“HDF5 as a critical component in the Julia HPC ecosystem”

BoF: HDF5: Building on 25 Years of Success

William F Godoy

Oak Ridge National Laboratory
Contents

• Motivation, about DOE Facilities

• Julia value proposition

• HDF5.jl

• Lessons and Final Thoughts
Motivation

- The US Department of Energy (DOE) computing user facilities are “heterogeneous” GPU-accelerated systems
- Oak Ridge Leadership Computing Facility (OLCF): Summit, Frontier
- Argonne Leadership Computing Facility (ALCF): Theta, Aurora (coming soon)
- National Energy Research Scientific Computing Center (NERSC): Cori, Pelmutter
- We are driven and funded by science https://nationallabs.org/our-labs/what-we-do/
- Fortran and CPUs are still very important!
- More data is produced more than ever!
Rethink how we do Computing

- Scientific programming is HARD (specially on our Leadership Computing Facilities, LCFs)
- Software is our “specialized science equipment” for science
- There is still a lot of plumbing to be done
- Programming productivity is always a challenge
- Barrier to entry from idea to portable performance
- AI/ML+HPC is a multidisciplinary co-design challenge
- How to leverage US DOE Exascale Computing Project legacy?

“Can a machine translate a sufficiently rich mathematical language into a sufficiently economical program at a sufficiently low cost to make the whole affair feasible?” --- Backus on Fortran (1980)

Key question: “What novel approaches to software design and implementation can be developed to provide performance portability for applications across radically diverse computing architectures?” Reimagining Codesign for Advanced Scientific Computing: Unlocking Transformational Opportunities for Future Computing Systems for Science. DOE Report https://doi.org/10.2172/1822198
I/O Performance is more than POSIX write/reads

- Even with a performance language processing + I/O can be “slow” (fragile)
- The NeXus schema on top of HDF5 is used across several neutron facilities

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Entry Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>/entry</td>
</tr>
<tr>
<td>attribute</td>
<td>/entry/NX_class</td>
</tr>
<tr>
<td></td>
<td>/entry/NX_class/attribute</td>
</tr>
<tr>
<td>group</td>
<td>/entry/DASlogs</td>
</tr>
<tr>
<td>attribute</td>
<td>/entry/DASlogs/NX_class/attribute</td>
</tr>
<tr>
<td></td>
<td>/entry/DASlogs/BL6_CS/DataType/NX_class</td>
</tr>
<tr>
<td>dataset (SDS)</td>
<td>/entry/DASlogs/BL6_CS/DataType/average_value</td>
</tr>
<tr>
<td>dataset (SDS)</td>
<td>/entry/DASlogs/BL6_CS/DataType/average_value_error</td>
</tr>
<tr>
<td>group</td>
<td>/entry/bank1_events</td>
</tr>
<tr>
<td>attribute</td>
<td>/entry/bank1_events/NX_class/attribute</td>
</tr>
<tr>
<td>dataset (SDS)</td>
<td>/entry/bank1_events/event_id</td>
</tr>
<tr>
<td>dataset (SDS)</td>
<td>/entry/bank1_events/event_index</td>
</tr>
<tr>
<td></td>
<td>/entry/bank1_events/attribute</td>
</tr>
<tr>
<td></td>
<td>/entry/bank1_events/NX_class/attribute</td>
</tr>
<tr>
<td></td>
<td>/entry/bank1_events/event_id</td>
</tr>
<tr>
<td></td>
<td>/entry/bank1_events/event_index</td>
</tr>
</tbody>
</table>

TABLE I: Schematic representation of the hierarchical NeXus schema [5] for recorded raw event-based neutron data.

