# Code Size Compiler Optimizations and Techniques for Embedded Systems

#### **ADITYA KUMAR**

Compiler Engineer at Snap Inc. Distinguished Speaker at ACM







#### Disclaimer

Any opinions expressed are solely my own and do not express the views or opinions of my employer.

#### Code size of mobile apps

- Energy cost of download\*: 5.12 kWh/GB
- Total apps downloaded in 2020\*\*: 143.0B
- Saving 1MB on each download:
  - 1MB \* 143B \* 5.12kWh/GB = Some large number...

TLDR: Code size matters...

# Compiler optimizations and techniques for code size

#### Measurement techniques

- size (binutils)
- strings (binutils)
- bloaty (https://github.com/google/bloaty)

#### Compiler optimizations

- Common compiler flags
- Compiler specific flags

#### C++ Library optimizations

- Reduce Inlining
- Template instantiations

#### Source code optimizations

- Code restructuring and annotations
- Using cheaper data structures and algorithms
- Using tools

#### Getting insights into source code

• Compiler instrumentation

#### Optimizations yet to be implemented

- In C++ standard libraries
- In Compilers like LLVM and GCC

### Measurement techniques

#### Size

- •size gcc/11/libstdc++.dylib
- \_\_TEXT \_\_DATA \_\_OBJC others dec hex •1703936 65536 0 1851392 3620864 374000

#### Strings

- •strings gcc/11/libstdc++.dylib
- •2180 strings totalling 36kb

#### Bloaty

- •bloaty gcc/11/libstdc++.dylib
- FILE SIZE VM SIZE 29.1% 1.00Mi 29.0% 1.00Mi TEXT, text 25.0% 882Ki 25.0% 882Ki String Table 16.6% 583Ki 16.5% 583Ki Symbol Table 12.3% 433Ki 12.2% 433Ki TEXT, eh frame Export Info 5.0% 176Ki 5.0% 176Ki 4.1% 146Ki 4.1% 146Ki TEXT, const Weak Binding Info 2.5% 87.8Ki 2.5% 87.8Ki 1.2% 41.6Ki 1.2% 41.6Ki DATA, gcc except tab 1.0% 36.9Ki 1.0% 36.9Ki DATA CONST, const TEXT, text cold 0.9% 33.3Ki 0.9% 33.3Ki 0.5% 16.1Ki 0.5% 16.1Ki [10 Others] 0.5% 15.9Ki 0.0% 945 [ DATA] 0.4% 15.0Ki TEXT, cstring 0.4% 15.0Ki 0.0% 0.3% 11.3Ki [ LINKEDIT] 0.0% 0.2% 8.12Ki DATA, bss 0.2% 8.01Ki 0.2% 8.01Ki [ DATA CONST] Function Start Addresses 0.2% 7.43Ki 0.2% 7.43Ki 0.0% 0.2% 6.88Ki DATA, common 6.08Ki Indirect Symbol Table 0.2% 0.2% 6.08Ki DATA, la symbol ptr 0.1% 4.59Ki 0.1% 4.59Ki 0.1% 3.44Ki 0.1% 3.44Ki TEXT, stubs • 100.0% 3.44Mi 100.0% 3.45Mi TOTAL

## Analysis of strings in libstdc++

#### Strings

- •strings gcc/11/libstdc++.dylib
- 2180 strings totalling 36kb

#### What are these strings?

- String literals
- Function names
- Error/Exception messages
- ...

#### Examples

- •"attempt to splice a list into itself"
- "attempt to compare a %1.state; iterator to a %2.state; iterator"
- ...

git grep -l 'attempt to splice a list into
itself'

•libstdc++-v3/src/c++11/debug.cc

## Code size optimization flags

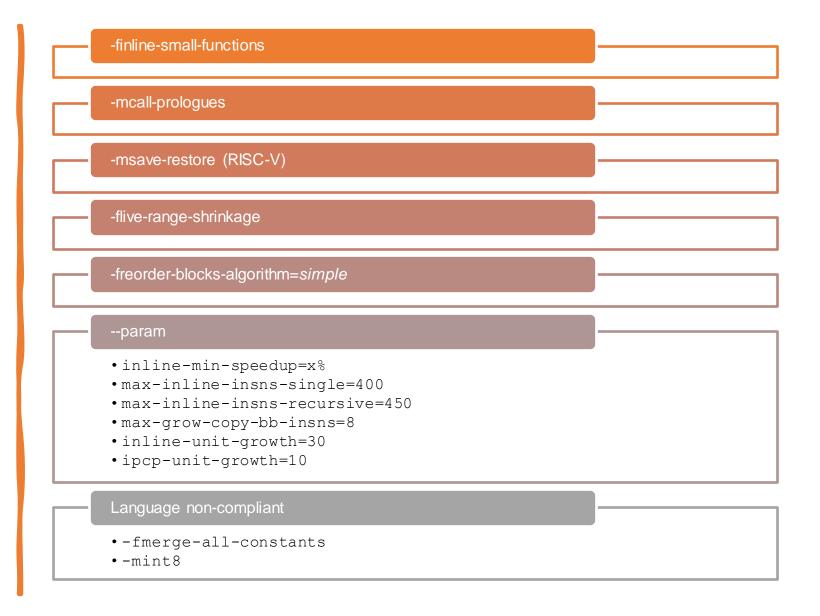
#### Always decrease code size

- -Os
- -flto
- -Wl, --strip-all (Or remove `-g` flag)
- -fno-unroll-loops
- -fno-exceptions
- -fno-rtti

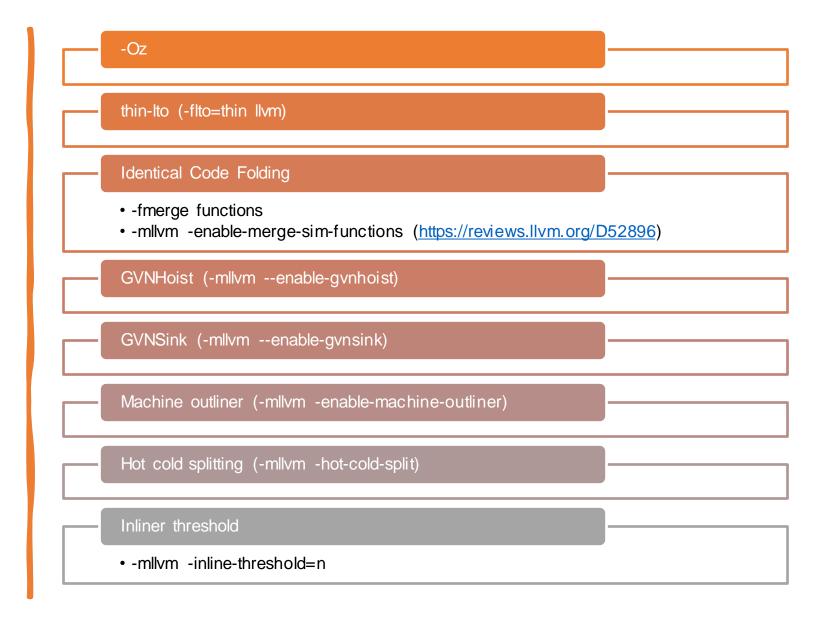
#### May increase code size in some codebase

- -ffunction-sections -Wl, --gcsections
- -fno-jump-tables

# Code size optimization flags (Only gcc)



# Code size optimization flags (Only Ilvm)



# C++ Library optimizations (libstdc++, libc++, boost, eigen)

#### Function definitions in header files

- Explicit template instantiations
  - Reduces code size
  - Reduces compile time
  - Not always possible :(

#### Function attributes

• \_\_attribute\_\_((noinline)) to commonly used functions

#### Source code level optimizations

Code refactoring

Source code annotations to help compiler

Using cheaper data structures

Using cheaper algorithms

Using standard library routines

Using external tools

#### Source code level optimizations (Code refactoring)

Using software engineering techniques

#### Move code out of header files

- Functions
- Classes
- Variables

#### Flattening classes

• Removing useless heirarchies

#### Early evaluation

constexpr

#### Lazy evaluation

- Lambda, Function Objects
- Set of functions -> Hashmap<Key, Lambda Function>

 Source code level optimizations (Code refactoring) Using language features

#### Explicitly generate definitions in .cpp file

- Constructors (Copy, Move etc.)
- Destructors
- Assignment operator
- Explicit instantiations of templates

#### Inheritance

- s/virtual//
- Empty base optimization

#### Member functions -> Free functions

Avoid copying

#### Source code level optimizations (Source code annotations)

#### Function attributes

- attribute ((cold))
- attribute ((noinline))

#### pragmas

- pragma pack
- pragma clang/gcc optimize off/on
  - Careful while putting in .h file
- pragma clang attribute push(\_\_attribute\_\_((noinline)),
   apply to = function)

Source code level optimizations (Using cheaper data structures)

