Updates from the Ecosystem

Valentin Churavy (@vchuravy)

Packages updates

- MPI.jl 0.20
  - Multi-ABI support through Preferences.jl
  - Better support for pre-built multi-MPI binaries

- CUDA v4
  - Also uses Preferences.jl instead of environment variables to select CUDA version
  - Much improved binary story

- AMDGPU.jl
  - Stabilization and readiness testing for LUMI and Frontier

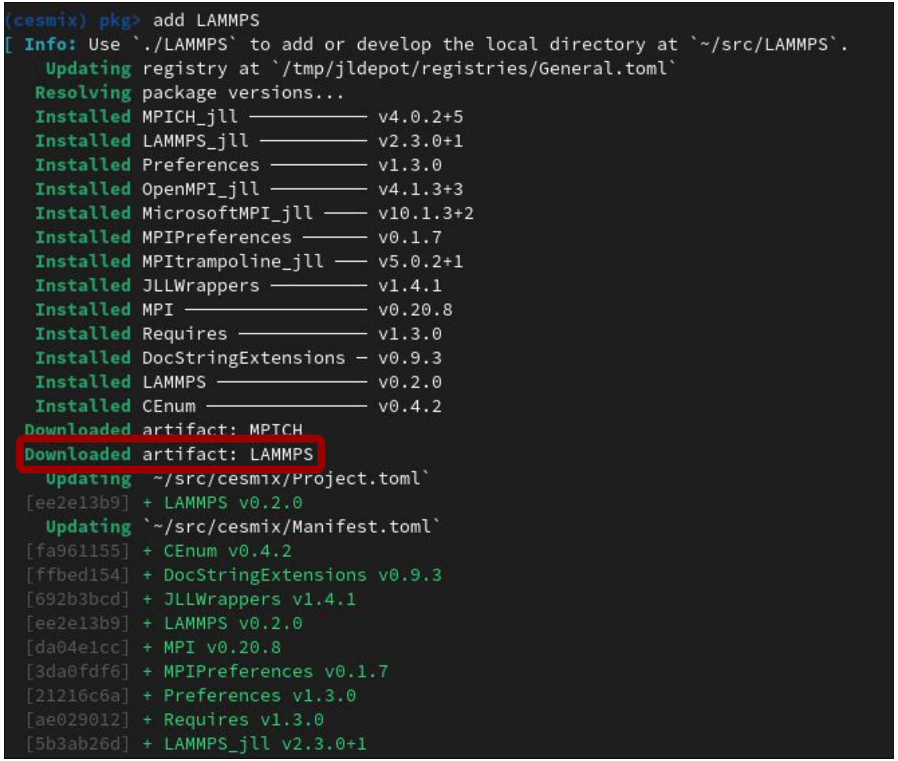
Deploying and Reproducibility

- Big appeal of Julia is the integrated package manager
  - Installing LAMMPS is as easy as doing `] add LAMMPS`
  - Binaries are automatically downloaded for the user
    - Big pain-point in the beginning with folks struggling to get LAMMPS installed properly
    - Opt-out still an option for experts
  - MPI ABI compatibility was a pain-point.

- Over the last year big community effort to make binary packages with MPI dependencies in Julia work.
  - Erik Schnetter, Simon Byrne, Mosè Giordano,
    Valentin Churavy, Elliot Saba

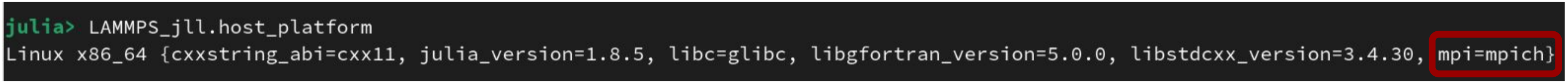
Interlude: Julia package manager

- Pkg.jl is Julia integrated Package manager
  - Reproducibility
  - Accessible through the REPL or as an API
- Project.toml
  - What does a "project"/package need
  - List all direct dependencies and compatibility bounds
- Manifest.toml
  - An instantiation of a Project.toml
  - Contains all direct & transitive dependencies
  - Is exactly reproducible with the same Julia version
  - "Docker" without the fuss

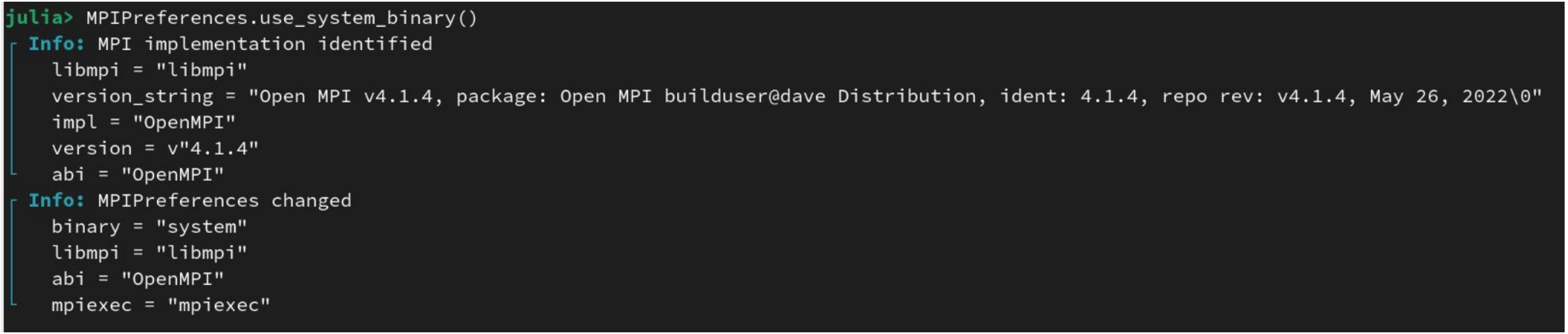


```
(cesmix) pkg> add LAMMPS
[ Info: Use `./LAMMPS` to add or develop the local directory at `~/src/LAMMPS`.
    Updating registry at `/tmp/jldepot/registries/General.toml`
   Resolving package versions...
   Installed MPICH_jll ─────────── v4.0.2+5
   Installed LAMMPS_jll ────────── v2.3.0+1
   Installed Preferences ───────── v1.3.0
   Installed OpenMPI_jll ───────── v4.1.3+3
   Installed MicrosoftMPI_jll ──── v10.1.3+2
   Installed MPIPreferences ────── v0.1.7
   Installed MPItrampoline_jll ─── v5.0.2+1
   Installed JLLWrappers ───────── v1.4.1
   Installed MPI ───────────────── v0.20.8
   Installed Requires ──────────── v1.3.0
   Installed DocStringExtensions ─ v0.9.3
   Installed LAMMPS ────────────── v0.2.0
   Installed CEnum ─────────────── v0.4.2
  Downloaded artifact: MPICH
  Downloaded artifact: LAMMPS
    Updating `~/src/cesmix/Project.toml`
  [ee2e13b9] + LAMMPS v0.2.0
    Updating `~/src/cesmix/Manifest.toml`
  [fa961155] + CEnum v0.4.2
  [ffbed154] + DocStringExtensions v0.9.3
  [692b3bcd] + JLLWrappers v1.4.1
  [ee2e13b9] + LAMMPS v0.2.0
  [da04e1cc] + MPI v0.20.8
  [3da0fdf6] + MPIPreferences v0.1.7
  [21216c6a] + Preferences v1.3.0
  [ae029012] + Requires v1.3.0
  [5b3ab26d] + LAMMPS_jll v2.3.0+1
```

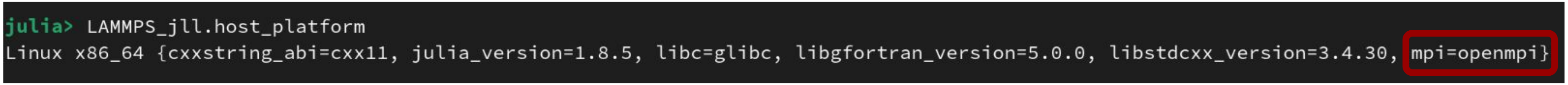
MPIPreferences in action



```
julia> LAMMPS_jll.host_platform
Linux x86_64 {cxxstring_abi=cxx11, julia_version=1.8.5, libc=glibc, libgfortran_version=5.0.0, libstdcxx_version=3.4.30, mpi=mpich}
```



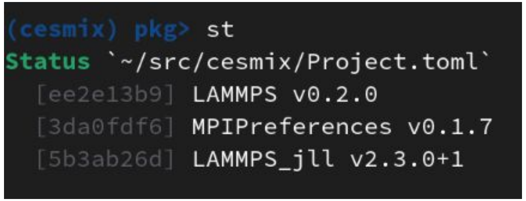
```
julia> MPIPreferences.use_system_binary()
┌ Info: MPI implementation identified
│   libmpi = "libmpi"
│   version_string = "Open MPI v4.1.4, package: Open MPI builduser@dave Distribution, ident: 4.1.4, repo rev: v4.1.4, May 26, 2022\0"
│   impl = "OpenMPI"
│   version = v"4.1.4"
└   abi = "OpenMPI"
┌ Info: MPIPreferences changed
│   binary = "system"
│   libmpi = "libmpi"
│   abi = "OpenMPI"
└   mpiexec = "mpiexec"
```



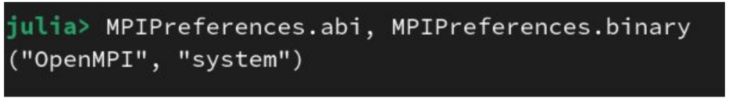
```
julia> LAMMPS_jll.host_platform
Linux x86_64 {cxxstring_abi=cxx11, julia_version=1.8.5, libc=glibc, libgfortran_version=5.0.0, libstdcxx_version=3.4.30, mpi=openmpi}
```

