



Code size compiler optimizations and techniques for embedded systems

Aditya Kumar

Code size compiler optimizations and techniques

Compiler optimizations	Flags Optimizations to enable
C++ Library optimizations	Reduce Inlining Template instantiations
Source code optimizations	Moving function definitions to .c/.cpp file Using cheaper data structures
Getting insights into source code	Compiler instrumentation Moving less frequently used features into a shared library



Code size optimization flags

- Code size optimization flags
 - -Os
 - -Oz (only LLVM)
 - -fno-function-sections
 - -fno-unroll-loops
 - -fno-exceptions
 - -fno-rtti
 - -finline-limit=n (gcc), -mllvm -inline-threshold=n (llvm)
 - -fno-jump-tables

Code size optimization flags (Only gcc)

-mcall-prologues

-mint8 (not language compliant)

-msave-restore (RISC-V)

-freorder-blocks-algorithm=*simple*

- --param
 - inline-min-speedup=x%
 - max-inline-insns-single=400
 - max-inline-insns-recursive=450
 - max-grow-copy-bb-insns=8
 - inline-unit-growth=30
 - ipcp-unit-growth =10

Compiler Optimizations to enable

lto (-flto)

thin-lto (-flto=thin llvm)

Identical Code Folding

- -fipa-icf(gcc)
- -fmerge functions (llvm)
- -mllvm -enable-merge-sim-functions (llvm <https://reviews.llvm.org/D52896>)

GVNHoist (-mllvm --enable-gvnhoist)

GVNSink (-mllvm --enable-gvnsink)

Machine outliner (-mllvm -enable-machine-outliner)

Hot cold splitting (-mllvm -hot-cold-split)

C++ Library optimizations

Widely used C++ libraries have function definitions in header files

- libstdc++, libc++, boost, eigen

Explicit template instantiations

- Reduces code size
- Reduces compile time

Function attributes

- `__attribute__((noinline))` to commonly used functions

Source code optimizations

Moving function definitions to .c/.cpp file

- Functions in header file can get inlined and contribute to code-size

C++ code

- Constructors (Copy, Move etc.)
- Destructors
- Operator overloading
- Explicit instantiations of templates

Source code optimizations

Using cheaper data structures

Commonly used	Cheaper alternative
----------------------	----------------------------

<code>std::vector,</code> <code>std::deque</code>	<code>std::list</code>
--	------------------------

<code>std::unordered_map,</code> <code>std::unordered_set</code>	<code>std::map, std::set</code>
---	---------------------------------

Source code optimizations

```
// clang++ -Oz test.cpp -o test.o, 13976 bytes
#include<map>
int main() {
    std::map<int, int> m;
    m[10] = 100;
    return m[0];
}
```

```
// clang++ -Oz test.cpp -o test.o, 15140 bytes
#include<unordered_map>
int main() {
    std::unordered_map<int, int> m;
    m[10] = 100;
    return m[0];
}
```

```
// clang++ -Oz test.cpp -o test.o, 12960 bytes
#include<list>
int main() {
    std::list<int> l;
    for (int i = 0; i < 1000; ++i)
        l.push_back(i);
    return *l.begin();
}
```

```
// clang++ -Oz test.cpp -o test.o, 14308 bytes
#include<vector>
int main() {
    std::vector<int> v;
    for (int i = 0; i < 1000; ++i)
        v.push_back(i);
    return v[0];
}
```

Getting insights into source code

Function entry instrumentation

- -finstrument-functions
- -fpatchable-function-entry

Cheap function entry instrumentation

- Very low overhead function entry instrumentation
- Lock free
- (Patch in review:
<https://reviews.llvm.org/D74362>)



Getting insights into source code

- Moving less frequently used features into a shared library
 - Reduces the time to launch the program

Binary compression

- Compressing binary to reduce codesize
 - libzlg (<https://libzlg.bitsnbites.eu/>) has low memory footprint and decoding is fast

References

- <https://reviews.llvm.org/D52896>
- <https://reviews.llvm.org/D74362>
- gcc.gnu.org
- llvm.org