~~Statically~~ Compiling Julia

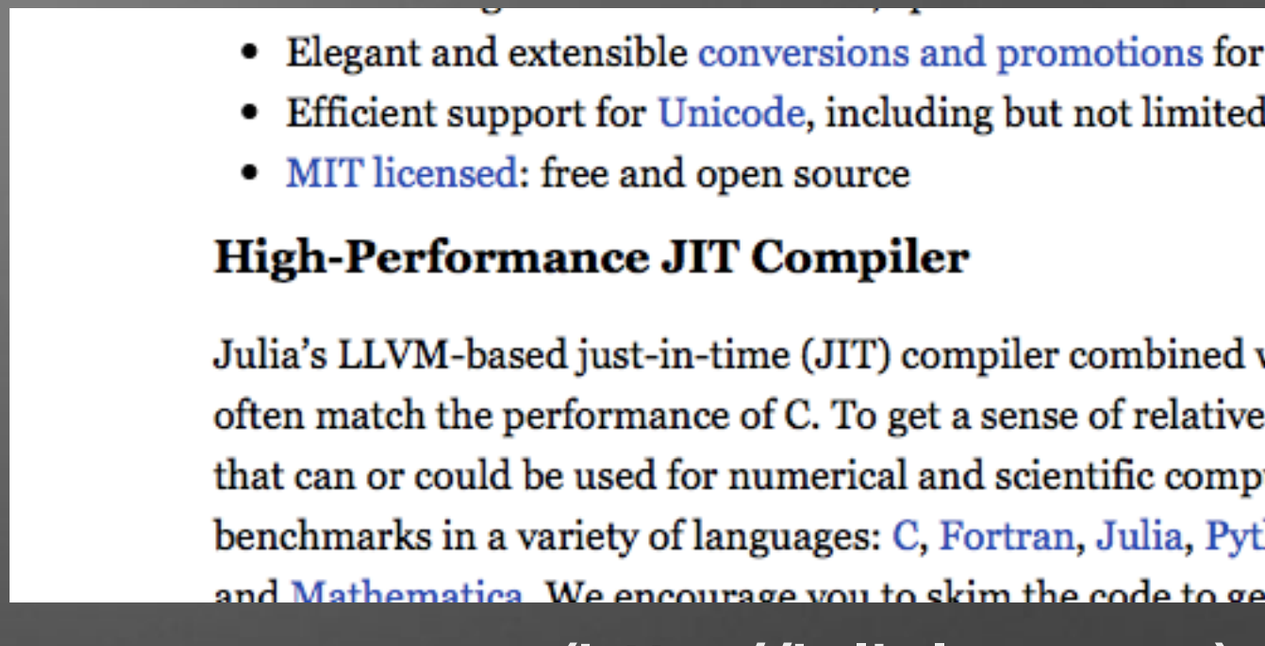
Jameson Nash

Julia is an Ahead-of-Time (AoT) statically-compiled language*!

(*excludes staged functions)

But doesn't it use a JIT?

The homepage of Julia describes the LLVM JIT compiler that helps to make Julia fast.



- Elegant and extensible conversions and promotions for
- Efficient support for Unicode, including but not limited
- MIT licensed: free and open source

High-Performance JIT Compiler

Julia's LLVM-based just-in-time (JIT) compiler combined v
often match the performance of C. To get a sense of relative
that can or could be used for numerical and scientific comp
benchmarks in a variety of languages: C, Fortran, Julia, Pyt
and Mathematica. We encourage you to skim the code to ge

(http://julialang.org)

What it doesn't mention, is that Julia's language design permits better Ahead-of-Time analysis than a traditional JIT must deal with.

What does AoT mean?

Full pre-analysis (type inference) valid

Methods can’t be changed on-stack

No runtime checks on types required by the JIT

Infamous issue #265: previous statements aren’t reliably enforced by the runtime / compiler :(

Adding a method requires updating the AoT compiled code:

- “toplevel” code vs. function code
- natural separation: defining code vs. running data

Julia: REPL or Compiler?

```
~/julia$ ./julia --help
julia [switches] -- [programfile] [args...]
 -v, --version             Display version information
 -h, --help                Print this message

 -J, --sysimage <file>     Start up with the given system image file
 --precompiled={yes|no}    Use precompiled code from system image if available
 --compilecache={yes|no}   Enable/disable incremental precompilation of modules
 -H, --home <dir>          Set location of julia executable
 --startup-file={yes|no}   Load ~/.juliarc.jl
 --handle-signals={yes|no} Enable or disable Julia's default signal handlers

 -e, --eval <expr>         Evaluate <expr>
 -E, --print <expr>        Evaluate and show <expr>
 -L, --load <file>         Load <file> immediately on all processors

--compile={yes|no|all|min}Enable or disable JIT compiler, or request exhaustive compilation
-C, --cpu-target <target> Limit usage of cpu features up to <target>
-O, --optimize={0,1,2,3}  Set the optimization level (default 2 if unspecified or 3 if specified as -O)
--inline={yes|no}         Control whether inlining is permitted (overrides functions declared as @inline)
--check-bounds={yes|no}   Emit bounds checks always or never (ignoring declarations)
--math-mode={ieee,fast}   Disallow or enable unsafe floating point optimizations (overrides @fastmath declaration)

--depwarn={yes|no|error}  Enable or disable syntax and method deprecation warnings ("error" turns warnings into errors)

--output-o name           Generate an object file (including system image data)
--output-ji name          Generate a system image data file (.ji)
--output-bc name          Generate LLVM bitcode (.bc)
--output-incremental=no   Generate an incremental output file (rather than complete)

--code-coverage={none|user|all}, --code-coverage
                          Count executions of source lines (omitting setting is equivalent to "user")
--track-allocation={none|user|all}, --track-allocation
                          Count bytes allocated by each source line
```

