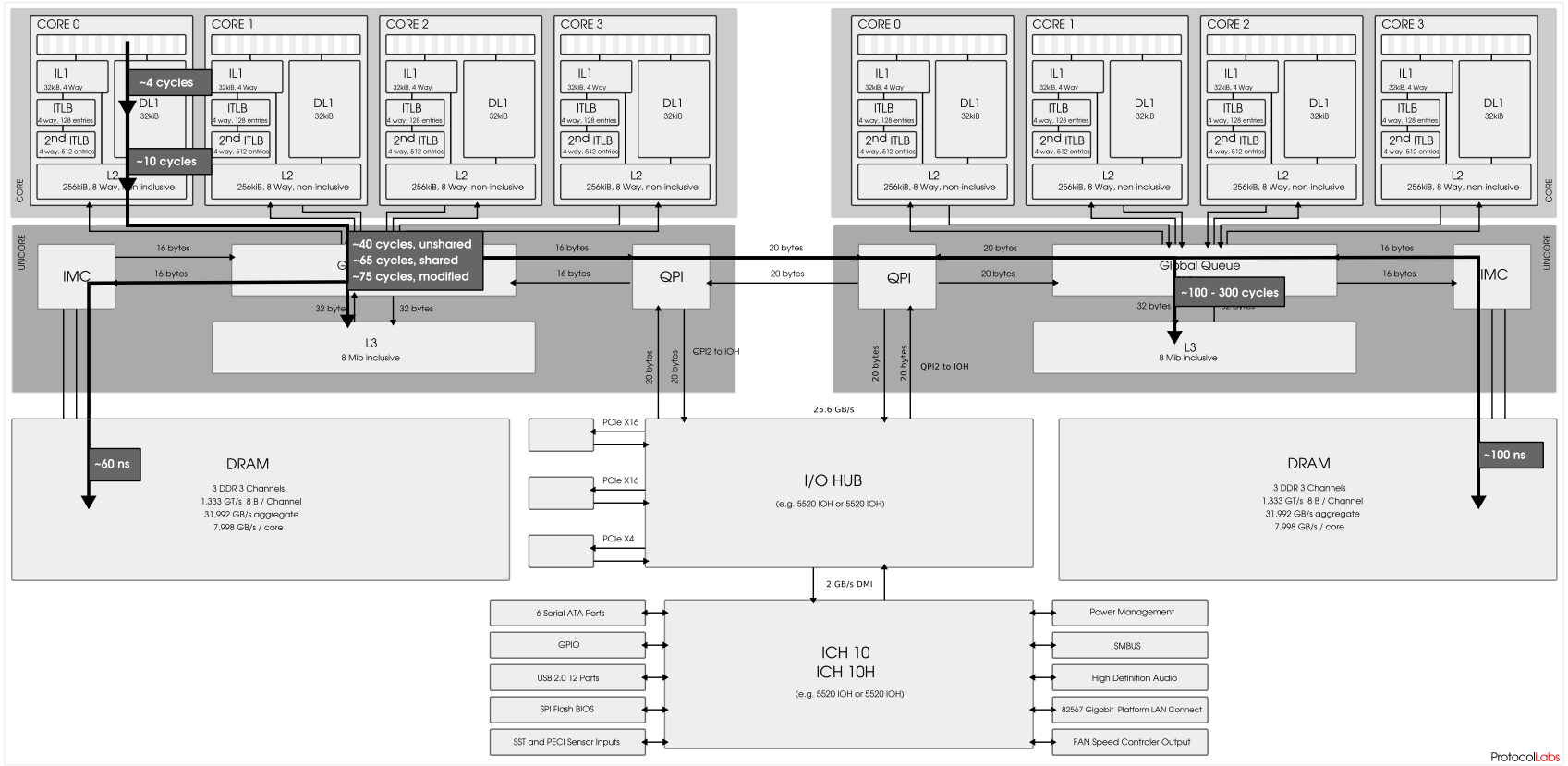
Bottom Line

All hardware vendor attempts have only one goal: *hide memory latency*

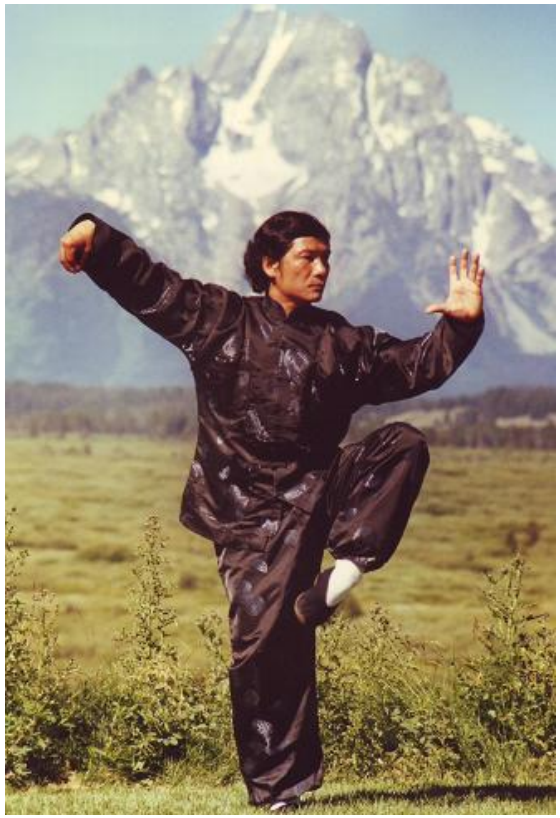
Memory Hierarchy Today – Intel XEON 5000 Sequence



Memory Hierarchy Today – Intel XEON 5000 Sequence



The Tao of Performance Analysis



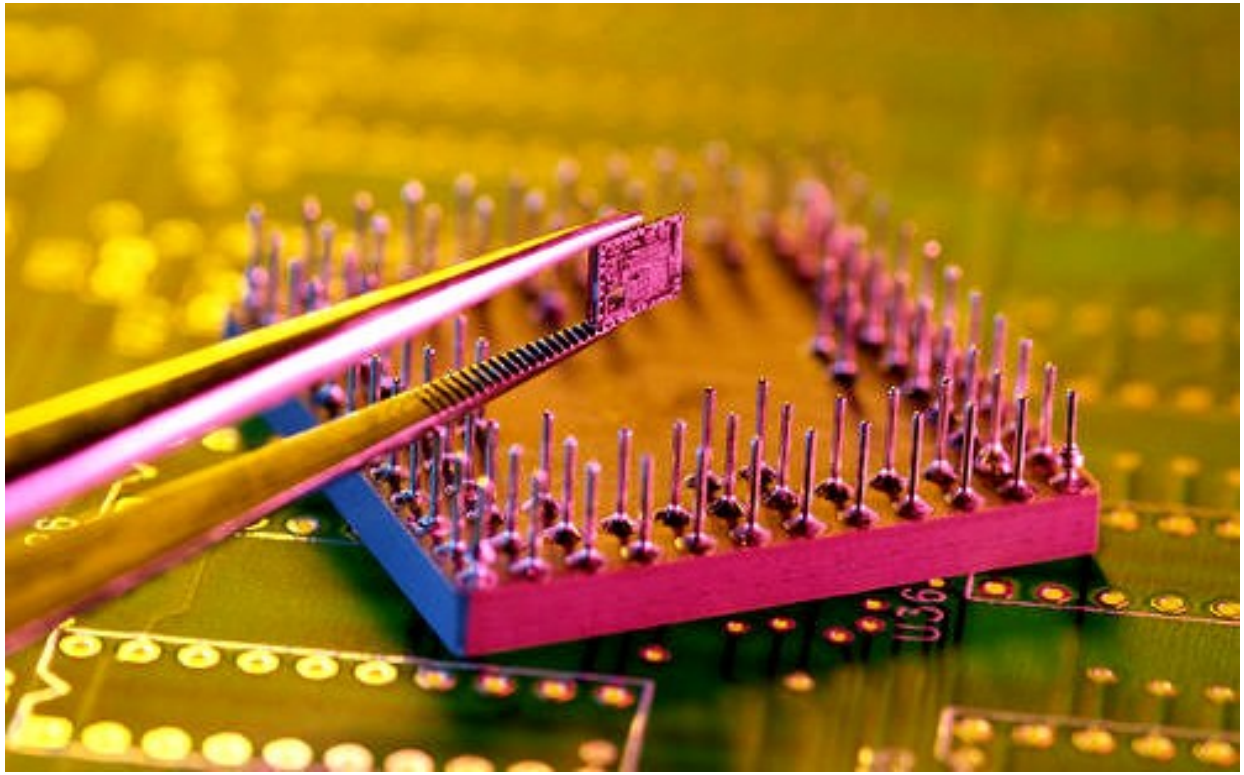
The Tao of Performance Analysis

- Identify hot spots
- Focus on hot spots, ignore the rest
- Reduce analyze complexity
  - Disable uninvolved services
  - Disable Hyperthreading, disable Turbo Boost
- Make analyze comparable: use same test-case
- Use realistic test-data

