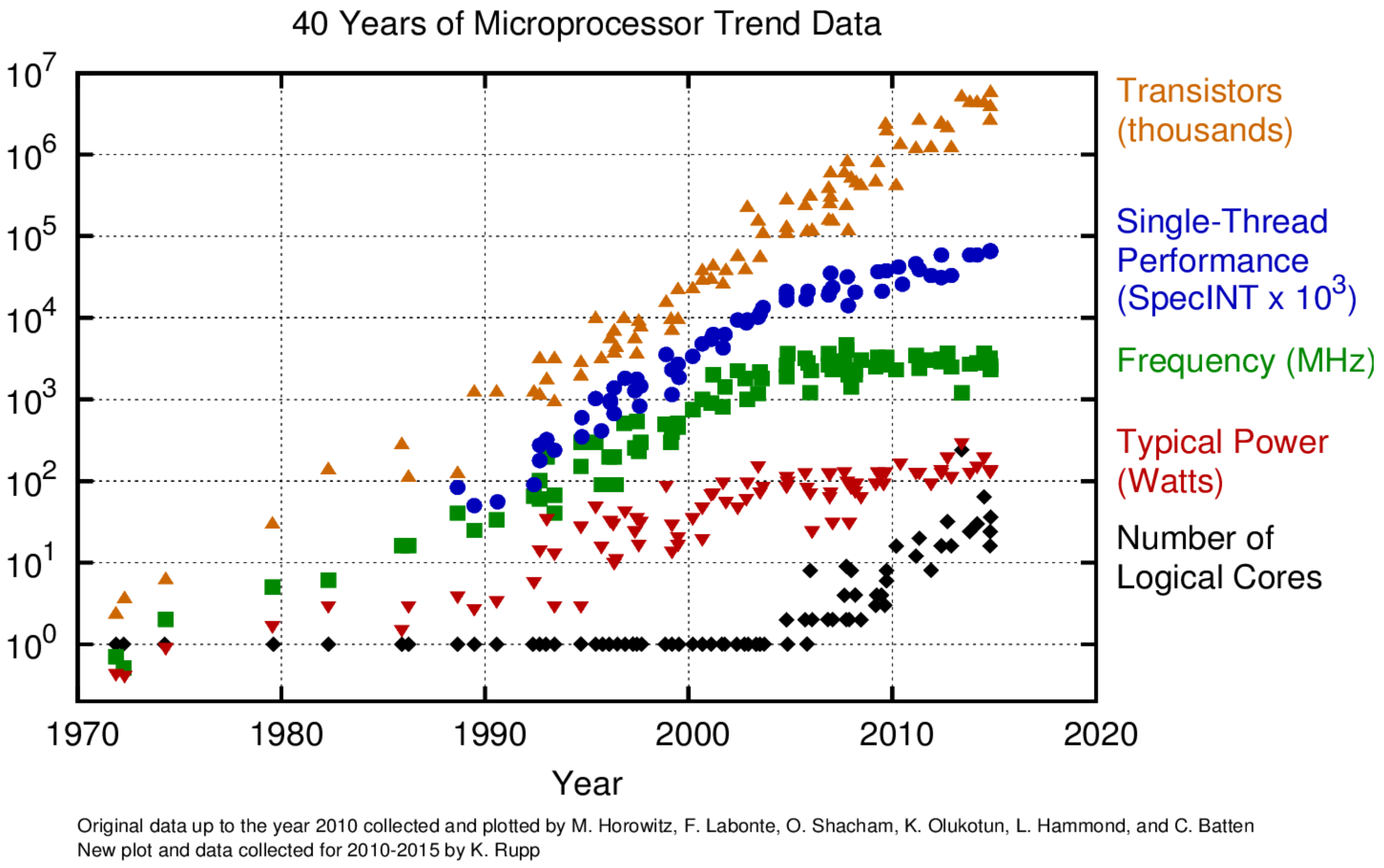
Performance analysis & tuning on modern CPU

