Effective GCC/Clang optimizations for Embedded systems

Khem Raj

@himvis
Agenda

• Introduction
• Tools
• Compiler Optimization Switches
• Data types
• Variables and Functions
• Optimization Tips
• Summary
Tools

- Know your compiler toolchain
  - GCC based, Clang bases, Other Vendors ..
- Read through compiler has to offer
  - Each one has few difference that could matter
Tools

• Understand the memory layout
  – Explicit Linker scripts
    • Common in bare-metal applications
  – Default linker scripts
    • Commonly used in hosted applications
Tools

- Linker map files (-Wl,-Map,mymap.map)
- Objdump – Disassemble Objects
- Size – Elf size dumper
- Readelf – Display Content of ELF files
- Nm – ELF symbol lister
- Strip – Remove Symbols and debug info
- More …
Optimization Options

- **O<n> Switches**
  - **O0**
    - No optimizations
  - **O/O1**
    - Somewhere between -O0 and -O2
  - **O2**
    - Moderate level of optimization which enables most optimizations
  - **Os**
    - Like -O2 with extra optimizations to reduce code size
  - **Og**
    - Like -O1, better debuggability
  - **Oz**
    - Like -Os (and thus -O2), but reduces code size further.
  - **O3**
    - Like -O2, except that it enables optimizations that take longer to perform or that may generate larger code (in an attempt to make the program run faster)
  - **Ofast**
    - Like O3 with more aggressive optimization (may violate standards compliance)
Optimization Options

- GCC

- Clang doesn’t have such a page
Optimization Options

• Security
  – -fstack-protector-strong
  – -D_FORTIFY_SOURCE=2
  – -Wformat -Wformat-security
  – -Werror=format-security
Compiler Optimizations

Does Compiler support –O4 ?
Data Types

• Know processor and word-size
  – Use data-types representable in processor word size
  – Smaller datatypes
    • Code size increase
  – Larger datatypes
    • Might degrade performance
  – Also depends on processing architecture
    • X86 might work fine, but ARM may exhibit above or vice-versa
Data Types

- Delegate to compiler
  - C99 provides
    - Fixed width – uint*_t
    - Minimum width – uint_least*_t
    - Fastest width – uint_fast*_t
  - Portable datatypes
    - uint<size>_t
Variables and Functions

• Using “const”
  – is a big hint to compiler
  – Immutable data
  – Documentation
  – Better diagnostics from compiler
  – Better optimization opportunity
Variables and Functions

- Function Parameter
  - Know the ABI and calling convention
    • Depends on processor architecture
  - ARM
    • 4 registers available for parameter passing
    • -mfloat-abi
      - Hard
      - Soft
      - Softvfp
  - Know alignment
    • Set parameters sequence such that no padding is needed
Variables and Functions

- Avoid global and static data in loops
- Use volatile when really needed
- Avoid function calls in loops
Variables and functions

• Compiler attributes
  – Clang
    – https://clang.llvm.org/docs/AttributeReference.html
  – GCC
    • https://gcc.gnu.org/onlinedocs/gcc/Function-Attributes.html
    • https://gcc.gnu.org/onlinedocs/gcc/Variable-Attributes.html#Variable-Attributes
Optimization Tips

• Create baselines
  – Accounts for Law of diminishing returns
  – Set an End goal

• You are as good as your tools
  – Find good measurement tools
  – Augment compilation with other tools

• Dare to Experiment
  – Dig deeper into generated code
Optimization Tips

• Consider Portability
  – Follow ISO C standards and demand it from compiler (–std=c99)
  – Its easy to give-up portability under stress
  – Use predefined compiler preprocessor macros
    • https://sourceforge.net/p/predef/wiki/Compilers/

#pragma GCC optimize ("unroll-loops")
Optimization Tips

- Which one is better?

\[ x = x \ ? \ 0 : 10 \]

\[
\text{if} \ (x) \ {\ }
\qquad x = 0; \\
\text{else} \ {\ }
\qquad x = 10; \\
\text{}\}
\]
Optimization Tips - Stack

- Know default stack size – It's not unlimited
- Local variables e.g. Large arrays
- Take a hard look at recursive functions
- End-call optimization

```c
int foo()
{
    ...

    if (..)
        return bar();
    else
        return 0;
}
```
Optimization Tips

• Put most likely code in hotpath
  – Cascade of if-then-else
  – Look for converting simple conditions to switch-case

• Help tail recursion Elimination
  – Return value of recursive call without modifications
int factorial(int x)
{
    if (x == 0) {
        return 1;
    } else {
        return x * factorial(x - 1);
    }
}

int factorial(int x, int f)
{
    if (x == 0) {
        return f;
    } else {
        return factorial(x - 1, f * n);
    }
}
Help the compiler and it will help you
Summary

- Compiler has to be conservative
  - It wont apply an optimization if its not sure
    - Pointer aliasing
  - Do-while is better than for-loops
    - Loop termination test can be optimized out
  - Use compiler provided annotations
    - Function and variable attributes
    - Pragmas
  - Use intrinsic where possible
Summary

• Recommendations
  – Avoid “Release” and “Debug” modes
    • Uniform optimization across production and development saves a lot of time
  – Know your system
    • Processor architecture, bus width, DRAM, Flash, clock speeds etc.
  – Profile before optimize
  – Delegate to tools as much as possible
    • Don’t make them do things they can’t do well
  – Avoid inline assembly
  – Make portability as priority
Thank you