#### COMMONLY USED

#### CHEAPER ALTERNATIVE

std::vector,
std::deque

std::list, std::array

std::unorderd\_map,
std::unordered\_set

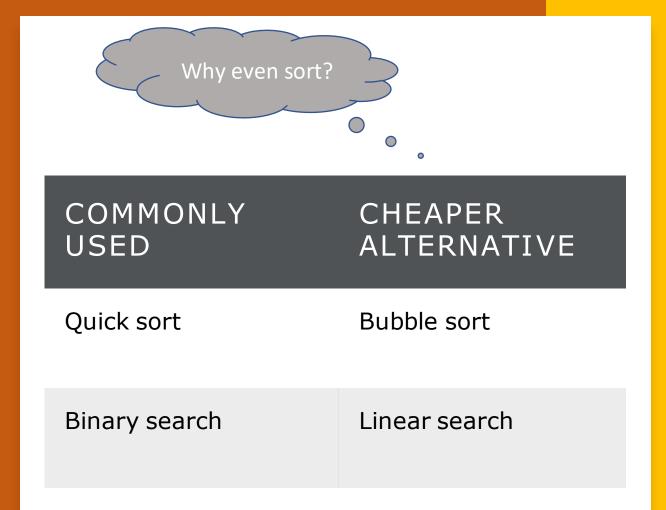
std::map,

std::set

### Source code level optimizations (Using cheaper data structures)

```
// clang++ -Oz map.cpp -o map.o, 13,976 bytes
                                                     // clang++ -Oz umap.cpp -o umap.o, 15,140 bytes
int main() {
                                                     int main() {
  std::map<int, int> m;
                                                        std::unordered map<int, int> m;
  m[10] = 100;
                                                        m[10] = 100;
// clang++ -Oz list.cpp -o list.o, 12,960 bytes
                                                     // clang++ -Oz vector.cpp -o vector.o, 14,308 bytes
int main() {
                                                     int main() {
  std::list<int> l;
                                                        std::vector<int> v;
                                                        for (int i = 0; i < 1000; ++i)
  for (int i = 0; i < 1000; ++i)
    1.push back(i);
                                                          v.push back(i);
  return *1.begin();
```

Source code level optimizations (Using cheaper algorithms)



```
// 10 instructions in assembly at -Oz, 90
                                                     // 3 instructions in assembly
instructions at -O3
                                                     #include<cstring>
                                                     void call_memcpy(int *p, int *q, int
void mymemcpy(int *p, int *q, int sz) {
 for (int i = 0; i < sz; ++i)
                                                     sz) {
  p[i] = q[i];
                                                      std::memcpy(p, q, 4*sz);
mymemcpy(int*, int*, int):
                                                     call memcpy(int*, int*, int):
 movsxd rax, edx
                                                      sal edx, 2
                                                      movsx rdx, edx
 xor ecx, ecx
.LBBO 1: # =>This Inner Loop Header: Depth=1
                                                      imp memcpy # TAILCALL
 cmp rcx, rax
 jge .LBBO 2
 mov edx, dword ptr [rsi + 4*rcx]
 mov dword ptr [rdi + 4*rcx], edx
 inc rcx
 jmp .LBB0 1
.LBB0 2:
 ret
```

Source code level optimizations (Using standard library functions)

#### Source code level optimizations (Using other tools)

#### Moving less frequently used features into a shared library

- Reduces code size of main binary
- Reduces the time to launch the program

#### Compressing less frequently used code

libzlg (<a href="https://liblzg.bitsnbites.eu">https://liblzg.bitsnbites.eu</a>/) has low memory footprint and decoding is fast

#### Compress sections

• libelf, eu-elfcompress

#### Strip symbols

- strip (binutils)
- Ilvm-strip

#### Getting insights into source code

- Function entry instrumentation
  - -finstrument-functions
  - -finstrument-function-entry-bare
  - -fpatchable-function-entry

#### Code size optimizations yet to be implemented

In C++ standard libraries

In Compilers like LLVM and GCC

## Code size optimizations yet to be implemented in C++ standard libraries

**Conditional** noexcept

```
#ifdef ADD UNSAFE NOEXCEPT
#define MAY NOEXCEPT true
#define MAY NOEXCEPT false
template <class Tp, class Allocator>
void vector< Tp, Allocator>:: vallocate(size type __n) noexcept(MAY NOEXCEPT)
if ( n > max size())
  this-> throw length error(); // throw = X
this->__begin_ = this->__end_ = __alloc_traits::allocate(this->__alloc(), __n);
this->__end_cap() = this->__begin_ + __n;
  annotate new(0);
```

## Code size optimizations yet to be implemented in compilers (gcc, clang)

- Rename functions to take advantage of linker deduplication
  - int get\_my\_favorite\_int() { return 10; } // rename to f\_returns\_10();
    int get\_dec\_base() { return 10; } // rename to f\_returns\_10();
- Aggressive outlining of cold regions
  - SESE, SEME
  - Merging functions after hot cold splitting
- Merging similar functions
  - Prototype in LLVM: https://reviews.llvm.org/D22051 https://reviews.llvm.org/D111912

## Code size optimizations (yet to be implemented in llvm)

- Outline prologue and epilogue
- Support for atrributes and pragmas
  - <u>attribute</u> ((optsize))pragma GCC optimize("Os")
- Basic Block Reordering to minimize code size
- Split function before inlining
  - Fuse hot cold splitting with inliner
- De-duplicate code from sibling branches <u>Bug#47215</u>
- Loop idiom recognition\*
  - memset
  - memcpy

\* Some memcpy, memset patterns aren't recognized by clang, but recognized by gcc: <a href="https://godbolt.org/z/dPvGY3edr">https://godbolt.org/z/dPvGY3edr</a>

#### References

- man gcc
- clang --help-hidden
- man elfcompress
- Ilvm-strip --help
- "-Os Matters" by Mark Zeren
  - https://www.youtube.com/watch?v=vGV5u1nx qd8
- https://github.com/google/bloaty
- <a href="https://www.mail-archive.com/gcc@gcc.gnu.org/msg91116.html">https://www.mail-archive.com/gcc@gcc.gnu.org/msg91116.html</a>
- http://gcc.gnu.org/
- http://llvm.org