Both?

=> Issue #15864: separate `julia-compile` from `julia`

Obligatory Word-cloud

Mini Julia

> Is there some easy hackable mini-version of Julia available? #16798

Not easily. Julia isn’t designed to be a “toy” language.

It is possible to simulate parts of Julia inside Julia:

https://github.com/JuliaLang/julia/blob/master/examples/juliatypes.jl : See Jeff’s JuliaCon 2015 talk

But Julia is easily hackable! Let’s go…

Main.Core

```
~/julia$ echo > nothing.jl

~/julia$ ./julia --output-ji minicore.ji \
> nothing.jl
ERROR: could not open file boot.jl

~/julia$ cd base
~/julia/base$ ../julia --output-ji minicore.ji \
> ../nothing.jl

~/julia/base$ ls -lh minicore.ji
-rw-r--r-- jameson 33K Jun 14 14:54 minicore.ji
```

REPL?

```
~/julia/base$ ../julia -J minicore.ji
WARNING: Base._start not defined, falling
back to economy mode repl.

julia> Core.print(“Hello world\n")
Hello world
nothing

julia>
```

REPL?

```
~/julia/base$ ../julia -J minicore.ji
WARNING: Base._start not defined, falling
back to economy mode repl.

julia> 1 + 1
error during run:
UndefVarError(var=:+)

julia> Base
error during run:
UndefVarError(var=:Base)
```

Going beyond Core

```
~/julia/base$ ../julia --output-ji inference.ji coreimg.jl

~/julia/base$ ../julia --output-o sys.o --sysimage inference.ji sysimg.jl

~/julia/base$ cc -shared -o sys.so sys.o -ljulia -L ../usr/lib

~/julia/base$ ../julia --sysimage sys.so
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: http://docs.julialang.org
   _ _   _| |_  __ _   |  Type "?help" for help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.5.0-dev+4711 (2016-06-14 03:43 UTC)
 _/ |\__'_|_|_|\__'_|  |  jn/demo/09e75d4* (fork: 4 commits, 1 day)
|__/                   |  x86_64-apple-darwin15.5.0

julia>
```

Deeper Inspection

But what did we just do?

–> Julia as a compiler for writing Julia programs in Julia

–> Static output with dynamic semantics

Overview: Translation stages

- ./julia (repl.c)

- eval/include (builtins.c)
  –> parser (julia-parser.scm)
    –> run macros (ast.c)
      –> lowered form (julia-syntax.scm)
        –> execute toplevel expressions (toplevel.c)
          –> call functions (gf.c)
            –> dispatch (typemap.c)
              –> interpreter (interpreter.c) OR
                 compiler (codegen.cpp)
                   + optimizations (inference.jl)

For details, see Jeff's talk from JuliaCon 2014.

Overview: Standard Library

- ./julia (repl.c)
  –> getopt parsing
  –> loads the standard library

- libjulia.so (or .dll or .dylib)
  –> Garbage Collector / Allocator
  –> Builtin Functions / Intrinsics
  –> Interpreter
  –> Builtin Types
  –> Type System
  –> Dispatch System
  –> Compiler
  –> Platform abstraction / integration
  –> System Image / serializer

- sys.so (and other lib/julia/ files *.so and *.ji)
  –> Core module
  –> Inference module
  –> Base module

- Packages

- Dependencies
  –> LLVM, BLAS, libgit2, etc.

Minimal ‘main(void)’

```c
// This file (repl.c) is a part of Julia. License is MIT: http://julialang.org/license

// Standard headers
#include <string.h>
#include <stdint.h>
// Julia headers (for initialization and gc commands)
#include "uv.h"
#include "julia.h"

int main(int argc, char *argv[])
{
    // Initialize Julia
    uv_setup_args(argc, argv);
    libsupport_init();
    // Setup any runtime options here
    jl_options.fast_math = JL_OPTIONS_FAST_MATH_OFF; // --math-mode=ieee
    julia_init(JL_IMAGE_JULIA_HOME);
    // Run the REPL
    jl_function_t *start_client = (jl_function_t*)jl_get_global(jl_base_module, jl_symbol("_start"));
    (void)jl_apply(&start_client, 1);
    // Cleanup and graceful exit
    jl_atexit_hook(0);
    return 0;
}
```

