Static Performance Analysis with LLVM

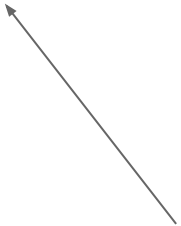
Clément Courbet
G. Chatelet, B. De Backer, O. Sykora,

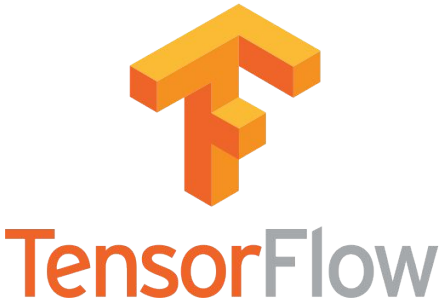
Google Compiler Research


TensorFlow

Low-precision matrix multiplication

github.com/google/**gemmlowp**





Optimize this at (nearly) all costs !

Tools

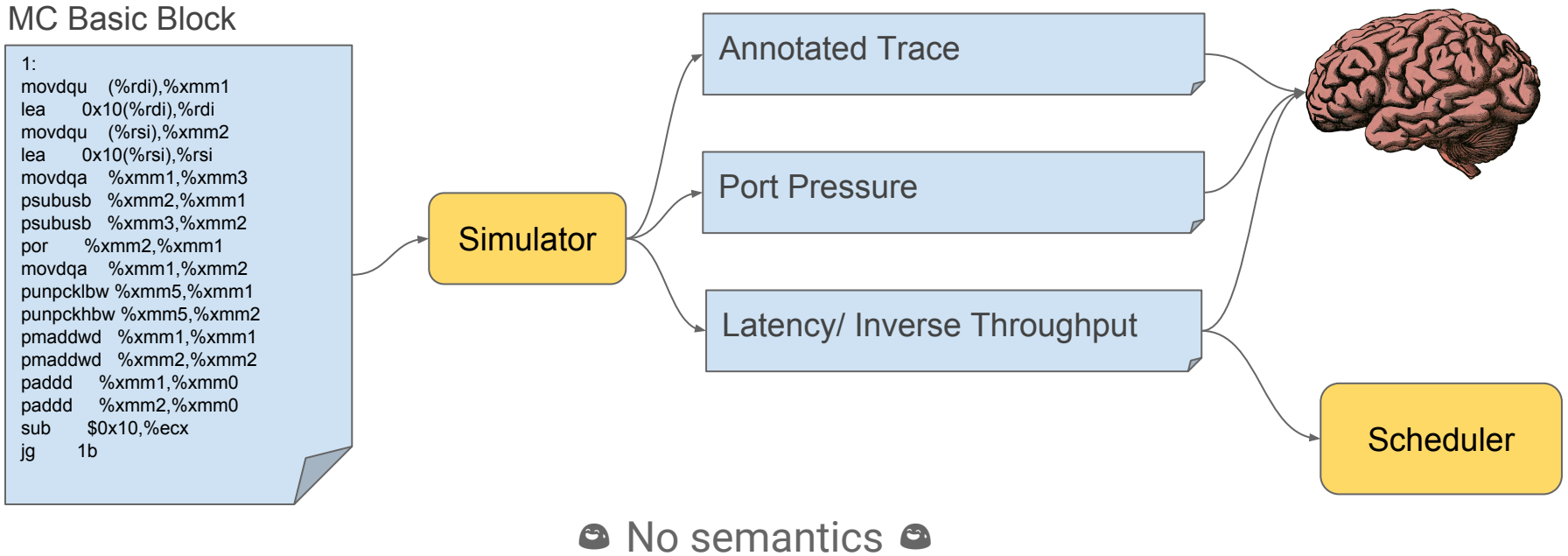
Benchmarks

- Closer to real-life performance
- Slooooow
- Requires access to hardware

Static Analysis

- Fast
- Reproducible
- Hard to model input-dependent behaviour (branches, cache)

Our Static Performance Analyzer



No semantics

Simulator API

Target-independent `Simulator` interface:

```
const vector<MCInst>& BasicBlock = ...;
auto Simulator = Target->createSimulator();
SimulationLog Log = Simulator.Run(BasicBlock);
```