MPIPreferences.jl

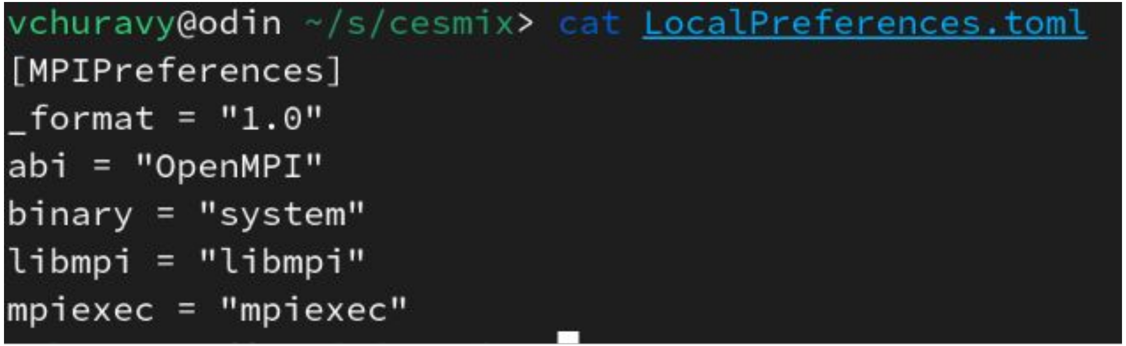
- Preferences.jl
  - Key-value store for package settings
  - Integrates with Julia package pre-compilation
  - No environment variables anymore
- MPIPreferences.jl:
  - Single-source of truth
  - Contains ABI + source of binary
  - Routines for detecting system ABI



```
(cesmix) pkg> st
Status `~/src/cesmix/Project.toml`
  [ee2e13b9] LAMMPS v0.2.0
  [3da0fdf6] MPIPreferences v0.1.7
  [5b3ab26d] LAMMPS_jll v2.3.0+1
```



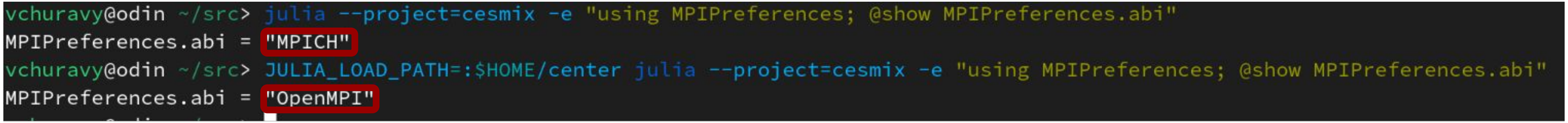
```
julia> MPIPreferences.abi, MPIPreferences.binary
("OpenMPI", "system")
```



```
vchuravy@odin ~/s/cesmix> cat LocalPreferences.toml
[MPIPreferences]
_format = "1.0"
abi = "OpenMPI"
binary = "system"
libmpi = "libmpi"
mpiexec = "mpiexec"
```

Global “HPC Center” setup

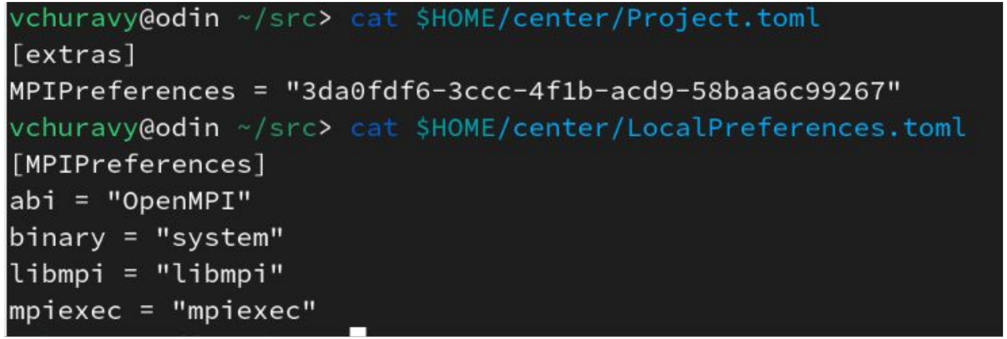
https://juliaparallel.org/tutorials/preferences/



```
vchuravy@odin ~/src> julia --project=cesmix -e "using MPIPreferences; @show MPIPreferences.abi"
MPIPreferences.abi = "MPICH"
vchuravy@odin ~/src> JULIA_LOAD_PATH=:$HOME/center julia --project=cesmix -e "using MPIPreferences; @show MPIPreferences.abi"
MPIPreferences.abi = "OpenMPI"
```