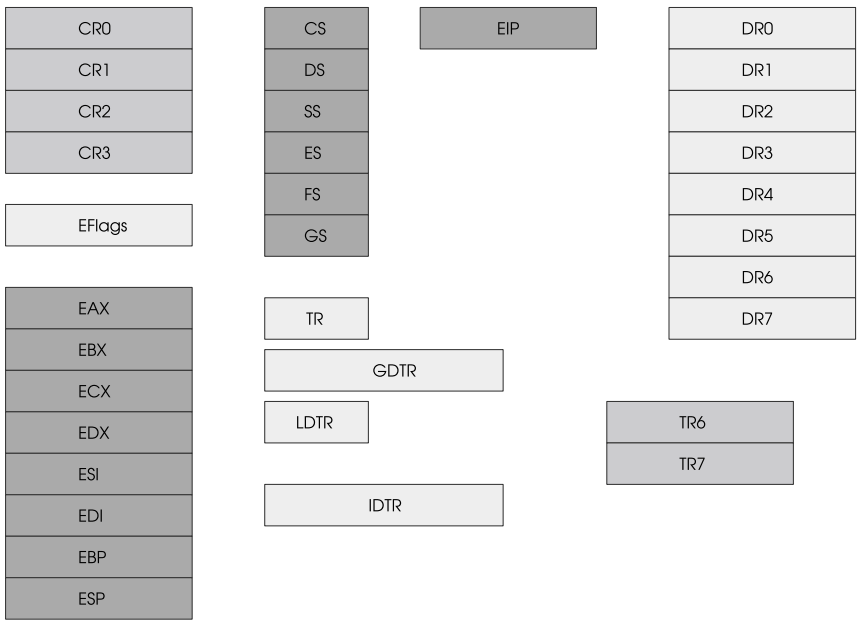
Determine Efficiency of Hotspots

- Execution stalls
  - `UOPS_EXECUTED.CORE_STALL_CYLES`, `UOPS_EXECUTED.CORE_ACTIVE_CYCLES`
  - Frontend stalls
  - Backend stalls
- Cycles per Instruction
  - `CPU_CLK_UNHALTED.CORE / INST_RETIRED.ANY`
  - Ideal CPI: 0.25, 1: OK, $>$ 2: somehow high
- Examine generated instructions
  - Yes, instruction and architecture knowledge is required

