How to Optimize for Arm and not get Eaten by a Bear

Performance Optimization in a World of Multiple Microarchitectures

john.Linford@arm.com
10 Feb 2022
Arm Enables Diversity

- NVIDIA: Grace
- Ampere
- AWS
- Fujitsu
- Huawei
- Marvell / Cavium: Graviton3, Graviton2, Kunpeng 920, ThunderX2
- SiPearl: Altra, Rhea
- ETRI: K-AB21
How to hunt the bear and not get eaten?

How to “optimize for Arm” without becoming tied to a specific chip?
Core Instruction Set Architecture (ISA)
A CPU’s vocabulary

- Armv8.0-A
- Armv8.1-A
- Armv8.2-A
- Armv8.2 SVE
- Armv8.3-A
- Armv8.3 Complex
- Armv8.4-A
- Armv8.5 Security
- Armv8.5 +
- Armv8.6 +
- Armv8.3 LDAPR
- Armv8.4 Dot Product
- Armv8.5 Security
ISA vs. u-arch

The base ISA

How the ISA is implemented

Theoretical Armv8.4-a chip with ThunderX u-arch

Arm Neoverse V1
Fujitsu A64FX
Arm Neoverse N1
Qualcomm Falkor
Marvell ThunderX2
Marvell ThunderX
AppliedMicro X-Gene
Arm Cortex-A72

Architecture (-march)

v8.0 v8.1 v8.2 v8.3 v8.4

v8.1 v8.0

v8.1 v8.2

v8.2

A64FX
How to specify ISA and u-arch?

GCC and LLVM (and LLVM-based compilers)

- **-march**
  - For aarch64 targets, this flag specifies the ISA
  - Fine-grained control over ISA extensions: “armv8.2-a+sve”
  - This flag behaves differently for x86 targets!

- **-mtune**
  - For aarch64 targets, this flag specifies the u-arch
  - See man pages for supported u-arches
  - This flag behaves differently for x86 targets!

- **-mcpu**
  - For aarch64 targets, this flag is a shortcut. It specifies both the ISA and the u-arch
  - Accepts the same parameters as –mtune
  - This flag is deprecated for x86 targets!
Execution vs. Optimization: \(-\text{march}=\text{armv8.1}\)

- **Arm Neoverse V1**: v8.1
- **Fujitsu A64FX**: v8.2
- **Arm Neoverse N1**: v8.2
- **Qualcomm Falkor**: v8.1
- **Marvell ThunderX2**: v8.1
- **Marvell ThunderX**: v8.0
- **AppliedMicro X-Gene**: v8.0
- **Arm Cortex-A72**: v8.0

**Execution**
- Binary will execute.

**Limited Optimization**
- No optimization attempted
- Optimization is limited because u-arch unknown
- Binary may not execute: *invalid instruction*
Execution vs. Optimization: \(-\text{mtune=\text{a64fx}}\)

- Highly portable binary
- Limited optimization: ISA is limited to v8.0
Execution vs. Optimization: \(-mcpu=a64fx\)

- Limited portability
  - Optimized only for A64FX

- Arm Neoverse V1
- Fujitsu A64FX
- Arm Neoverse N1
- Qualcomm Falkor
- Marvell ThunderX2
- Marvell ThunderX
- AppliedMicro X-Gene
- Arm Cortex-A72

Architecture (-march):

- v8.0
- v8.1
- v8.2
- v8.3
- v8.4
A more realistic view of $-\text{mcpu}=\text{a64fx}$

- Binary is expected to only be used on A64FX.
- Compiler is free to use extensions, take shortcuts, etc.