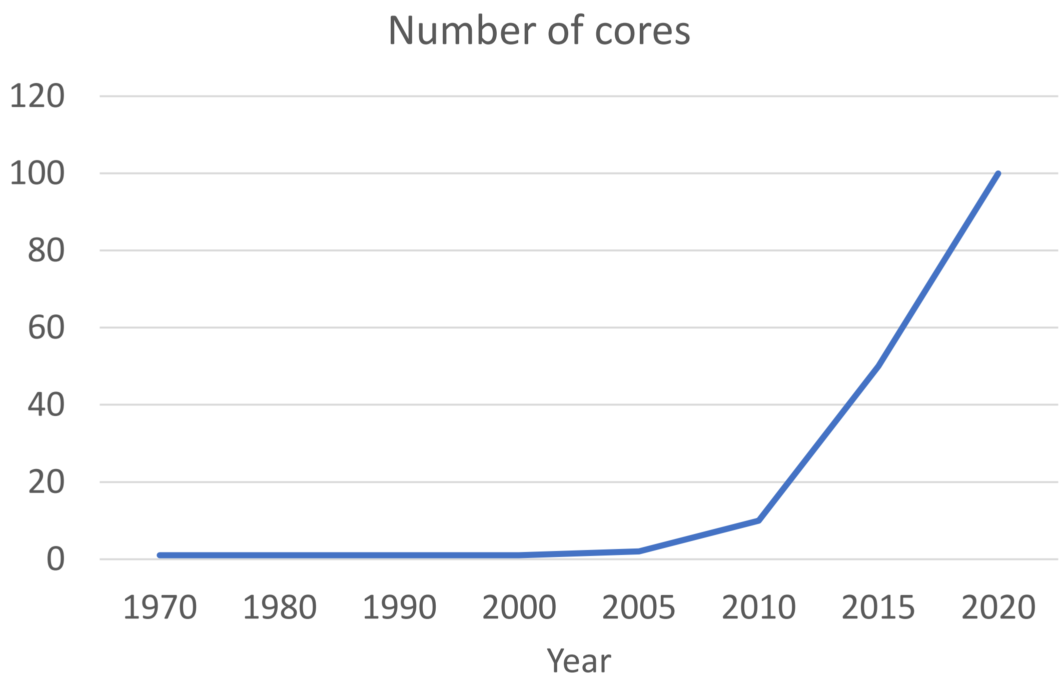
Denis Bakhvalov
C++ compiler dev

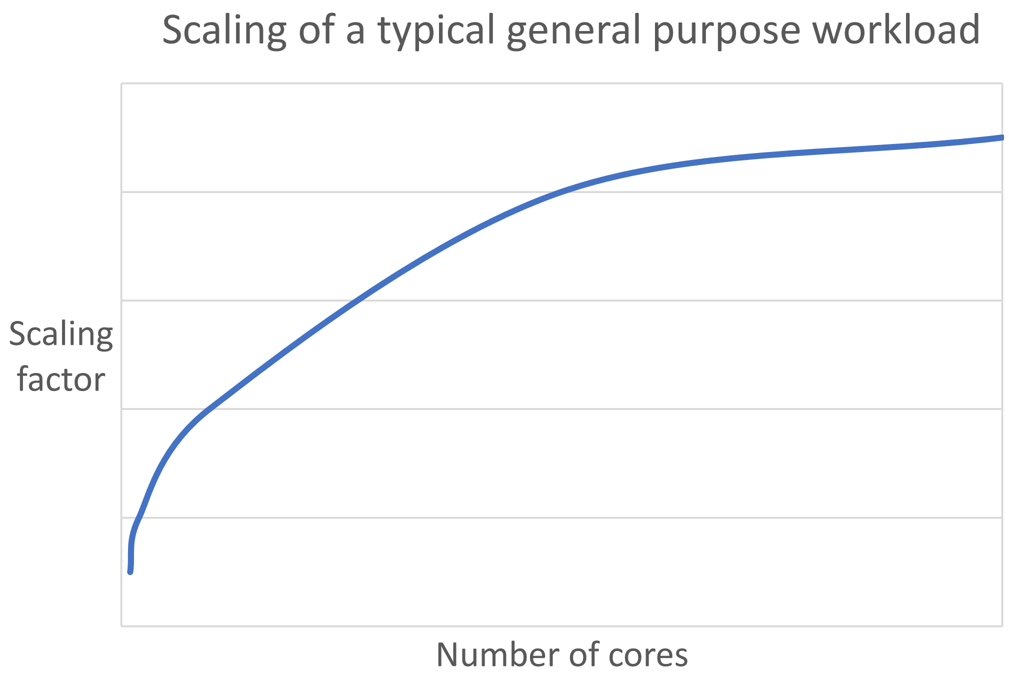
Performance



credit: [*https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/*](https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/)

Why do we need performance analysis?

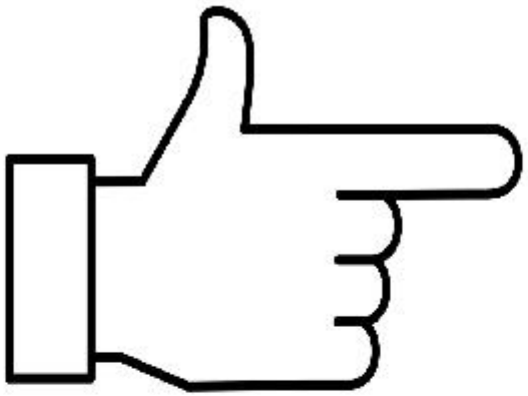


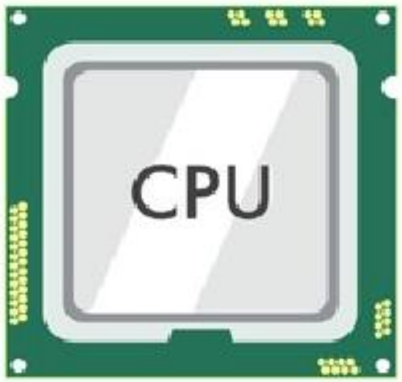


What is performance analysis?

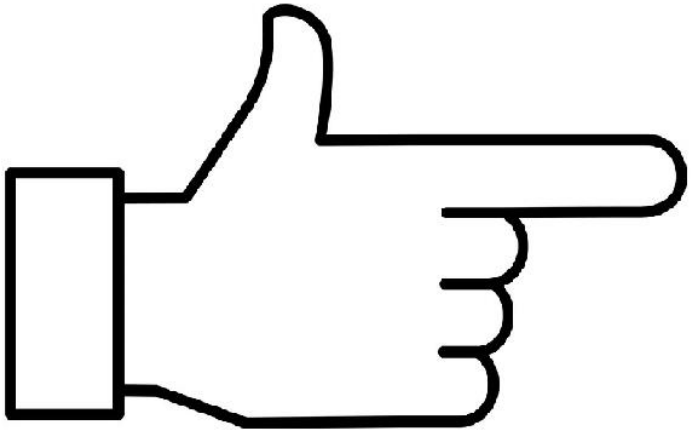
credit: image taken from istockphoto.com

Why HW does not solve all our problems?





Why compiler does not solve all our problems?



Agenda

- How to conduct fair experiments
- How to process measurements and compare results
- Application profiling
- How to find the CPU bottleneck
- CPU-specific code tuning examples

Motivational example

```cpp
// func.cpp




void benchmark_func(int* a)
{
	for (int i = 0; i < 32; ++i)
		a[i] += 1;
}
```

```cpp
// func.cpp
void foo(int* a)
{
	for (int i = 0; i < 32; ++i)
		a[i] += 1;
}

void benchmark_func(int* a)
{
	for (int i = 0; i < 32; ++i)
		a[i] += 1;
}
```

	Throughput, GB/s
Baseline	28.7
+ foo()	33.0 (+15%)

Motivational example

	Throughput, GB/s
Baseline	28.7
+ foo()	33.0 (+15%)
32B function alignment *	38.0 (+32%)
32B basic blocks alignment **	42.4 (+48%)

* -mllvm -align-all-functions=5

** -mllvm -align-all-blocks=5

https://easyperf.net/blog/2018/01/18/Code_alignment_issues

Agenda

- How to conduct fair experiments
- How to process measurements and compare results
- Application profiling
- How to find the CPU bottleneck
- CPU-specific code tuning examples

Conducting fair measurements

Have dedicated machine(s) that are tuned for performance measurements:

- minimal background processes
- no one allowed to log in (besides CI bots)
- non-deterministic features disabled

More details here:

https://easyperf.net/blog/2019/08/02/Perf-measurement-environment-on-Linux

Tool for setting up an environment for benchmarking (Linux and MacOS):

https://github.com/parttimenerd/temci

Agenda

- How to conduct fair experiments
- How to process measurements and compare results
- Application profiling
- How to find the CPU bottleneck
- CPU-specific code tuning examples

Processing measurements

Version A:

{ 90s ; 110s ; 120s }

Which is faster?

Version B:

{ 100s ; 105s ; 110s }

Advice

- No single metric is perfect.
- If possible, make many benchmark iterations (>30).
- Use mean/geomean carefully when the number of samples is small (<5), because it can be spoiled by outliers. Consider taking minimum/median.
- To be on a safe side, plot the distributions and let the readers drive their own conclusions.

https://easyperf.net/blog/2019/12/30/Comparing-performance-measurements

Agenda

- How to conduct fair experiments
- How to process measurements and compare results
- Application profiling
- How to find the CPU bottleneck
- CPU-specific code tuning examples

Performance analysis approaches

- Code instrumentation
- Profiling
- Tracing*

** not in this presentation, see more here:*
https://easyperf.net/blog/2019/08/23/Intel-Processor-Trace

Code instrumentation

```
int foo(int x) {
  printf("foo is called");
  // function body …
}
```

Profiling

Hotspots Hotspots by CPU Utilization

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | To

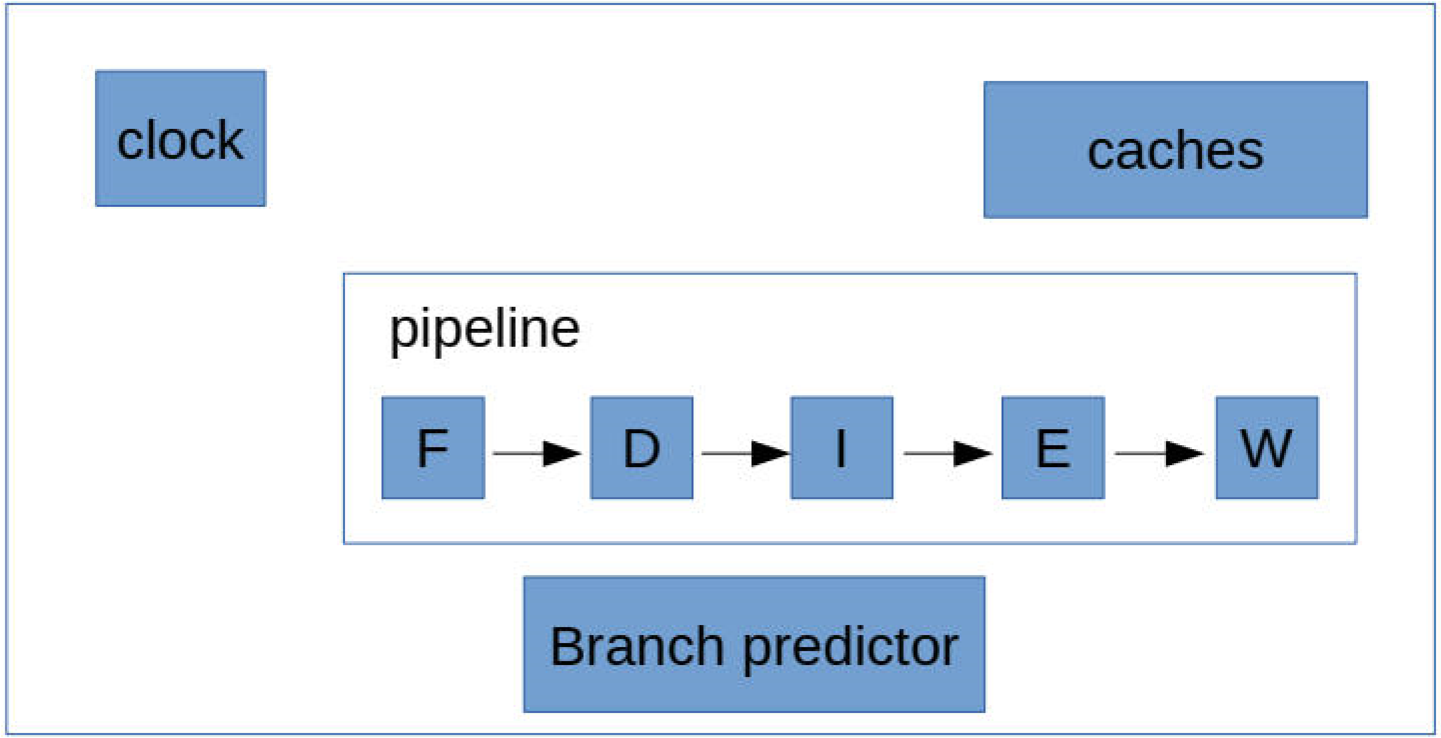
Grouping: Function / Call Stack

Function / Call Stack	CPU Time ▼	Instructions Retired
func1	88.002s	204,787,500,000
func2	85.125s	345,112,500,000
func3	65.018s	206,400,000,000
func4	36.546s	59,475,000,000
func5	35.368s	120,725,000,000
func6	28.482s	62,750,000,000

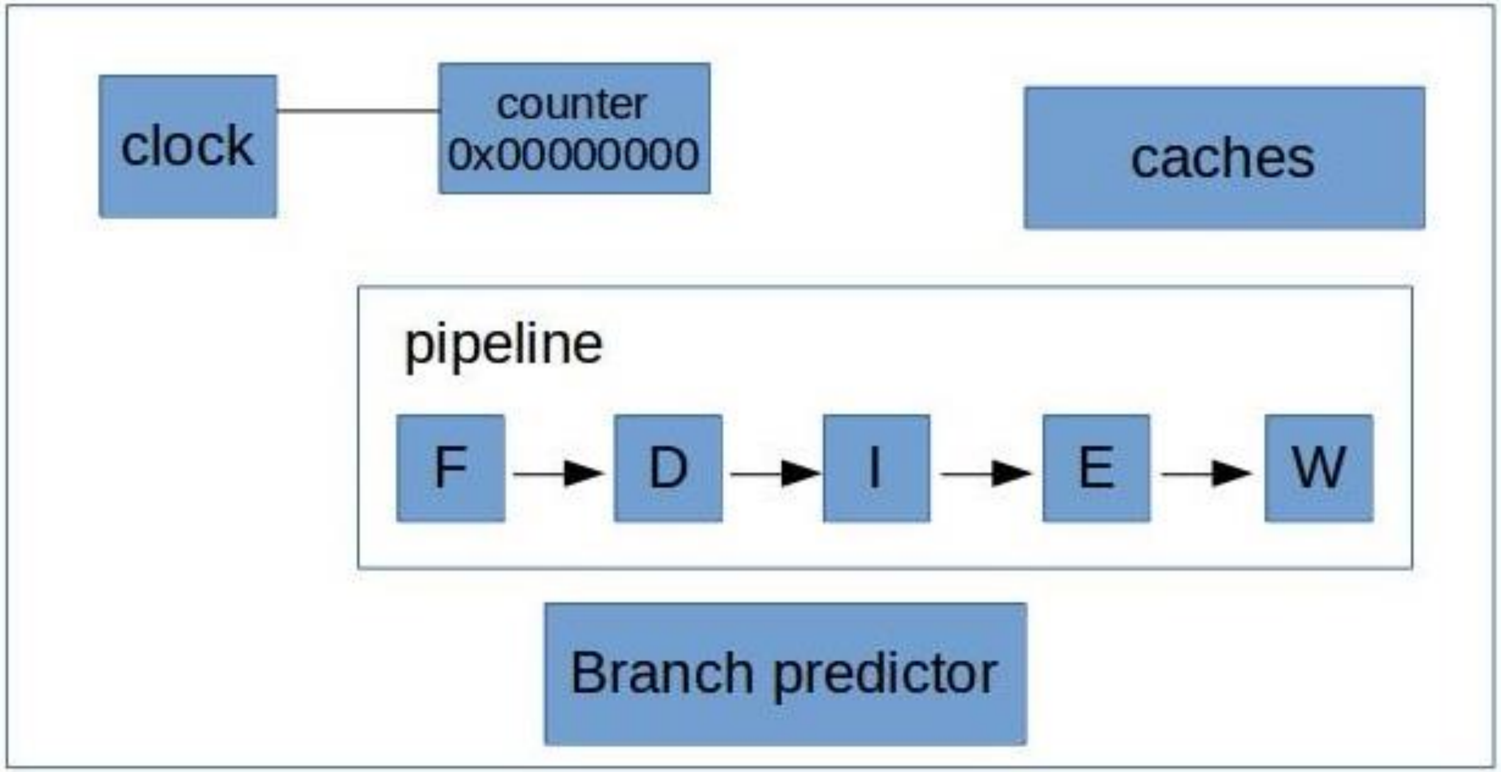
Easiest profiler ever?

A: Debugger!

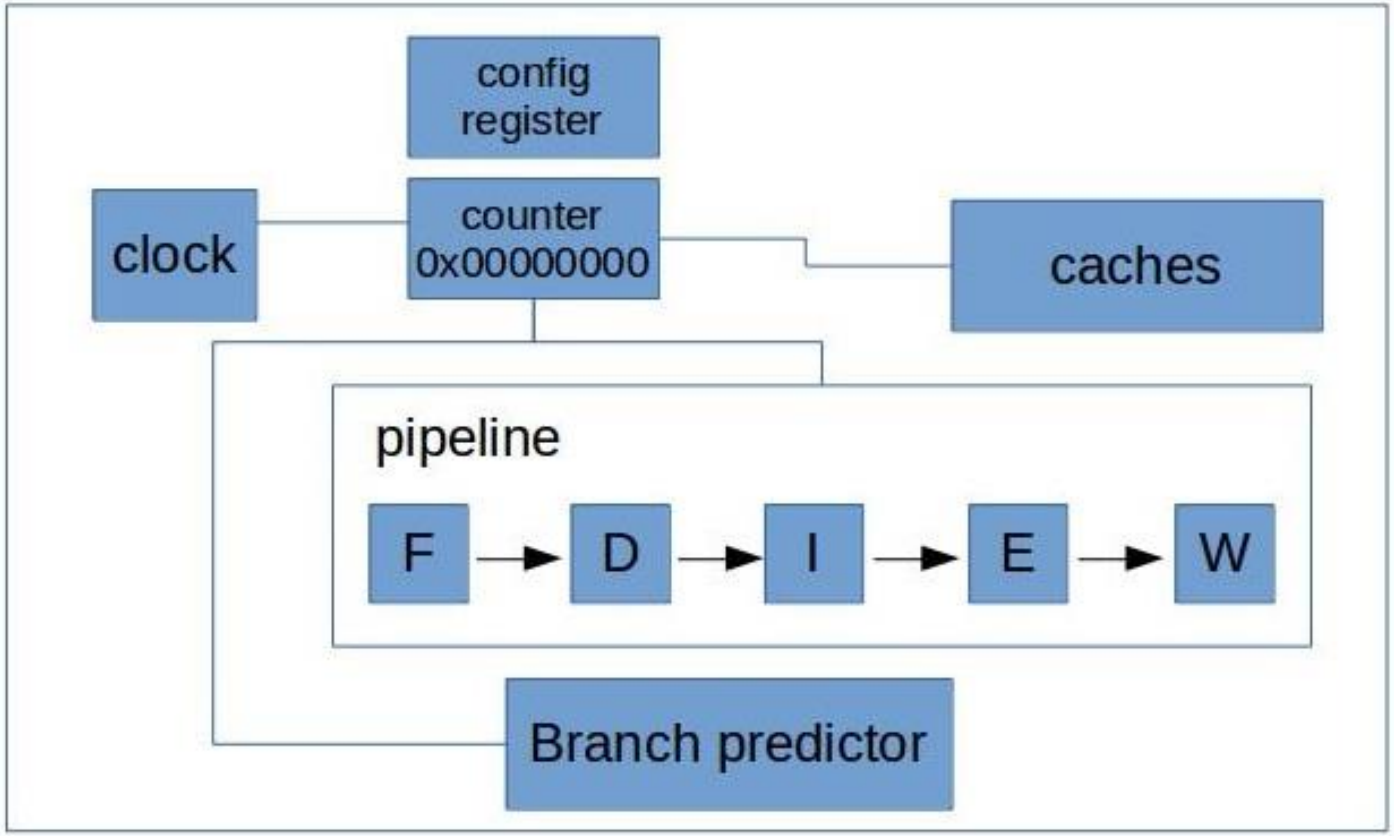
CPU mental model



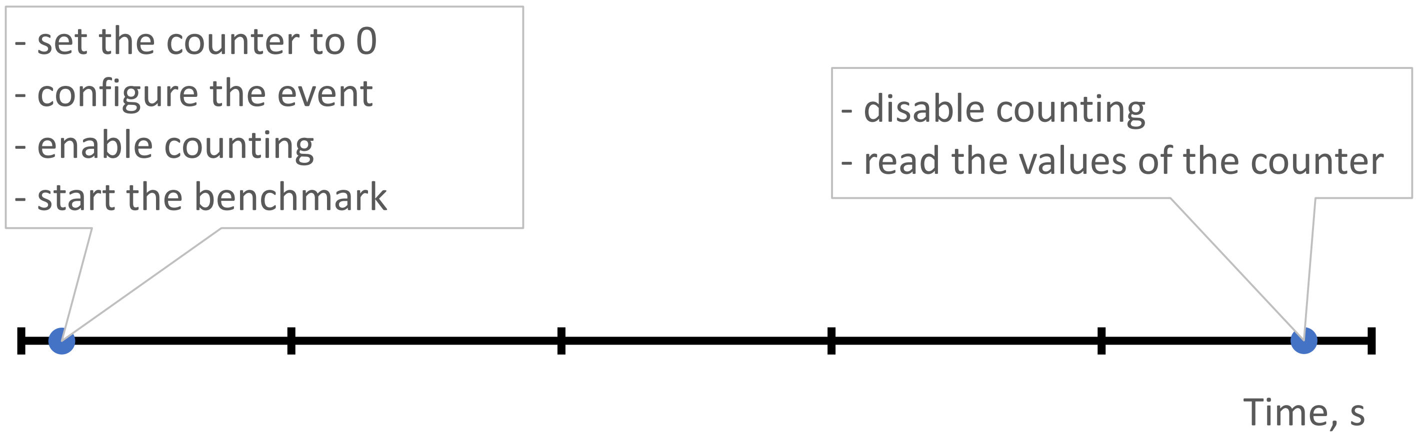
Counting cycles



Counting different events



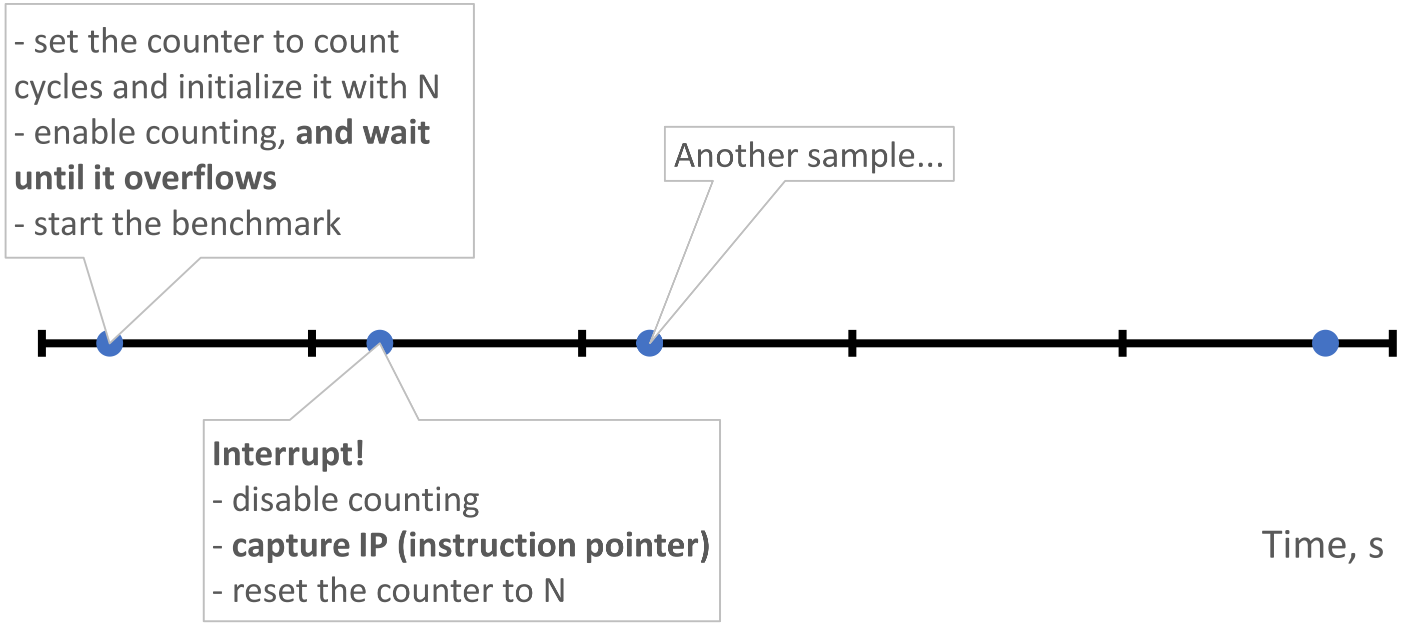
Counting events




```
$ perf stat -- ./a.exe

 10580290629   cycles          #    3,677 GHz
  8067576938   instructions    #    0,76  insn per cycle
  3005772086   branches        # 1044,472 M/sec
   239298395   branch-misses   #    7,96% of all branches
```

Sampling




```
$ perf record -- ./a.exe
$ perf report -n --stdio
# Samples: 434K of event 'cycles'
# Overhead       Samples  Shared Object  Symbol
# ........  ............  .............  .........
#
    68.53%        297420    a.exe        [.] foo
    15.76%         68398    a.exe        [.] bar
     3.64%         15797    a.exe        [.] main
```

Agenda

- How to conduct fair experiments
- How to process measurements and compare results
- Application profiling
- How to find the CPU bottleneck
- CPU-specific code tuning examples

Want to tune your code on CPU level?

- Fix major performance problems first.
- Avoid premature microarchitecture optimizations.

