

DEALING WITH PERFORMANCE ANALYSIS IN C/C++

Denis Bakhvalov

embo++, March 2018

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Overview

- How we do performance analysis
- How you can do it
- What you need to be aware of

Performance analysis that we do

Performance analysis of	Code of the benchmark	Code of the compiler
Applications	changing	fixed
Compiler	fixed	changing

Performance analysis that we do



What tools do I use?

- perf, Intel® VTune™ Amplifier
- binutils (objdump, nm, etc.)

Compiler options

You might want to use at least those:

- -O2/O3
- -march=<architecture>

For coremark-pro suite:

'-O3 -march=skylake' is +9% better than '-O2'

Case 1

Execution slows down by 15%.

Benchmark: EEMBC (networking/routelookup)

Performance degraded (1)

```
for (...)  
{  
    small_func(...);  
}
```

before (fast)

```
40:  mov     DWORD PTR [esp+0x4],esi  
      mov     DWORD PTR [esp],ebp  
      → call    <small_func>  
      mov     DWORD PTR [esp+0x14],eax  
      lea     eax,[ebp+0x4]  
      mov     DWORD PTR [esp+0x4],esi  
      mov     DWORD PTR [esp],eax  
      → call    <small_func>  
      add     ebp,0x8  
      add     edi,0xfffffffffe  
      mov     DWORD PTR [esp+0x14],eax  
      jne     40
```

after (slow)

```
40:  mov     DWORD PTR [esp+0x4],esi  
      mov     DWORD PTR [esp],ebp  
      → call    <small_func>  
      add     ebp,0x4  
      dec     edi  
      mov     DWORD PTR [esp+0x38],eax  
      jne     40
```

performance
-15%

Case 2

Execution slows down by 10%.

Benchmark: EEMBC (coremark-pro/loops-all-mid-10k-sp)

Performance degraded (2)

before (fast)

```
5a0: vmovd  xmm0,eax
      vpbroa xmm0,xmm0
      vmovd  xmm1,esi
      vpbroa xmm1,xmm1
      vpaddw xmm0,xmm0,xmm1
      vpaddw xmm0,xmm0,xmm4
      vpunpc xmm1,xmm0,xmm0
      vpmovz ymm1,xmm1

      ...

      vpcmpb xmm1,xmm1,xmm1
      vpgath xmm2,DWORD PTR [ymm0*1+0x0],xmm1
      vinsrb ymm0,ymm2,xmm3,0x1
      vmovdq YMMWORD PTR [rbx+rsi*4],ymm0
      add    rsi,0x8
      cmp    rsi,rdx
      jl     5a0
```

after (slow)

```
830: lea     esi,[rax+rdx*1]
      and     esi,0xffff
      mov     esi,DWORD PTR [rcx+rsi*4]
      mov     DWORD PTR [rbx+rdx*4],esi
      cmp     rdx,rbp
      lea     rdx,[rdx+0x1]
      jl     830
```

performance
-10%

Compiler Opt Reports

```
// a.cpp
void foo(int* a)
{
    for (int i = 0; i < 1024; i++)
    {
        a[i] += 1;
    }
}
```

\$ clang++ a.cpp -O2 -Rpass=.

<source>:4:5: remark: **unrolled loop by a factor of 2** with a breakout at trip 0 [-Rpass=loop-unroll]

for(int i = 0; i < 1024; i++)

^

<source>:4:5: remark: **vectorized loop** (vectorization width: 8, interleaved count: 1) [-Rpass=loop-vectorize]

for(int i = 0; i < 1024; i++)

^

Opt Reports in compiler explorer

The screenshot displays the Compiler Explorer interface with three panels. The left panel shows the C++ source code for a function `foo` that increments an array `a` of 1024 elements. The middle panel shows the assembly output for `x86-64 clang 5.0.0` with the `-O2` optimization level. The right panel shows the optimization reports for the same code.