The Call for Hardware Performance Monitoring Support

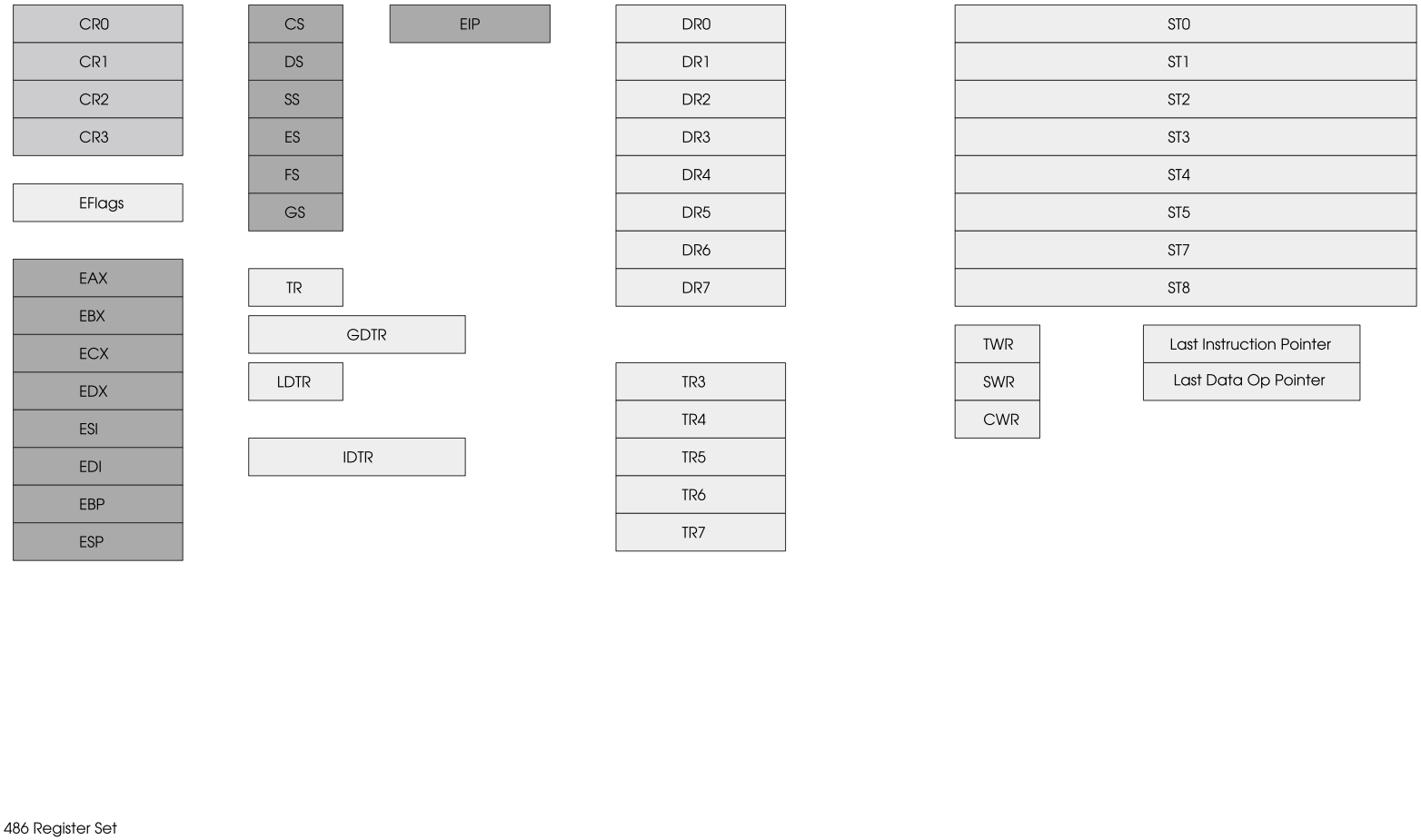


Register Set Evolution

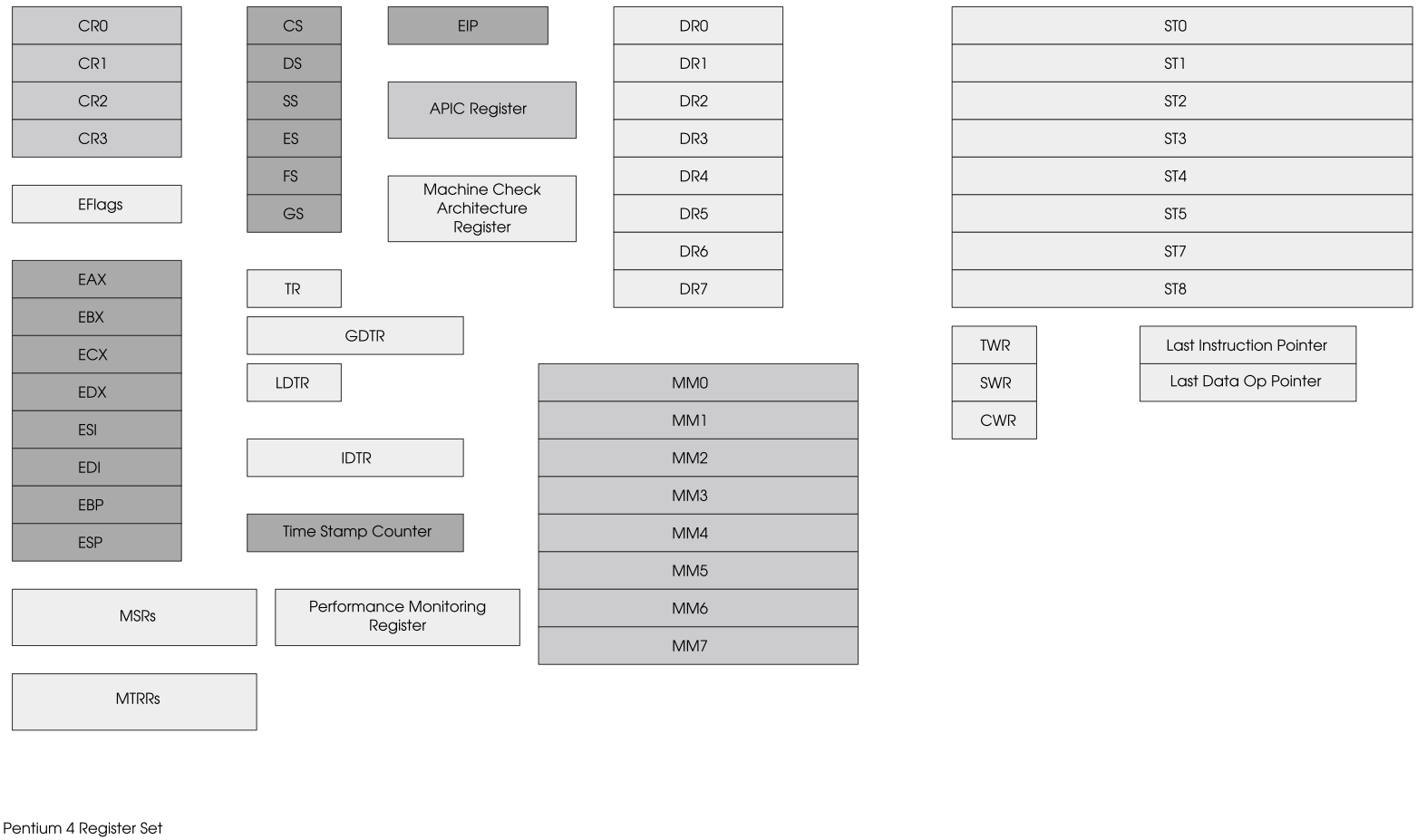


386 Register Set

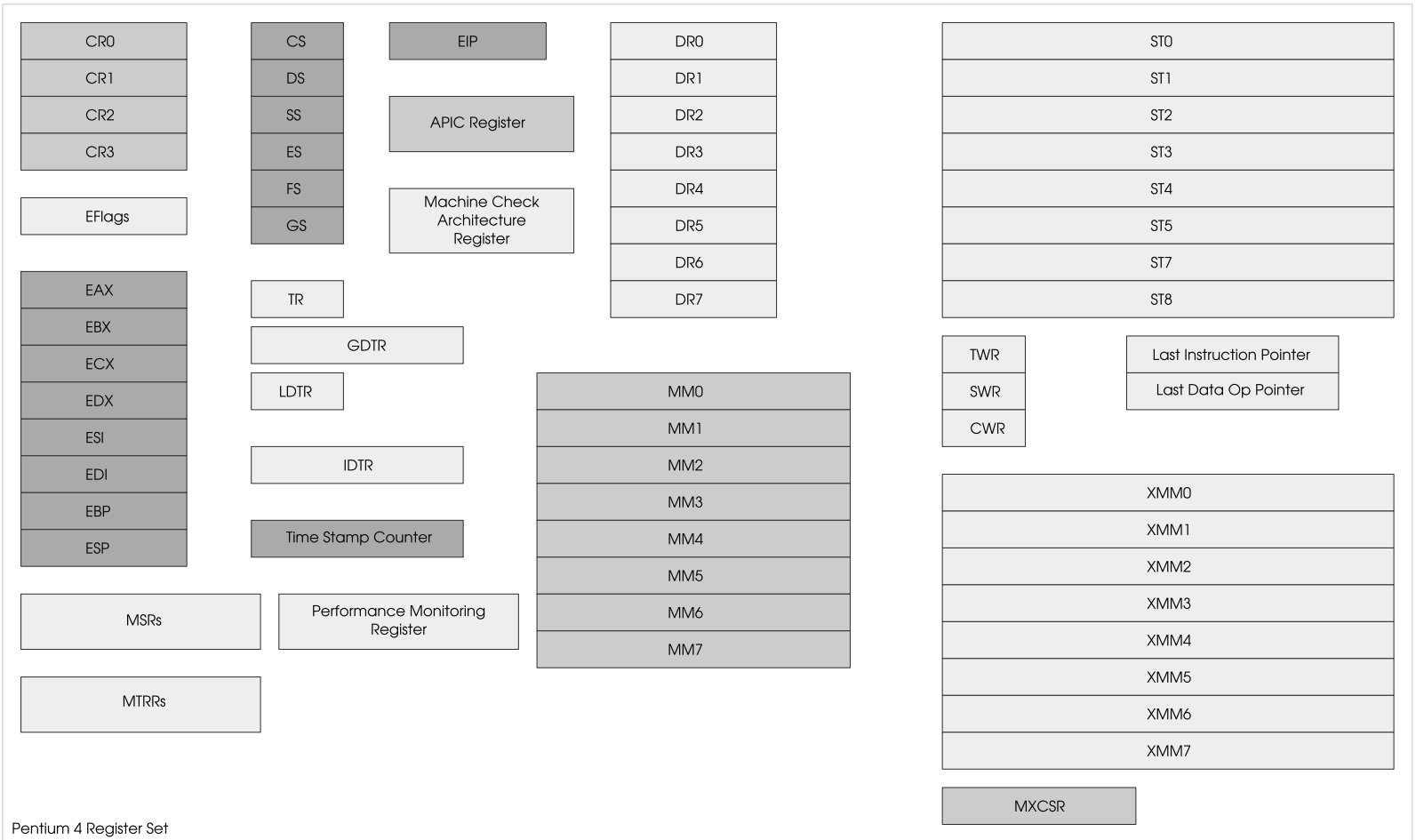
Register Set Evolution



Register Set Evolution

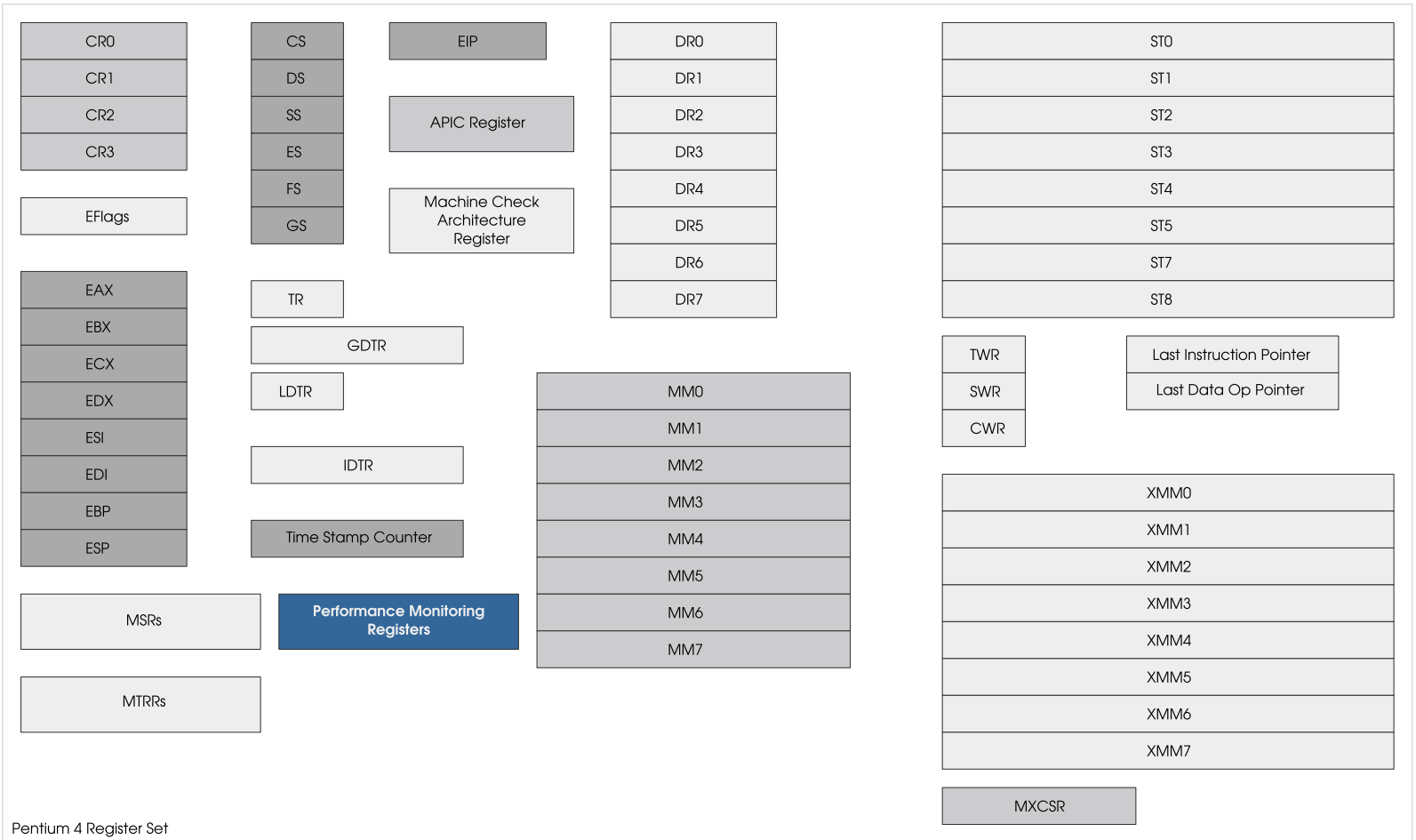


Register Set Evolution



Pentium 4 Register Set

Register Set Evolution



Pentium 4 Register Set

Architectural Complexity

- With increasing architectural complexity it is almost impossible to know what happens „ inside“.
- Modern CPU provides therefore insights and mechanism to get intrinsic data[3]

[3] or: current hardware trends makes monitoring critical

Performance Monitoring Units – Overview

- One PMU in each (logical) Core
  - `CPU_CLK_UNHALTED`
  - `INST_RETIRED`
  - `LLC_MISSES`
  - `LLC_REFS`
  - `BR_INST_RETIRED`
  - `BR_MISS_PRED_RETIRED`
  - `...`

Performance Monitoring Units – Overview

- One PMU in each (logical) Core
  - `CPU_CLK_UNHALTED`
  - `INST_RETIRED`
  - `LLC_MISSES`
  - `LLC_REFS`
  - `BR_INST_RETIRED`
  - `BR_MISS_PRED_RETIRED`
  - `...`
- One PMU in each Uncore
  - `UNC_DRAM_PAGE_MISS`
  - `UNC_DRAM_READ_CAS`
  - `UNC_GQ_ALLOC`
  - `UNC_GQ_CYCLES_FULL`

Performance Monitoring Units – Overview

- One PMU in each (logical) Core
  - `CPU_CLK_UNHALTED`
  - `INST_RETIRED`
  - `LLC_MISSES`
  - `LLC_REFS`
  - `BR_INST_RETIRED`
  - `BR_MISS_PRED_RETIRED`
  - `...`
- One PMU in each Uncore
  - `UNC_DRAM_PAGE_MISS`
  - `UNC_DRAM_READ_CAS`
  - `UNC_GQ_ALLOC`
  - `UNC_GQ_CYCLES_FULL`
- PMUs are highly specific to a CPU!

Core PMU

- 4 fully programmable counters per CPU Core
  - Each count a certain event
  - Each counter can be used simultaneously
- 1 fixed counter
  - Clock Counter
  - Reference Clock Counter
  - Instruction Counter

UNCore PMU

- 8 fully programmable counters per CPU Core
  - Each count a certain event
  - Each counter can be used simultaneously
- 1 fixed counter

PMU Modes

- Event based Counting (EBC)
  - No interrupts
  - Very small overhead
- Event based Sampling (EBS)