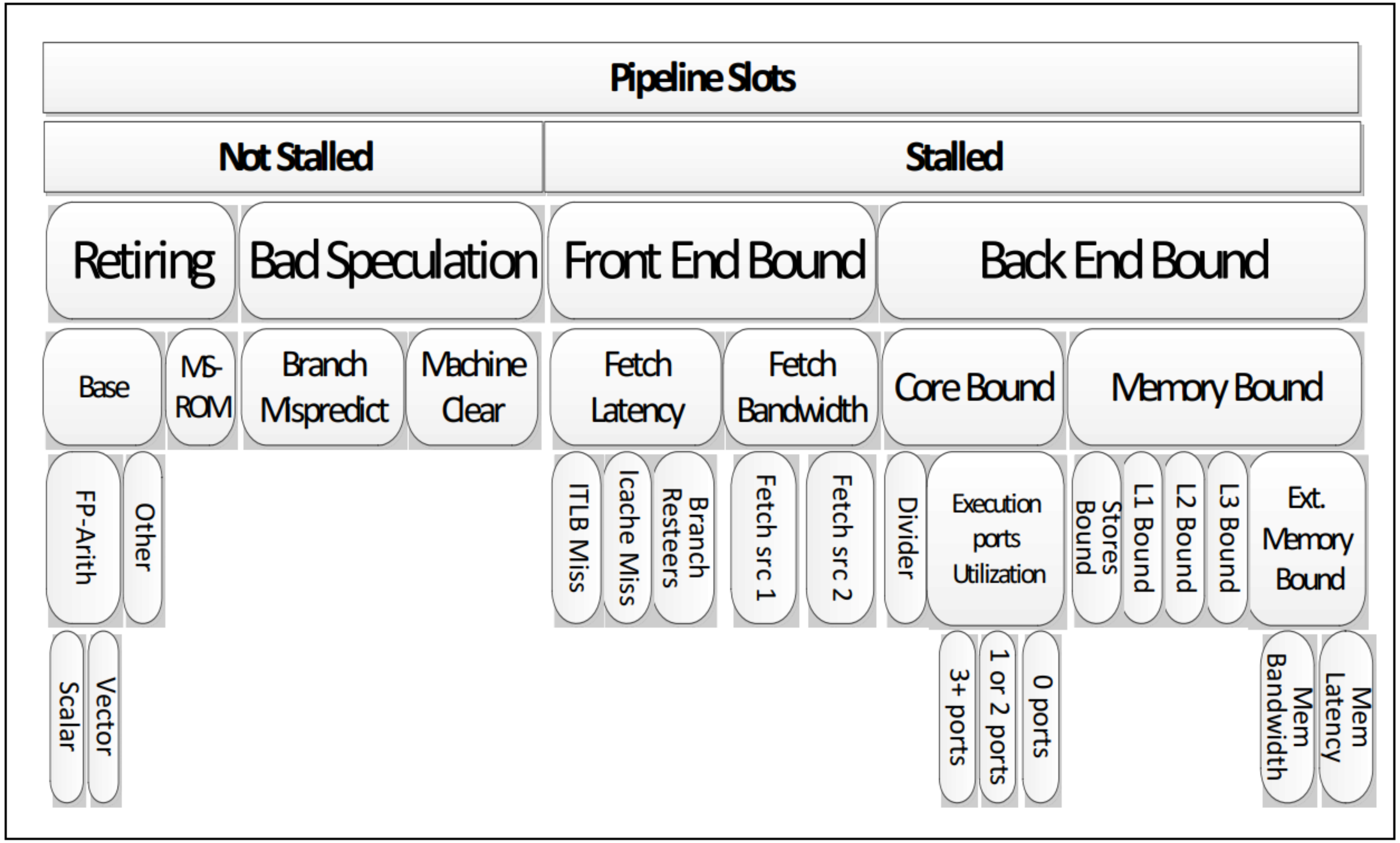
High level optimization tips

- If using a very high level / interpreted language (python, javascript, etc.), rewrite in a language with less overhead. <…>
- Analyze the algorithms and data structures you are using, see if you can find better ones. <…>
- Tune compiler options. Check that you use at least those 3: -O3/-Ofast, -march, -flto.
- If your problem is a highly parallelizable computation, make it threaded, or consider running it on the GPU.
- If you are doing a lot of IO, see if you can use async IO to avoid blocking while waiting for IO operations.
- See if you can leverage using more RAM to reduce the amount of CPU and IO you have to use (memoization, look-up tables, caching of data retrieved using IO, etc.)
- Buy beefier hardware. Consider renting some heavy compute instances in the cloud.