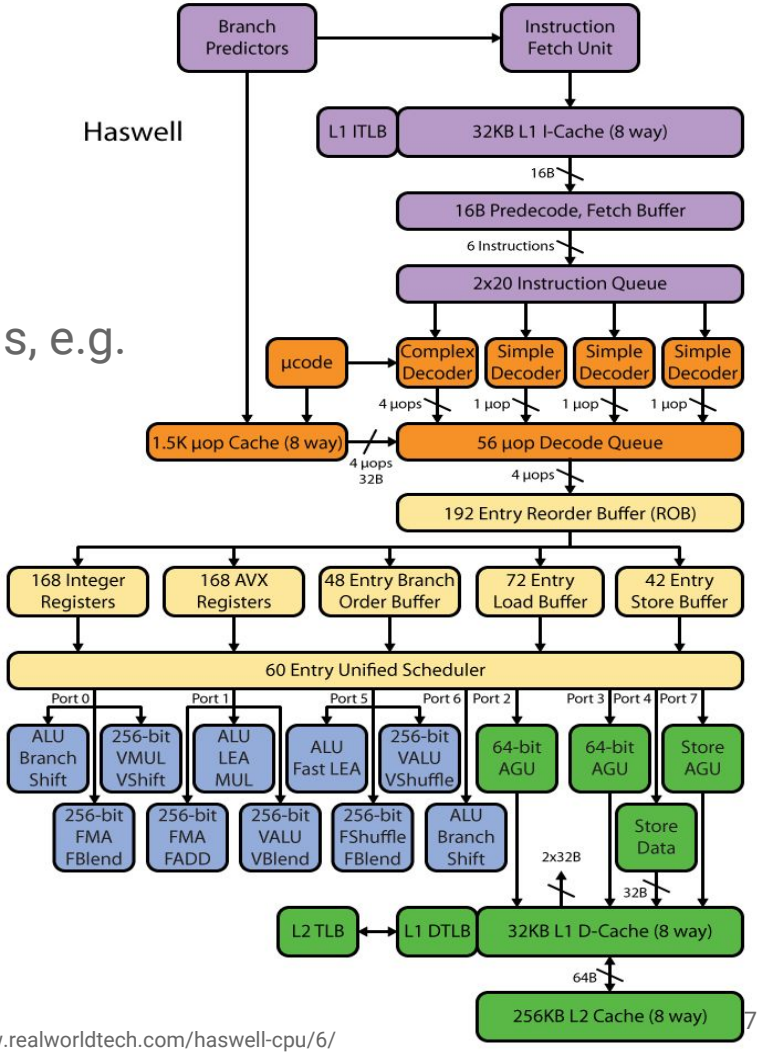
Simulator Internals: Components

- Generic →
  Reuse LLVM's target-independent descriptions, e.g.

  Scheduler: `llvm::MCSchedModel`

  RegisterRenamer: `llvm::MCRegisterInfo`

- Target-specific, e.g. Intel Fetcher



Source:
http://www.realworldtech.com/haswell-cpu/6/

Analysis: IACA-like frontend

```
analyzing 'libyuv_sumsquareerrorsse2.s'
ran 20 iterations in 132 cycles
Block Inverse Throughput: [5-6] cycles per iteration

Port Pressure (cycles per iteration):
```

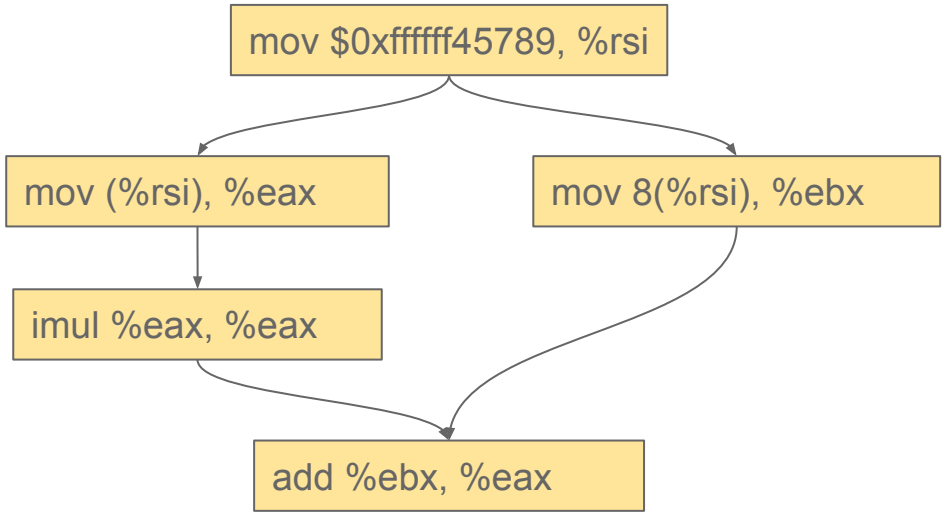
Port	HWDivider	HWPort0	HWPort1	HWPort2	HWPort3	HWPort4	HWPort5	HWPort6	HWPort7
Cycles		4.35	4.40	1.00	1.00		4.30	1.95	