`JULIA_LOAD_PATH`

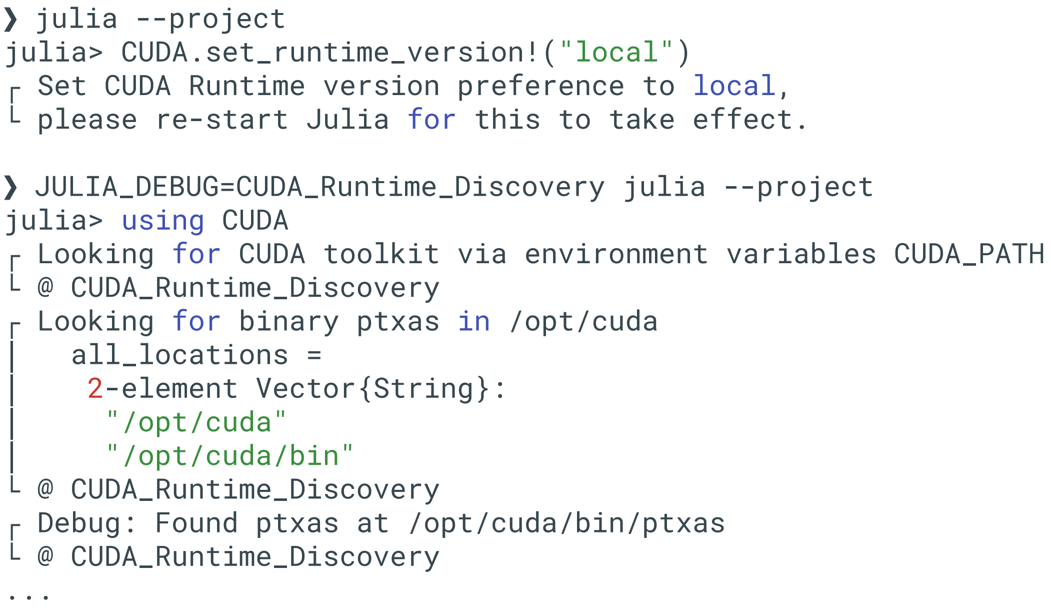
Determines the visibility of Julia packages and we can also use it to set preferences on a center level
Important: Note the **colon** as part of the definition.



```
vchuravy@odin ~/src> cat $HOME/center/Project.toml
[extras]
MPIPreferences = "3da0fdf6-3ccc-4f1b-acd9-58baa6c99267"
vchuravy@odin ~/src> cat $HOME/center/LocalPreferences.toml
[MPIPreferences]
abi = "OpenMPI"
binary = "system"
libmpi = "libmpi"
mpiexec = "mpiexec"
```

CUDA v4

https://juliagpu.org/post/2023-02-01-cuda_4.0/#jlls_for_cuda_artifacts



```
❯ julia --project
julia> CUDA.set_runtime_version!("local")
┌ Set CUDA Runtime version preference to local,
└ please re-start Julia for this to take effect.

❯ JULIA_DEBUG=CUDA_Runtime_Discovery julia --project
julia> using CUDA
┌ Looking for CUDA toolkit via environment variables CUDA_PATH
└ @ CUDA_Runtime_Discovery
┌ Looking for binary ptxas in /opt/cuda
│   all_locations =
│    2-element Vector{String}:
│     "/opt/cuda"
│     "/opt/cuda/bin"
└ @ CUDA_Runtime_Discovery
┌ Debug: Found ptxas at /opt/cuda/bin/ptxas
└ @ CUDA_Runtime_Discovery
...
```

Julia 1.9

- Update to LLVM 14
- Computational Float16
- Improved support for A64FX
- PkgImages
- Weak dependencies / Package extensions

Taking Float16 seriously

First attempt: Naively lowering `Float16` to LLVM’s `half` type.