∀ microarchitecture and architecture

- Execution
- Optimization

- Arm Neoverse V1
- Fujitsu A64FX
- Arm Neoverse N1
- Qualcomm Falkor
- Marvell ThunderX2
- Marvell ThunderX
- AppliedMicro X-Gene
- Arm Cortex-A72

- v8.0
- v8.1
- v8.2
- v8.3
- v8.4

© 2021 Arm
__sync_fetch_and_add(&var, num);

GCC 11.1.0 on A64FX

-march=armv8.2-a

```assembly
.arch armv8.2-a+crc
.file  "foo.c"
.text
 .section .rodata
 .align 3

mov x29, sp
str w0, [sp, 28]
str x1, [sp, 16]
ldr w0, [sp, 28]
str w0, [sp, 44]

mov w0, 1
str w0, [sp, 40]
ldr w0, [sp, 40]
move w1, w0
add x0, sp, 44
ldaddal w1, w0, [x0]
ldr w0, [sp, 44]

mov w1, w0
adrp x0, .LC0
add x0, x0, :lo12:.LC0
bl printf
mov w0, 0
ldp x29, x30, [sp], 48
```

No SVE

-atomic Add

libgcc call

-mtune=a64fx

```assembly
.arch armv8-a
.file  "foo.c"
.text
.global __aarch64_ladd4_acq_acq_rel
.text
 .section .rodata
 .align 3

mov x29, sp
str w0, [sp, 28]
str x1, [sp, 16]
ldr w0, [sp, 28]
str w0, [sp, 44]

mov w0, 1
str w0, [sp, 40]
ldr w0, [sp, 40]
move w2, w0
add x0, sp, 44
mov x1, x0
mov w0, w2
bl __aarch64_ladd4_acq_acq_rel
ldr w0, [sp, 44]
move w1, w0
adrp x0, .LC0
add x0, x0, :lo12:.LC0
bl printf
mov w0, 0
ldp x29, x30, [sp], 48
```

Minimal ISA

-atomic Add

-- Best ISA

-mcpu=a64fx

```assembly
.arch armv8.2-a+crc+sve
.file  "foo.c"
.text
 .section .rodata
 .align 3

mov x29, sp
str w0, [sp, 28]
str x1, [sp, 16]
ldr w0, [sp, 28]
str w0, [sp, 44]

mov w0, 1
str w0, [sp, 40]
ldr w0, [sp, 40]
move w2, w0
add x0, sp, 44
mov x1, x0
mov w0, w2
bl __aarch64_ladd4_acq_acq_rel
ldr w0, [sp, 44]
move w1, w0
adrp x0, .LC0
add x0, x0, :lo12:.LC0
bl printf
mov w0, 0
ldp x29, x30, [sp], 48
```

Atomic Add

libgcc call

© 2021 Arm
Compiler flags for tuned portable binaries
https://gcc.gnu.org/onlinedocs/gcc/AArch64-Options.html#aarch64-feature-modifiers

- V1-optimized, runs on A64FX
  - \texttt{-march=armv8.2-a+sve}
  - \texttt{-mtune=neoverse-v1}
  - \texttt{-msve-vector-bits=\texttt{scalable}}
  - Targets V1 u-arch and limits the ISA to A64FX
  - Uses both SVE and NEON, and will occasionally prefer NEON over SVE

- V1-optimized, runs on N1
  - \texttt{-march=armv8.2-a+nosve+dotprod}
  - \texttt{-mtune=neoverse-v1}
  - Targets V1 u-arch and limits the ISA to N1
  - Uses only NEON (which performs well on V1)

- N1-optimized, runs on V1
  - \texttt{-mcpu=neoverse-n1}
  - V1’s features are a superset of N1’s
How to hunt the bear and not get eaten
a.k.a how to get good performance without being tied to a particular chip

• Let someone else hunt the bear
  • NVIDIA NGC
  • wiki.arm-hpc.org

• Only hunt the bear where it is safe
  • Link against portable optimized libraries
  • Use autovectorizing compilers – don’t hand-tune SIMD code

• If you must hunt the bear, stay outside the cave
  • Compile for a common base architecture
  • If extensions are critical to your code’s performance, understand that cost

• If you enter the cave, be sure you can run out
  • Build from source with the appropriate flags (e.g. Spack or EasyBuild)
  • Distribute multiple binaries and dynamically load as appropriate
Thank You
Danke
Gracias
谢谢
ありがとうございます
Asante
Merci
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה