FACEBOOK

Code size compiler optimizations and techniques for embedded systems

Aditya Kumar

@_hiraditya_

Code size compiler optimizations and techniques

Compiler optimizations	Flags Optimizations to enable
C++ Library optimizations	Reduce Inlining Template instantiations
Source code optimizations	Code restructuring and annotations Using cheaper data structures and algorithms
Getting insights into source code	Compiler instrumentation Moving less frequently used features into a shared library
Compiler optimizations yet to be implemented in LLVM	Loop idiom Split function before inlining etc.

Code size optimization flags

- -Os
- -Oz
- -Wl,--strip-all (Or do not pass `-g` flag)
- -fno-function-sections
- -fno-unroll-loops
- -fno-exceptions
- -fno-rtti
- -fno-jump-tables

Compiler Optimizations to enable

- lto (-flto)
- thin-lto (-flto=thin llvm)
- Identical Code Folding
  - -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)
- Play with inliner threshold
  - -mllvm -inline-threshold=n

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 level optimizations

- Code refactoring
- Source code annotations to help compiler
- Using cheaper data structures
- Using cheaper algorithms

Source code level optimizations

Code refactoring

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 level optimizations

Source code annotations

Function attributes

- `__attribute__((cold))`
- `__attribute__((noinline))`

C++ code

- `pragma clang 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 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];
}
```

Source code level optimizations

Using cheaper algorithms

Commonly used	Cheaper alternative
Quick sort	Bubble sort
Binary search	Linear search

Source code level optimizations

Using standard library algorithms

// 10 instructions in assembly at –Oz, 90 instructions at -O3

```
void mymemcpy(int *p, int *q, int sz) {
  for (int i = 0; i < sz; ++i)
    p[i] = q[i];
}
```

```
mymemcpy(int*, int*, int):
  movsxd rax, edx
  xor ecx, ecx
.LBB0_1: # =>This Inner Loop Header: Depth=1
  cmp rcx, rax
  jge .LBB0_2
  mov edx, dword ptr [rsi + 4*rcx]
  mov dword ptr [rdi + 4*rcx], edx
  inc rcx
  jmp .LBB0_1
.LBB0_2:
  ret
```

// 2 instructions in assembly

```
#include<cstring>
void mcpy(int *p, int *q, int sz) {
  memcpy(p, q, sz);
}
```

```
mcpy(int*, int*, int):
  movsxd rdx, edx
  jmp memcpy # TAILCALL
```

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
- Compressing less frequently used code
  - libzlg (https://liblzg.bitsnbites.eu/) has low memory footprint and decoding is fast
- Compress sections (elfcompress)
- Strip symbols (llvm-strip)

Code size optimizations (yet to be implemented in llvm)

- Outline prologue and epilogue
- Short int
- Support for atrributes and pragmas
  - __attribute__((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 PR47215

Code size optimizations (yet to be implemented in llvm)

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();

Loop idiom recognition*

- memset
- memcpy

* Even `memcpy`, `memset` aren't recognized by clang: https://godbolt.org/z/d5Mxea

References

- https://reviews.llvm.org/D52896
- https://reviews.llvm.org/D74362
- `man gcc`
- `clang –help-hidden`
- `man elfcompress`
- `llvm-strip --help`

@_hiraditya_