Compiler Flags for Code Size and Performance

ADITYA KUMAR

 @_hiraditya_

Code size impact on download of mobile apps [2022]

Energy cost of download*: 0.072 kWh/GB

Total apps downloaded in 2022**: 143.0B

Energy savings for 1MB reduction on each download: **1MB * 143B * 0.072kWh/GB = 10.3 gWh**

Equivalent to 765 million tree seedlings grown for 10 years***

TLDR: Code size matters...

* https://www.carbonbrief.org/factcheck-what-is-the-carbon-footprint-of-streaming-video-on-netflix/

** https://www.businessofapps.com/data/app-statistics/

*** https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator#results

Measurement techniques for code size

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

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
•   5.0%   176Ki   5.0%   176Ki    Export Info
•   4.1%   146Ki   4.1%   146Ki    __TEXT,__const
•   2.5%  87.8Ki   2.5%  87.8Ki    Weak Binding Info
•   1.2%  41.6Ki   1.2%  41.6Ki    __DATA,__gcc_except_tab
•   1.0%  36.9Ki   1.0%  36.9Ki    __DATA_CONST,__const
•   0.9%  33.3Ki   0.9%  33.3Ki    __TEXT,__text_cold
•   0.5%  16.1Ki   0.5%  16.1Ki    [10 Others]
•   0.5%  15.9Ki   0.0%     945    [__DATA]
•   0.4%  15.0Ki   0.4%  15.0Ki    __TEXT,__cstring
•   0.0%       4   0.3%  11.3Ki    [__LINKEDIT]
•   0.0%       0   0.2%  8.12Ki    __DATA,__bss
•   0.2%  8.01Ki   0.2%  8.01Ki    [__DATA_CONST]
•   0.2%  7.43Ki   0.2%  7.43Ki    Function Start Addresses
•   0.0%       0   0.2%  6.88Ki    __DATA,__common
•   0.2%  6.08Ki   0.2%  6.08Ki    Indirect Symbol Table
•   0.1%  4.59Ki   0.1%  4.59Ki    __DATA,__la_symbol_ptr
•   0.1%  3.44Ki   0.1%  3.44Ki    __TEXT,__stubs
• 100.0%  3.44Mi 100.0%  3.45Mi    TOTAL
```

Code size optimization flags

Always decrease code size

- `-Os`
- `-Oz` (https://godbolt.org/z/zdMKhfe7G)
- `-flto`
- `-fno-unroll-loops`
- `-fno-exceptions`
- `-fno-rtti`

May increase code size in some codebase

- `-ffunction-sections -Wl,--gc-sections`
- `-fno-jump-tables`

Additional Code size optimization flags

- thin-lto (-flto=thin llvm)
- GVNHoist (-mllvm --enable-gvnhoist)
- GVNSink (-mllvm --enable-gvnsink)
- Machine outliner (-mllvm -enable-machine-outliner)
- Hot cold splitting (-mllvm -hot-cold-split)
- Change Inliner threshold
  - -mllvm -inline-threshold=n

Impact of performance optimizations

- On smartphones
  - Android, iPhone
- Power consumption (4kWh per year)
  - Very small number but..
  - * 6.8 billion
- Improving 1% of power consumption on all smartphones
  - 1% * 4kWh * 6.8 billion = 272 gWh = 20 billion tree seedlings grown for 10 years

Compiler optimizations for performance

- Optimization level -O2, -O3, -Ofast (https://godbolt.org/z/4vPr6dhWb)
- LTO (-flto, -flto=thin)
- -fno-exceptions
- -fno-rtti
- -mllvm -hot-cold-split (https://godbolt.org/z/zhddPh5ex)
- -mllvm --enable-gvnhoist
- -mllvm --enable-gvnsink

Performance analysis tools

- Valgrind
- Linux Perf
- Visual studio performance tool
- Intel Vtune
- XCode Instruments

Performance Analysis with Valgrind

valgrind [--tool=memcheck]

- valgrind mostly known for its memory leak checker

valgrind --tool=cachegrind

- cache and branch simulator
- count read, write, and branch instructions

valgrind --tool=callgrind

- execution call graph
- visualization tool kcachegrind

Valgrind: Example – SQLite

```
$ valgrind --tool=cachegrind ./sqlite_llvm <test.sql >/dev/null
[...]
--------------------------------------------------------------------------------
           Ir       I1mr   ILmr          Dr      D1mr    DLmr          Dw      D1mw    DLmw
--------------------------------------------------------------------------------
1,278,771,731 29,231,219 35,783 359,414,267 6,707,514 528,920 197,515,528 2,594,262 171,968  PROGRAM TOTALS

--------------------------------------------------------------------------------
         Ir      I1mr  ILmr         Dr      D1mr    DLmr         Dw    D1mw   DLmw  file:function
--------------------------------------------------------------------------------
363,052,233 7,560,087 3,122 97,707,865 1,084,529  77,197 44,505,055 217,826 29,838  src/sqlite3.c:sqlite3VdbeExec
 95,048,357    80,721   111 33,248,107    59,086   7,273 20,173,275      91      7  src/sqlite3.c:vdbeRecordCompareWithSkip
 68,045,026   695,509 1,144 14,883,933   114,698   1,918  5,525,733 272,507 19,249  src/sqlite3.c:balance
 56,713,554 1,101,002   276 18,416,705   683,914  21,085  3,453,665   1,947     25  src/sqlite3.c:sqlite3BtreeMovetoUnpacked
 45,344,891    59,660    66 13,589,490    66,121  18,775 12,795,281  59,451     86  src/sqlite3.c:sqlite3VdbeRecordUnpack
 36,550,248    47,192    94  9,615,816   217,845  11,567          0       0      0  src/sqlite3.c:cellSizePtr
 35,156,491 1,031,905   859  7,810,853   489,509   1,936  6,546,085 175,469 26,159  /build/glibc-2.19/malloc/malloc.c:_int_malloc
 34,402,967   219,015    40 12,316,213    31,625   1,007          0       0      0  src/sqlite3.c:vdbeRecordCompareInt
```

Performance Analysis with Linux Perf

perf stat

- sum up all counters

perf record

- record events

perf report

- Shows the profile collected using `perf record`

Perf stat: Example – SQLite

```
 $ perf stat ./sqlite_llvm <test.sql >/dev/null

Performance counter stats for './sqlite_llvm':

      1045.856070      task-clock (msec)         #    1.000 CPUs utilized
                1      context-switches          #    0.001 K/sec
                0      cpu-migrations            #    0.000 K/sec
              809      page-faults               #    0.774 K/sec
    1,636,720,010      cycles                    #    1.565 GHz                      [83.16%]
      548,530,227      stalled-cycles-frontend   #   33.51% frontend cycles idle     [83.16%]
      218,991,051      stalled-cycles-backend    #   13.38% backend  cycles idle     [67.04%]
    3,385,841,295      instructions              #    2.07  insns per cycle
                                                 #    0.16  stalled cycles per insn [83.54%]
      709,436,490      branches                  #  678.331 M/sec                    [83.54%]
        2,586,354      branch-misses             #    0.36% of all branches          [83.17%]

      1.045918998 seconds time elapsed
```

Perf record: Example – xalancbmk

```
$ perf record ./xalancbmk
$ perf report
  0.20 629a84:   ldr    w9, [x0,#24]
 18.71 629a88:   ldr    w8, [x1,#24]
 12.93 629a8c:   cmp    w9, w8
  2.74 629a90:   b.ne   629af8 <xalanc_1_8::XalanDOMString::equals
  2.00 629a94:   ldp    x8, x10, [x0]
  2.43 629a98:   cmp    x8, x10
  1.80 629a9c:   ldp    x10, x12, [x1]
  1.03 629aa0:   adrp   x11, 704000 <vtable for xalanc_1_8::ReusableArenaBlock+0x8>
  0.53 629aa4:   add    x11, x11, #0xb08
  0.03 629aa8:   csel   x8, x11, x8, eq
  1.33 629aac:   cmp    x10, x12
  0.34 629ab0:   csel   x10, x11, x10, eq
  1.78 629ab4:   cbz    w9, 629b00 <xalanc_1_8::XalanDOMString::equals
  0.02 629ab8:   ldrh   w11, [x8]
  4.02 629abc:   ldrh   w12, [x10]
  3.75 629ac0:   cmp    w11, w12
  1.03 629ac4:   b.ne   629b08 <xalanc_1_8::XalanDOMString::equals
  1.16 629ac8:   lsl    x9, x9, #1
```

Analyzing System Performance

Vary one Component of the System at a time

- Measure impact of one component on the System
- Run multiple times

Disable frequency scaling

- cpufrequtils

Performance metrics

- Dynamic profiles, compiler logs

Systematic performance analysis

- Monitor performance regression over time
- Time series: track performance of system over time
- Git bisect performance changes

Performance analysis pitfalls

Central tendencies

- The median instead of the mean

Use the quantile values instead of a single median value

- Helps with prioritization

Outlier detection

- Filter outliers

Weighted samples for combining historical data

- Recent data more important than the previous data.

Case studies

std::vector copy constructor

```cpp
// clang++ -std=c++17 -O3 -fno-exceptions -stdlib=libc++
#include<vector>


using T = int;
T vec_copy(const std::vector<T> &v1) {
    auto v(v1);
    return 10;
}


/*
// Or
T vec_copy(std::vector<T> v1) {
    auto v(v1);
    return 10;
}
*/
```

```
vec_copy(std::__1::vector<int, std::__1::allocator<int>
> const&): # @vec_copy(std::__1::vector<int,
std::__1::allocator<int> > const&)
push r15
push r14
push rbx
mov r15, qword ptr [rdi]
mov r14, qword ptr [rdi + 8]
mov rbx, r14
sub rbx, r15
je .LBB0_10
js .LBB0_11
mov rdi, rbx
call operator new(unsigned long)@PLT
add rbx, -4
cmp rbx, 28
jb .LBB0_3
mov rcx, rax
sub rcx, r15
cmp rcx, 32
jb .LBB0_3
shr rbx, 2
inc rbx
mov rsi, rbx
and rsi, -8
lea rcx, [r15 + 4*rsi]
lea rdx, [rax + 4*rsi]
xor edi, edi
.LBB0_6: # =>This Inner Loop Header: Depth=1
movups xmm0, xmmword ptr [r15 + 4*rdi]
movups xmm1, xmmword ptr [r15 + 4*rdi + 16]
movups xmmword ptr [rax + 4*rdi], xmm0
movups xmmword ptr [rax + 4*rdi + 16], xmm1
add rdi, 8
cmp rsi, rdi
jne .LBB0_6
cmp rbx, rsi
jne .LBB0_8
jmp .LBB0_9
.LBB0_3:
mov rcx, r15
mov rdx, rax
.LBB0_8: # =>This Inner Loop Header: Depth=1
mov esi, dword ptr [rcx]
mov dword ptr [rdx], esi
add rcx, 4
add rdx, 4
cmp rcx, r14
jne .LBB0_8
.LBB0_9:
mov rdi, rax
call operator delete(void*)@PLT
.LBB0_10:
mov eax, 10
pop rbx
```

[copy] After optimization D147741

```
// clang++ -std=c++17 -O3 -fno-
exceptions -stdlib=libc++
#include<vector>


using T = int;
T vec_copy(const std::vector<T> &v1)
{
    auto v(v1);
    return 10;
}
```

https://godbolt.org/z/7q9PdrT81

```
vec_copy(std::__1::vector<int,
std::__1::allocator<int> > const&):
sub rsp, 24
xorps xmm0, xmm0
movaps xmmword ptr [rsp], xmm0
mov qword ptr [rsp + 16], 0
mov rax, qword ptr [rdi + 8]
sub rax, qword ptr [rdi]
je .LBB0_2
js .LBB0_3