1. What to do on platforms with no/limited hardware support
2. Extended precision (thanks x87) rears it’s ugly head

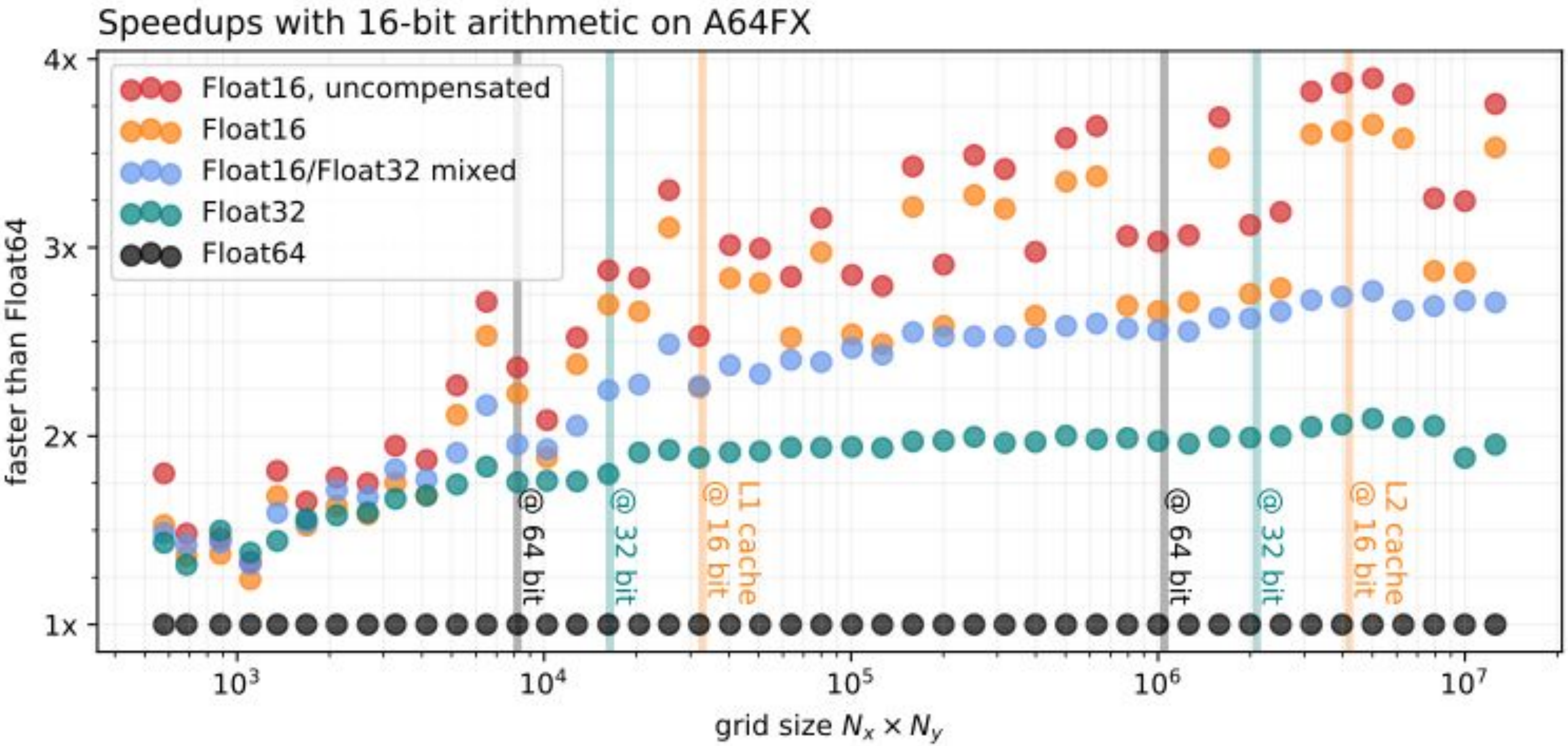
Lesson: In order to implement numerical routines that are portable we must be very careful in what semantics we promise.

Solution: On targets without hardware support for `Float16`, truncate after each operation.

GCC 12 supports this as: `-fexcess-precision=16`

```
define half @julia_muladd(half %0,
half %1, half %2) {
top:
  %3 = fmul half %0, %1
  %4 = fadd half %3, %2
  ret half %4
}
```

```
define half @julia_muladd(half %0, half %1, half %2){
top:
  %3 = fpext half %0 to float
  %4 = fpext half %1 to float
  %5 = fmul float %3, %4
  %6 = fptrunc float %5 to half
  %7 = fpext half %6 to float
  %8 = fpext half %2 to float
  %9 = fadd float %7, %8
  %10 = fptrunc float %9 to half
  ret half %10
```