#Uops	HWDivider	HWPort0	HWPort1	HWPort2	HWPort3	HWPort4	HWPort5	HWPort6	HWPort7	
1				1.00						movdqu xmm1, xmmword ptr [rdi]
1			0.30				0.70			lea rdi, [rdi + 0x10]
1					1.00					movdqu xmm2, xmmword ptr [rsi]
1			0.55				0.45			lea rsi, [rsi + 0x10]
1		0.35	0.65							movdqa xmm3, xmm1
1			0.70				0.30			psubusb xmm1, xmm2
1			0.40				0.60			psubusb xmm2, xmm3
1		0.95	0.05							por xmm1, xmm2
1		1.00								movdqa xmm2, xmm1
1							1.00			punpcklbw xmm1, xmm5
1							1.00			punpckhbw xmm2, xmm5
1		1.00								pmaddwd xmm1, xmm1
1		1.00								pmaddwd xmm2, xmm2
1			0.75				0.25			paddd xmm0, xmm1
1			1.00							paddd xmm0, xmm2
1		0.05						0.95		sub ecx, 0x10
1								1.00		jg .Ltmp0

Automatic Scheduling

Minimize the simulated latency (alternative to `PostRAMachineSched`)

- Random/Exhaustive Search (CP Solver) ~hours-days
- Genetic Algorithms (Biased Random-Key Genetic Algorithms) <seconds



Exhaustive Search

- gemmlowp's `SSE4_32_Kernel4x4Depth2`:
  0-2% faster (vs. implementation contributed by Intel).

- libwebp's `FTransform()`:
  0-5% faster.

On benchmarks; no performance regressions

Genetic Algorithms

- 100 milliseconds
- 10% improvement (in theory)

Original: 9.5 Cycles/Iter

```
movd  mm0, dword ptr [r8]
punpcklbw  mm0, mm7
movq  mm1, mm0
movq  mm2, mm0
pmullw     mm1, mm1
paddw mm1, mm3
psrlw mm1, 0x8
pmullw     mm0, mm1
paddw mm0, mm3
psrlw mm0, 0x8
pmullw     mm0, mm4
pmullw     mm1, mm5
pmullw     mm2, mm6
paddw mm0, mm1
paddw mm0, mm2
psrlw mm0, 0x6
packuswb   mm0, mm7
movd  dword ptr [r8], mm0
add   r8, 0x4
dec   ecx
jne   .Ltmp0
```

Rescheduled: 8.5 Cycles/Iter

```
movd  mm0, dword ptr [r8]
punpcklbw  mm0, mm7
movq  mm1, mm0
pmullw     mm1, mm1
movq  mm2, mm0
pmullw     mm2, mm6
paddw mm1, mm3
psrlw mm1, 0x8
pmullw     mm0, mm1
pmullw     mm1, mm5
paddw mm0, mm3
psrlw mm0, 0x8
pmullw     mm0, mm4
paddw mm0, mm1
paddw mm0, mm2
psrlw mm0, 0x6
packuswb   mm0, mm7
movd  dword ptr [r8], mm0
add   r8, 0x4
dec   ecx
jne   .Ltmp0
```

Future Work

- Integrating into llvm-mca (in particular frontend simulation)
- Genetic scheduler → `MachineFunctionPass`

Try It Out!

https://github.com/google/EXEgesis/tree/master/llvm_sim