  - Interrupts the CPU after a certain number of events[4]
  - Collect execution events
  - Statistical method
- Precice Event based Sampling (PEBS)
  - Not all events
  - Processor save (exact) context (instruction pointer)

[4] Sample After Value – SAV

Linux Perf

Perf Overview

- Ingo Molnar and Thomas Gleixner
- Milestone 8647093: from Documentation to new tool directory (2009)
- Userspace tool plus Kernel Subsystem[5]
- git like subcommands
- Per thread/per workload/per CPU/system wide
- No daemon mode
- Today: Arnaldo, Ingo, Frederic, Masami and many others

[5] maybe the term subsystem is to big, compared to real Linux subsystems

Perf Modules

```
$ perf list
The most commonly used perf commands are:
  annotate        Read perf.data (created by perf record) and display annotated code
  archive         Create archive with object files with build-ids found in perf.data file
  bench           General framework for benchmark suites
  buildid-cache   Manage build-id cache.
  buildid-list    List the buildids in a perf.data file
  diff            Read two perf.data files and display the differential profile
  evlist          List the event names in a perf.data file
  inject          Filter to augment the events stream with additional information
  kmem            Tool to trace/measure kernel memory(slab) properties
  kvm             Tool to trace/measure kvm guest os
  list            List all symbolic event types
  lock            Analyze lock events
  probe           Define new dynamic tracepoints
  record          Run a command and record its profile into perf.data
  report          Read perf.data (created by perf record) and display the profile
  sched           Tool to trace/measure scheduler properties (latencies)
  script          Read perf.data (created by perf record) and display trace output
  stat            Run a command and gather performance counter statistics
  test            Runs sanity tests.
  timechart       Tool to visualize total system behavior during a workload
  top             System profiling tool.
```

