Introduction to julia

Oliver Schulz
Max Planck Institute for Physics
oschulz@mpp.mpg.de

MAX-PLANCK-GESELLSCHAFT

Max-Planck-Institut für Physik
(Werner-Heisenberg-Institut)

September 2021

Course material

Git-clone or download the course material:

https://github.com/oschulz/julia-course

Why Julia?

Science needs code - but how to write it?

- Choice of programming language(s) matter!
- Need to balance:
  - Learning time
  - Productivity
  - Performance
- Usually involves compromises

Programming Language Options

- C++:
  - Pro: Very fast (in expert hands)
  - Pro: Really cool new concepts (even literally) in C++11/14/17/...
  - Con: Complex, takes long time to learn and much longer to master
  - Con: Straightforward tasks often result in lengthy code
  - Con: No memory management (General protection faults)
  - Con: No universal package management
  - Con: Composability isn't great

Programming Language Options

- Python:
  - Pro: Broad user base, popular first programming language
  - Pro: Easy to learn, good standard library
  - Con: Can't write time-critical loops in Python, workarounds like Numba/Cython have many limitations, don't compose well
  - Con: Language itself fairly primitive, not very expressive
  - Con: Duck-Typing necessitates lots of test code
  - Con: No effective multi-threading
  - Con: Composability isn't great

What else is there?

- Fortran:
  - Pro: Math can be really fast
  - Con: Old language, few modern concepts
  - Con: Shrinking user base
  - Con: Composability isn't great
  - Do you *really* want to ...?
- Scala, Go, Kotlin etc.:
  - Pro: Lots of individual strengths
  - Con: Math either fast *or* generic *or* or complicated
  - Con: Calling C, Fortran or Phython code often difficult
  - Con: Composability isn't great

The 97 and the 3 Percent

> *We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.*

Donald E. Knuth

- Some programming languages (e.g. Python) great for the 97% - but can't make the 3% fast.
- Some other languages (e.g. C/C++, Fortran) can handle the 3% - but makes the 97% complicated.

The Two-language Problem

- Common approach nowadays:
  Write time critical code in C/C++, rest in Python

- Pro: End-user can code comfortably in Python, with good performance

- Con: Complexity of C/C++ **plus** complexity of Python

- Con: Need proficiency in **two** languages, barrier that prevents
  non-expert users from contributing to important parts of code

- Con: Limits generic implementation of algorithms

- Con: Severely limits metaprogramming, automatic differentiation, etc.

The Expression Problem

> *The expression problem is a new name for an old problem. The goal is to define a datatype by cases, where one can add new cases to the datatype and new functions over the datatype, without recompiling existing code, and while retaining static type safety (e.g., no casts).*

Philip Wadler

- In other words: The capability to add both new subtypes and new functionality for a type defined in a package you don't own
- Object oriented languages typically can't do this
  (Ruby has a dirty way, Scala a clean workaround)
- If you have programming experience, you have felt this, even if you didn't name it
- Result: Packages tend not to compose well

We were looking for a language ...

- as fast as C/C++/Fortran
- as easy to learn and productive as Python
- with a solution for the expression problem
- with first class math support (vectors, matrices, etc.)
- with true functional programming
- with great Fortran/C/C++/Python integration
- with true metaprogramming (like Lisp or Scala)
- good at parallel and distributed programming
- suitable for for interactive, small and large applications

Julia

- Designed for scientific/technical computing
- Originated at MIT, first public version 2012
- Covers the whole wish-list
- Clear focus on user productivity and software quality
- Rapid growth of user base and software packages
- Current version: Julia v1.5 (v1.6 will be out soon)

Julia Language Properties

- Fast: JAOT compilation to native CPU and GPU code
- Multiple-dispatch (more powerful than object-oriented): solves the expression problem
- Dynamically typed
- Very powerful type system, types are first-class values
- Functional programming and metaprogramming
- First-class math support (like Fortran or Matlab)
- ...

Julia Language Properties, cont.

- ...
- Local and distributed code execution
- State-of-the-art multi-threading: parallel code can call parallel code that can call parallel code, ..., without oversubscribing threads
- Software package management: Trivial to create and install packages
- Excellent REPL (console)
- Easy to call Fortran, C/C++ and Python code

Julia large-scale use case examples

- Celeste: Variational Bayesian inference for astronomical images (doi:10.1214/19-AOAS1258), 1.54 petaflops using 1.3 million threads on 9,300 Knights Landing (KNL) nodes on Cori at NERSC
- Clima: Full-earth climate simulation, https://clima.caltech.edu, large team, uses everything from MPI to GPUs
- ...

When (not) to use Julia

- *Do* use Julia for computations, visualization, data processing ... pretty much anything scientific/technical
- *Do not* use Julia for scripts what will only run for a second (code gen overhead), use Python or shell scripts
- *Do not* use Julia for non-computing web apps, etc. (*at least not yet*), use Go or Node.js

Julia 101

Verbs and nouns - functions and types

- Julia is not Java: Verbs aren't owned by nouns
- Julia has: types, functions and methods
- Methods belong to *functions*, not to types!

Functions

Short one-liner function:

In [4]:

```
f(x) = x^2
```

Out[4]:

```
f (generic function with 1 method)
```

In [5]:

```
f(3)
```

Out[5]:

```
9
```

Function that needs more than one line:

In [6]:

```
function f(x)
    # ... something ...
    x^2
end
```

Out[6]:

```
f (generic function with 1 method)
```

is equivalent to

In [7]:

```
function f(x)
    # ... something ...
    return x^2
end
```

Out[7]:

```
f (generic function with 1 method)
```

Note: `return` is optional, and often not used explicitly. Last expression in a function, block, etc. is automatically returned (like in Mathematica).

Types

An abstract type, must be empty:

```julia
abstract type MySuperType end
```

An immutable type, value of `i` can't change:

```julia
struct MySubType <: MySuperType
    i::Int
end
```

A mutable type, value of `i` can change:

```julia
mutable struct MyMutableSubType <: MySuperType
    i::Int
end
```

Type parameters

Julia has a powerful parametric type system, based on set theory:

```
struct MyRealArray{T<:Real,N} <: AbstractArray{T,N}
    # ...
end
```

defines an array type with real-valued elements.

```
foo(A::AbstractArray{<:Real}) = do_something_with(A)
```

is equivalent to

```
bar(A::AbstractArray{T,N}) where {T<:Real,N} = do_something_with(A)
```

defines a function covariant in the element type of `A`. Can also be contravariant:

```
baz(A::AbstractArray{>:Real}) = do_something_with(A)
```

Type aliases and union types

Type aliases are just const values:

```
const Abstract2DArray{T} = AbstractArray{T,2}
rand(2, 2) isa Abstract2DArray == true
```

Type unions are unions of set of types.

```
const RealVecOrMat{T} where {T<:Real} = Union{AbstractArray{T,1},
AbstractArray{T,2}}
```

is the union of a 1D and 2D array types with real-valued elements.

Syntax: Variables

```julia
# Global variables:
const a = 42
b = 24

function foo(x)
    # Local variables:

    c = a * x
    d = b * x # Avoid, type of b can change!
    #...
end
```

Loops

For loop:

```julia
for i in something_iterable
    # ...
end
```

`something_iterable` can be a range, an array, anything that implements the Julia iterator API.

While loop:

```julia
while condition
    # do something
end
```

Control flow

If-else, evaluate only one branch:

```
if condition
    # do something
elseif condition
    # do something else
else
    # or something different
end
```

Ternary operator, evaluate only one branch:

```
condition ? result_if_true : result_if_false
```

`ifelse`, evaluate both results but return only one:

```
ifelse(condition, result_if_true, result_if_false)
```

Blocks and scoping

Begin/end-block:

```julia
begin
    # *Not* a new scope in here
    # ...
end
```

Let-block:

```julia
b = 24

let my_b = b
    # New scope in here.
    # If b is bound to new value, my_b won't change.
    # ...
end
```

Arrays

Vectors:

```
v = [1, 2, 3]
```

```
v = rand(5)
```

Matrices:

```
A = [1 2; 3 4]
```

```
A = rand(4, 5)
```

- Column-first memory layout!
- Almost anything array-like is subtype of `AbstractArray`.

Array indexing

Get `i`-th element of vector `v`:

```
v[i]
```

Most higher-dimensional array types support cartesian and linear indexing (usually faster):

```
A[i, j]
A[lin_idx]
```

Use `eachindex(A)` to get indices of best type for given `A` (usually linear).

In Julia, anything array-like can usually be an index as well

```
A[2:3, [1, 4, 5]]
```

Array comprehension and generators

Returns an array:

```
[f(x) for x in some_collection]
```

Returns an iterable generator:

```
(f(x) for x in some_collection)
```

Hello World (and more) in Julia

In [8]:

```
println("Hello, World!")
```

```
Hello, World!
```

Hello World (and more) in Julia

In [8]:

```
println("Hello, World!")
```

```
Hello, World!
```

Let's define a function

In [9]:

```
f(x, y) = x * y
f(20, 2.1)
```

Out[9]:

```
42.0
```

Multiplication is also defined for vectors, so this works, too:

In [10]:

```
f(4.2, [1, 2, 3, 4])
```

Out[10]:

```
4-element Vector{Float64}:
  4.2
  8.4
 12.600000000000001
 16.8
```

Let's Look Under the Hood

In [11]:

```
@code_llvm debuginfo=:none f(20, 2.1)
```

```
define double @julia_f_1769(i64 signext %0, double %1) {
top:
  %2 = sitofp i64 %0 to double
  %3 = fmul double %2, %1
  ret double %3
}
```

In [12]:

```
@code_native debuginfo=:none f(20, 2.1)
```

```
        .text
        vcvtsi2sd       %rdi, %xmm1, %xmm1
        vmulsd  %xmm0, %xmm1, %xmm0
        retq
        nopw    (%rax,%rax)
```

Multiple Dispatch

In [13]:
```
foo(x::Integer, y::Number) = x * y
foo(x::Integer, y::AbstractString) = join(fill(y, x))
```

Out[13]:
```
foo (generic function with 2 methods)
```

In [14]:
```
foo(3, 4)
```

Out[14]:
```
12
```

In [15]:
```
foo(3, "abc")
```

Out[15]:
```
"abcabcabc"
```

In [16]:
```
foo(4.5, 3)
```

```
MethodError: no method matching foo(::Float64, ::Int64)
Closest candidates are:
  foo(::Integer, ::Number) at In[13]:1

Stacktrace:
 [1] top-level scope
   @ In[16]:1
 [2] eval
   @ ./boot.jl:360 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::St
```

```
ring, filename::String)
   @ Base ./loading.jl:1116
```

Functional Programming

In [17]:
```
A = rand(10)
```

Out[17]:
```
10-element Vector{Float64}:
 0.6371208023031867
 0.671824106672500l
 0.474889305066802б
 0.42663698605676403
 0.7906983796911875
 0.12541113346976673
 0.45340111998340027
 0.47037648329103954
 0.776373809609l002
 0.857030414638802l
```

In [18]:
```
idxs = findall(x -> 0.2 < x < 0.6, A)
```

Out[18]:
```
4-element Vector{Int64}:
 3
 4
 7
 8
```

In [19]:
```
A[idxs]
```

Out[19]:
```
4-element Vector{Float64}:
 0.474889305066802б
 0.42663698605676403
```

0.45340111998340027
0.47037648329103954

Even types are first-class values:

In [20]:
```
mytype = Number
```

Out[20]:
```
Number
```

In [21]:
```
subtypes(mytype)
```

Out[21]:
```
2-element Vector{Any}:
 Complex
 Real
```

Julia type hierarchy extends all the way down to primitive types:

In [22]:
```
Float64 <: AbstractFloat <: Real <: Number <: Any
```

Out[22]:
```
true
```

Broadcasting

In [23]:

```
A = [1.1, 2.2, 3.3]
B = [4.4, 5.5, 6.6]
broadcast((x, y) -> (x + y)^2, A, B)
```

Out[23]:

```
3-element Vector{Float64}:
 30.25
 59.290000000000006
 98.00999999999998
```

Shorter broadcast syntax:

In [24]:

```
(A .+ B) .^ 2
```

Out[24]:

```
3-element Vector{Float64}:
 30.25
 59.290000000000006
 98.00999999999998
```

Loop Fusion and SIMD Vectorization

In [25]:

```julia
foo(X, Y) = (X .+ Y) .^ 2
@code_llvm raw=false debuginfo=:none foo(A, B)
```

```llvm
define nonnull {}* @japi1_foo_2441({}* %0, {}** %1, i32 %2) #0 {
top:
  %3 = alloca [4 x {}*], align 8
  %gcframe173 = alloca [4 x {}*], align 16
  %gcframe173.sub = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe1
73, i64 0, i64 0
  %.sub = getelementptr inbounds [4 x {}*], [4 x {}*]* %3, i64 0, i64 0
  %4 = bitcast [4 x {}*]* %gcframe173 to i8*
  call void @llvm.memset.p0i8.i32(i8* nonnull align 16 dereferenceable(32)
%4, i8 0, i32 32, i1 false)
  %5 = alloca {}**, align 8
  store volatile {}** %1, {}*** %5, align 8
  %6 = alloca [1 x [1 x i64]], align 8
  %7 = alloca [1 x [1 x i64]], align 8
  %thread_ptr = call i8* asm "movq %fs:0, $0", "=r"() #8
  %ptls_i8 = getelementptr i8, i8* %thread_ptr, i64 -32768
  %8 = bitcast [4 x {}*]* %gcframe173 to i64*
  store i64 8, i64* %8, align 16
  %9 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe173, i64 0, i6
4 1
  %10 = bitcast i8* %ptls_i8 to i64*
  %11 = load i64, i64* %10, align 8
  %12 = bitcast {}** %9 to i64*
  store i64 %11, i64* %12, align 8
  %13 = bitcast i8* %ptls_i8 to {}***
  store {}** %gcframe173.sub, {}*** %13, align 8
  %14 = load {}*, {}** %1, align 8
```

```
  %15 = getelementptr inbounds {}*, {}** %1, i64 1
  %16 = load {}*, {}** %15, align 8
  %17 = bitcast {}* %14 to {}**
  %18 = getelementptr inbounds {}*, {}** %17, i64 3
  %19 = bitcast {}** %18 to i64*
  %20 = load i64, i64* %19, align 8
  %21 = bitcast {}* %16 to {}**
  %22 = getelementptr inbounds {}*, {}** %21, i64 3
  %23 = bitcast {}** %22 to i64*
  %24 = load i64, i64* %23, align 8
  switch i64 %24, label %L20 [
    i64 0, label %L13
    i64 1, label %L17
  ]

L13:                                              ; preds = %top
  %25 = icmp eq i64 %20, 0
  br label %L23

L17:                                              ; preds = %top
  %26 = icmp eq i64 %20, 1
  br label %L23

L20:                                              ; preds = %top
  %27 = icmp eq i64 %24, %20
  br label %L23

L23:                                              ; preds = %L20, %L17, %L
13
  %value_phi.in = phi i1 [ %25, %L13 ], [ %26, %L17 ], [ %27, %L20 ]
  %28 = icmp eq i64 %20, 1
  %value_phi1.v = or i1 %28, %value_phi.in
  br i1 %value_phi1.v, label %L57, label %L31
```

```
L31:                                                    ; preds = %L23
  switch i64 %20, label %L40 [
    i64 0, label %L33
    i64 1, label %L37
  ]

L33:                                                    ; preds = %L31
  %29 = icmp eq i64 %24, 0
  br label %L43

L37:                                                    ; preds = %L31
  %30 = icmp eq i64 %24, 1
  br label %L43

L40:                                                    ; preds = %L31
  %31 = icmp eq i64 %20, %24
  br label %L43

L43:                                                    ; preds = %L40, %L37, %L
33
  %value_phi20.in = phi i1 [ %29, %L33 ], [ %30, %L37 ], [ %31, %L40 ]
  %32 = icmp eq i64 %24, 1
  %value_phi21.v = or i1 %32, %value_phi20.in
  br i1 %value_phi21.v, label %L57, label %L51

L51:                                                    ; preds = %L43
  %33 = call noalias nonnull {}* @jl_gc_pool_alloc(i8* %ptls_i8, i32 1400,
i32 16) #1
  %34 = bitcast {}* %33 to i64*
  %35 = getelementptr inbounds i64, i64* %34, i64 -1
  store atomic i64 140310350102144, i64* %35 unordered, align 8
  store i64 %20, i64* %34, align 8
  %36 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe173, i64 0, i
64 2
```

```
  store {}* %33, {}** %36, align 16
  store {}* %33, {}** %.sub, align 8
  %37 = call nonnull {}* @jl_apply_generic({}* inttoptr (i64 140310368871456 to {}*), {}** nonnull %.sub, i32 1)
  %38 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe173, i64 0, i64 3
  store {}* %37, {}** %38, align 8
  %39 = call noalias nonnull {}* @jl_gc_pool_alloc(i8* %ptls_i8, i32 1400, i32 16) #1
  %40 = bitcast {}* %39 to i64*
  %41 = getelementptr inbounds i64, i64* %40, i64 -1
  store atomic i64 140310350102144, i64* %41 unordered, align 8
  store i64 %24, i64* %40, align 8
  store {}* %39, {}** %36, align 16
  store {}* %39, {}** %.sub, align 8
  %42 = call nonnull {}* @jl_apply_generic({}* inttoptr (i64 140310368871456 to {}*), {}** nonnull %.sub, i32 1)
  store {}* %42, {}** %36, align 16
  store {}* inttoptr (i64 140310375323856 to {}*), {}** %.sub, align 8
  %43 = getelementptr inbounds [4 x {}*], [4 x {}*]* %3, i64 0, i64 1
  store {}* %37, {}** %43, align 8
  %44 = getelementptr inbounds [4 x {}*], [4 x {}*]* %3, i64 0, i64 2
  store {}* inttoptr (i64 140310375323952 to {}*), {}** %44, align 8
  %45 = getelementptr inbounds [4 x {}*], [4 x {}*]* %3, i64 0, i64 3
  store {}* %42, {}** %45, align 8
  %46 = call nonnull {}* @jl_apply_generic({}* inttoptr (i64 140310352358976 to {}*), {}** nonnull %.sub, i32 4)
  store {}* %46, {}** %36, align 16
  store {}* %46, {}** %.sub, align 8
  %47 = call nonnull {}* @jl_apply_generic({}* inttoptr (i64 140310417078240 to {}*), {}** nonnull %.sub, i32 1)
  call void @jl_throw({}* %47)
  unreachable
```

```
L57:                                              ; preds = %L43, %L23
  %.sroa.026.0 = phi i64 [ %24, %L23 ], [ %20, %L43 ]
  %48 = getelementptr inbounds [1 x [1 x i64]], [1 x [1 x i64]]* %6, i64 0
, i64 0, i64 0
  store i64 %.sroa.026.0, i64* %48, align 8
  %49 = call nonnull {}* inttoptr (i64 140310593210021 to {}* ({}*, i64)*)
({}* inttoptr (i64 140310355132272 to {}*), i64 %.sroa.026.0)
  %50 = bitcast {}* %49 to {}**
  %51 = getelementptr inbounds {}*, {}** %50, i64 3
  %52 = bitcast {}** %51 to i64*
  %53 = load i64, i64* %52, align 8
  switch i64 %53, label %L90 [
    i64 0, label %L80
    i64 1, label %L84
  ]

L80:                                              ; preds = %L57
  %54 = icmp eq i64 %.sroa.026.0, 0
  br i1 %54, label %L98, label %L223

L84:                                              ; preds = %L57
  %55 = icmp eq i64 %.sroa.026.0, 1
  br i1 %55, label %L98, label %L223

L90:                                              ; preds = %L57
  %56 = icmp eq i64 %53, %.sroa.026.0
  br i1 %56, label %L98, label %L223

L98:                                              ; preds = %L90, %L84, %L
80
  %.not34 = icmp eq {}* %49, %14
  br i1 %.not34, label %L124, label %L101

L101:                                             ; preds = %L98
```

```
  %57 = load i8, i8* inttoptr (i64 140310355132345 to i8*), align 1
  %58 = and i8 %57, 1
  %.not42.not = icmp eq i8 %58, 0
  br i1 %.not42.not, label %L107, label %L124

L107:                                             ; preds = %L101
  %59 = bitcast {}* %49 to i64*
  %60 = load i64, i64* %59, align 8
  %61 = bitcast {}* %14 to i64*
  %62 = load i64, i64* %61, align 8
  %.not58 = icmp eq i64 %60, %62
  br i1 %.not58, label %L119, label %L124

L119:                                             ; preds = %L107
  %63 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe173, i64 0, i
64 3
  store {}* %49, {}** %63, align 8
  %64 = call nonnull {}* inttoptr (i64 140310593210237 to {}* ({}*)*)({}*
nonnull %14)
  %.pre166 = bitcast {}* %64 to {}**
  br label %L124

L124:                                             ; preds = %L119, %L107,
 %L101, %L98
  %.pre-phi168 = phi {}** [ %17, %L107 ], [ %17, %L101 ], [ %17, %L98 ], [
%.pre166, %L119 ]
  %value_phi11 = phi {}* [ %14, %L107 ], [ %14, %L101 ], [ %14, %L98 ], [
 %64, %L119 ]
  %65 = getelementptr inbounds {}*, {}** %.pre-phi168, i64 3
  %66 = bitcast {}** %65 to i64*
  %67 = load i64, i64* %66, align 8
  %.not38 = icmp eq i64 %67, 1
  %.not35 = icmp eq {}* %49, %16
  br i1 %.not35, label %L159, label %L136
```

```
L136:                                             ; preds = %L124
  %68 = load i8, i8* inttoptr (i64 140310355132345 to i8*), align 1
  %69 = and i8 %68, 1
  %.not40.not = icmp eq i8 %69, 0
  br i1 %.not40.not, label %L142, label %L159

L142:                                             ; preds = %L136
  %70 = bitcast {}* %49 to i64*
  %71 = load i64, i64* %70, align 8
  %72 = bitcast {}* %16 to i64*
  %73 = load i64, i64* %72, align 8
  %.not = icmp eq i64 %71, %73
  br i1 %.not, label %L154, label %L159

L154:                                             ; preds = %L142
  %74 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe173, i64 0, i
64 3
  store {}* %49, {}** %74, align 8
  %75 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe173, i64 0, i
64 2
  store {}* %value_phi11, {}** %75, align 16
  %76 = call nonnull {}* inttoptr (i64 140310593210237 to {}* ({}*)*)({}*
nonnull %16)
  %.pre169 = bitcast {}* %76 to {}**
  br label %L159

L159:                                             ; preds = %L154, %L142,
 %L136, %L124
  %.pre-phi172 = phi {}** [ %21, %L142 ], [ %21, %L136 ], [ %21, %L124 ],
[ %.pre169, %L154 ]
  %value_phi12 = phi {}* [ %16, %L142 ], [ %16, %L136 ], [ %16, %L124 ], [
%76, %L154 ]
  %77 = getelementptr inbounds {}*, {}** %.pre-phi172, i64 3
```

```
  %78 = bitcast {}** %77 to i64*
  %79 = load i64, i64* %78, align 8
  %.not39 = icmp eq i64 %79, 1
  %.not36 = icmp eq i64 %.sroa.026.0, 0
  br i1 %.not36, label %L233, label %L185.lr.ph

L185.lr.ph:                                       ; preds = %L159
  %80 = bitcast {}* %value_phi11 to double**
  %81 = load double*, double** %80, align 8
  %bc116 = bitcast double* %81 to i8*
  %82 = bitcast {}* %value_phi12 to double**
  %83 = load double*, double** %82, align 8
  %bc = bitcast double* %83 to i8*
  %84 = bitcast {}* %49 to double**
  %85 = load double*, double** %84, align 8
  %86 = bitcast double* %85 to i8*
  %min.iters.check140 = icmp ult i64 %.sroa.026.0, 16
  br i1 %.not38, label %L185.lr.ph.split.us, label %L185.lr.ph.L185.lr.ph.
split_crit_edge

L185.lr.ph.L185.lr.ph.split_crit_edge:            ; preds = %L185.lr.ph
  br i1 %.not39, label %L185.us52.preheader, label %L185.preheader

L185.preheader:                                   ; preds = %L185.lr.ph.L1
85.lr.ph.split_crit_edge
  br i1 %min.iters.check140, label %scalar.ph, label %vector.memcheck

vector.memcheck:                                  ; preds = %L185.preheade
r
  %scevgep = getelementptr double, double* %85, i64 %.sroa.026.0
  %scevgep63 = getelementptr double, double* %81, i64 %.sroa.026.0
  %scevgep65 = getelementptr double, double* %83, i64 %.sroa.026.0
  %bound0 = icmp ult double* %85, %scevgep63
  %bound1 = icmp ult double* %81, %scevgep
```

```
  %found.conflict = and i1 %bound0, %bound1
  %bound067 = icmp ult double* %85, %scevgep65
  %bound168 = icmp ult double* %83, %scevgep
  %found.conflict69 = and i1 %bound067, %bound168
  %conflict.rdx = or i1 %found.conflict, %found.conflict69
  br i1 %conflict.rdx, label %scalar.ph, label %vector.ph

vector.ph:                                        ; preds = %vector.memche
ck
  %n.vec = and i64 %.sroa.026.0, 9223372036854775792
  br label %vector.body

vector.body:                                      ; preds = %vector.body,
 %vector.ph
  %index = phi i64 [ 0, %vector.ph ], [ %index.next, %vector.body ]
  %87 = getelementptr inbounds double, double* %81, i64 %index
  %88 = bitcast double* %87 to <4 x double>*
  %wide.load = load <4 x double>, <4 x double>* %88, align 8
  %89 = getelementptr inbounds double, double* %87, i64 4
  %90 = bitcast double* %89 to <4 x double>*
  %wide.load70 = load <4 x double>, <4 x double>* %90, align 8
  %91 = getelementptr inbounds double, double* %87, i64 8
  %92 = bitcast double* %91 to <4 x double>*
  %wide.load71 = load <4 x double>, <4 x double>* %92, align 8
  %93 = getelementptr inbounds double, double* %87, i64 12
  %94 = bitcast double* %93 to <4 x double>*
  %wide.load72 = load <4 x double>, <4 x double>* %94, align 8
  %95 = getelementptr inbounds double, double* %83, i64 %index
  %96 = bitcast double* %95 to <4 x double>*
  %wide.load73 = load <4 x double>, <4 x double>* %96, align 8
  %97 = getelementptr inbounds double, double* %95, i64 4
  %98 = bitcast double* %97 to <4 x double>*
  %wide.load74 = load <4 x double>, <4 x double>* %98, align 8
  %99 = getelementptr inbounds double, double* %95, i64 8
```

```
  %100 = bitcast double* %99 to <4 x double>*
  %wide.load75 = load <4 x double>, <4 x double>* %100, align 8
  %101 = getelementptr inbounds double, double* %95, i64 12
  %102 = bitcast double* %101 to <4 x double>*
  %wide.load76 = load <4 x double>, <4 x double>* %102, align 8
  %103 = fadd <4 x double> %wide.load, %wide.load73
  %104 = fadd <4 x double> %wide.load70, %wide.load74
  %105 = fadd <4 x double> %wide.load71, %wide.load75
  %106 = fadd <4 x double> %wide.load72, %wide.load76
  %107 = fmul <4 x double> %103, %103
  %108 = fmul <4 x double> %104, %104
  %109 = fmul <4 x double> %105, %105
  %110 = fmul <4 x double> %106, %106
  %111 = getelementptr inbounds double, double* %85, i64 %index
  %112 = bitcast double* %111 to <4 x double>*
  store <4 x double> %107, <4 x double>* %112, align 8
  %113 = getelementptr inbounds double, double* %111, i64 4
  %114 = bitcast double* %113 to <4 x double>*
  store <4 x double> %108, <4 x double>* %114, align 8
  %115 = getelementptr inbounds double, double* %111, i64 8
  %116 = bitcast double* %115 to <4 x double>*
  store <4 x double> %109, <4 x double>* %116, align 8
  %117 = getelementptr inbounds double, double* %111, i64 12
  %118 = bitcast double* %117 to <4 x double>*
  store <4 x double> %110, <4 x double>* %118, align 8
  %index.next = add i64 %index, 16
  %119 = icmp eq i64 %index.next, %n.vec
  br i1 %119, label %middle.block, label %vector.body

middle.block:                                     ; preds = %vector.body
  %cmp.n = icmp eq i64 %.sroa.026.0, %n.vec
  br i1 %cmp.n, label %L233, label %scalar.ph

scalar.ph:                                        ; preds = %middle.block,
```

```
%vector.memcheck, %L185.preheader
  %bc.resume.val = phi i64 [ %n.vec, %middle.block ], [ 0, %L185.preheader
], [ 0, %vector.memcheck ]
  br label %L185

L185.us52.preheader:                                  ; preds = %L185.lr.ph.L1
85.lr.ph.split_crit_edge
  br i1 %min.iters.check140, label %scalar.ph78, label %vector.memcheck94

vector.memcheck94:                                    ; preds = %L185.us52.pre
header
  %scevgep82 = getelementptr double, double* %85, i64 %.sroa.026.0
  %scevgep84 = getelementptr double, double* %81, i64 %.sroa.026.0
  %uglygep = getelementptr i8, i8* %bc, i64 1
  %bound086 = icmp ult double* %85, %scevgep84
  %bound187 = icmp ult double* %81, %scevgep82
  %found.conflict88 = and i1 %bound086, %bound187
  %bound089 = icmp ugt i8* %uglygep, %86
  %bound190 = icmp ult double* %83, %scevgep82
  %found.conflict91 = and i1 %bound089, %bound190
  %conflict.rdx92 = or i1 %found.conflict88, %found.conflict91
  br i1 %conflict.rdx92, label %scalar.ph78, label %vector.ph95

vector.ph95:                                          ; preds = %vector.memche
ck94
  %n.vec97 = and i64 %.sroa.026.0, 9223372036854775792
  %.pre = load double, double* %83, align 8
  br label %vector.body79

vector.body79:                                        ; preds = %vector.body7
9, %vector.ph95
  %index98 = phi i64 [ 0, %vector.ph95 ], [ %index.next99, %vector.body79
]
  %120 = getelementptr inbounds double, double* %81, i64 %index98
```

```
  %121 = bitcast double* %120 to <4 x double>*
  %wide.load102 = load <4 x double>, <4 x double>* %121, align 8
  %122 = getelementptr inbounds double, double* %120, i64 4
  %123 = bitcast double* %122 to <4 x double>*
  %wide.load103 = load <4 x double>, <4 x double>* %123, align 8
  %124 = getelementptr inbounds double, double* %120, i64 8
  %125 = bitcast double* %124 to <4 x double>*
  %wide.load104 = load <4 x double>, <4 x double>* %125, align 8
  %126 = getelementptr inbounds double, double* %120, i64 12
  %127 = bitcast double* %126 to <4 x double>*
  %wide.load105 = load <4 x double>, <4 x double>* %127, align 8
  %128 = insertelement <4 x double> undef, double %.pre, i32 0
  %129 = shufflevector <4 x double> %128, <4 x double> undef, <4 x i32> ze
roinitializer
  %130 = fadd <4 x double> %wide.load102, %129
  %131 = fadd <4 x double> %wide.load103, %129
  %132 = fadd <4 x double> %wide.load104, %129
  %133 = fadd <4 x double> %wide.load105, %129
  %134 = fmul <4 x double> %130, %130
  %135 = fmul <4 x double> %131, %131
  %136 = fmul <4 x double> %132, %132
  %137 = fmul <4 x double> %133, %133
  %138 = getelementptr inbounds double, double* %85, i64 %index98
  %139 = bitcast double* %138 to <4 x double>*
  store <4 x double> %134, <4 x double>* %139, align 8
  %140 = getelementptr inbounds double, double* %138, i64 4
  %141 = bitcast double* %140 to <4 x double>*
  store <4 x double> %135, <4 x double>* %141, align 8
  %142 = getelementptr inbounds double, double* %138, i64 8
  %143 = bitcast double* %142 to <4 x double>*
  store <4 x double> %136, <4 x double>* %143, align 8
  %144 = getelementptr inbounds double, double* %138, i64 12
  %145 = bitcast double* %144 to <4 x double>*
  store <4 x double> %137, <4 x double>* %145, align 8
```

```
  %index.next99 = add i64 %index98, 16
  %146 = icmp eq i64 %index.next99, %n.vec97
  br i1 %146, label %middle.block77, label %vector.body79

middle.block77:                                   ; preds = %vector.body79
  %cmp.n101 = icmp eq i64 %.sroa.026.0, %n.vec97
  br i1 %cmp.n101, label %L233, label %scalar.ph78

scalar.ph78:                                      ; preds = %middle.block7
7, %vector.memcheck94, %L185.us52.preheader
  %bc.resume.val100 = phi i64 [ %n.vec97, %middle.block77 ], [ 0, %L185.us
52.preheader ], [ 0, %vector.memcheck94 ]
  br label %L185.us52

L185.lr.ph.split.us:                              ; preds = %L185.lr.ph
  br i1 %.not39, label %L185.us.us.preheader, label %L185.us.preheader

L185.us.preheader:                                ; preds = %L185.lr.ph.sp
lit.us
  br i1 %min.iters.check140, label %scalar.ph107, label %vector.memcheck12
5

vector.memcheck125:                               ; preds = %L185.us.prehe
ader
  %scevgep111 = getelementptr double, double* %85, i64 %.sroa.026.0
  %uglygep113 = getelementptr i8, i8* %bc116, i64 1
  %scevgep114 = getelementptr double, double* %83, i64 %.sroa.026.0
  %bound0117 = icmp ugt i8* %uglygep113, %86
  %bound1118 = icmp ult double* %81, %scevgep111
  %found.conflict119 = and i1 %bound0117, %bound1118
  %bound0120 = icmp ult double* %85, %scevgep114
  %bound1121 = icmp ult double* %83, %scevgep111
  %found.conflict122 = and i1 %bound0120, %bound1121
  %conflict.rdx123 = or i1 %found.conflict119, %found.conflict122
```

```
  br i1 %conflict.rdx123, label %scalar.ph107, label %vector.ph126

vector.ph126:                                     ; preds = %vector.memche
ck125
  %n.vec128 = and i64 %.sroa.026.0, 9223372036854775792
  %.pre165 = load double, double* %81, align 8
  br label %vector.body108

vector.body108:                                   ; preds = %vector.body10
8, %vector.ph126
  %index129 = phi i64 [ 0, %vector.ph126 ], [ %index.next130, %vector.body
108 ]
  %147 = insertelement <4 x double> undef, double %.pre165, i32 0
  %148 = shufflevector <4 x double> %147, <4 x double> undef, <4 x i32> ze
roinitializer
  %149 = getelementptr inbounds double, double* %83, i64 %index129
  %150 = bitcast double* %149 to <4 x double>*
  %wide.load133 = load <4 x double>, <4 x double>* %150, align 8
  %151 = getelementptr inbounds double, double* %149, i64 4
  %152 = bitcast double* %151 to <4 x double>*
  %wide.load134 = load <4 x double>, <4 x double>* %152, align 8
  %153 = getelementptr inbounds double, double* %149, i64 8
  %154 = bitcast double* %153 to <4 x double>*
  %wide.load135 = load <4 x double>, <4 x double>* %154, align 8
  %155 = getelementptr inbounds double, double* %149, i64 12
  %156 = bitcast double* %155 to <4 x double>*
  %wide.load136 = load <4 x double>, <4 x double>* %156, align 8
  %157 = fadd <4 x double> %148, %wide.load133
  %158 = fadd <4 x double> %148, %wide.load134
  %159 = fadd <4 x double> %148, %wide.load135
  %160 = fadd <4 x double> %148, %wide.load136
  %161 = fmul <4 x double> %157, %157
  %162 = fmul <4 x double> %158, %158
  %163 = fmul <4 x double> %159, %159
```

```
  %164 = fmul <4 x double> %160, %160
  %165 = getelementptr inbounds double, double* %85, i64 %index129
  %166 = bitcast double* %165 to <4 x double>*
  store <4 x double> %161, <4 x double>* %166, align 8
  %167 = getelementptr inbounds double, double* %165, i64 4
  %168 = bitcast double* %167 to <4 x double>*
  store <4 x double> %162, <4 x double>* %168, align 8
  %169 = getelementptr inbounds double, double* %165, i64 8
  %170 = bitcast double* %169 to <4 x double>*
  store <4 x double> %163, <4 x double>* %170, align 8
  %171 = getelementptr inbounds double, double* %165, i64 12
  %172 = bitcast double* %171 to <4 x double>*
  store <4 x double> %164, <4 x double>* %172, align 8
  %index.next130 = add i64 %index129, 16
  %173 = icmp eq i64 %index.next130, %n.vec128
  br i1 %173, label %middle.block106, label %vector.body108

middle.block106:                                  ; preds = %vector.body10
8
  %cmp.n132 = icmp eq i64 %.sroa.026.0, %n.vec128
  br i1 %cmp.n132, label %L233, label %scalar.ph107

scalar.ph107:                                     ; preds = %middle.block1
06, %vector.memcheck125, %L185.us.preheader
  %bc.resume.val131 = phi i64 [ %n.vec128, %middle.block106 ], [ 0, %L185.
us.preheader ], [ 0, %vector.memcheck125 ]
  br label %L185.us

L185.us.us.preheader:                             ; preds = %L185.lr.ph.sp
lit.us
  br i1 %min.iters.check140, label %scalar.ph138, label %vector.memcheck15
6

vector.memcheck156:                               ; preds = %L185.us.us.pr
```

```
eheader
  %scevgep142 = getelementptr double, double* %85, i64 %.sroa.026.0
  %uglygep144 = getelementptr i8, i8* %bc116, i64 1
  %uglygep145 = getelementptr i8, i8* %bc, i64 1
  %bound0147 = icmp ugt i8* %uglygep144, %86
  %bound1148 = icmp ult double* %81, %scevgep142
  %found.conflict149 = and i1 %bound0147, %bound1148
  %bound0151 = icmp ugt i8* %uglygep145, %86
  %bound1152 = icmp ult double* %83, %scevgep142
  %found.conflict153 = and i1 %bound0151, %bound1152
  %conflict.rdx154 = or i1 %found.conflict149, %found.conflict153
  br i1 %conflict.rdx154, label %scalar.ph138, label %vector.ph157

vector.ph157:                                     ; preds = %vector.memche
ck156
  %n.vec159 = and i64 %.sroa.026.0, 9223372036854775792
  br label %vector.body139

vector.body139:                                   ; preds = %vector.body13
9, %vector.ph157
  %index160 = phi i64 [ 0, %vector.ph157 ], [ %index.next161, %vector.body
139 ]
  %174 = load double, double* %81, align 8
  %175 = insertelement <4 x double> undef, double %174, i32 0
  %176 = load double, double* %83, align 8
  %177 = insertelement <4 x double> undef, double %176, i32 0
  %178 = fadd <4 x double> %175, %177
  %179 = fmul <4 x double> %178, %178
  %180 = shufflevector <4 x double> %179, <4 x double> undef, <4 x i32> ze
roinitializer
  %181 = getelementptr inbounds double, double* %85, i64 %index160
  %182 = bitcast double* %181 to <4 x double>*
  store <4 x double> %180, <4 x double>* %182, align 8
  %183 = getelementptr inbounds double, double* %181, i64 4
```

```
  %184 = bitcast double* %183 to <4 x double>*
  store <4 x double> %180, <4 x double>* %184, align 8
  %185 = getelementptr inbounds double, double* %181, i64 8
  %186 = bitcast double* %185 to <4 x double>*
  store <4 x double> %180, <4 x double>* %186, align 8
  %187 = getelementptr inbounds double, double* %181, i64 12
  %188 = bitcast double* %187 to <4 x double>*
  store <4 x double> %180, <4 x double>* %188, align 8
  %index.next161 = add i64 %index160, 16
  %189 = icmp eq i64 %index.next161, %n.vec159
  br i1 %189, label %middle.block137, label %vector.body139

middle.block137:                                  ; preds = %vector.body13
9
  %cmp.n163 = icmp eq i64 %.sroa.026.0, %n.vec159
  br i1 %cmp.n163, label %L233, label %scalar.ph138

scalar.ph138:                                     ; preds = %middle.block1
37, %vector.memcheck156, %L185.us.us.preheader
  %bc.resume.val162 = phi i64 [ %n.vec159, %middle.block137 ], [ 0, %L185.
us.us.preheader ], [ 0, %vector.memcheck156 ]
  br label %L185.us.us

L185.us.us:                                       ; preds = %L185.us.us, %
scalar.ph138
  %value_phi1351.us.us = phi i64 [ %195, %L185.us.us ], [ %bc.resume.val16
2, %scalar.ph138 ]
  %190 = load double, double* %81, align 8
  %191 = load double, double* %83, align 8
  %192 = fadd double %190, %191
  %193 = fmul double %192, %192
  %194 = getelementptr inbounds double, double* %85, i64 %value_phi1351.u
s.us
  store double %193, double* %194, align 8
```

```
  %195 = add nuw nsw i64 %value_phi1351.us.us, 1
  %exitcond57.not = icmp eq i64 %195, %.sroa.026.0
  br i1 %exitcond57.not, label %L233, label %L185.us.us

L185.us:                                          ; preds = %L185.us, %sca
lar.ph107
  %value_phi1351.us = phi i64 [ %202, %L185.us ], [ %bc.resume.val131, %sc
alar.ph107 ]
  %196 = load double, double* %81, align 8
  %197 = getelementptr inbounds double, double* %83, i64 %value_phi1351.us
  %198 = load double, double* %197, align 8
  %199 = fadd double %196, %198
  %200 = fmul double %199, %199
  %201 = getelementptr inbounds double, double* %85, i64 %value_phi1351.us
  store double %200, double* %201, align 8
  %202 = add nuw nsw i64 %value_phi1351.us, 1
  %exitcond55.not = icmp eq i64 %202, %.sroa.026.0
  br i1 %exitcond55.not, label %L233, label %L185.us

L185.us52:                                        ; preds = %L185.us52, %s
calar.ph78
  %value_phi1351.us53 = phi i64 [ %209, %L185.us52 ], [ %bc.resume.val100,
%scalar.ph78 ]
  %203 = getelementptr inbounds double, double* %81, i64 %value_phi1351.us
53
  %204 = load double, double* %203, align 8
  %205 = load double, double* %83, align 8
  %206 = fadd double %204, %205
  %207 = fmul double %206, %206
  %208 = getelementptr inbounds double, double* %85, i64 %value_phi1351.us
53
  store double %207, double* %208, align 8
  %209 = add nuw nsw i64 %value_phi1351.us53, 1
  %exitcond56.not = icmp eq i64 %209, %.sroa.026.0
```