Source Code (C++):

```
1 void foo(int* a)
2 {
3     for (int i = 0; i < 1024; i++)
4     {
5         a[i] += 1;
6     }
7 }
```

Assembly Output (x86-64 clang 5.0.0):

```
1 _Z3fooP1: # @_Z3fooP1
2 xor eax, eax
3 pcmpeqd xmm0, xmm0
4 .LBB0_1: # =>This Inner Loop Header: Depth=1
5 movdqu xmm1, xmmword ptr [rdi + 4*rax]
6 movdqu xmm2, xmmword ptr [rdi + 4*rax + 16]
7 movdqu xmm3, xmmword ptr [rdi + 4*rax + 32]
8 movdqu xmm4, xmmword ptr [rdi + 4*rax + 48]
9 psubd xmm1, xmm0
10 psubd xmm2, xmm0
11 movdqu xmmword ptr [rdi + 4*rax], xmm1
12 movdqu xmmword ptr [rdi + 4*rax + 16], xmm2
13 psubd xmm3, xmm0
14 psubd xmm4, xmm0
15 movdqu xmmword ptr [rdi + 4*rax + 32], xmm3
16 movdqu xmmword ptr [rdi + 4*rax + 48], xmm4
17 add rax, 16
18 cmp rax, 1024
19 jne .LBB0_1
20 ret
```

Optimization Reports (x86-64 clang 5.0.0 Opt Viewer):

```
1 void foo(int* a)
2 {
3     for (int i = 0; i < 1024; i++)
4     {
5         a[i] += 1;
6     }
7 }
```

Optimization Reports:

- Passed - vectorized loop (vectorization width: 4, interleaved count: 2)
- Passed - unrolled loop by a factor of 2 with a breakout at trip 0

Using optimization reports

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

\$ clang++ -Rpass=. -Rpass-analyses=.
<source>:4:5: remark: **loop not vectorized**
<source>:4:5: remark: the cost-model indicates
that **vectorization is not beneficial**

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

\$ clang++ -Rpass=. -Rpass-analyses=.
<source>:5:5: remark: **vectorized loop...**

Finding optimization opportunities

1. Measure the baseline
2. Identify the hotspots
3. Do analysis of the hotspots:
 - Check optimization reports for the hot places
 - Look at generated assembly of the hotspots
 - Collect hardware events and performance counters

Tricky cases

Code placement

```
// func.cpp
```

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

```
// 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;
}
```