Life Demo

Demo

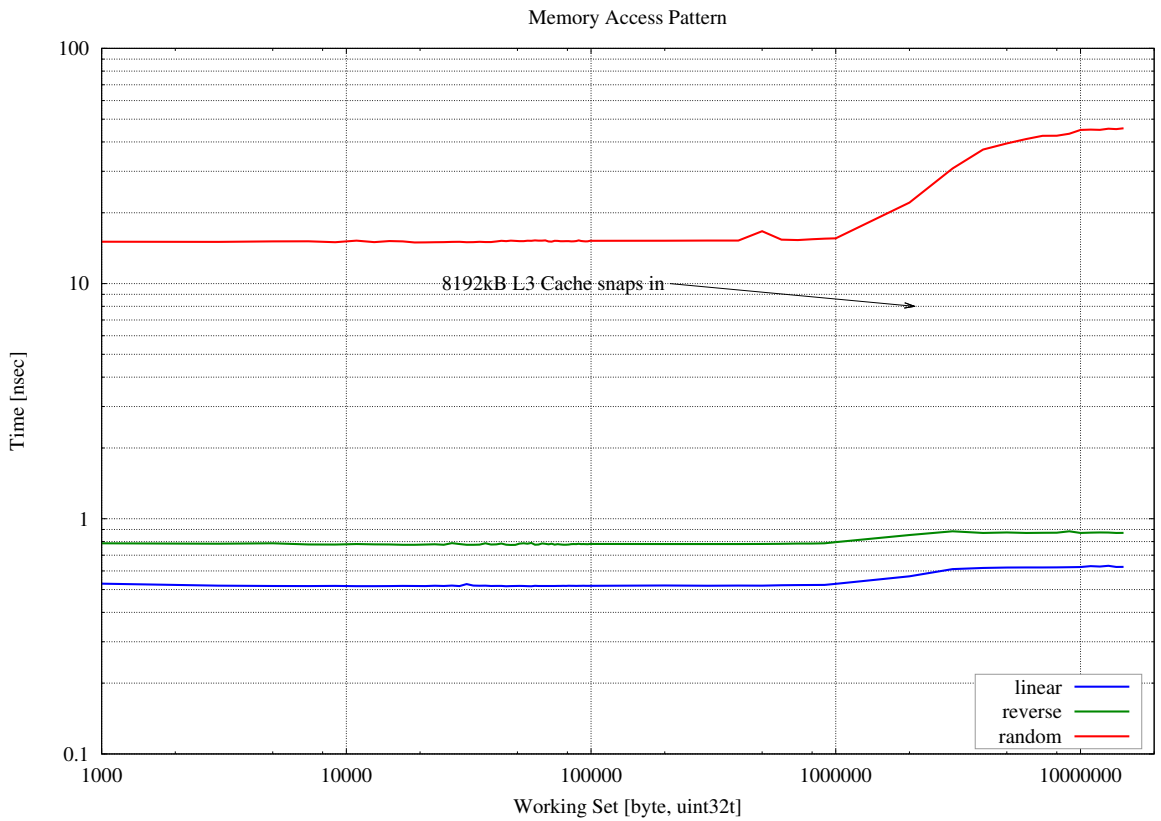
Example 1

Memory Access Pattern Matters

Access Pattern

- Think about data structures
  - Array
  - List
  - Hashtable
  - Btree

Access Pattern



Access Pattern - Pseudo Random

```
 139508,315690 task-clock                #    5,996 CPUs utilized            ( +-  0,03% ) 100,00%
        12.673 context-switches          #    0,091 K/sec                    ( +-  0,24% ) 100,00%
           118 CPU-migrations            #    0,001 K/sec                    ( +- 13,19% ) 100,00%
         2.902 page-faults               #    0,021 K/sec                    ( +-  3,91% )
78.854.265.280 cycles                    #    0,565 GHz                      ( +-  1,10% ) 40,58%
 2.321.529.838 stalled-cycles-frontend   #    2,94% frontend cycles idle     ( +-  4,28% ) 38,57%
30.009.130.064 stalled-cycles-backend    #   38,06% backend  cycles idle     ( +-  0,12% ) 42,99%
62.716.912.813 instructions              #    0,80  insns per cycle
                                         #    0,48  stalled cycles per insn  ( +-  4,37% ) 40,51%
18.228.924.903 branches                  #  130,666 M/sec                    ( +-  2,11% ) 38,56%
    65.516.732 branch-misses             #    0,36% of all branches          ( +-  8,56% ) 38,27%
31.569.908.790 L1-dcache-loads           #  226,294 M/sec                    ( +-  1,30% ) 37,04%
    18.172.049 L1-dcache-load-misses     #    0,06% of all L1-dcache hits    ( +- 22,86% ) 40,92%
    50.183.885 LLC-loads                 #    0,360 M/sec                    ( +- 14,13% ) 40,83%
    28.160.133 LLC-load-misses           #   56,11% of all LL-cache hits     ( +-  7,80% ) 41,73%

  23,266494724 seconds time elapsed                                          ( +-  0,03% )
```

Access Pattern - Linear

```
  8042,299034 task-clock                #    5,998 CPUs utilized            ( +-  0,05% ) 100,00%
          878 context-switches          #    0,109 K/sec                    ( +- 13,15% ) 100,00%
           24 CPU-migrations            #    0,003 K/sec                    ( +- 44,68% ) 100,00%
        2.646 page-faults               #    0,329 K/sec                    ( +-  1,76% )
5.957.240.006 cycles                    #    0,741 GHz                      ( +- 30,84% ) 35,44%
1.419.666.336 stalled-cycles-frontend   #   23,83% frontend cycles idle     ( +- 10,87% ) 62,51%
1.096.067.572 stalled-cycles-backend    #   18,40% backend  cycles idle     ( +-  9,10% ) 75,54%
3.087.516.698 instructions              #    0,52  insns per cycle
                                        #    0,46  stalled cycles per insn  ( +- 11,80% ) 77,89%
1.567.151.858 branches                  #  194,864 M/sec                    ( +- 17,85% ) 59,37%
    3.965.022 branch-misses             #    0,25% of all branches          ( +- 30,58% ) 31,14%
3.060.755.224 L1-dcache-loads           #  380,582 M/sec                    ( +- 22,17% ) 19,07%
    5.074.888 L1-dcache-load-misses     #    0,17% of all L1-dcache hits    ( +- 46,08% )  9,92%
   12.268.349 LLC-loads                 #    1,525 M/sec                    ( +- 44,56% ) 10,60%
    2.566.738 LLC-load-misses           #   20,92% of all LL-cache hits     ( +- 47,68% ) 18,55%

  1,340719886 seconds time elapsed                                          ( +-  0,05% )
```

Example 2

Shared Resources

Example 2

Input: global variable i
pthread_barrier_wait(&barrier);
for $i \leftarrow 0$ **to** 2^{30} **do**
    i++;
end

Input: global variable i
pthread_barrier_wait(&barrier);
for $i \leftarrow 0$ **to** 2^{30} **do**
    i++;
end

Shared Resources

```
$ perf stat -d -r 10 ./a.out
  197170,885666 task-clock                #    5,318 CPUs utilized            ( +-  3,90% )
         12.611 context-switches          #    0,064 K/sec                    ( +- 20,75% )
             93 CPU-migrations            #    0,000 K/sec                    ( +- 22,33% )
            207 page-faults               #    0,001 K/sec                    ( +-  0,18% )
642.275.775.062 cycles                    #    3,257 GHz                      ( +-  3,90% ) 40,00%
 14.223.466.981 stalled-cycles-frontend   #    2,21% frontend cycles idle     ( +-  2,44% ) 40,01%
608.715.881.283 stalled-cycles-backend    #   94,77% backend  cycles idle     ( +-  3,98% ) 40,00%
  4.020.392.445 instructions              #    0,01  insns per cycle
                                          #  151,41  stalled cycles per insn  ( +-  1,02% ) 40,01%
    692.627.784 branches                  #    3,513 M/sec                    ( +-  1,25% ) 40,02%
      5.289.831 branch-misses             #    0,76% of all branches          ( +- 14,30% ) 40,01%
 14.730.027.536 L1-dcache-loads           #   74,707 M/sec                    ( +-  2,59% ) 40,01%
    392.392.754 L1-dcache-load-misses     #    2,66% of all L1-dcache hits    ( +-  2,72% ) 40,01%
  1.088.906.661 LLC-loads                 #    5,523 M/sec                    ( +-  2,64% ) 40,01%
  1.015.331.233 LLC-load-misses           #   93,24% of all LL-cache hits     ( +-  2,76% ) 40,00%

  37,072694341 seconds time elapsed                                          ( +-  2,64% )
```

Pipelining

- Modern processors employ a technique called pipelining to increase instruction throughput
- Various pieces of hardware perform various operation at the same time (parallel)
  - Part A of the pipeline is executing instruction FOO
  - Part B fetch instruction Bar
  - Part C decode instruction X
  - Part D commit results from instruction FOO

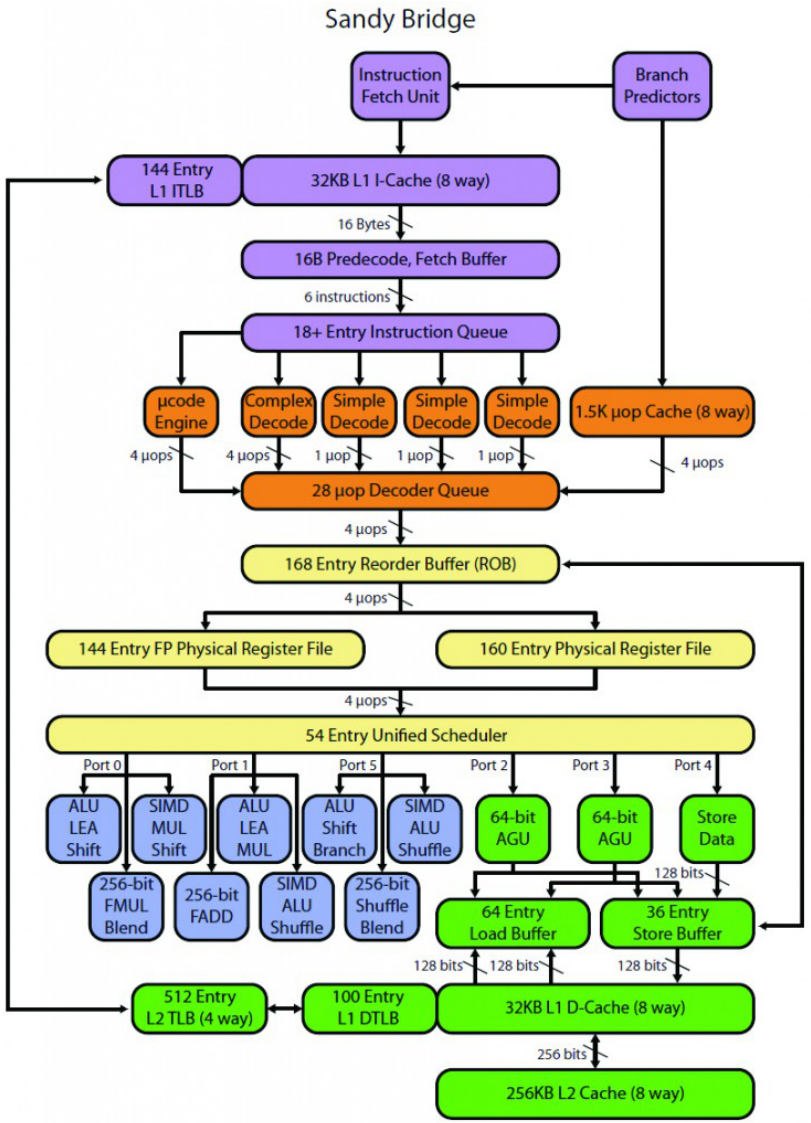
Frontend

- Responsible for providing a stream of work for the back-end
- Front-end operate ”in-order”
- Working on instructions (but generate uops for backend, called decoding)
- Sandy Bridge Front-end is capable of delivering up to 4 instructions/cycle (4 decoders) to the back.

