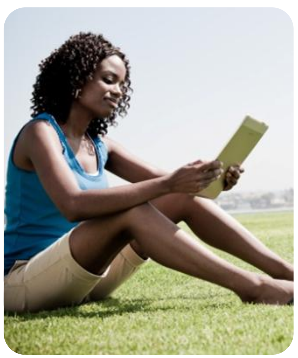
Open *Source* | Open *Possibilities*



Porting LLVM to a Next Generation DSP

Presented by: L . Taylor Simpson

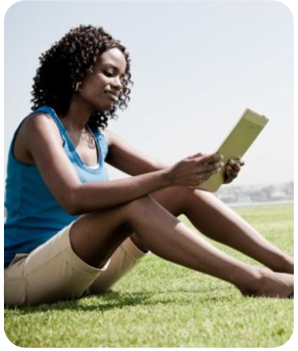
LLVM Developers' Meeting: 11/18/2011

QuIC
QUALCOMM INNOVATION CENTER, INC.

Agenda

- Hexagon DSP
- Initial porting
- Performance improvement
- Future plans

Hexagon DSP



QuIC QUALCOMM INNOVATION CENTER, INC.

Hexagon – Typical DSP Features

- Wide computation engine
  - 8-MAC design, dual 64-bit loads or stores
  - Performance meets or exceeds highest-performance industry DSPs

- Native numerical support
  - Fractionals, complex
  - Saturation, scaling, rounding

- Exploits parallelism at 3 levels
  - Unique multi-threaded architecture
  - VLIW (up to 4 instructions in parallel)
  - SIMD

Hexagon – Typical CPU features

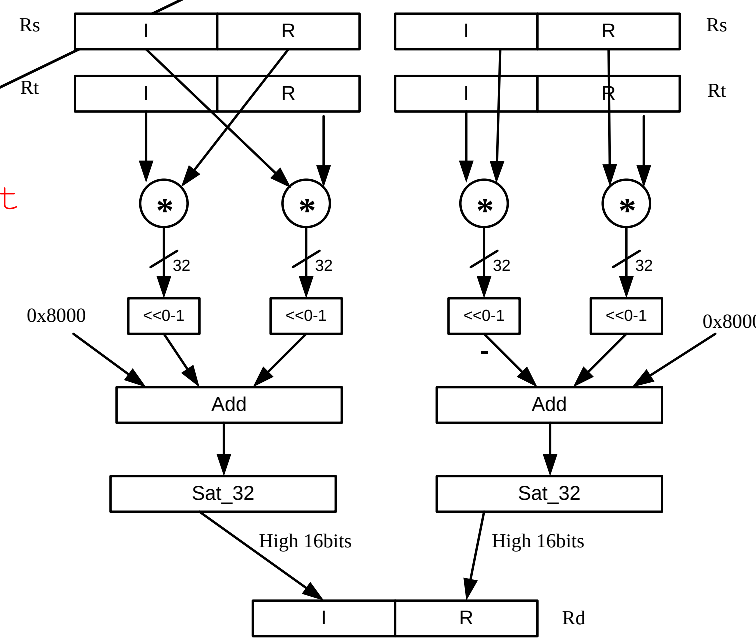
- Not your grandfather's DSP!
  - Capable of supporting RTOS or high-level OS
  - Can run all of SPEC on target
- Supports C/C++ modern programming environment
  - High-quality compilers and tools
  - Reduces development cost of extensive assembly programming
- Cache-based, hardware-managed memory
  - Simplifies programming model and reduces power
- Advanced system architecture
  - Precise exceptions
  - MMU with address translation and protection
  - HW support for virtual machines
- Excellent control code performance
  - Can offload work from main CPU

Hexagon Instruction Example

- Single packet from inner loop of FFT
- Performs 29 “RISC ops” in 1 cycle
- All threads can all be doing this (or something else) in parallel

64-bit Load and
64-bit Store with post-update addressing

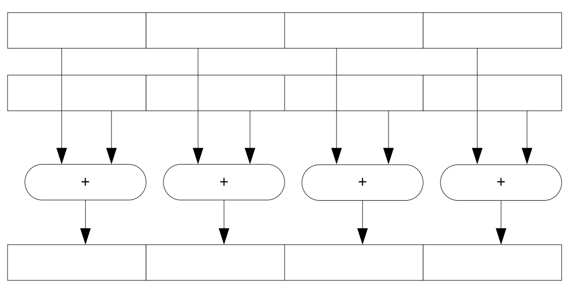
Complex Multiply



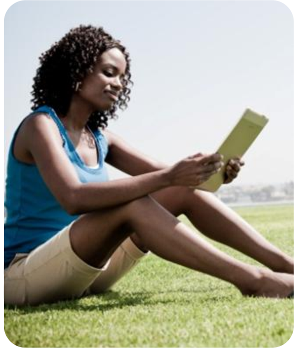

```
{ R17:16 = MEMD(R0++M1)
  MEMD(R6++M1) = R25:24
  R20 = CMPY(R20, R8):<<1:rnd:sat
  R11:10 = VADDH(R11:10, R13:12)
}:endloop0
```