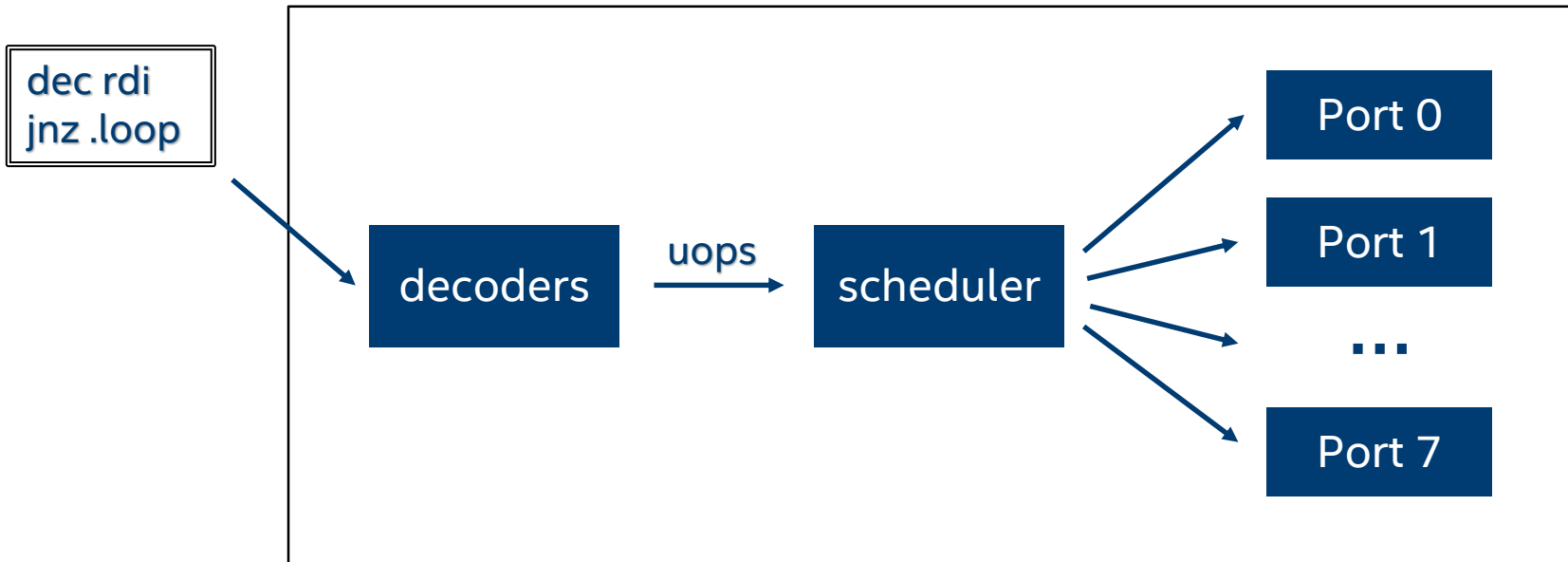
Throughput
+15%

Code placement

```
$ perf stat -e r53019c -- ...
```

	Throughput, GB/s	IDQ_UOPS_NOT_ DELIVERED.CORE
Baseline	28.7	<later>
+ foo()	33.0 (+15%)	<later>

Understanding IDQ_UOPS_NOT_DELIVERED.CORE



Code placement

```
$ perf stat -e r53019c -- ...
```

	Throughput, GB/s	IDQ_UOPS_NOT_ DELIVERED.CORE
Baseline	28.7	$34 * 10^9$
+ foo()	33.0 (+15%)	$30 * 10^9$

Code placement

```
$ perf stat -e r53019c -- ...
```

	Throughput, GB/s	IDQ_UOPS_NOT_ DELIVERED.CORE
Baseline	28.7	$34 * 10^9$
+ foo()	33.0 (+15%)	$30 * 10^9$
32B function alignment *	38.0 (+32%)	$24 * 10^9$
32B basic blocks alignment **	42.4 (+48%)	$17 * 10^9$

* -mllvm -align-all-functions=5

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

Code placement

Function aligned at 32B boundary

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80																
90																
a0	mov	mov	mov	mov	mov	mov	mov	vpcm	vpcm	vpcm	vpcm	nop	nop	nop	nop	nop
b0	vmov	vmov	vmov	vmov	vmov	vmov	vmov	vmov	vmov	vpsub	vpsub	vpsub	vpsub	vmov	vmov	vmov
c0	vmov	vmov	vmov	vmov	vmov	vmov	add	add	add	add	jne	jne	vzero	vzero	vzero	ret
d0																
a0																
b0																

} cache line

Loop aligned at 32B boundary

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80																
90																
a0	mov	mov	mov	mov	mov	mov	mov	vpcmp	vpcmp	vpcmp	vpcmp	nop	nop	nop	nop	nop
b0	nop	nop	nop	nop	nop	nop	nop	nop	nop	nop	nop	nop	nop	nop	nop	nop
c0	vmov	vmov	vmov	vmov	vmov	vmov	vmov	vmov	vmov	vpsub	vpsub	vpsub	vpsub	vmov	vmov	vmov
d0	vmov	vmov	vmov	vmov	vmov	vmov	add	add	add	add	jne	jne	nop	nop	nop	nop
e0	vzero	vzero	vzero	ret												
f0																

Code placement

Additional resources

- My blog post: “Code alignment issues”
(https://dendibakh.github.io/blog/2018/01/18/Code_alignment_issues)
- 2016 LLVM Developers' Meeting: Z. Ansari "[Causes of Performance Instability due to Code Placement in IA](#)“
- Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.: [Producing Wrong Data Without Doing Anything Obviously Wrong!](#) In: Proc. of Int'l Conf. on Architectural Support for Programming Languages and Operating System, March, 2009, pp. 265-276. ACM, Washington (2009)

One more tricky case

Instruction fusion

before (fused)

```
inc DWORD [<memory address>]
```

after (unfused)

```
mov edx, DWORD [<memory address>]  
inc edx  
mov DWORD [<memory address>], edx
```

~5%
performance gap

Instruction fusion

before (fused)

```
inc DWORD [<memory address>]
```

after (unfused)

```
mov edx, DWORD [<memory address>]  
inc edx  
mov DWORD [<memory address>], edx
```

	INSTRUCTIONS_ RETIRED	UOPS_RETIRED. ALL	CYCLES/ ITERATION
Fused	1	3	1
Unfused	3	3	1

Instruction fusion

before (fused)

```
inc DWORD [<memory address>]
```

after (unfused)

```
mov edx, DWORD [<memory address>]  
inc edx  
mov DWORD [<memory address>], edx
```

performance on par
(in equal conditions)

Know your hardware!

You can find me

- My blog: dendibakh.github.io
- Twitter: @dendibakh

Q&A