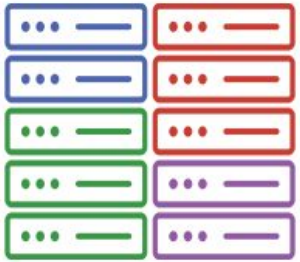
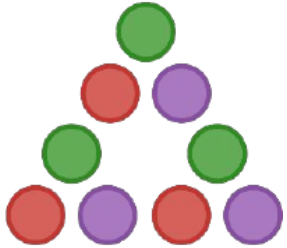




Julia and MPI:
O ABI, ABI, wherefore art thou ABI?

Valentin Churavy
Erik Schnetter
Simon Byrne
vchuravy@uni-mainz.de

UNIA

julia

JG|U

Jeff Hammond https://c.im/@jeffscience
@science_dot

Replying to @science_dot, @miguelraz_ and @JuliaLanguage

Julia is of course great because it's basically Fortran for people who are too lazy to declare types and has an interpreter.

1:41 pm · 22 Feb 2023 · **1,257** Views

1 Quote Tweet **22** Likes

Yet another high-level language?

Dynamically typed, high-level syntax

Open-source, permissive license

Built-in package manager

Interactive development

```
julia> function mandel(z)
           c = z
           maxiter = 80
           for n = 1:maxiter
               if abs(z) > 2
                   return n-1
               end
               z = z^2 + c
           end
           return maxiter
       end

julia> mandel(complex(.3, -.6))
14
```

Yet another high-level language?

Typical features

Dynamically typed, high-level syntax

Open-source, permissive license

Built-in package manager

Interactive development

Unusual features

Great performance!

JIT AOT-style compilation

Most of Julia is written in Julia

Reflection and metaprogramming

Pkg.jl — Julia's Package Manager

- `Project.toml`: Describes the dependencies and compatibilities
- `Manifest.toml`: Record of precise versions of all direct & indirect dependencies

```
(@v1.5) pkg> st
Status `~/.julia/environments/v1.5/Project.toml`
  [7da242da] Enzyme v0.3.0

(@v1.5) pkg> up
   Updating registry at `~/.julia/registries/General`
######################################################## 100.0%
Updating `~/.julia/environments/v1.5/Project.toml`
  [7da242da] ↑ Enzyme v0.3.0 ⇒ v0.3.1
Updating `~/.julia/environments/v1.5/Manifest.toml`
  [7da242da] ↑ Enzyme v0.3.0 ⇒ v0.3.1
  [7cc45869] ↑ Enzyme_jll v0.0.5+0 ⇒ v0.0.6+0
```

- Binarybuilder: (https://binarybuilder.org/)
  - Sandboxed cross-compiler
  - Encodes best practices

- JLL Packages:
  Binary dependencies packaged as Artifacts

- Yggdrasil:
  Collection of build recipes

Pkg3: Julia's New Package Manager
https://www.youtube.com/watch?v=-yUiLCGegJs

What's new in Pkg: Artifacts & Binaries
https://www.youtube.com/watch?v=xPhnJCAkI4k

_jll Packages

- A binary released as a Julia package that the package manager can install
- Transparently loading dependencies and makes libraries available

BUT! What about my bespoke HPC cluster? I must use HPE/Cray...

1. Everything is overwritable/exchangeable
2. `JLLWrappers.jl` uses `Preferences.jl` to make library location configurable
3. Extended platform tags support things like MPI ABI

HPC — Interacting with the System

1. Julia has direct foreign call support for C & Fortran
   - https://docs.julialang.org/en/v1/base/c/#Base.@ccall
   - https://docs.julialang.org/en/v1/manual/calling-c-and-fortran-code

```
@ccall getenv(var::Cstring)::Cstring
```

2. Automatic wrapper generation with Clang.jl
   - Core assumption: No C compiler available on user system

3. @cfunction creates a C-function pointer to use as a callback
   - Calling Julia from C with MPI: https://github.com/omlins/libdiffusion

4. Be careful around GC interactions!

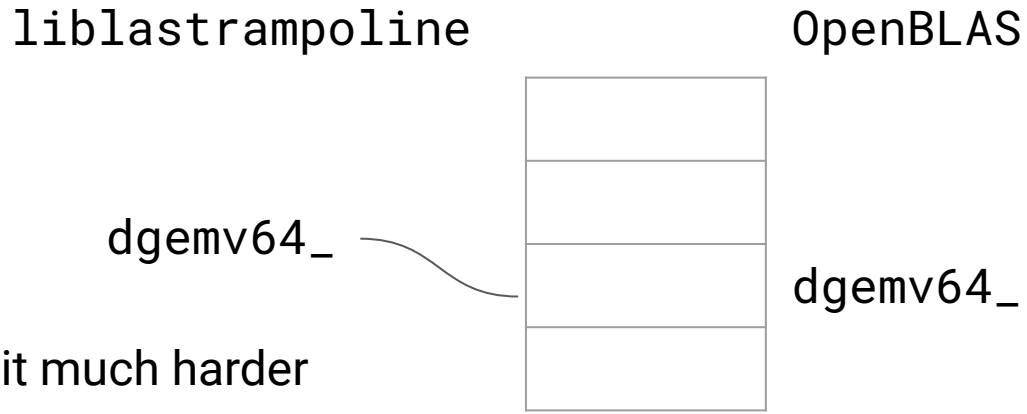
ABI => C/System ABI

How this works for BLAS

`Libblastrampoline` — one C library to link other C-libraries against

- dlopen/dlsym to obtain addresses and fill jump vector
- LBT provides entry points to link against
- Dynamically redicrects
  to runtime selected target
  Library: MKL/OpenBLAS/BLIS

