DEALING WITH PERFORMANCE ANALYSIS IN C/C++

Denis Bakhvalov

embo++, March 2018

Legal Disclaimer & Optimization Notice

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,0xfffffffe
    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
     │
     │  ...
     │
     │ vpcmpe  xmm1,xmm1,xmm1
     │ vpgath  xmm2,DWORD PTR [ymm0*1+0x0],xmm1
     │ vinser  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,0xfff
     │ 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%

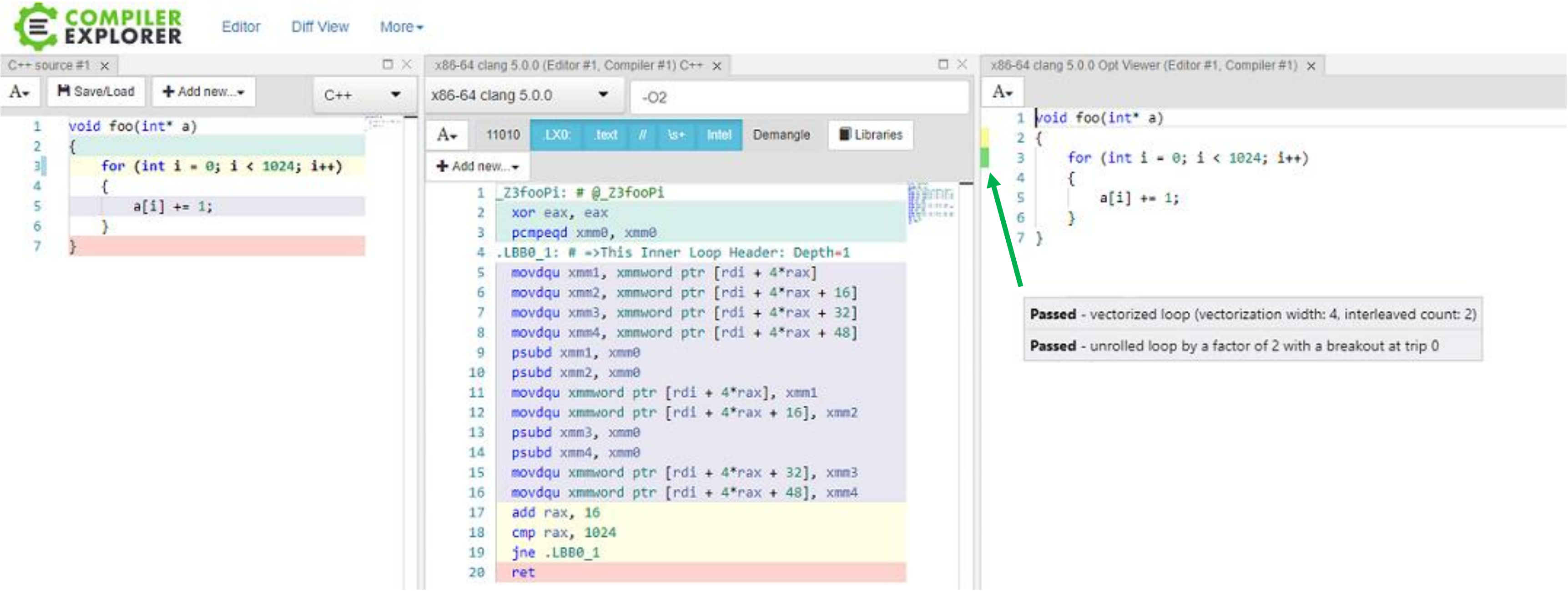
Compiler Opt Reports

```
// 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



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

```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;
}
```

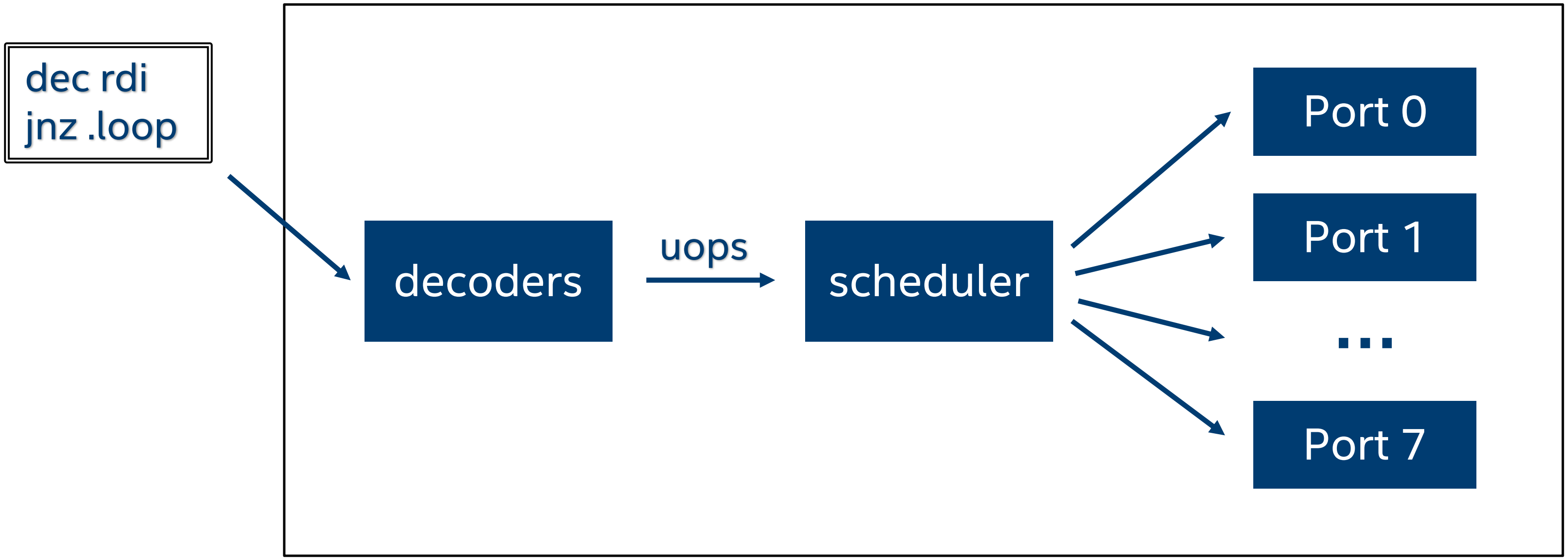
Throughput
+15%

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	vpcmp	vpcmp	vpcmp	vpcmp	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