In [26]:

```julia
@code_native debuginfo=:none foo(A, B)
```

```
	.text
	pushq	%rbp
	movq	%rsp, %rbp
	pushq	%r15
	pushq	%r14
	pushq	%r13
	pushq	%r12
	pushq	%rbx
	andq	$-32, %rsp
	subq	$128, %rsp
	vxorpd	%xmm0, %xmm0, %xmm0
	vmovapd	%ymm0, (%rsp)
	movq	%rsi, 96(%rsp)
	movq	%fs:0, %rdi
	movq	$8, (%rsp)
	movq	-32768(%rdi), %rax
	movq	%rax, 8(%rsp)
	movq	%rsp, %rax
	movq	%rax, -32768(%rdi)
	movq	(%rsi), %r12
	movq	8(%rsi), %r13
	movq	24(%r12), %r15
	movq	24(%r13), %rbx
	cmpq	$1, %rbx
	je	L112
	testq	%rbx, %rbx
	jne	L121
	testq	%r15, %r15
	sete	%al
	jmp	L127
```

```
L112:
        cmpq    $1, %r15
        sete    %al
        jmp     L127
L121:
        cmpq    %r15, %rbx
        sete    %al
L127:
        addq    $-32768, %rdi                   # imm = 0x8000
        movabsq $jl_system_image_data, %r14
        cmpq    $1, %r15
        movq    %rdi, 40(%rsp)
        je      L225
        testb   %al, %al
        jne     L225
        cmpq    $1, %r15
        je      L184
        testq   %r15, %r15
        jne     L202
        testq   %rbx, %rbx
        sete    %al
        cmpq    $1, %rbx
        jne     L214
        jmp     L197
L184:
        cmpq    $1, %rbx
        sete    %al
        cmpq    $1, %rbx
        jne     L214
L197:
        movq    %r15, %rbx
        jmp     L225
L202:
        cmpq    %rbx, %r15
```

```
        sete    %al
        cmpq    $1, %rbx
        je      L197
L214:
        testb   %al, %al
        je      L1357
        movq    %r15, %rbx
L225:
        movq    %rbx, 56(%rsp)
        leaq    243107877(%r14), %rax
        movabsq $jl_system_image_data, %rdi
        movq    %rbx, %rsi
        vzeroupper
        callq   *%rax
        movq    %rax, %r11
        movq    24(%rax), %rax
        cmpq    $1, %rax
        je      L450
        testq   %rax, %rax
        jne     L489
        testq   %rbx, %rbx
        jne     L460
L290:
        cmpq    %r12, %r11
        je      L340
        testb   $1, 5030201(%r14)
        jne     L340
        movq    (%r11), %rax
        cmpq    (%r12), %rax
        jne     L340
        movq    %r11, 24(%rsp)
        leaq    243108093(%r14), %rax
        movq    %r12, %rdi
        movq    %r11, %r15
```

```
        callq   *%rax
        movq    %r15, %r11
        movq    %rax, %r12
L340:
        movq    24(%r12), %r15
        cmpq    %r13, %r11
        je      L401
        testb   $1, 5030201(%r14)
        jne     L401
        movq    (%r11), %rax
        cmpq    (%r13), %rax
        jne     L401
        movq    %r11, 24(%rsp)
        movq    %r12, 16(%rsp)
        addq    $243108093, %r14                # imm = 0xE7D88FD
        movq    %r13, %rdi
        movq    %r11, %r13
        callq   *%r14
        movq    %r13, %r11
        movq    %rax, %r13
L401:
        testq   %rbx, %rbx
        je      L1323
        movq    24(%r13), %rsi
        movq    (%r12), %rax
        movq    (%r13), %rcx
        movq    (%r11), %rdx
        cmpq    $1, %r15
        jne     L500
        cmpq    $1, %rsi
        jne     L523
        cmpq    $16, %rbx
        jae     L557
        xorl    %esi, %esi
```

```
        jmp     L688
L450:
        cmpq    $1, %rbx
        je      L290
L460:
        movq    %rax, 48(%rsp)
        movabsq $throwdm, %rax
        leaq    48(%rsp), %rdi
        leaq    56(%rsp), %rsi
        callq   *%rax
        ud2
L489:
        cmpq    %rbx, %rax
        je      L290
        jmp     L460
L500:
        cmpq    $1, %rsi
        jne     L540
        cmpq    $16, %rbx
        jae     L718
        xorl    %esi, %esi
        jmp     L880
L523:
        cmpq    $16, %rbx
        jae     L911
        xorl    %esi, %esi
        jmp     L1072
L540:
        cmpq    $16, %rbx
        jae     L1103
        xorl    %esi, %esi
        jmp     L1296
L557:
        leaq    (%rdx,%rbx,8), %rsi
```

```
        leaq    1(%rax), %rdi
        leaq    1(%rcx), %r8
        cmpq    %rdx, %rdi
        seta    %r9b
        cmpq    %rsi, %rax
        setb    %r10b
        cmpq    %rdx, %r8
        seta    %dil
        cmpq    %rsi, %rcx
        setb    %r8b
        xorl    %esi, %esi
        testb   %r10b, %r9b
        jne     L688
        andb    %r8b, %dil
        jne     L688
        movq    %rbx, %rsi
        andq    $-16, %rsi
        xorl    %edi, %edi
        nop
L624:
        vmovsd  (%rax), %xmm0                   # xmm0 = mem[0],zero
        vaddsd  (%rcx), %xmm0, %xmm0
        vmulsd  %xmm0, %xmm0, %xmm0
        vbroadcastsd    %xmm0, %ymm0
        vmovupd %ymm0, (%rdx,%rdi,8)
        vmovupd %ymm0, 32(%rdx,%rdi,8)
        vmovupd %ymm0, 64(%rdx,%rdi,8)
        vmovupd %ymm0, 96(%rdx,%rdi,8)
        addq    $16, %rdi
        cmpq    %rdi, %rsi
        jne     L624
        cmpq    %rsi, %rbx
        je      L1323
        nopw    (%rax,%rax)
```

```
L688:
        vmovsd  (%rax), %xmm0                  # xmm0 = mem[0],zero
        vaddsd  (%rcx), %xmm0, %xmm0
        vmulsd  %xmm0, %xmm0, %xmm0
        vmovsd  %xmm0, (%rdx,%rsi,8)
        incq    %rsi
        cmpq    %rsi, %rbx
        jne     L688
        jmp     L1323
L718:
        leaq    (%rdx,%rbx,8), %rsi
        leaq    (%rax,%rbx,8), %rdi
        leaq    1(%rcx), %r8
        cmpq    %rdi, %rdx
        setb    %r9b
        cmpq    %rsi, %rax
        setb    %r10b
        cmpq    %rdx, %r8
        seta    %dil
        cmpq    %rsi, %rcx
        setb    %r8b
        xorl    %esi, %esi
        testb   %r10b, %r9b
        jne     L880
        andb    %r8b, %dil
        jne     L880
        movq    %rbx, %rsi
        andq    $-16, %rsi
        vmovsd  (%rcx), %xmm0                  # xmm0 = mem[0],zero
        xorl    %edi, %edi
        vbroadcastsd    %xmm0, %ymm0
        nopw    %cs:(%rax,%rax)
L800:
        vaddpd  (%rax,%rdi,8), %ymm0, %ymm1
```

```
        vaddpd  32(%rax,%rdi,8), %ymm0, %ymm2
        vaddpd  64(%rax,%rdi,8), %ymm0, %ymm3
        vaddpd  96(%rax,%rdi,8), %ymm0, %ymm4
        vmulpd  %ymm1, %ymm1, %ymm1
        vmulpd  %ymm2, %ymm2, %ymm2
        vmulpd  %ymm3, %ymm3, %ymm3
        vmulpd  %ymm4, %ymm4, %ymm4
        vmovupd %ymm1, (%rdx,%rdi,8)
        vmovupd %ymm2, 32(%rdx,%rdi,8)
        vmovupd %ymm3, 64(%rdx,%rdi,8)
        vmovupd %ymm4, 96(%rdx,%rdi,8)
        addq    $16, %rdi
        cmpq    %rdi, %rsi
        jne     L800
        cmpq    %rsi, %rbx
        je      L1323
L880:
        vmovsd  (%rax,%rsi,8), %xmm0            # xmm0 = mem[0],zero
        vaddsd  (%rcx), %xmm0, %xmm0
        vmulsd  %xmm0, %xmm0, %xmm0
        vmovsd  %xmm0, (%rdx,%rsi,8)
        incq    %rsi
        cmpq    %rsi, %rbx
        jne     L880
        jmp     L1323
L911:
        leaq    (%rdx,%rbx,8), %rsi
        leaq    1(%rax), %rdi
        leaq    (%rcx,%rbx,8), %r8
        cmpq    %rdx, %rdi
        seta    %r9b
        cmpq    %rsi, %rax
        setb    %r10b
        cmpq    %r8, %rdx
```

```
        setb    %dil
        cmpq    %rsi, %rcx
        setb    %r8b
        xorl    %esi, %esi
        testb   %r10b, %r9b
        jne     L1072
        andb    %r8b, %dil
        jne     L1072
        movq    %rbx, %rsi
        andq    $-16, %rsi
        vmovsd  (%rax), %xmm0                  # xmm0 = mem[0],zero
        xorl    %edi, %edi
        vbroadcastsd    %xmm0, %ymm0
        nopw    %cs:(%rax,%rax)
L992:
        vaddpd  (%rcx,%rdi,8), %ymm0, %ymm1
        vaddpd  32(%rcx,%rdi,8), %ymm0, %ymm2
        vaddpd  64(%rcx,%rdi,8), %ymm0, %ymm3
        vaddpd  96(%rcx,%rdi,8), %ymm0, %ymm4
        vmulpd  %ymm1, %ymm1, %ymm1
        vmulpd  %ymm2, %ymm2, %ymm2
        vmulpd  %ymm3, %ymm3, %ymm3
        vmulpd  %ymm4, %ymm4, %ymm4
        vmovupd %ymm1, (%rdx,%rdi,8)
        vmovupd %ymm2, 32(%rdx,%rdi,8)
        vmovupd %ymm3, 64(%rdx,%rdi,8)
        vmovupd %ymm4, 96(%rdx,%rdi,8)
        addq    $16, %rdi
        cmpq    %rdi, %rsi
        jne     L992
        cmpq    %rsi, %rbx
        je      L1323
L1072:
        vmovsd  (%rax), %xmm0                  # xmm0 = mem[0],zero
```

```
        vaddsd  (%rcx,%rsi,8), %xmm0, %xmm0
        vmulsd  %xmm0, %xmm0, %xmm0
        vmovsd  %xmm0, (%rdx,%rsi,8)
        incq    %rsi
        cmpq    %rsi, %rbx
        jne     L1072
        jmp     L1323
L1103:
        leaq    (%rdx,%rbx,8), %rsi
        leaq    (%rax,%rbx,8), %rdi
        leaq    (%rcx,%rbx,8), %r8
        cmpq    %rdi, %rdx
        setb    %r9b
        cmpq    %rsi, %rax
        setb    %r10b
        cmpq    %r8, %rdx
        setb    %dil
        cmpq    %rsi, %rcx
        setb    %r8b
        xorl    %esi, %esi
        testb   %r10b, %r9b
        jne     L1296
        andb    %r8b, %dil
        jne     L1296
        movq    %rbx, %rsi
        andq    $-16, %rsi
        xorl    %edi, %edi
        nopw    %cs:(%rax,%rax)
L1184:
        vmovupd (%rax,%rdi,8), %ymm0
        vmovupd 32(%rax,%rdi,8), %ymm1
        vmovupd 64(%rax,%rdi,8), %ymm2
        vmovupd 96(%rax,%rdi,8), %ymm3
        vaddpd  (%rcx,%rdi,8), %ymm0, %ymm0
```

```
        vaddpd  32(%rcx,%rdi,8), %ymm1, %ymm1
        vaddpd  64(%rcx,%rdi,8), %ymm2, %ymm2
        vaddpd  96(%rcx,%rdi,8), %ymm3, %ymm3
        vmulpd  %ymm0, %ymm0, %ymm0
        vmulpd  %ymm1, %ymm1, %ymm1
        vmulpd  %ymm2, %ymm2, %ymm2
        vmulpd  %ymm3, %ymm3, %ymm3
        vmovupd %ymm0, (%rdx,%rdi,8)
        vmovupd %ymm1, 32(%rdx,%rdi,8)
        vmovupd %ymm2, 64(%rdx,%rdi,8)
        vmovupd %ymm3, 96(%rdx,%rdi,8)
        addq    $16, %rdi
        cmpq    %rdi, %rsi
        jne     L1184
        cmpq    %rsi, %rbx
        je      L1323
        nopw    (%rax,%rax)
L1296:
        vmovsd  (%rax,%rsi,8), %xmm0             # xmm0 = mem[0],zero
        vaddsd  (%rcx,%rsi,8), %xmm0, %xmm0
        vmulsd  %xmm0, %xmm0, %xmm0
        vmovsd  %xmm0, (%rdx,%rsi,8)
        incq    %rsi
        cmpq    %rsi, %rbx
        jne     L1296
L1323:
        movq    8(%rsp), %rax
        movq    40(%rsp), %rcx
        movq    %rax, (%rcx)
        movq    %r11, %rax
        leaq    -40(%rbp), %rsp
        popq    %rbx
        popq    %r12
        popq    %r13
```

```
        popq    %r14
        popq    %r15
        popq    %rbp
        vzeroupper
        retq
L1357:
        movabsq $140310593213369, %rax          # imm = 0x7F9C9B1113B9
        movl    $1400, %esi                     # imm = 0x578
        movl    $16, %edx
        movq    %rdi, %r12
        vzeroupper
        callq   *%rax
        movq    %r14, -8(%rax)
        movq    %r15, (%rax)
        movq    %rax, 16(%rsp)
        movq    %rax, 64(%rsp)
        movabsq $jl_apply_generic, %rax
        movabsq $jl_system_image_data, %rdi
        leaq    64(%rsp), %r15
        movq    %r15, %rsi
        movl    $1, %edx
        callq   *%rax
        movq    %rax, %r13
        movq    %rax, 24(%rsp)
        movq    %r12, %rdi
        movl    $1400, %esi                     # imm = 0x578
        movl    $16, %edx
        movabsq $140310593213369, %rax          # imm = 0x7F9C9B1113B9
        callq   *%rax
        movq    %r14, -8(%rax)
        movq    %rbx, (%rax)
        movq    %rax, 16(%rsp)
        movq    %rax, 64(%rsp)
        movabsq $jl_system_image_data, %rdi
```

```
movq    %r15, %rsi
movl    $1, %edx
movabsq $jl_apply_generic, %rbx
callq   *%rbx
movq    %rax, 16(%rsp)
movabsq $jl_system_image_data, %rcx
movq    %rcx, 64(%rsp)
movq    %r13, 72(%rsp)
movabsq $jl_system_image_data, %rcx
movq    %rcx, 80(%rsp)
movq    %rax, 88(%rsp)
movabsq $jl_system_image_data, %rdi
movq    %r15, %rsi
movl    $4, %edx
callq   *%rbx
movq    %rax, 16(%rsp)
movq    %rax, 64(%rsp)
movabsq $jl_system_image_data, %rdi
movq    %r15, %rsi
movl    $1, %edx
callq   *%rbx
movabsq $jl_throw, %rcx
movq    %rax, %rdi
callq   *%rcx
```

