Supporting Julia Users on NERSC’s Cori and Perlmutter Systems

NERSC Data Days 2022
Oct 26-27, 2022

Johannes Blaschke
Data Science Engagement Group, NERSC
The Need for High-Performance Glue Code

- **Objective**: Establish High-Productivity High-Performance Programming Languages

- **Common Design Pattern**: High-Productivity Language (eg. Python) as Glue Code
  - At NERSC: Julia, Python + C/C++/CUDA
  - Pro: Use appropriate language for algorithms requiring high performance
  - Con: N+1-language problem (code maintainability)
  - Con: Context switching between interpreted and compiled languages

<table>
<thead>
<tr>
<th>Function signature</th>
<th>Pybind11</th>
<th>ccall</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>int fn0()</td>
<td>132 ±14.9</td>
<td>2.34 ±1.24</td>
<td>56×</td>
</tr>
<tr>
<td>int fn1(int)</td>
<td>217 ±20.9</td>
<td>2.35 ±1.33</td>
<td>92×</td>
</tr>
<tr>
<td>double fn2(int, double)</td>
<td>232 ±11.7</td>
<td>2.32 ±0.189</td>
<td>100×</td>
</tr>
<tr>
<td>char* fn3(int, double, char*)</td>
<td>267 ±28.9</td>
<td>6.27 ±0.396</td>
<td>42×</td>
</tr>
</tbody>
</table>
Julia Usage Trends at NERSC

- Growing interest in Julia at NERSC:

<table>
<thead>
<tr>
<th>Do you use Julia locally or at NERSC?</th>
<th>Responses</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;I do not use Julia (locally or at NERSC)&quot;</td>
<td>308</td>
<td>74</td>
</tr>
<tr>
<td>&quot;I use Julia locally but not at NERSC&quot;</td>
<td>81</td>
<td>20</td>
</tr>
<tr>
<td>&quot;I use Julia locally and at NERSC&quot;</td>
<td>24</td>
<td>6.8</td>
</tr>
<tr>
<td>&quot;I use Julia at NERSC but not locally&quot;</td>
<td>2</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Do you plan to use Julia in future?
Julia Support at NERSC

- **Objective**: Enable users to “roll their own” Julia install / environment
- Support different “levels” of Julia users:
  a. Provide documentation and use cases
  b. Provide system-wide settings using `Preferences.jl`: user can load a module, and all packages that need vendor libs (MPI, HPF5, etc) gets correctly compiled)
  c. Provide compatibility interfaces, eg. MPItrampoline
  d. Modules include pre-compiled packages in the `JULIA_DEPOT_PATH` and `JULIA_LOAD_PATH`
Julia Usage Trends at NERSC

• Growing use of Julia modules at NERSC

229 module users (04/21-07/22)
Julia Usage Trends at NERSC

- Julia Users like new versions
- Difficult for center software release cycle to keep up with latest Julia version
  - Use CI/CD to keep up to date
  - Enable users to be productive with their own Julia versions
Ongoing and Future Work

- **Detailed Usage Monitoring:** Use `startup.jl` to register `atexit` hook which monitors loaded packages.

  ![Monitoring Scientific Python Usage on a Supercomputer](image)

  Monitoring Scientific Python Usage on a Supercomputer

  Rollin Thomas¹, Laurie Stephey², Annette Greiner³, Brandon Cook⁴

- **Production-Level Support:** Optimize Julia performance (eg. GPUs) on NERSC systems and integrate support into center operations.

- **Advanced Workflow Control:** Explore how workflow managers interact with center resource scheduler (eg. Slurm) *in situ* using API (eg. PMI2).

- **Documentation, Use Cases, and Training**
Noteworthy Julia Packages (for HPC)

- **JuliaIO**: https://github.com/JuliaIO
  JuliaData: https://github.com/JuliaData
  Collects many Julia packages around I/O and Data

- **JuliaParallel**: https://github.com/JuliaParallel
  Collects many Julia packages around distributed and parallel computing

- **JuliaGPU**: https://github.com/JuliaGPU
  Collects many Julia packages used for GPU computing
Noteworthy I/O Packages

- **Pidfile.jl**: Provides the linux/unix pidfile mechanism to hold mutex’es – useful for locking files
- **HDF5.jl**: HDF5-file support
- **Zarr.jl**: Julia Zarr (N-D array compressed data) support
- **JLD.jl / JLD2.jl**: Julia-native serialization support
- **Tables.jl / DataFrames.jl / CSV.jl**: Tabular data support
- **JuliaDB.jl**: A distributed database for tables (implemented in pure Julia)
Noteworthy REST and Web Frameworks

- **HTTP.jl**: Send and receive HTTP requests
- **Mux.jl / Oxygen.jl**: Routing middleware for HTTP requests – Oxygen is newer and makes multithreading easier (considered an all-Julia replacement for FastAPI)
- **Genie.jl**: Fully-fledged web development framework (Julia’s answer to Flask)
Noteworthy HPC Packages

“Traditional” HPC support: (https://github.com/JuliaParallel)

- **MPI.jl**: no explanation needed (it is CUDA/ROCM-aware)
- **ClusterManagers.jl**: manager HPC resources on the fly (also note SlurmClusterManager.jl and MPIClusterManagers.jl for HPC clusters)
- **ImplicitGlobalGrid.jl / MPIArrays.jl**: implement a global address space (using the Array interface) built on MPI.jl
Noteworthy HPC Packages

Tasking (producer-consumer) style HPC support: (https://github.com/JuliaParallel)

• **Distributed.jl / Dagger.jl**: task-based parallelism (like Dask and Ray)
• **DTables.jl / DistributedArrays.jl**: arrays and tables build on distributed

ML support: **Flux.jl** (like pytorch, but different)
Noteworthy HPC Packages

GPU Support:
(https://github.com/JuliaGPU)

- **CUDA.jl / AMDGPU.jl / oneAPI.jl**: low-level GPU support (expose GPU Array interface + helper functions to manage GPU resources)
- **KernelAbstractions.jl**: lets you write portable code by writing portable kernels (a bit “like” Kokkos)
- + Many Many more
Demo Time!
(sources: https://jblaschke.github.io/HPC-Julia/)