HW-loop end
- Dec count
- Compare
- Jump top

Vector 4x16-bit Add



Initial Porting



QuIC QUALCOMM INNOVATION CENTER, INC.

LLVM for Hexagon – Initial Porting Effort

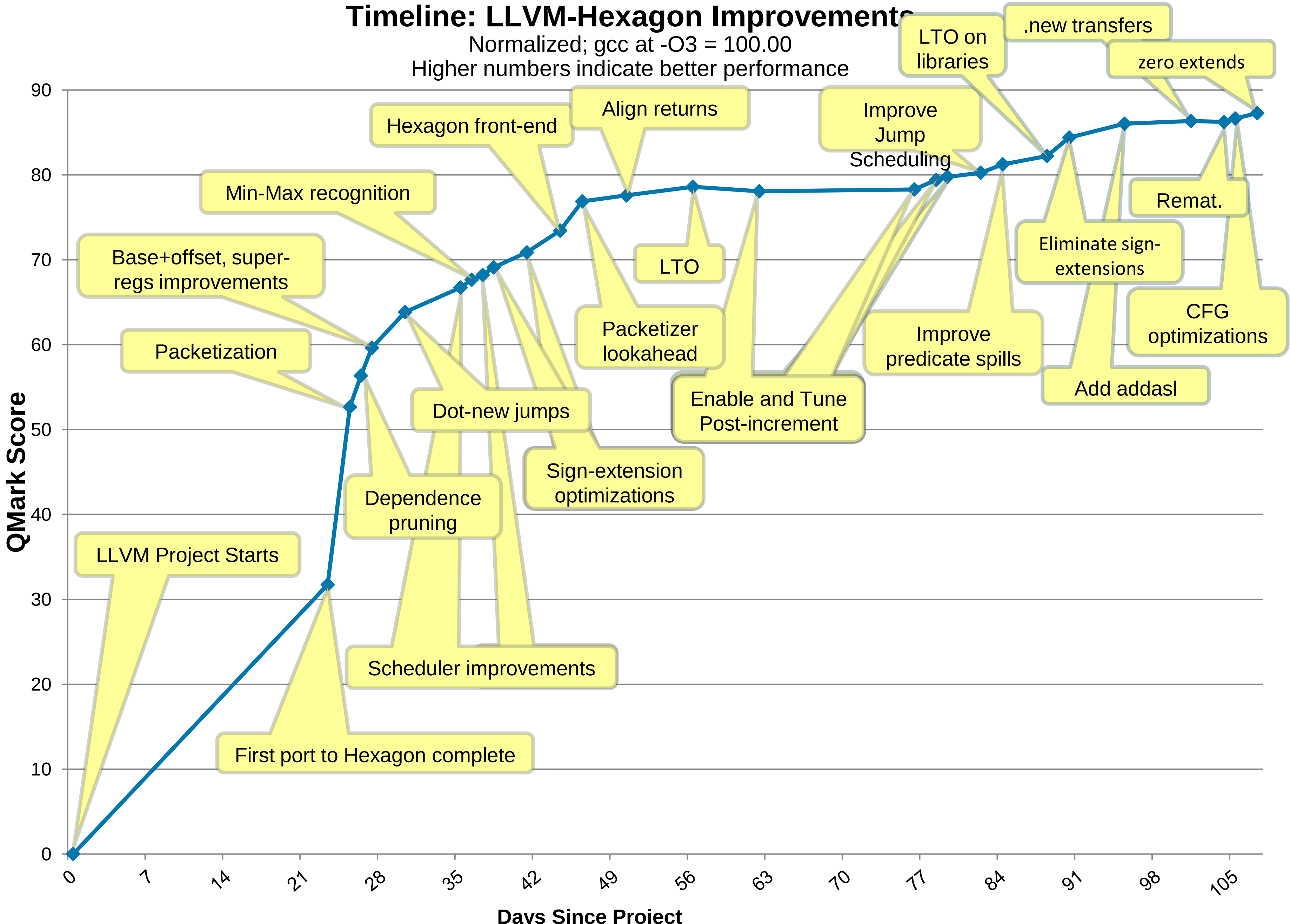
- It took 2 engineers 23 days to get Hexagon back end working
  - Passing DSP benchmark suite
- It took 107 calendar days to get to 87% performance of GCC
- Leveraged existing assembler, linker, test suite