Backend

- Front-end operate ”out-of-order” (often)
- Working on micro-operations (uops)

Sandy Bridge Microarchitecture



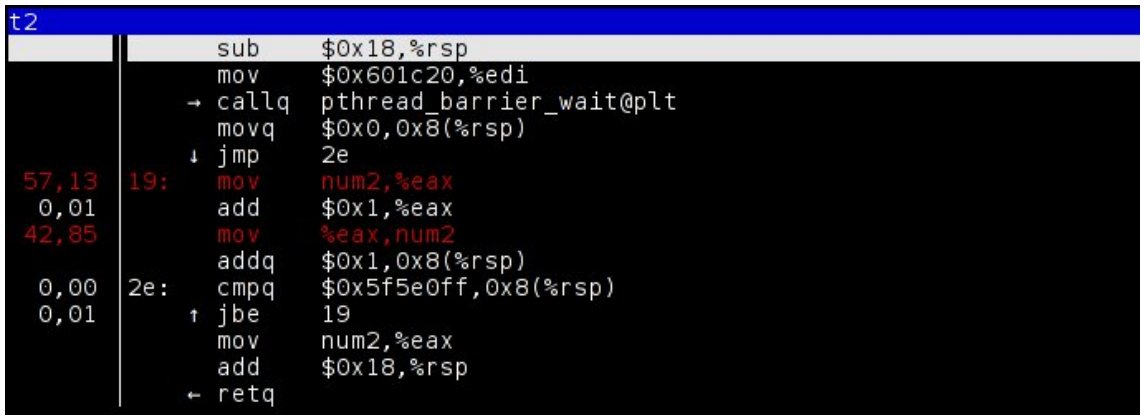
Find the hot spot

```
perf record -a -g -e stalled-cycles-backend ./a.out
perf report
```

Find the hot spot

```
Samples: 814K of event 'stalled-cycles-backend', Event count (approx.): 627665590172
+   18,35%  false-sharing  false-sharing                [.] t2
+   18,32%  false-sharing  false-sharing                [.] t4
+   18,24%  false-sharing  false-sharing                [.] t3
+   17,80%  false-sharing  false-sharing                [.] t1
+   13,21%  false-sharing  false-sharing                [.] t6
+   12,69%  false-sharing  false-sharing                [.] t5
+    1,23%         swapper  [kernel.kallsyms]           [k] native_safe_halt
+    0,02%            Xorg  [kernel.kallsyms]           [k] delay_tsc
+    0,02%            Xorg  [kernel.kallsyms]           [k] in_lock_functions
+    0,01%            Xorg  [kernel.kallsyms]           [k] native_read_tsc
```

Find the hot spot



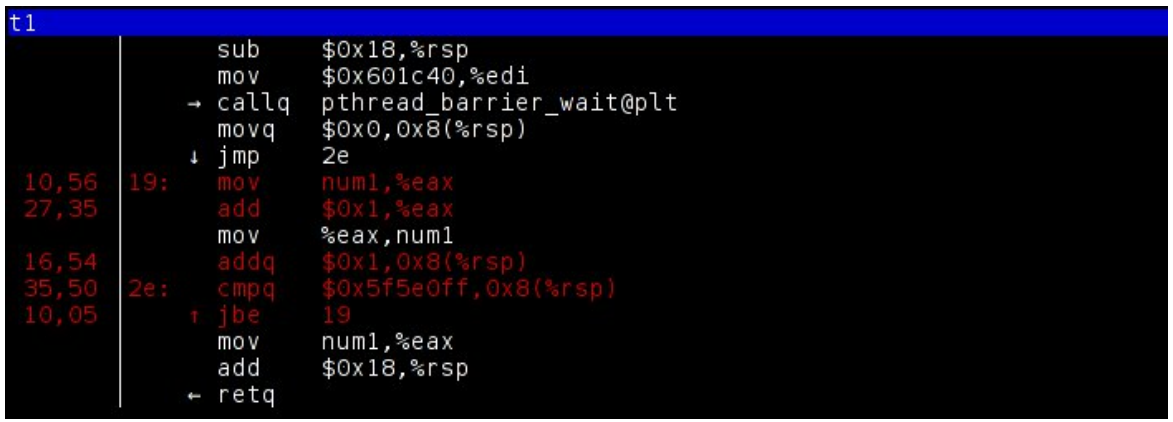
False Sharing

- False sharing is when different threads access non-overlapping areas of a cachline
- Here: one cachline is shared between 6 cores!
- Causing lines to be renewed regularly, if any thread writes to it
- BTW: Linux Kernel have `PER_CPU` (`DEFINE_PER_CPU(type, name),...`)

Try to fix this

```
$ diff -Nuar false-sharing.c.old false-sharing.c
--- false-sharing.c.old 2012-05-23 22:07:52.000000000 +0000
+++ false-sharing.c     2012-05-23 22:08:16.000000000 +0000
@@ -33,12 +33,12 @@

  static pthread_barrier_t barrier;

   -static unsigned num1;
   -static unsigned num2;
   -static unsigned num3;
   -static unsigned num4;
   -static unsigned num5;
   -static unsigned num6;
   +static unsigned num1 ____cacheline_aligned;
   +static unsigned num2 ____cacheline_aligned;
   +static unsigned num3 ____cacheline_aligned;
   +static unsigned num4 ____cacheline_aligned;
   +static unsigned num5 ____cacheline_aligned;
   +static unsigned num6 ____cacheline_aligned;

$ make
$ perf stat -a -d -r 10 ./false-sharing
   1257,529627 task-clock                #    6,011 CPUs utilized           ( +-  0,82% ) 100,00%
           422 context-switches          #    0,335 K/sec                   ( +-  1,42% ) 100,00%
            22 CPU-migrations            #    0,017 K/sec                   ( +-  4,24% ) 100,00%
           212 page-faults               #    0,169 K/sec                   ( +-  0,18% )
 3.701.493.777 cycles                    #    2,943 GHz                     ( +-  0,98% ) 40,82%
   190.292.288 stalled-cycles-frontend   #    5,14% frontend cycles idle    ( +-  6,17% ) 41,67%
 2.207.591.340 stalled-cycles-backend    #   59,64% backend  cycles idle    ( +-  0,72% ) 41,52%
 3.623.979.984 instructions              #    0,98  insns per cycle
                                         #    0,61  stalled cycles per insn ( +-  0,49% ) 40,91%
   630.872.872 branches                  #  501,676 M/sec                   ( +-  0,90% ) 39,90%
       270.216 branch-misses             #    0,04% of all branches         ( +-  7,45% ) 38,93%
 2.547.084.258 L1-dcache-loads           # 2025,467 M/sec                   ( +-  1,03% ) 38,56%
       337.080 L1-dcache-load-misses     #    0,01% of all L1-dcache hits   ( +- 13,59% ) 38,85%
       509.484 LLC-loads                 #    0,405 M/sec                   ( +- 10,60% ) 39,17%
       120.115 LLC-load-misses           #   23,58% of all LL-cache hits    ( +-  9,60% ) 39,82%

   0,209195268 seconds time elapsed                                         ( +-  0,82% )
```

SNAFU


t1
sub $0x18,%rsp
mov $0x601c40,%edi
callq pthread_barrier_wait@plt
movq $0x0,0x8(%rsp)
jmp 2e
10,56 19: mov num1,%eax
27,35 add $0x1,%eax
mov %eax,num1
16,54 addq $0x1,0x8(%rsp)
35,50 2e: cmpq $0x5f5e0ff,0x8(%rsp)
10,05 jbe 19
mov num1,%eax
add $0x18,%rsp
retq