Reproduced from https://doi.org/10.1029/2021MS002684

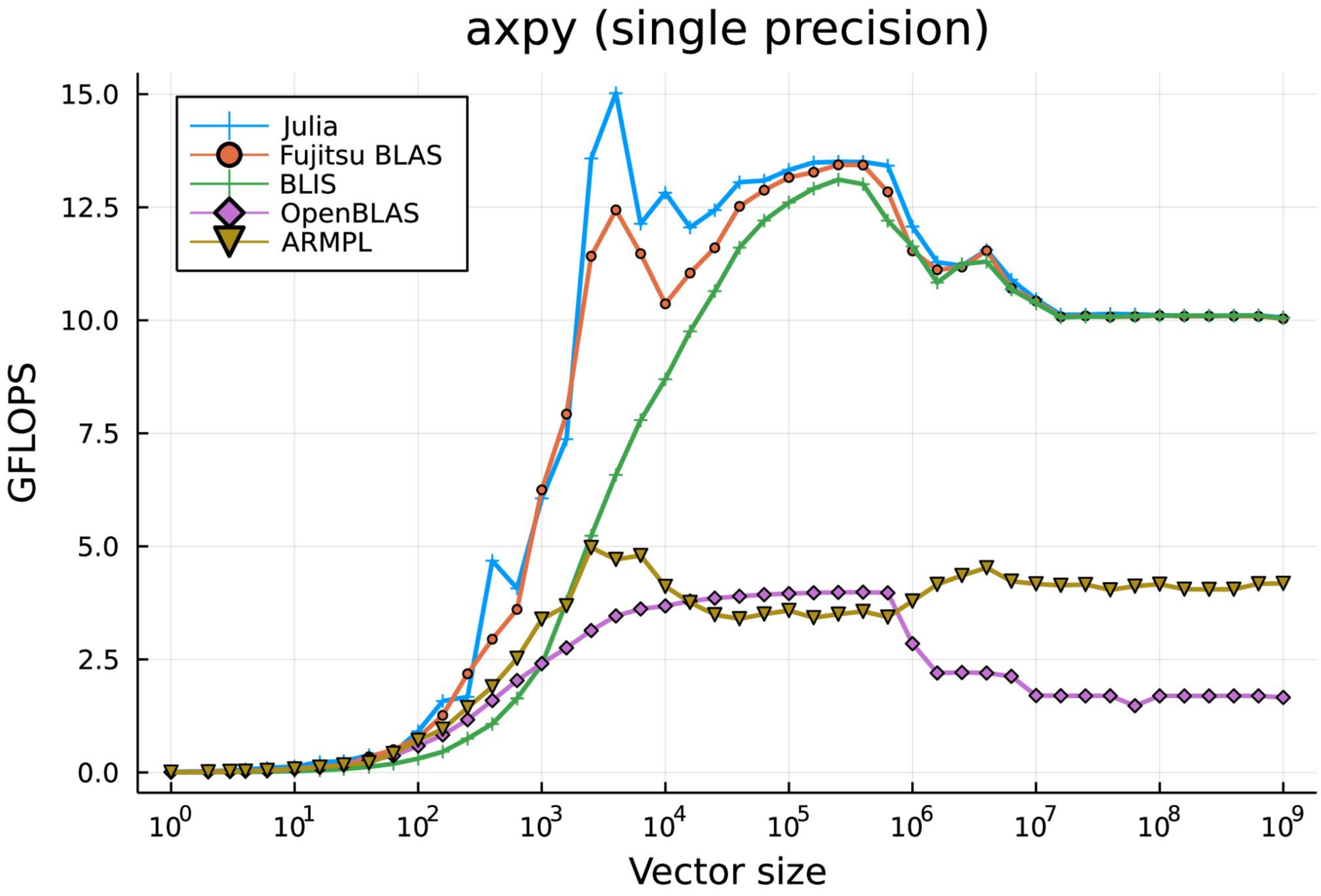
Performance and Scalability on Fugaku

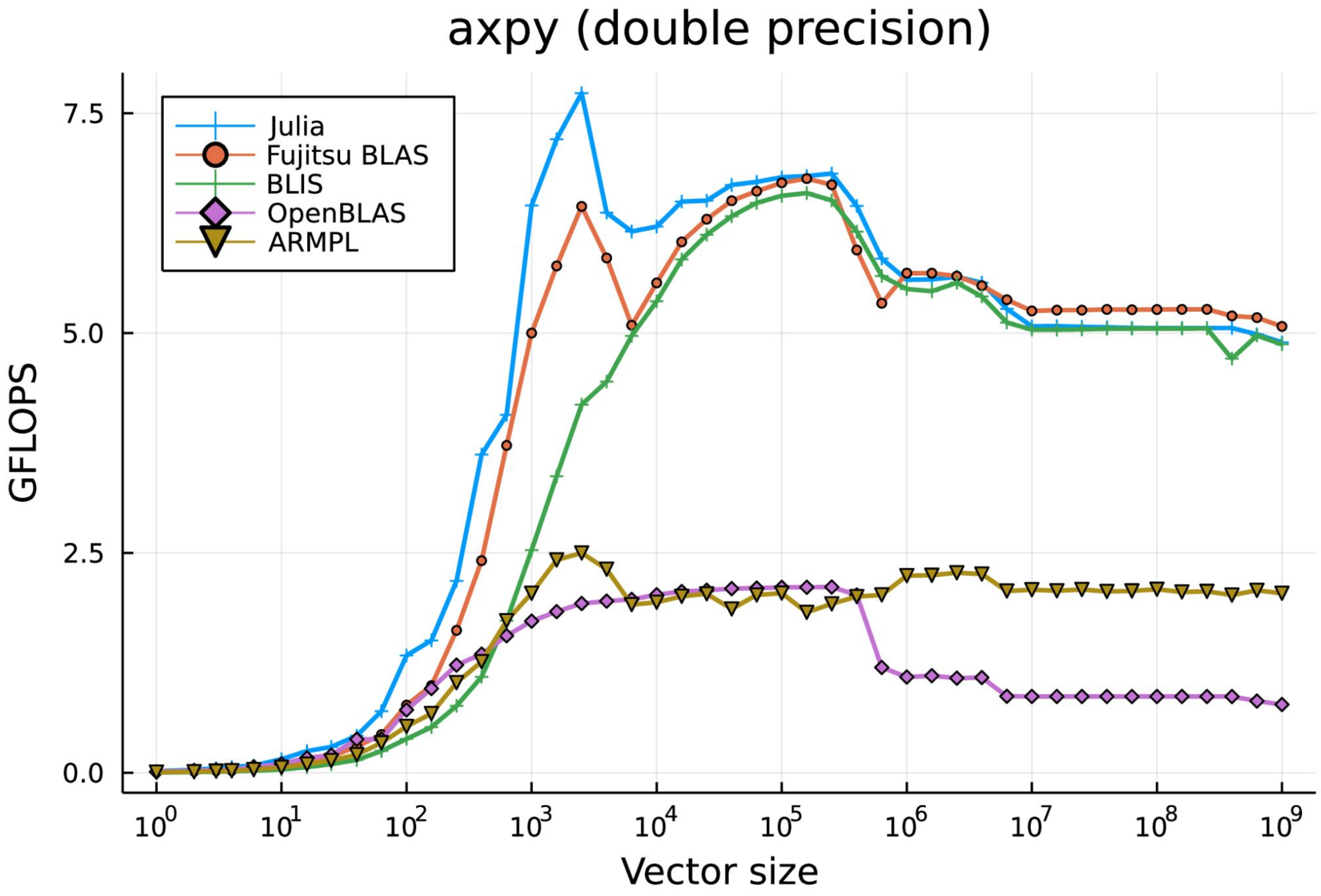
Level 1 BLAS showdown