- Points of efficacy for LLVM
  - Robust and easy to port
  - Very well designed and documented
  - Carefully engineered for compiler construction
  - Excellent infrastructure for writing mid-level compiler optimizations

Timeline: LLVM-Hexagon Improvements

Normalized; gcc at -O3 = 100.00

Higher numbers indicate better performance



Open *Source* | Open *Possibilities*

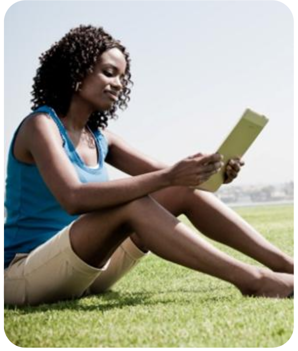
HEXAGON QuIC QUALCOMM INNOVATION CENTER, INC.

Transition Time

- Simultaneously to LLVM work, GCC moved forward
  - New version of GCC for Hexagon released
  - Version 4 of Hexagon core released with significant support in GCC
  - LLVM only 72% performance of GCC
- Quickly improved pass rate to 98%
  - Leverage existing compiler test suite
  - Initial pass rate for –O0: 49%
  - Initial pass rate for –O3: 63%
  - Most of the remaining issues are corner cases in C++ front end
- Current status
  - LLVM achieves 89% performance of GCC for Hexagon

Open *Source* | Open *Possibilities*

Performance Improvement



Performance Improvement – Instruction Scheduling

- Optimal performance for VLIW requires precise scheduling
- Hexagon packetizer
  - Originally a post-pass to form packets from scheduled code
- Alias information in scheduler
- Use machine resource constraints during scheduling

Performance Improvement – Loop Unroller

- Enable loops with runtime trip counts
- We have seen both large improvements and losses
  - We will likely need some target-specific information
- Patch currently under review

Performance Improvement - Miscellaneous

- Hardware loop support
- Post-increment
- Loop strength reduction
  - Addressing modes: base+offset, post-increment, base+index
- New version of core released
  - Numerous new instruction combinations
  - More relaxed packet forming rules
  - Enhanced predication support

What is a hardware loop

- Execute loops with zero overhead
- Hexagon has two special instructions
- Hexagon sets up two registers
  - Loop start address, SA0/SA1
  - Loop count, LC0/LC1

Here's a loop

```
for (i =0; i < n; i++) {
  a += b[i];
}
```

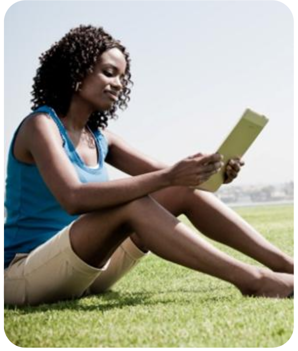
The generated code

```
.L1: {
       r3 = memw(r1++#4)
       r0 = add(r0, #-1)
      }
      {
        p0 = cmp.eq(r0, #0)
        r2 = add(r3, r2)
        if (!p0.new) jump:t .L1
      }
```

With hardware loop

```
        loop0(.L1, r0)
.L1: {
        r3 = memw(r1++#4)
      }
      {
        r2 = add(r3, r2)
      }:endloop0
```

Next Steps

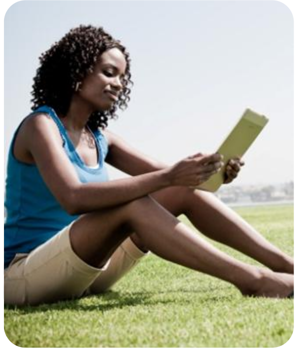


QUALCOMM INNOVATION CENTER, INC.

Next Steps

- Upstreaming our changes
- Code size reduction
- Represent VLIW packets in back end
- Multi-basic-block scheduling
- Enable loop unrolling for loops with multiple exits
- Improve alias analysis
  - Very important for VLIW scheduling
  - Have seen issues with type-based disambiguation
- Expose machine-dependent information to optimizer
  - Which addressing modes are supported?
  - Which loop unrolling factor is best for target?
- Software pipelining

Questions?



QuIC QUALCOMM INNOVATION CENTER, INC.