Suggested Readings

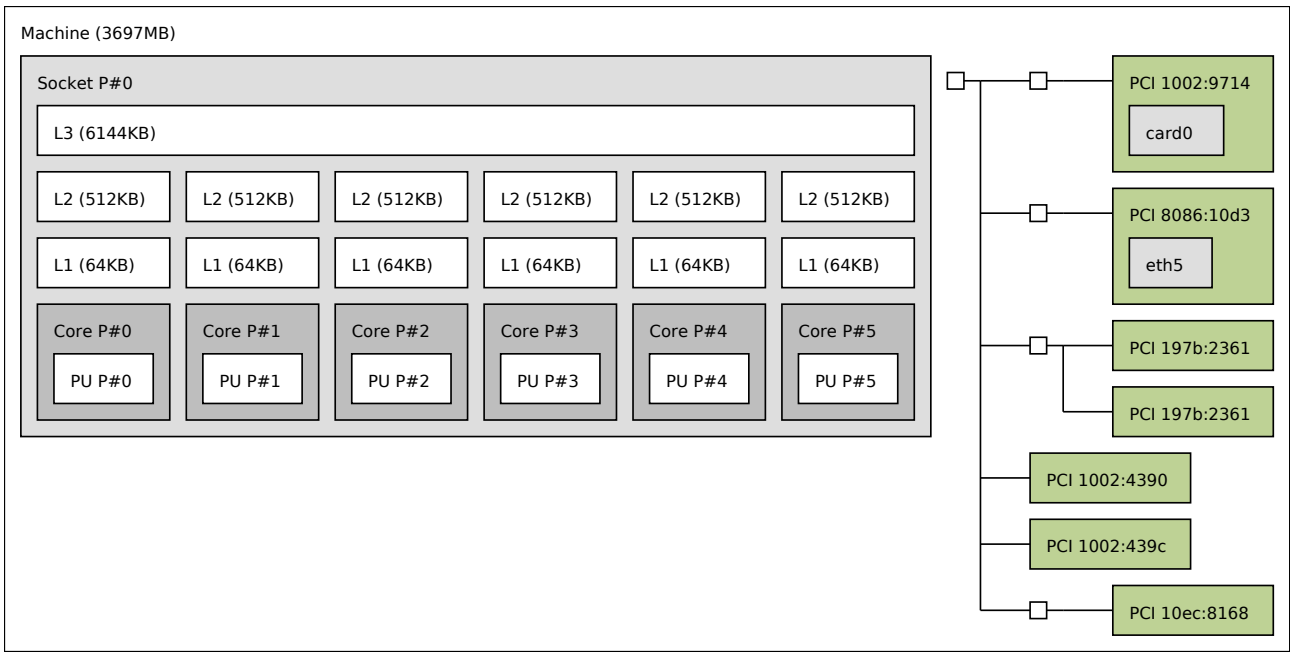
- Books
  - *Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture* from Jon Stokes
  - *Multi-Core Programmierung* Intel Press
- Online Articles:
  - Intel Guide for Developing Multithreaded Applications `http://software.intel.com/en-us/articles/intel-guide-for-developing-multithreaded-applications/`
  - The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms, Michael E. Thomadakis, Supercomputing Facility, Texas A&M University
  - BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors
  - Intel XEON Processor 7500 Series Uncore Programming Guide
  - Intel 5520 Chipset and Intel 5500 Chipset
  - Intel Microarchitecture Codename Nehalem Performance Monitoring Unit Programming Guide `http://software.intel.com/file/30320`
  - Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processor

Thank you very much!

Questions?

Modern Memory Architecture

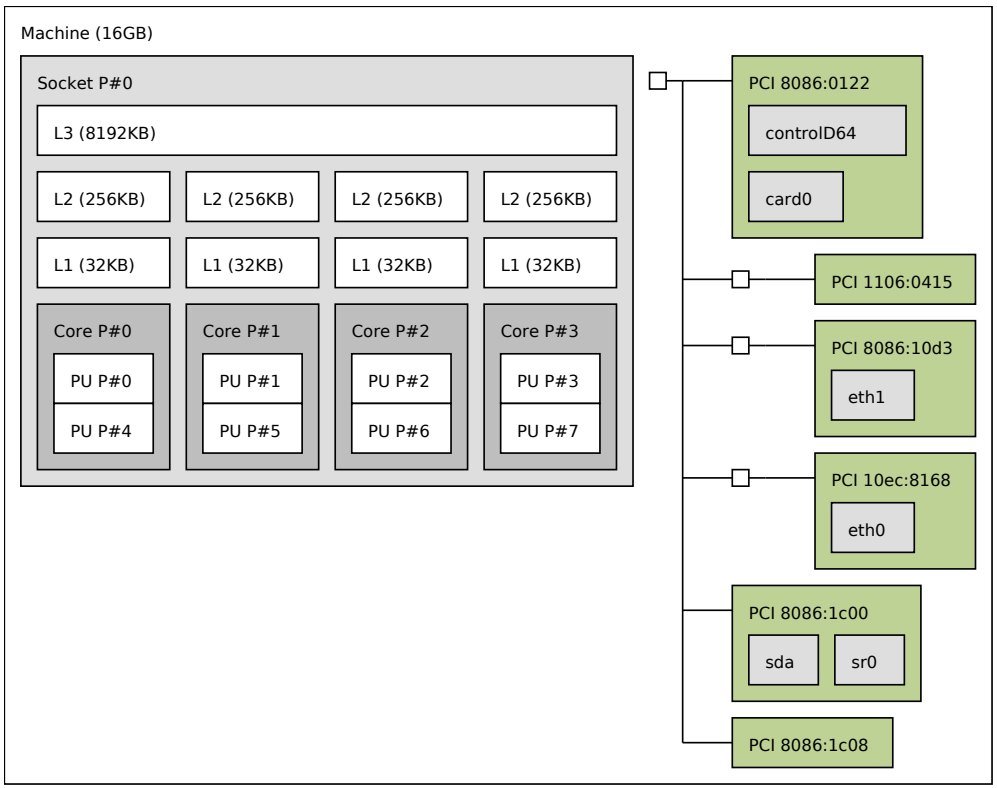
AMD Phenom(tm) II X6 1055T Processor[6]



[6] generated via `lstopo --no-legend mm-arch-x2-1055.pdf`; Debian package: *hwloc*