- Drastically simplifies shipping
  binaries to users
- MPItrampoline provides similar
  for MPI, but ABI concerns make it much harder



Lack of MPI ABI

- Binary explosion, recent build of LAMMPS 44 separate builds
  - 4 MPI ABIs
  - Linux/Windows/Mac/FreeBSD
  - x86/x86_64/AArch64/ARM6/PowerPCle
  - Two C++ ABIs
  - CUDA versions -> 156 builds
- Uncertainty
  - https://github.com/JuliaParallel/MPI.jl/blob/master/src/api/openmpi.jl
  - https://github.com/JuliaParallel/MPI.jl/blob/master/src/api/mpich.jl
  - HPE MPT / Microsoft MPI /MPICH / OpenMPI / MPItrampoline

- A stable ABI would vastly simplify deploying binaries to the user, Improve the user experience and remove fragility from writing wrappers

Callbacks

```
typedef void MPI_User_function(
    void *invec, void *inoutvec,
    int *len, MPI_Datatype *datatype);

int MPI_Op_create(MPI_User_function *user_fn,
                  int commute, MPI_Op *op)


@cfunction(callback, Cvoid,
           (Ptr{Cvoid}, Ptr{Cvoid},
            Ptr{Cint},  Ptr{MPI_Datatype}))
```

Callback problem

1. How do I pass **context**?
   In Julia: closure / anonymous functions

2. Still need to pass a function pointer..
   Julia uses GCC-style
   https://gcc.gnu.org/onlinedocs/gccint/Trampolines.html

3. Undefined in the ABI...
   -> Not defined on AArch64
   -> Descriptor support needs recompilation of all binaries...

Callback solution

Recent APIs often include a context argument!

```
typedef void (*ucp_am_recv_data_nbx_callback_t) (
    void * request , ucs_status_t status ,
    size_t length , void * user_data )
```

For future API designs, include a context argument.

https://juliaparallel.org/MPI.jl/stable/examples/03-reduce/

Monthly HPC user group calls

- Open to all / Birds of a feather style
- Provides a repeated point of contact for folks who want to use Julia in HPC
- Concentrated development efforts: example MPI.jl

2nd Thursday of the Month at 2pm CET (next Oct 10th)

4th Tuesday of the Month at 2pm EST (next Oct 22nd)

Details: https://julialang.org/community/ — Events

Whishlist

- MPI ABI
  - Goal: Supporting existing libraries and shipping efficient binaries to users
- MPI_Op callbacks lack:
  - Context argument
  - Proper exception handling
- Integration between task based runtimes and MPI
  - UCX is a nice starting point
- GPU stream ordered operations

Contact me at vchuravy@uni-mainz.de if you want to chat about those

Appendix

What about binaries?

- We need to support:
  - Windows (32bit&64bit); Mac OS (x86_64&aarch64); Linux (x86,ARM,PPC); FreeBSD
  - Can’t assume compiler available on user system
- Artifacts: Immutable, platform specific “archives”
- Doesn't overburden the Package manager
  (version control -> artifact selection)
- https://binarybuilder.org & https://github.com/JuliaPackaging/Yggdrasil
  - Cross-compilation toolchain for building binaries reliably
  - Repository of (user-submitted) recipes
  - Buildfarm + automatic release as JLL packages
  - https://www.youtube.com/watch?v=S__x3K31qnE
  - https://www.youtube.com/watch?v=_jI8CbN_-IE

Example UCX.jl

```
function ucp_put_nb(ep, buffer, length, remote_addr, rkey, cb)
    ccall(
        (:ucp_put_nb, libucp),
        ucs_status_ptr_t,
        (ucp_ep_h, Ptr{Cvoid}, Csize_t, UInt64, ucp_rkey_h, ucp_send_callback_t),
        ep, buffer, length, remote_addr, rkey, cb)
end


function send_callback(req::Ptr{Cvoid}, status::API.ucs_status_t, user_data::Ptr{Cvoid})
    @assert user_data !== C_NULL
    request = UCXRequest(user_data)
    request.status = status
    notify(request)
    API.ucp_request_free(req)
    nothing
end

function put!(ep::UCXEndpoint, request, data::Ptr, nbytes, remote_addr, rkey)
    cb = @cfunction(send_callback, Cvoid, (Ptr{Cvoid}, API.ucs_status_t, Ptr{Cvoid}))
    ptr = ucp_put_nb(ep, data, nbytes, remote_addr, rkey, cb)
    return handle_request(request, ptr)
end

function put!(ep::UCXEndpoint, buffer, nbytes, remote_addr, rkey)
    request = UCXRequest(ep, buffer) # rooted through ep.worker
    GC.@preserve buffer begin
        data = pointer(buffer)
        put!(ep, request, data, nbytes, remote_addr, rkey)
    end
end
```