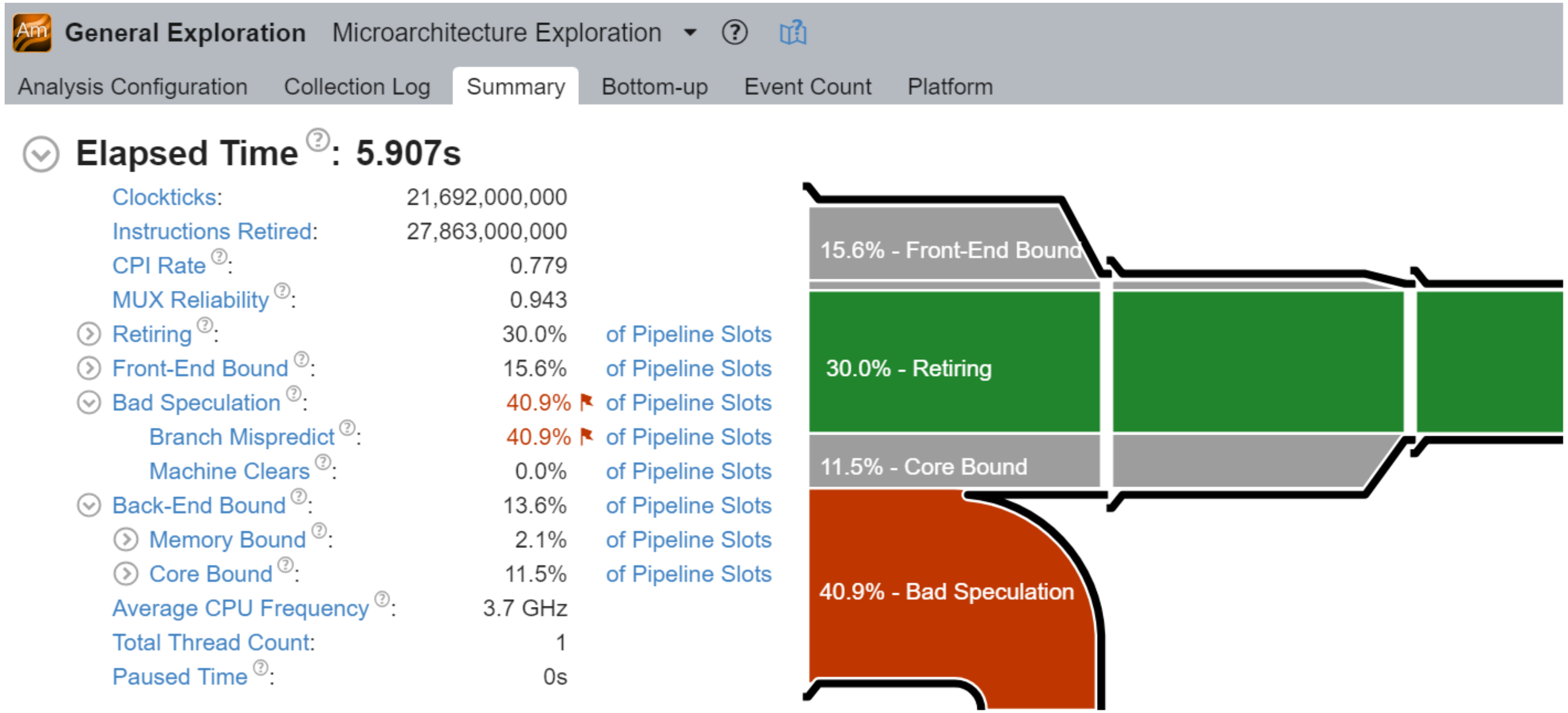
- *By u/AndreasTPC @Reddit*

TMAM

Top-Down Microarchitecture Analysis

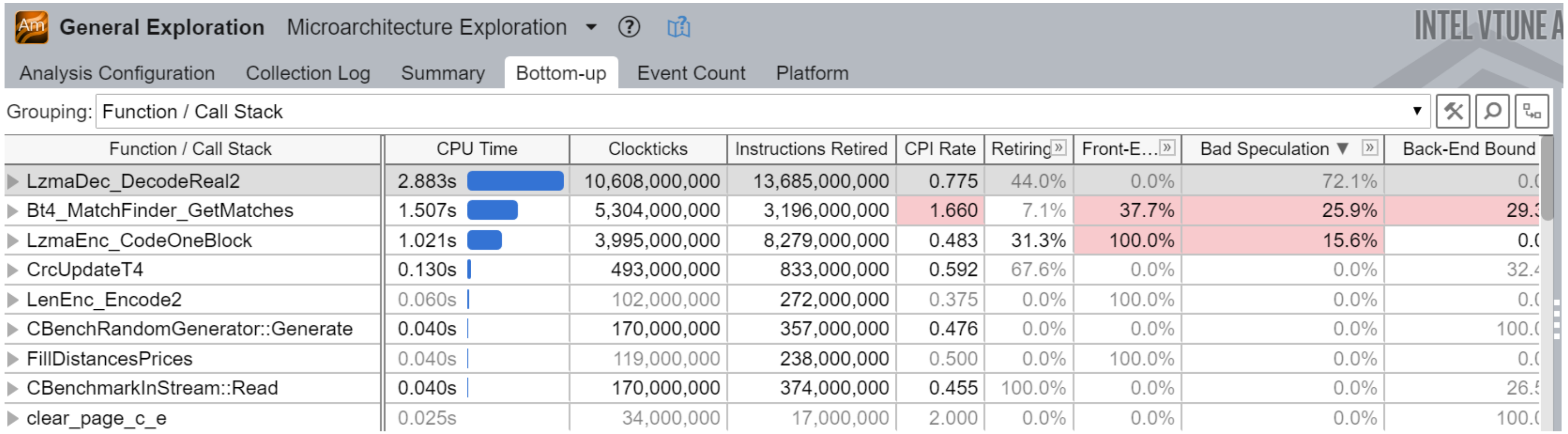


TMAM In VTune

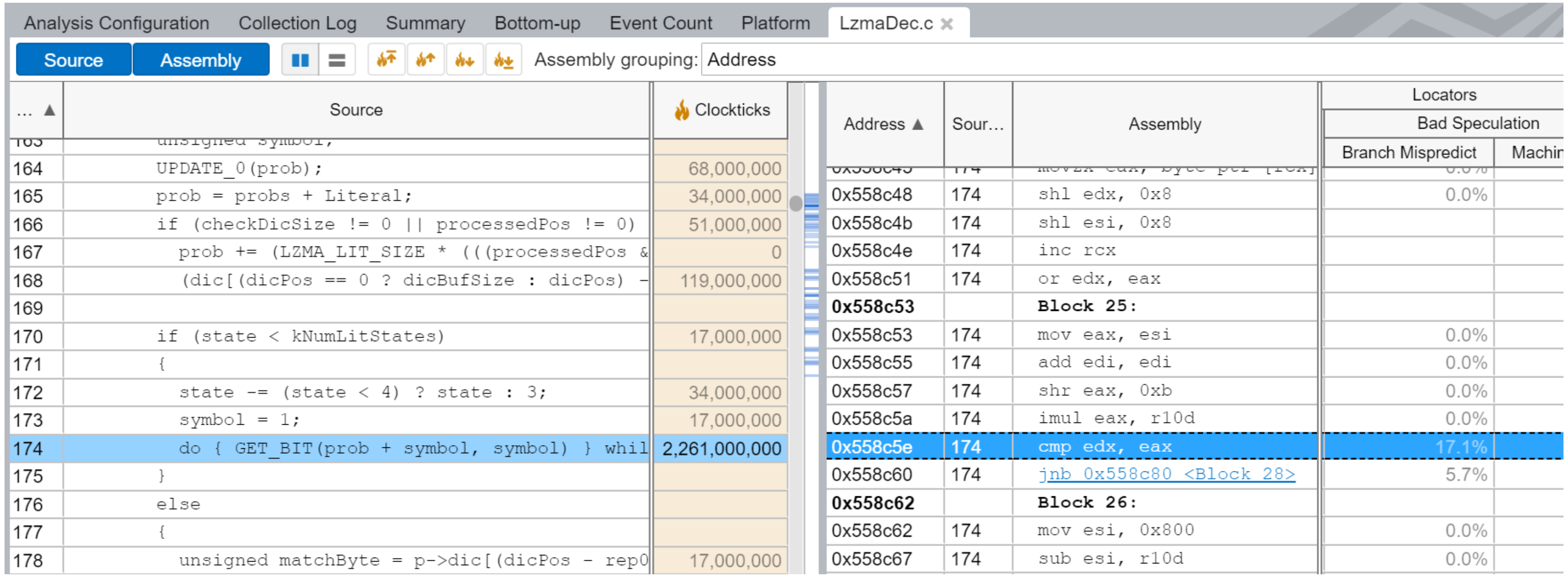


https://github.com/llvm-mirror/test-suite/tree/master/MultiSource/Benchmarks/7zip

VTune detailed view



Function / Call Stack	CPU Time	Clockticks	Instructions Retired	CPI Rate	Retiring	Front-E…	Bad Speculation	Back-End Bound
LzmaDec_DecodeReal2	2.883s	10,608,000,000	13,685,000,000	0.775	44.0%	0.0%	72.1%	0.
Bt4_MatchFinder_GetMatches	1.507s	5,304,000,000	3,196,000,000	1.660	7.1%	37.7%	25.9%	29.
LzmaEnc_CodeOneBlock	1.021s	3,995,000,000	8,279,000,000	0.483	31.3%	100.0%	15.6%	0.
CrcUpdateT4	0.130s	493,000,000	833,000,000	0.592	67.6%	0.0%	0.0%	32.
LenEnc_Encode2	0.060s	102,000,000	272,000,000	0.375	0.0%	100.0%	0.0%	0.
CBenchRandomGenerator::Generate	0.040s	170,000,000	357,000,000	0.476	0.0%	0.0%	0.0%	100.
FillDistancesPrices	0.040s	119,000,000	238,000,000	0.500	0.0%	100.0%	0.0%	0.
CBenchmarkInStream::Read	0.040s	170,000,000	374,000,000	0.455	100.0%	0.0%	0.0%	26.