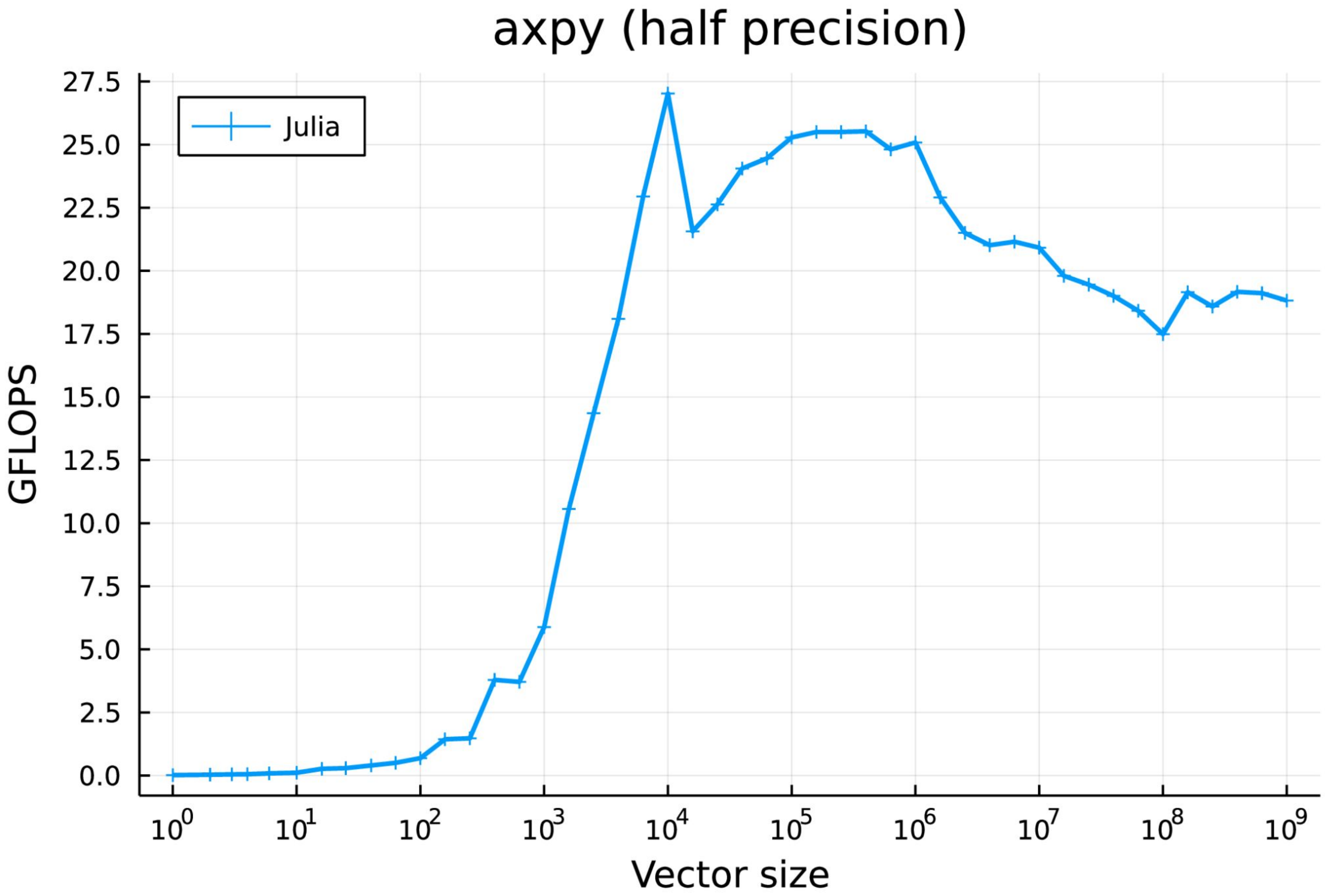
```
function axpy!(a, x, y)
    @simd for i in eachindex(x, y)
        @inbounds y[i] = muladd(a, x[i], y[i])
   end
   return y
end
```