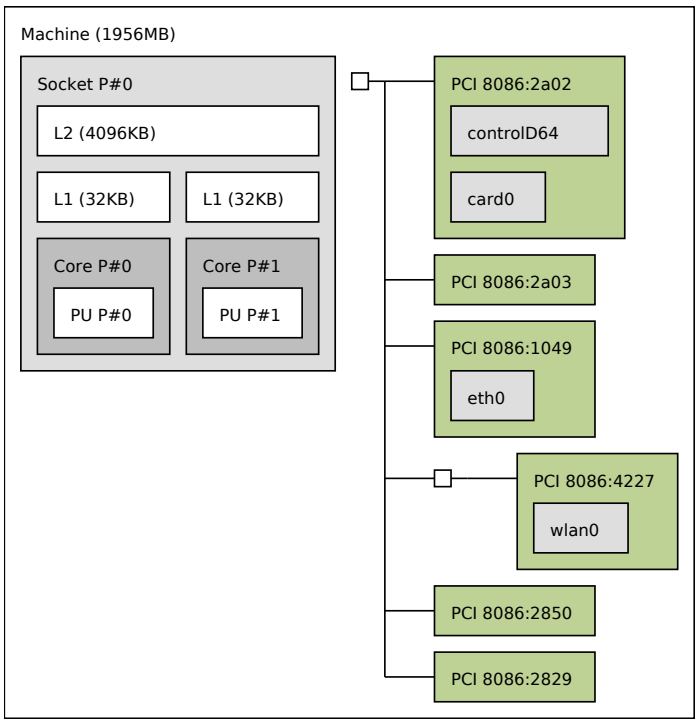
Modern Memory Architecture

Intel(R) Core(TM) i7-2700K CPU @ 3.50GHz



Modern Memory Architecture

Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz

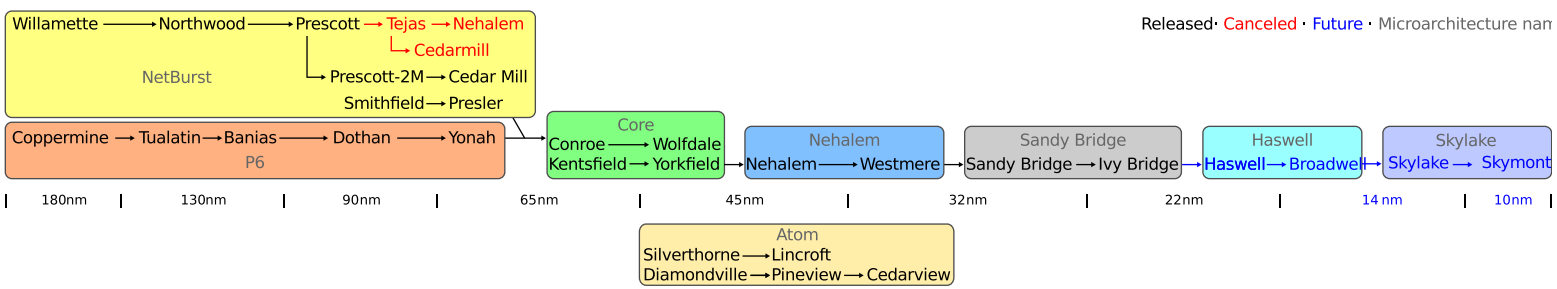


Intel Sandy Bridge Architecture

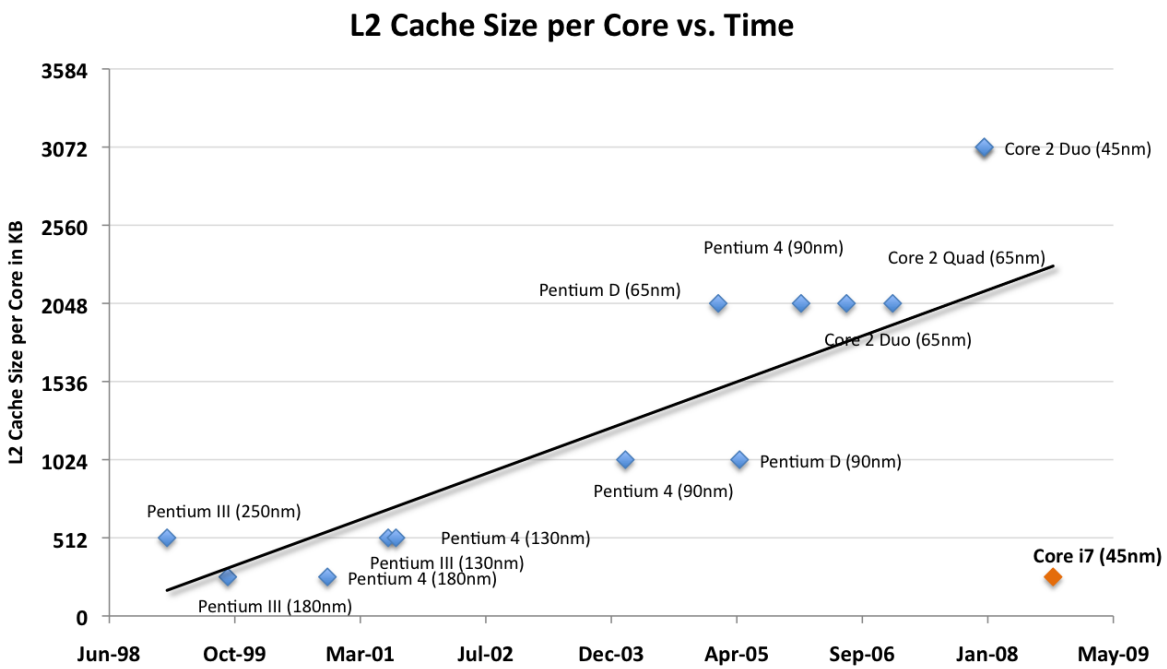
Target segment	Cores (Threads)	Processor Branding & Model		CPU Clock rate Normal	CPU Clock rate Turbo	Graphics Clock rate Normal	Graphics Clock rate Turbo	L3 Cache	TDP	Release Date (Y-M-D)	Price (USD)	Motherboard Socket	Motherboard Interface	Motherboard Memory
Extreme / High-End	6 (12)	Core i7 Extreme	3960X	3.3 GHz	3.9 GHz	N/A		15 MB	130 W	2011-11-14	$999	LGA 2011	DMI 2.0 PCIe 2.0[14]	Up to quad channel DDR3-1600[15]
		Core i7	3930K	3.2 GHz	3.8 GHz			12 MB			$583			
	4 (8)		3820	3.6 GHz				10 MB		2012-02-13[16]	$294			
Performance			**2700K**	3.5 GHz	3.9 GHz	850 MHz	1350 MHz	8 MB	95 W	2011-10-24	$332	LGA 1155	DMI 2.0 PCIe 2.0	Up to dual channel DDR3-1333
			2600K	3.4 GHz	3.8 GHz					2011-01-09	$317			
			2600								$294			
			2600S	2.8 GHz					65 W		$306			
	4 (4)	Core i5	2550K	3.4 GHz		N/A		6 MB	95 W	2012-01-30	$225			
			2500K	3.3 GHz	3.7 GHz	850 MHz	1100 MHz			2011-01-09	$216			
			2500								$205			
			2500S	2.7 GHz					65 W		$216			
			2500T	2.3 GHz	3.3 GHz	650 MHz	1250 MHz		45 W					
			2450P	3.2 GHz	3.5 GHz	N/A			95 W	2012-01-30	$195			
			2400	3.1 GHz	3.4 GHz	850 MHz	1100 MHz			2011-01-09	$184			
			2405S	2.5 GHz	3.3 GHz				65 W	2011-05-22	$205			
			2400S							2011-01-09	$195			
			2380P	3.1 GHz	3.4 GHz	N/A			95 W	2012-01-30	$177			
			2320	3.0 GHz	3.3 GHz	850 MHz				2011-09-04				
			2310	2.9 GHz	3.2 GHz					2011-05-22				
			2300	2.8 GHz	3.1 GHz					2011-01-09				
			2390T	2.7 GHz	3.5 GHz	650 MHz			35 W	2011-02-20	$195			
		Core i3	2120T	2.6 GHz						2011-09-04	$127			
			2100T	2.5 GHz						2011-02-20				
			2130	3.4 GHz							$138			

Intel Sandy Bridge Architecture

- Replace Nehalem microarchitecture
- Released first January 2011, developed beginning in 2005
- Major features
  - 32 kB data + 32 kB instruction L1 cache (3 clocks) and 256 kB L2 cache (8 clocks) per cor
  - 64-byte cache line size
  - Shared L3 cache includes the processor graphics
  - Decoded micro-operation cache and enlarged, optimized branch predictor
  - Two load/store operations per CPU cycle for each memory channel
- Roadmap:



Cache Evolution



source: anandtech.com

Use Latest Perf Version

- You can use Arnaldo's or Peter's branch directly (sometimes I do), but I suggest to use Ingo's development master branch! (tip:master)
- You are free to use perf/core from Ingo too.

```
git remote add tip git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip.git
git remote update tip
git checkout -b perf/core tip/perf/core
or
git checkout -b tip/masster tip/master
```

Latency Values

- Mutex lock/unlock 100 ns
- Send 2K bytes over 1 Gbps network 20,000 ns
- Read 1 MB sequentially from memory 250,000 ns
- Disk seek 10,000,000 ns
- ...

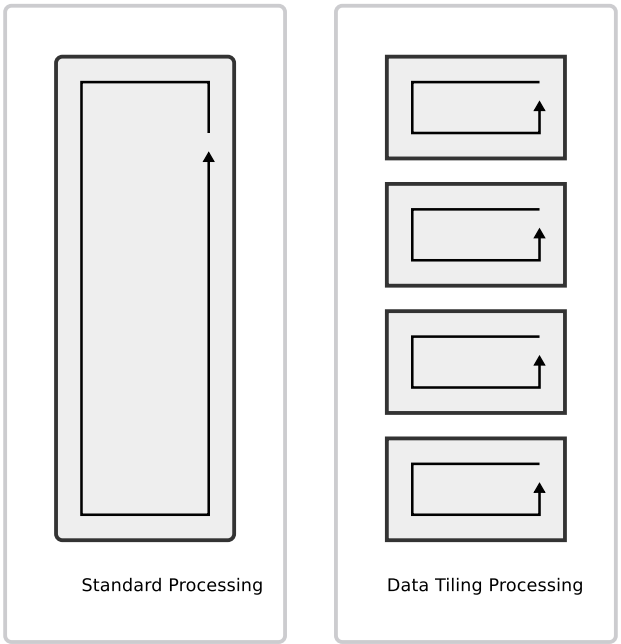
Perfmon2

- Alternative for perf
- Support sampling and counting
- System/Thread wide
- Perfmon provides a library, useable for self-monitoring!
- ...

Perfmon2

- Alternative for perf
- Support sampling and counting
- System/Thread wide
- Perfmon provides a library, useable for self-monitoring!
- ...
- Great tool to understand PMUs in detail!
  `http://perfman2.sf.net`

Data Tiling - Cache Blocking



Thread - CPU Detail

- watch -n ,1 ps -aLo pcpu,cpuid,pid,args