arm

Performance Improvements in Open Source Toolchains for Arm

Linaro Connect San Diego 2019

James Greenhalgh

2019-09-26

Compiler Performance Team

- Who are we?
  - C/C++ Compiler and system library (glibc, etc.) optimization experts
  - Experienced GCC and LLVM engineers
  - Arm microarchitecture experts
  - Performance Analysts
- What are we good at?
  - Improving benchmark performance
    - Spec for servers
    - Embedded benchmarks for microcontrollers
  - Improving library performance
    - glibc memory, string, and math routine optimization
  - Working with high-level compiler transforms
    - Loop distribution in GCC
  - Working in open source communities
    - Arm, AArch64 maintainers and code owners in compiler communities

Why might you be interested in our work?

- Our goal is to make software on Arm fast
- Primarily we do this through improving benchmark code
- We want to remove duplication in the ecosystem on performance work with impact on GCC and LLVM

Three strands

- Open Source GCC for Linux for Infrastructure
  - Boost the Spec2017 Integer score
- Open Source LLVM for Embedded devices
  - Enable the M-Profile vector extensions
  - Support M-Class microarchitectures
- Open Source LLVM for Client devices
  - Optimize intrinsic implementations for performance critical code

Spec Benchmarks and Me

- Spec2000
  - As we brought up AArch64 on prototypes and models
  - Many small improvements, broadly applicable
- Spec2006 as a driver for math libraries and loop transforms
  - Huge improvements to glibc
  - Double-Digit overall improvements to benchmark score
- Spec2017
  - Heroic optimizations at rate=N

Spec Benchmarks and the wider world

- We’re behind other architectures on compiler optimization technology for Spec
  - Better memory utilization
  - Better vectorization
- How can we get to the point where we compete microarchitecture to microarchitecture
  - Neoverse N1 is great!
  - How can we ensure that software on Neoverse N1 is great too!
- We’re going to try to do this in GCC
  - The most important Open Source toolchain on Linux
- We’d love to do this with your support

Optimized Routines

- **Attend SAN19-511 (Joey Ye)**

  Arm and partners have been contributing extensively to library functions projects. It will be in the best interests of Arm software eco-system to maximize the value of those contributions. An amplifier will be reusing the open-source implementations in multiple projects. An aggregator is for anyone to collaborate the contribution by submitting new functions or enhancing/fixing existing functions.

  However, reuse and collaboration will not just happen without appropriate license and copyright model. There are legal risks like license compatibility and copyright assignment that have to be addressed properly.

  Optimized Routines is an open source project that serves as the core of a solution to clear obstacles of reusing and collaboration. It uses a liberal license, the MIT license, and is the copyright of a single organization. This method of releasing software under a more permissive license in the first instance prevents the software being locked into restrictive licenses and allows much greater freedom for the Arm eco-system.

  This presentation will the issues in reusing and collaborating of open source libraries, show how the Optimized Routines addressed them and the progress of collabrating contributions into this project.

Spec So Far

- Idiom recognition
  - Popcnt
  - FMIN/FMAX
- High level loop transforms
  - Loop interchange
  - Loop distribution
- Vectorization improvements
  - Sum of absolute differences
  - Vectorized average

Where next for Spec?

- Three major workloads for our team
  - 505.mcf – Memory layout optimizations
  - 525.x264 – More aggressive auto-vectorization
  - 548.exchange2 – Recursive inlining and Fortran library optimization
- More details available on the GCC wiki
- Why is this different?
  - These optimizations are closer to “heroic”
  - They require a lot of compiler effort for a benefit to limited codebases

Why do we care about LLVM performance?

- LLVM is a foundational technology for Arm Compiler
- LLVM is the base for the toolchain for Android
- LLVM is increasingly used in embedded toolchains

Arm M-Profile Vector Extension (MVE)

- Bringing vector instructions to M-Profile
- Performance opportunities for Machine Learning, DSP
- 16-bit floating point support
- Autovectorization and Intrinsics
- Gather load and scatter store support enables more vectorization
- Tail-loop predication
- Low-overhead loop instructions

Low-Overhead Loops

```
WLS   LR, R2, loopEnd
loopStart:
      LDRB R3, [R1], #1
      STRB R3, [R0], #1
      LE   LR, loopStart
loopEnd:
```

Why was this cool?

- Worked with the LLVM community
- Brought together other architectures with a similar feature
- Generic framework for low-overhead loops/hardware loops

What else do we care about in LLVM?

- Real world mobile applications
  - What can we do for Chromium?
- Embedded performance and code size
  - Upstream-first strategy
  - Arm Compiler gets early access for new architecture/microarchitecture

What would I like to happen next?

- If you like what we're doing, please support us in the Open Source communities
- If you'd like to help, please email me!
- If you'd like information on the awesome work Linaro is doing to optimize SVE in GCC and code size for LLVM, please email Maxim and talk to the TCWG

- I want to see hardware and software for the Arm ecosystem go fast!
  - I'd love to see us collaborate to make that happen!

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה