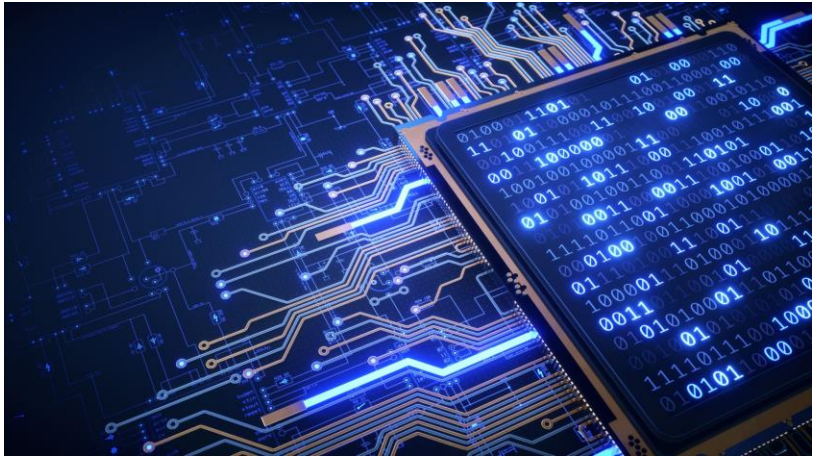


Compiler optimizations and software engineering techniques for improving application launch time

Aditya Kumar

Software Engineer

@_hiraditya_

linkedin.com/in/hiraditya

Program startup types

- Cold start
- Warm start
  - Cached/Prewarmed state
- Hot start

App startup compiler optimizations and techniques [cold start]

Program launch sequence

- Loader
- Rebase+Binding
- Static initializer
- crt etc.

Factors affecting startup

- Page faults
- extern functions
- Language runtime
- Static initializers and attribute constructors

Measurement and optimization techniques of different stages of startup

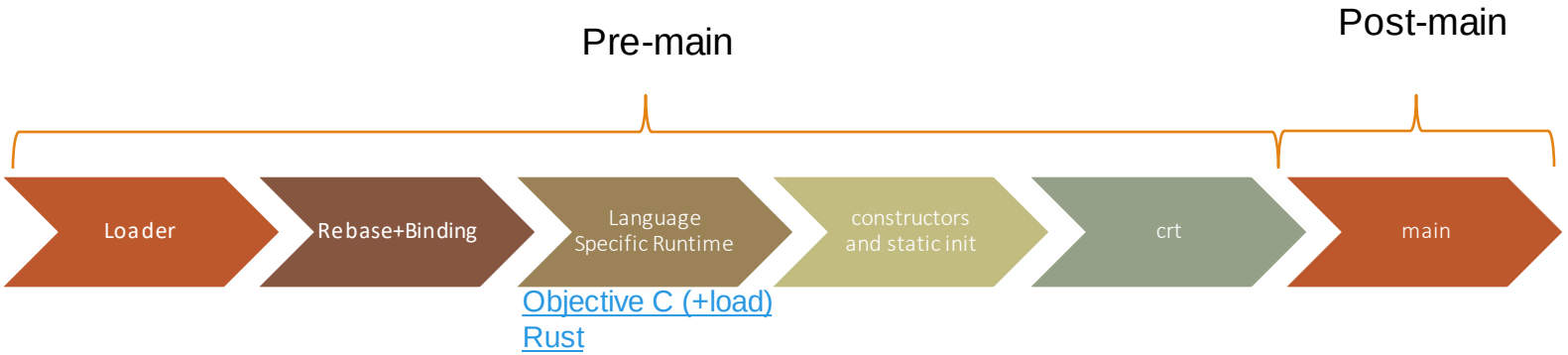
- Build time and runtime measurement techniques
- Build time and runtime optimizations

Compiler optimizations and software engineering techniques

- Compiler optimizations that helps with startup
- Software engineering techniques
- Compiler optimizations yet to be implemented in LLVM

Program launch sequence

Program launch sequence



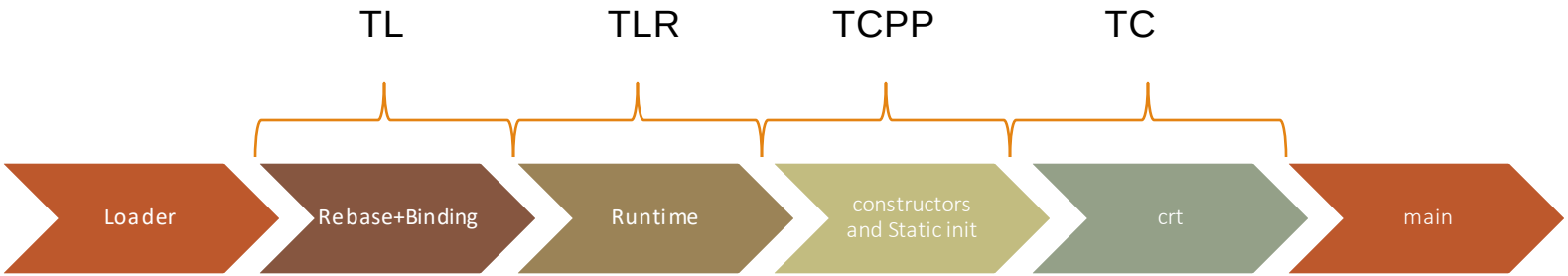
Measuring launch time spans

- By inserting timers wherever possible

Helps with

- Detect regressions
- Find bottlenecks

Measuring launch time spans



TL = Time to finish Loading the binary into memory
TLR = Time taken to execute Language runtime code
TCPP = Time taken by C++ static initializers and static constructors
TC = Time taken by the C runtime

Factors affecting cold start

Launch time is affected by

- TL
  - Page faults
  - extern functions
- TLR
  - Language runtime, pre-main methods
- TCPP
  - Static initializers and attribute constructors
  - C++ standard library static constructors
- TC
  - C runtime library setup

TL = Time to finish Loading the binary into memory
TLR = Time taken to execute Language runtime code
TCPP = Time taken by C++ static initializers and static constructors
TC = Time taken by the C runtime

Measurement & optimization techniques

Launch time can be improved by

- TL
  - Reducing Page faults
  - Reducing number of rebases and bindings
- TLR
  - Reducing work done by the language runtime
- TCPP
  - Reducing number of static initializers and static constructors
- TC
  - N/A for the most part

TL = Time to finish Loading the binary into memory
TLR = Time taken to execute Language runtime code
TCPP = Time taken by C++ static initializers and static constructors
TC = Time taken by the C runtime

[TL] Page faults, rebases and bindings

Measuring page faults

- getrusage
  - `struct rusage::ru_majflt`
- task_info
  - `struct task_events_info::faults`
- ps
  - majflt

Compiler optimizations that helps reduce page faults

- Compiler optimizations to reduce binary size (page faults ∝ binary size):
  - `-Oz, -Os`
  - `-fno-exceptions`
  - `-fno-rtti`
  - `-flto, -flto=thin`
  - Identical code folding (function merging)
  - Machine outliner
  - Devirtualization
- Reduce working set size
  - hot-cold-splitting

Other optimizations that helps reduce page faults

- Order file
- Code restructuring and annotations

Order file

- Helps with spatial locality
- Order file is a line separated list of function names
  - Mangled names for C++

```
$ clang++ hellp.cpp -o a.out -O2
$ objdump -D a.out | bash_magic
_main
__put_character_sequence
__pad_and_output

$ cat hello.order
__ZNSt3__124__put_character_sequenceIc...
__ZNSt3__116__pad_and_outputIcNS_11char_traitsI...
_main

$ clang++ hello.cpp -o b.out -O2 -Wl,-order_file -Wl,hello.order
$ objdump -D b.out | bash_magic
__put_character_sequence
__pad_and_output
_main
```

Steps to generate order file

- Execute the program
  - Instrumented or Bare
- Collect traces
  - Sampling (ftrace, dtrace)
  - Instrumented (-finstrument-function-entry-bare, -forder-file-instrumentation)
  - lldb (set breakpoint on all the functions)
- Generate order file by merging traces
  - Functions, static initializers
- Build the app with order file
  - `ld -order_file <file> -order_file_statistics <args> <binaries> -o a.out`

Measuring rebases and bindings

- Rebases
  - `objdump [-macho] --rebase Binary | wc`
- Bindings
  - `llvm-objdump [-macho] --bind` *`Binary`* `| wc`
  - `llvm-objdump [-macho] --lazy-bind` *`Binary`* `| wc`
  - `llvm-objdump [-macho] --weak-bind` *`Binary`* `| wc`
- Using dyldinfo
  - `dyldinfo [-rebase|-bind|-weak-bind|...] | wc`

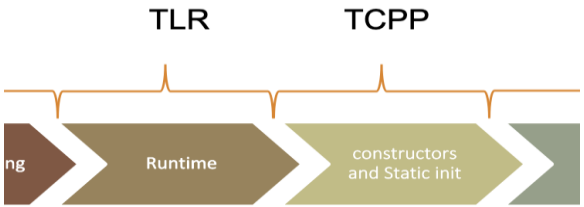
Reducing number of rebases and bindings

- Compiler optimizations to reduce rebases and bindings:
  - `-flto, -flto=thin`
- Making extern functions static when possible

[TLR] Language Runtime

Measuring impact of language runtime

- Use debugger or binutil tools to find the first called function
- Add 'start' timer
  - In the first function called.
- The 'end' timer
  - In the last function called or,
  - In the first function of static initializer (next stage)