Package management

- Julia probably has the best package management to date
- Press "]" to enter package management console
- Typically `add PACKAGE_NAME` is sufficient, can also do `add PACKAGE_NAME@VERSION`
- To get an unreleased version, use `add PACKAGE_NAME#BRANCH_NAME`
- Easy to start modifying a package via `dev PACKAGE_NAME`
- Multiple package versions can be installed, selection via Pkg.jl environments.
- Also useful: `julia> using Pkg; pkg"<Pkg console command>"`

Package creation

- A Julia package needs:
    - A "Project.toml" file
    - A "src/PackageName.jl" file
- That's it: Push to GitHub, and package is installable via `add PACKAGE_URL`
- Use Documenter.jl to document your package
- To enable `add PACKAGE_NAME`, package must be registered, there are some rules
- Use PkgTemplates.jl to generate new package with CI config (Travis, Appveyor, ...), docs generation, etc.

No free lunch

- Package loading and code-gen can sometime take a long time, but mitigations available:
- Revise.jl: Hot code-reloading at runtime
- [PackageCompiler.jl](https://github.com/JuliaLang/PackageCompiler.jl]: Ahead-of-time compilation, producing custom Julia system images

Performance tips

- Read the official Julia performance tips!
- Do *not* call on (non-const) global variables from time-critical code
- Type-stable code is fast code. Use `@code_warntype` and `Test.@inferred` to check!
- In some situations, closures can be troublesome, using `let` can help the compiler

This is efficient (not runtime reflection):

In [27]:

```
half_dynrange(T::Type{<:Number}) = (Int(typemax(T)) - Int(typemin(T))) / 2
half_dynrange(Int16)
```

Out[27]:

```
32767.5
```

In [28]:

```
@code_llvm half_dynrange(Int16)
```

```
;  @ In[27]:1 within `half_dynrange'
define double @julia_half_dynrange_2449() {
top:
  ret double 3.276750e+04
}
```

SIMD

Demo

Shared-memory parallelism

- Julia has native multithreading support
- Simple cases: Use `@threads` macro
- Since Julia v1.3: Cache-efficient composable multi-threaded parallelism

Processes, Clusters, MPI

- Julia brings a full API for remote processes and compute clusters
- Native support for local processes and remote processes via SSH
- MPI support via MPI.jl and MPIClusterManagers.jl

Benchmarking and profiling, digging deeper

Demo

Docs and help

- Official Julia docs
- Julia Cheat Sheet
- ThinkJulia
- https://julialang.org/learning/
- Julia Discourse
- Julia Slack
- Julia Gitter
- Julia on Youtube
- JuliaCon 2020

Statistics

Demo

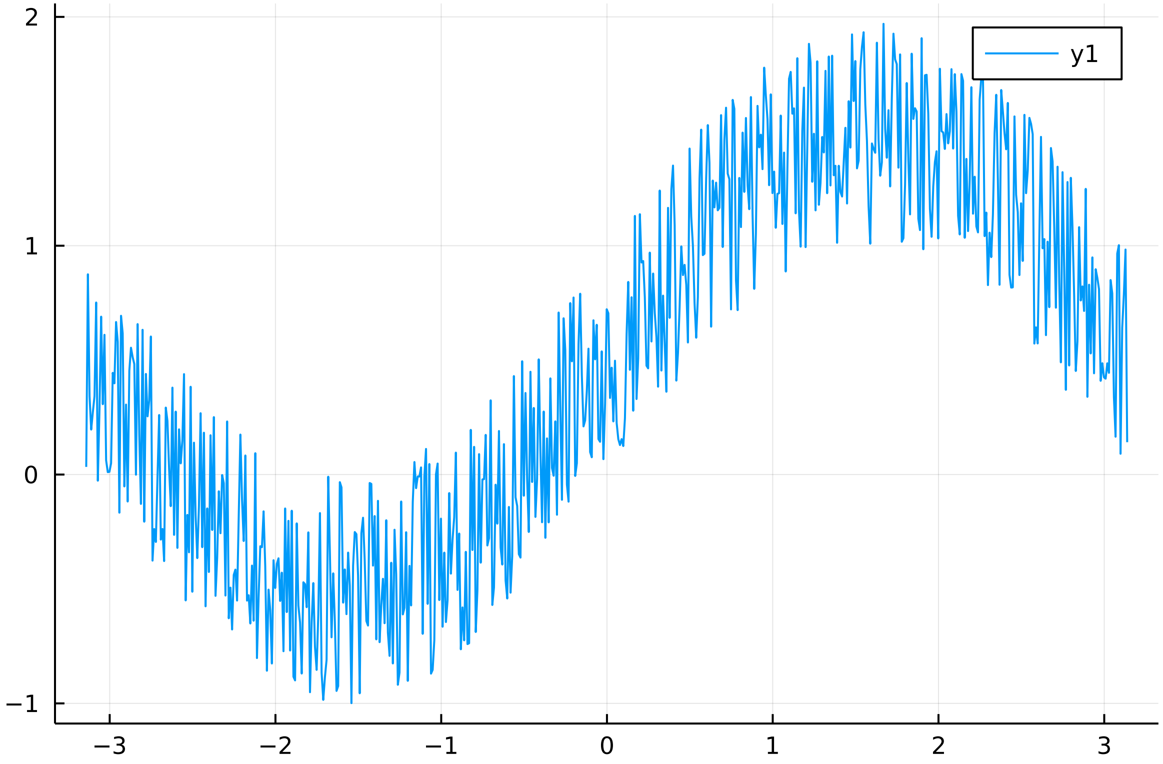
Visualization/Plotting: Plots, Makie, plotting recipes