.LBB0_2:
mov eax, 10
add rsp, 24
ret


.LBB0_3:
mov rdi, rsp
call std::__1::vector<int,
std::__1::allocator<int>
>::__throw_length_error[abi:v170000]()
const
.L.str:
.asciz "vector"


.L.str.1:
.asciz "length_error was thrown in -fno-
exceptions mode with message \"%s\""
```

std::vector<int> access

```
#include <vector>
void f(int);


void subscript_operator(std::vector<int> v)
{
    for (std::vector<int>::size_type i = 0; i < v.size(); i++)
        f(v[i]);
}


void iterator(std::vector<int> v)
{
    for (std::vector<int>::const_iterator i = v.begin(); i != v.end(); ++i)
        f(*i);
}


void range_loop(std::vector<int> v)
{
    for (auto i : v)
        f(i);
}
```

std::vector<int> access

```cpp
#include <vector>
void f(int);


void subscript_operator(std::vector<int> v)
{
    for (std::vector<int>::size_type i = 0; i < v.size(); i++)
        f(v[i]);
}


void iterator(std::vector<int> v)
{
    for (std::vector<int>::const_iterator i = v.begin(); i != v.end(); ++i)
        f(*i);
}


void range_loop(std::vector<int> v)
{
    for (auto i : v)
        f(i);
}
```

```asm
.LBB0_2: # => Inner Loop Header:
mov edi, dword ptr [rax + 4*r14]
call f(int)@PLT
inc r14
mov rax, qword ptr [rbx]
mov rcx, qword ptr [rbx + 8]
sub rcx, rax
sar rcx, 2
cmp r14, rcx
jb .LBB0_2
```

```asm
.LBB2_2: # =>Inner Loop Header:
mov edi, dword ptr [r14]
call f(int)@PLT
add r14, 4
cmp r14, qword ptr [rbx + 8]
jne .LBB2_2
```

```asm
.LBB1_1: # =>Inner Loop Header
mov edi, dword ptr [rbx]
call f(int)@PLT
add rbx, 4
cmp rbx, r14
jne .LBB1_1
```

https://godbolt.org/z/zMGTaEfYM

References (performance)

- https://github.com/hiraditya/std-benchmark/blob/master/docs/slides/slide-DAC-2017.pdf
- Profile-based Indirect Call Promotion
- Developer Tools #WWDC16
- Eliminating Virtual Function Calls in C++ Programs
- http://boost-sandbox.sourceforge.net/libs/proto/doc/html/boost_proto/users_guide/getting_started/hello_world.html
- Linkers and Loaders by John R. Levine
- https://man7.org/linux/man-pages/man2/getrusage.2.html
- https://www.gnu.org/software/hurd/gnumach-doc/Task-Information.html
- https://en.wikipedia.org/wiki/Lazy_initialization
- https://stackoverflow.com/a/28146199/811335
- http://www.dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html
- https://gist.github.com/C0deH4cker/80b53de22012146ea9d8
- https://blog.timac.org/2016/0804-dump-decrypted-mach-o-files/
- https://embeddedartistry.com/blog/2019/05/20/exploring-startup-implementations-os-x/
- https://engineering.fb.com/2015/11/20/ios/optimizing-facebook-for-ios-start-time/

References (code size)

- https://github.com/hiraditya/std-benchmark/blob/master/docs/slides/CppConCodesizeCompilerOptimizationAndTechniques.pdf
- man gcc
- clang --help-hidden
- "-Os Matters" by Mark Zeren
  - https://www.youtube.com/watch?v=vGV5u1nxqd8
- https://github.com/google/bloaty
- https://www.mail-archive.com/gcc@gcc.gnu.org/msg91116.html
- http://gcc.gnu.org/

Compiler Flags for Code Size and Performance

ADITYA KUMAR

 @_hiraditya_