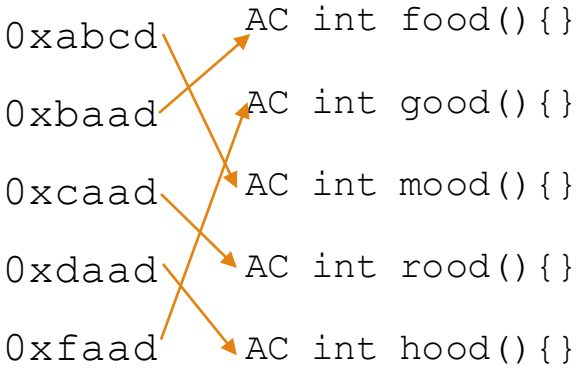
Reducing language runtime

- Removing pre-main functions as much as possible
  - Objective-C: Reduce number of +load in your program
- Do minimal work in pre-main functions
- Also helps with reliability of the app

[TCPP] Static initializers, constructors

Init array & SIOF

- Pointers to functions stored as array
  - Static constructor
- Each function in the init array is invoked in a sequence
- The order of init array is non-deterministic
  - Depends on the order of definition, as well as order of linking



Measuring impact of static initializers

- Approach #1:
  - Find which function executes first, and last (with debugger, or binutil tools)
  - Insert timers
- Approach #2
  - Add two constructors (begin, end) with init_priority
    - [gnu linker] respects priority across translation units
  - Manipulate the init_array (mod_init_func) to reorder the constructors to begin and end of the array
    - Good Starting point: https://gist.github.com/C0deH4cker/80b53de22012146ea9d8

Reducing number of static initializers and attribute constructors

- Removing global static objects
  - Lazy evaluation by initializing the static object in function [Meyer's singleton]
- Removing unnecessary `__attribute__((constructors))`

Compiler Optimizations

Compiler optimizations that affect startup

- Optimization level -Os, -Oz, -O3
- LTO
- -fno-exceptions
- -fno-rtti
- -mllvm -hot-cold-split=true
- -fmerge-functions

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

- Outlines cold subgraphs and puts them in a separate section
- Reduces page fault
- Enabled for all iPhone apps (in swift-llvm)

Profile guided indirect call promotion

- Helps with inlining and other optimization
- Enabled in lto and thinlto

Devirtualization

- Whole program devirtualization
  - -flto, -flto=thin
- [Objective-C] Annotating classes+methods with `objc_direct`.
  - Calls to `objc_direct` methods become C function calls
  - Objective-C metadata w.r.t. methods are removed

Software Engg Techniques

Code restructuring and annotations

Early evaluation

- `constexpr` (C++)

Lazy evaluation

- Creating functions objects
- Set of functions -> Hashmap<Key, Lambda Function>
- Moving the code not used in startup to a separate shared library

Caching

- Possibly with value profiling

Codegen at runtime

- vector<lambda>
- unordered_map<key, lambda>

Removing Dead Code

- Remove test code shipped to production
  - Code used only in tests ending up in the shipped binary
    - nm <Binary> | grep -i "test\|debug" :)

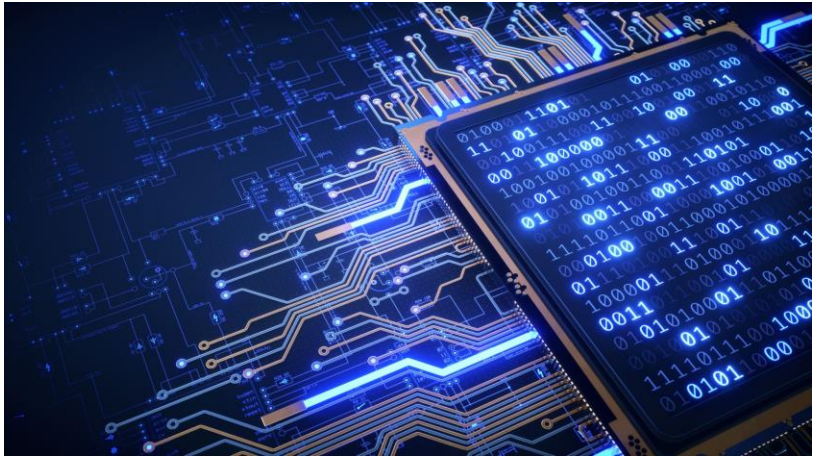
Compiler Optimizations yet to be implemented

Measuring static init time

- Limitations of linker, SIOF, C++
- Binary reordering of static initialization sequence

References

- 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/



App Startup compiler optimizations and techniques for embedded systems

Aditya Kumar

Software Engineer

@_hiraditya_

linkedin.com/in/hiraditya

Backup