clear_page_c_e	0.025s	34,000,000	17,000,000	2.000	0.0%	0.0%	0.0%	100.



…	Source	Clockticks
164	UPDATE_0(prob);	68,000,000
165	prob = probs + Literal;	34,000,000
166	if (checkDicSize != 0 || processedPos != 0)	51,000,000
167	prob += (LZMA_LIT_SIZE * (((processedPos &	0
168	(dic[(dicPos == 0 ? dicBufSize : dicPos) -	119,000,000
169		
170	if (state < kNumLitStates)	17,000,000
171	{	
172	state -= (state < 4) ? state : 3;	34,000,000
173	symbol = 1;	17,000,000
174	do { GET_BIT(prob + symbol, symbol) } whil	2,261,000,000
175	}	
176	else	
177	{	
178	unsigned matchByte = p->dic[(dicPos - rep0	17,000,000

Address	Sour…	Assembly	Locators: Bad Speculation: Branch Mispredict	Machir
0x558c48	174	shl edx, 0x8	0.0%	
0x558c4b	174	shl esi, 0x8		
0x558c4e	174	inc rcx		
0x558c51	174	or edx, eax		
0x558c53		**Block 25:**		
0x558c53	174	mov eax, esi	0.0%	
0x558c55	174	add edi, edi	0.0%	
0x558c57	174	shr eax, 0xb	0.0%	
0x558c5a	174	imul eax, r10d	0.0%	
0x558c5e	174	cmp edx, eax	17.1%	
0x558c60	174	jnb 0x558c80 <Block 28>	5.7%	
0x558c62		**Block 26:**		
0x558c62	174	mov esi, 0x800	0.0%	
0x558c67	174	sub esi, r10d	0.0%	