Let's Make a Plot

In [29]:

```
using Plots
range = -π:0.01:π
plot(range, sin.(range) + rand(length(range)))
```

Out[29]:



Histograms are easy, too

In [30]:

```
using Distributions
dist = Normal(0.0, 5.0)
```

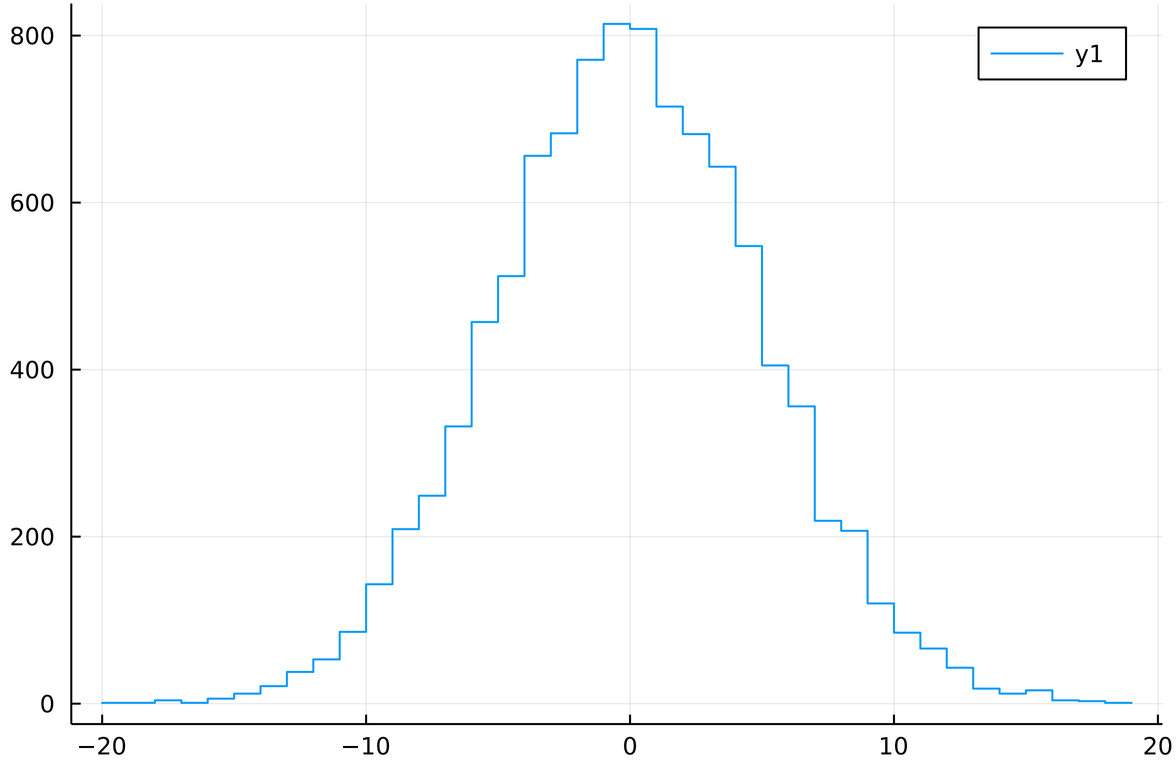
Out[30]:

```
Normal{Float64}(μ=0.0, σ=5.0)
```

In [31]:

```
stephist(rand(dist, 10000))
```

Out[31]:



Automatic differentiation

Let's define a simple neural network layer and loss function and auto-differentiate through it.

In [32]:

```
struct DenseLayer{M<:AbstractMatrix{<:Real},V<:AbstractVector{<:Real},F<:Function} <: Functi
    A::M
    b::V
    f::F
end

(l::DenseLayer)(x::AbstractVector{<:Real}) = (l.f).(l.A * x + l.b)

f_loss(y) = sum(y .^ 2);
```

In [33]:

```
mylayer = DenseLayer(rand(5,5), rand(5), x -> ifelse(x > zero(x), x, zero(x)));
```

In [34]:

```
x = rand(5)
mylayer(x)
```

Out[34]:

```
5-element Vector{Float64}:
 1.7041810782876734
 1.163826388903705
 1.682990185750071
 2.6543044638188427
 2.161791402654909
```

In [35]:

```
f_loss(mylayer(x))
```

Out[35]:

```
18.809855231674746
```

In [36]:

```julia
using Zygote
g = Zygote.gradient((mylayer, x) -> f_loss(mylayer(x)), mylayer, x)
g[1].A
```

Out[36]:

```
5×5 Matrix{Float64}:
 2.78885  1.12708   3.20878  1.92882  1.42191
 1.90457  0.769712  2.19135  1.31724  0.971056
 2.75417  1.11307   3.16888  1.90484  1.40423
 4.3437   1.75546   4.99775  3.00419  2.21466
 3.53772  1.42973   4.07041  2.44675  1.80372
```

In [37]:

```julia
g[1].b
```

Out[37]:

```
5-element Vector{Float64}:
 3.408362156575347
 2.32765277780741
 3.365980371500142
 5.308608927637685
 4.323582805309818
```

Calling code written in other language, REPL modes

Shell REPL mode

- The Julia shell is multi-language
- Press ";" for a system shell

PyCall, RCall

Calling Python from Julia is easy, can even use inline Python code:

```julia
using PyCall
numpy = pyimport("numpy")
numpy.zeros(5) isa Array

A = rand(5)
py"""type($A)""" isa PyObject
```

An incomplete tour of the Julia package ecosystem

Math

- ApproxFun.jl: Powerful function approximations
- FFTW.jl: Fast fourier transforms via FFTW
- DifferentialEquations.jl: A suite for numerically solving differential equations
- ...

Optimization

- JuMP.jl: Modeling language for Mathematical Optimization
- NLopt.jl: Optimization via NLopt
- Optim: Julia native nonlinear optimization

TypedTables and DataFrames

- Tables.jl: Abstract API for tabular data
- DataFrames.jl: Python/R-like dataframes
- TypedTables.jl: Type-stable tables
- Query.jl LINQ-inspired data query and transformation

Plotting and Visualization

- IJulia.jl: Julia Jupyter kernel
- Images.jl: Image processing
- PyPlot.jl: Use matplotlib/PyPlot from Julia
- Makie.jl: Hardware-accelerated plotting
- Plots.jl: Plotting with generic recipes and multiple backends

Statistics

- Distributions.jl: Probability distributions and associated functions

- StatsBase.jl: Statistics, histograms, etc.

- Many specialized packages

Automatic Differentiation

- ForwardDiff.jl: Forward-mode automatic differentiation
- Zyote.jl: Source-level reverse-mode automatic differentiation
- Enzyme.jl: LLVM-level reverse-mode automatic differentiation
- Several other packages available (ReverseDiff.jl, Nabla.jl, Yota.jl, ...)
- Exciting developements to come with new Julia Compiler features

Bayesian analysis and probabilistic programming

- BAT.jl: Bayesian analysis toolkit
- Gen.jl: General-Purpose Probabilistic Programming System
- Mamba.jl: MCMC for Bayesian analysis
- Turing.jl: Probabilistic machine learning and Bayesian statistics
- ...

Machine learning

- Flux.jl: Julia native deep learning library
- Knet.jl:Koc University deep learning framework
- MXNet.jl: MXNet Julia API
- ...

Calling code in other languages

- Cxx.jl: Call C++ from Julia
- PyCall.jl: Call Python from Julia
- RCall.jl: Call R from Julia
- ...

Efficient memory layout

- ArraysOfArrays.jl: Duality of flat and nested arrays
- StructArrays.jl, TypedTables.jl: AoS and SoA duality
- ValueShapes.jl: Duality of flat and nested structures
- ...

GPU Programming

- AMDGPU.jl: Julia on AMD GPUs (WIP)
- CUDA.jl: Julia on NVIDIA GPUs
- GPUArrays.jl: Generic CPU programming API
- XLA.jl: Julia on Google TPUs
- Maybe more architectures in the future?

IDEs

- julia-vscode: Visual Studio Code based Julia IDE
- Juno: Atom based Julia IDE (now deprecated in favor of VS code)

Final Remarks

- Julia is productive, fast and fun - give it a chance!
- Multiple dispatch opens up powerful ways of combining code