vs

```
LinearAlgebra.BLAS.axpy!(a, x, y)
```


axpy (single precision)
Julia
Fujitsu BLAS
BLIS
OpenBLAS
ARMPL
GFLOPS
15.0
12.5
10.0
7.5
5.0
2.5
0.0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
Vector size

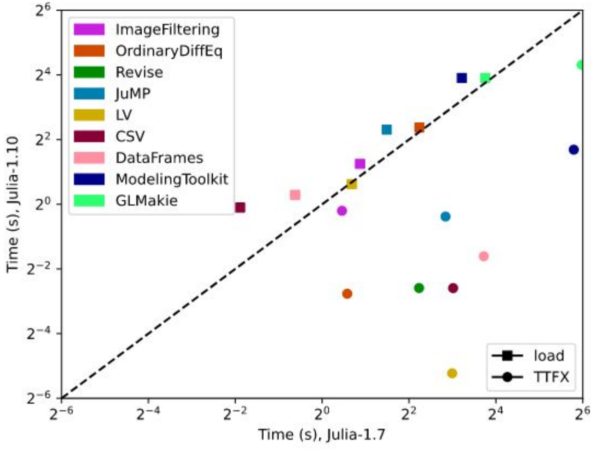

axpy (double precision)
Julia
Fujitsu BLAS
BLIS
OpenBLAS
ARMPL
GFLOPS
0.0
2.5
5.0
7.5
Vector size
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9


axpy (half precision)
Julia
GFLOPS
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
Vector size

Package images — Native code caching

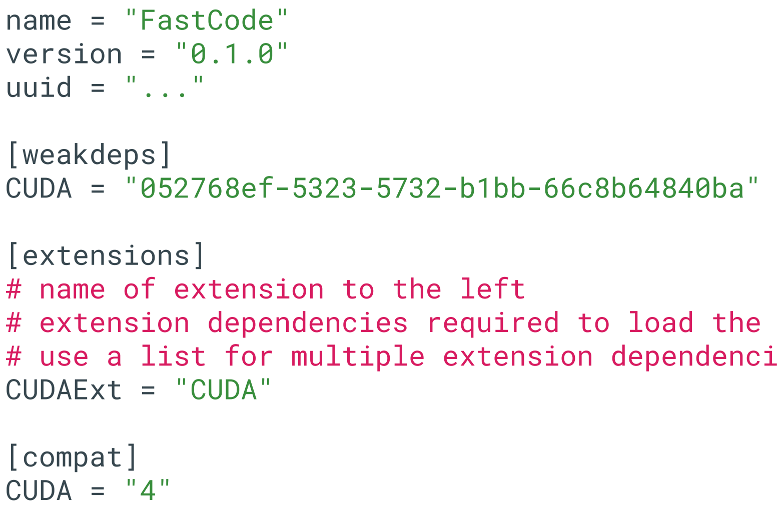
- Julia 1.9 will now by default cache native code during precompilation.
- Tradeoff: Precompilation got slower by quite a bit and cache file size increased.
- As a HPC center you will want to set `JULIA_CPU_TARGET` both during build , but also in your module to enable multi-versioning for both the sysimg as well as package images.

https://docs.julialang.org/en/v1/devdocs/sysimg/#System-image-optimized-for-multiple-microarchitectures



Weak dependencies and package extensions

- GPU backends are often heavy dependencies
- Ideally user would only need one backend, but we often need to add methods to support different backends.
- https://pkgdocs.julialang.org/dev/creating-packages/#Conditional-loading-of-code-in-packages-(Extensions)



```toml
name = "FastCode"
version = "0.1.0"
uuid = "..."

[weakdeps]
CUDA = "052768ef-5323-5732-b1bb-66c8b64840ba"

[extensions]
# name of extension to the left
# extension dependencies required to load the extension to the right
# use a list for multiple extension dependencies
CUDAExt = "CUDA"

[compat]
CUDA = "4"
```