This file is a part of Julia. License is MIT: http://julialang.org/license

More Standard Library

coreimg.jl

```
Main.Core.eval(Main.Core, :(baremodule Inference
using Core.Intrinsics
import Core: print, println, show, write, unsafe_write, STDOUT,
STDERR
ccall(:jl_set_istopmod, Void, (Bool,), false)

eval(x) = Core.eval(Inference,x)
eval(m,x) = Core.eval(m,x)

include = Core.include

## Load essential files and libraries
include("essentials.jl")
include("generator.jl")
include("reflection.jl")
include("options.jl")

# core operations & types
typealias Cint Int32
typealias Csize_t UInt
include("promotion.jl")
include("tuple.jl")
include("range.jl")
include("expr.jl")
include("error.jl")

# core numeric operations & types
include("bool.jl")
include("number.jl")
include("int.jl")
include("operators.jl")
include("pointer.jl")
const checked_add = +
const checked_sub = -
if !isdefined(Main, :Base)
    # conditional to allow redefining Core.Inference after base exists
```

sysimg.jl

```
baremodule Base
using Core.Intrinsics
ccall(:jl_set_istopmod, Void, (Bool,), true)
include = Core.include
include("coreio.jl")

eval(x) = Core.eval(Base,x)
eval(m,x) = Core.eval(m,x)

include("exports.jl")

## Load essential files and libraries
include("essentials.jl")
include("base.jl")
include("generator.jl")
include("reflection.jl")
include("options.jl")

# core operations & types
include("promotion.jl")
include("tuple.jl")
include("range.jl")
include("expr.jl")
include("error.jl")

# core numeric operations & types
include("bool.jl")
include("number.jl")
include("int.jl")
include("operators.jl")
include("pointer.jl")
include("refpointer.jl")
(::Type{T}){T}(arg) = convert(T, arg)::T
include("checked.jl")
importall .Checked
```

But not done yet…

userimg.jl

`--compile-incremental=yes`

using MyPackage
–> Base.require(:MyPackage)
  –> isdefined(Main, :MyPackage) ?
    –> Base._include_from_serialized or Base.include
      –> Base.cachecompile(:MyPackage)
        –> Recursion!
      –> MyPackage.__init__( )
–> symbols are imported from MyPackage

Deployment

Future work in progress.

BuildExecutable.jl and contrib/build_sysimg.jl

int main(int argc, char *argv[ ]);

Output C code

Bundling incremental compile into application

Installer tarball

Project.generate( )

Profiling

```
~/julia/test$ ../julia runtests.jl core
     * core
    SUCCESS
REMAINDER                    :  8.50%    5312822108
GC                           : 13.15%    8218142751
LOWERING                     : 31.01%   19385805839
PARSING                      :  0.73%     456448707
INFERENCE                    : 11.16%    6975897589
CODEGEN                      :  1.28%     799324355
METHOD_LOOKUP_SLOW           :  0.15%      93144624
METHOD_LOOKUP_FAST           :  5.70%    3566210459
LLVM_OPT                     : 25.35%   15844106508
LLVM_MODULE_FINISH           :  0.86%     534538848
METHOD_LOOKUP_COMPILE        :  0.02%      10894604
TYPE_LOOKUP                  :  2.10%    1315693342
```

Memory Usage

```
$ size -A sys.dylib
sys.dylib  :
section                   size        addr
__text                 4382405        3920
__stubs                    366     4386326
__stub_helper              626     4386692
__const                26109544    4387328
__cstring                 13108    30496880
__unwind_info             68376    30509988
__eh_frame              1251632    30578368
__nl_symbol_ptr              16    31830016
__got                        64    31830032
__la_symbol_ptr             488    31830096
__const                  406232    31830592
__bss                    167488    32236824
Total                  32400345
```

```
~/julia/usr/lib$ size -A libjulia.dylib
libjulia.dylib  :
section                   size        addr
__text                 1098861        3424
__stubs                   4350     1102286
__stub_helper             7166     1106636
__cstring                57318     1113808
__const                 240682     1171136
__unwind_info             5416     1411820
__eh_frame              110560     1417240
__nl_symbol_ptr             16     1527808
__got                      488     1527824
__la_symbol_ptr           5800     1528312
__mod_init_func             32     1534112
__mod_term_func             16     1534144
__const                 327272     1534160
__data                  219760     1861440
__thread_vars               24     2081200
__thread_bss              2368     2081224
__bss                   806656     2083600
__common               1346169     2890256
Total                  4232954
```

POWER8 Support

https://www-356.ibm.com/partnerworld/wps/ent/pdp/web/MyProgramAccess

Future Work

Disable JIT

Better generic code

Incremental linking

Implicit main function support

Transparent shlib embedding

System image compression / “strip”

Extract much of Base into a standard package library

Ports: webassembly, microkernel, embedded

Fix #265 (but disable fix for release code / static compile?)

Further addressing package load times?

Native code caching archive database

Eliminate Core.Intrinsics (move to all BuiltinFunctions)

Questions?

http://juliacomputing.com/blog/2016/02/09/static-julia.html