TMAM with toplev/perf

```
$ ~/pmu-tools/toplev.py --core S0-C0 -l1 taskset -c 0 ./a.out
...
S0-C0  Frontend_Bound:   13.81 +- 0.00 % Slots below
S0-C0  Bad_Speculation:   0.22 +- 0.00 % Slots below
S0-C0  Backend_Bound:    53.43 +- 0.00 % Slots          <==
S0-C0  Retiring:         32.53 +- 0.00 % Slots below

$ ~/pmu-tools/toplev.py --core S0-C0 -l2 -v --no-desc taskset -c 0 ./a.out
...

$ perf record -e cpu/event=0xd1,umask=0x20,name=MEM_LOAD_RETIRED.L3_MISS/ppp ./a.out

$ perf report -n --stdio
# Samples: 33K of event 'MEM_LOAD_RETIRED.L3_MISS'
# Overhead       Samples  Command  Shared Object      Symbol
# ........  ............  .......  .................  ...............................
#
    99.95%         33811  a.out    a.out              [.] foo                         <==
```

https://easyperf.net/blog/2019/02/09/Top-Down-performance-analysis-methodology

Agenda

- How to conduct fair experiments
- How to process measurements and compare results
- Application profiling
- How to find the CPU bottleneck
- CPU-specific code tuning examples

Inlining functions with hot prolog/epilog

```
 Percent |      Source code & Disassembly of foo
--------------------------------------------------------------------------
    3.77 :        418be0:       push   r15           <== prolog
    4.62 :        418be2:       mov    r15d,0x64
    2.14 :        418be8:       push   r14
    1.34 :        418bea:       mov    r14,rsi
    3.43 :        418bed:       push   r13
    3.08 :        418bef:       mov    r13,rdi
    1.24 :        418bf2:       push   r12
    1.14 :        418bf4:       mov    r12,rcx
    3.08 :        418bf7:       push   rbp
    3.43 :        418bf8:       mov    rbp,rdx
    1.94 :        418bfb:       push   rbx
    0.50 :        418bfc:       sub    rsp,0x8
    ...
    #                                                <== function body
    ...
    4.17 :        418d43:       add    rsp,0x8       <== epilog
    3.67 :        418d47:       pop    rbx
    0.35 :        418d48:       pop    rbp
    0.94 :        418d49:       pop    r12
    4.72 :        418d4b:       pop    r13
    4.12 :        418d4d:       pop    r14
    0.00 :        418d4f:       pop    r15
    1.59 :        418d51:       ret
```

Prolog + Epilog takes 50% of the time

https://easyperf.net/blog/2019/05/28/Performance-analysis-and-tuning-contest-3#inlining-functions-with-hot-prolog-and-epilog-265

Unrolling/vectorization compiler hints

```
for (int i = 0; i < N; ++i) {
  //...
}
```

$ clang++ -Rpass=. –Rpass-analyses=.
<source>:4:5: remark: **loop not vectorized** …

```
=>
```

```
#pragma clang vectorize(enable)
for (int i = 0; i < N; ++i) {
  //...
}
```

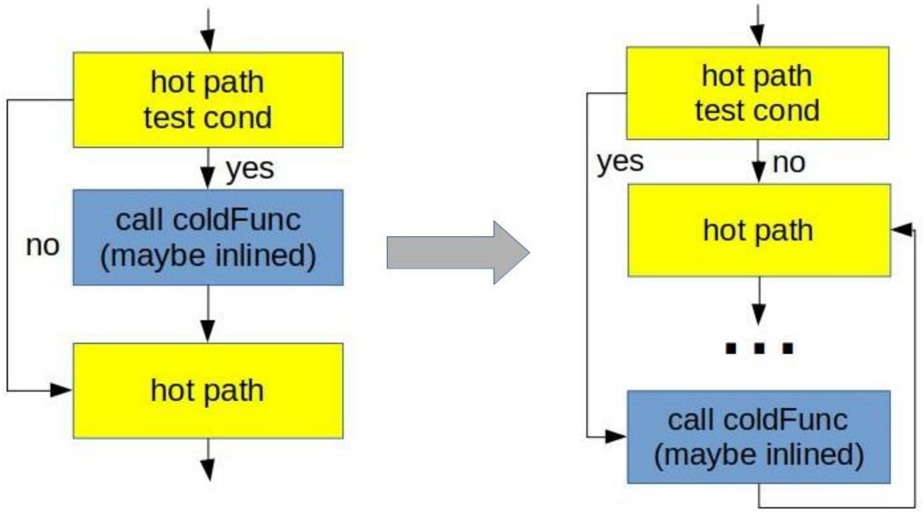
https://easyperf.net/blog/2017/11/09/Multiversioning_by_trip_counts

Inserting likely/unlikely compiler hints

```
// hot path
if (cond)
  coldFunc();
// hot path again

=>

// hot path
if (__builtin_expect(cond, 0)) // unlikely
  coldFunc();
// hot path again
```



https://easyperf.net/blog/2019/05/28/Performance-analysis-and-tuning-contest-3#improving-machine-block-placement-162

Inserting likely/unlikely compiler hints

```
for (;;) {
  switch (instruction) {
    // handle different instructions
  }
}

=>

for (;;) {
  switch (__builtin_expect(instruction, ADD)) {
    // handle different instructions
  }
}
```

https://easyperf.net/blog/2019/11/22/data-driven-tuning-specialize-switch

Branch to CMOV conversion

```
if (cond) { // branch has high misprediction rate
  // compute x1
  a = x1;
} else {
  // compute y1
  a = y1;
}
```

=>

```
// compute x1
// compute y1
a = cond ? x1 : y1;
```

Likely to generate CMOV instructions instead of CMP+JMP

https://easyperf.net/blog/2019/04/10/Performance-analysis-and-tuning-contest-2#fighting-branch-mispredictions-9

Memory prefetching

```
for (int i = 0; i < N; ++i) {
  ...
  x = arr[i]; // misses in L3 cache a lot
}

=>

for (int i = 0; i < N; ++i) {
  __builtin_prefetch(a + i, 0, 1);
  ...
  x = arr[i];
}
```

https://easyperf.net/blog/2019/02/09/Top-Down-performance-analysis-methodology#fixing-the-issue

Takeaways

- Configure environment properly
- Process results carefully
- Know how you can identify the bottleneck and headroom of the application
- Start low-level tuning only when all the major performance problems are fixed
- Know the code tuning recipes for modern CPU

Want More?

easyperf.net:

- Advanced CPU features: LBR, PEBS, Intel PT.
- Analysis of multithreaded apps: expensive locks, false sharing, etc.
- Tuning examples.
- CPU microarchitecture.
- And more…

Be sure to subscribe!

Q&A

my blog: easyperf.net

twitter: @dendibakh