- Big Data workflows are very sensitive to I/O + compute interactions

Fig. 5: Mantid’s “LoadEventNexus” CPU profiling flame graph representation for (a) Mantid v5.0, (b) Mantid latest implementation with our proposed strategy. The reduction of metadata-related CPU operations bottlenecks is illustrated in this comparison.
Julia's value proposition for HPC

- Designed for “scientific computing” (Fortran-like) and “data science” (Python-like) with **performant kernel code via LLVM compilation**
- Lightweight **interoperability** with existing Fortran and C libraries
- Julia is a **unifying workflow language** with a coordinated ecosystem

"Julia does not replace Python, but the costly workflow process around Fortran+Python+X, C+X, Python+X or Fortran+X (e.g. GPUs)"

where $X = \{\text{conda, pip, pybind11, Python, C, Fortran, C++, OpenMP, OpenACC, CUDA, HIP, CMake, numpy, scipy, matplotlib, Jupyter, ...}\}$

---

https://pde-on-gpu.vaw.ethz.ch/lecture7


https://quantumzeitgeist.com/learning-the-julia-programming-language-for-free/

https://juliadatascience.io/
Gray-Scott app: [https://github.com/JuliaORNL/GrayScott.jl](https://github.com/JuliaORNL/GrayScott.jl)

Simple 3D 2-variable diffusion-reaction solver
- CPU Threads, CUDA.jl and AMDGPU.jl backends using multiple dispatch
- Parallel I/O, can be visualized with ParaView
- MPI.jl for communication
- Configuration and job scripts for Frontier, Crusher and Summit under ./scripts/
- Data analysis on JupyterHub

Research Question: Can we write an entire HPC workflow in Julia?

\[
\begin{align*}
\frac{\partial U}{\partial t} &= D_U \nabla^2 U - U V^2 + F (1 - U) + nr \\
\frac{\partial V}{\partial t} &= D_V \nabla^2 V + UV^2 + -(F + k) V
\end{align*}
\]
Data analysis on JupyterHub at OLCF

- [https://jupyter.olcf.ornl.gov/](https://jupyter.olcf.ornl.gov/)
- We launched a Julia kernel on JupyterHub to read and analyze data (ADIOS2.jl)
- Visualize with Makie.jl
- JIT and TTFX (time to first plot) can be a nuance
- Need to support HDF5 via HDF5.jl

HDF5.jl: Julia bindings to HDF5

- Get all the benefits from HDF5 and Julia
- Low-level and high-level APIs
- Unlike Python, you can write a performant workflow in a single language (I/O + computation)

```julia
julia> h5open("example2.h5", "w") do fid
    g = create_group(fid, "mygroup")
    dset = create_dataset(g, "myvector", Float64, (10,))
    write(dset, rand(10))
end
```
HDF5.jl: Julia bindings to HDF5, interesting features

- Optionally reuse underlying HDF5 system-installed library (HPC)
- Julia has types, so memory alignment happens naturally
- Map compound types -> Julia types

In-memory HDF5 files

It is possible to use HDF5 files without writing or reading from disk. This is useful when receiving or sending data over the network. Typically, when sending data, one might want to:

1. Create a new file in memory. This can be achieved by passing `Drivers.Core(backing_store=false)` to `h5open(...)`
2. Add data to the HDF5.File object
3. Get a representation of the file as a byte vector. This can be achieved by calling `Vector{UInt8}(...)` on the file object.

Using HDF5

```julia
# Creates a new file object without storing to disk by setting `backing_store=false`
file_as_bytes = h5open("AnyName_InMemory", "w"; driver=Drivers.Core(backing_store=false)) do fid
    fid["MyData"] = randn(5, 5) # add some data
    return Vector{UInt8}(fid) # get a byte vector to send, e.g., using HTTP, MQTT or similar.
end
```

Supported data types

HDF5.jl knows how to store values of the following types: signed and unsigned integers of 8, 16, 32, and 64 bits, Float32, Float64; Complex versions of these numeric types; Arrays of these numeric types (including complex versions); String; and Arrays of String. Arrays of strings are supported using HDF5’s variable-length-string facility. By default, Complex numbers are stored as compound types with `r` and `i` fields following the `h5py` convention. When reading data, compound types with matching field names will be loaded as the corresponding Complex Julia type. These field names are configurable with the `HDF5.set_complex_field_names` and `AbstractString(Imag)`, and `HDF5.enable/disable_complex_SUPPORT()` function and complex support can be completely enabled/disabled with `HDF5.enable/disable_complex_SUPPORT()`.

As of HDF5.jl version 0.16.13, support was added to map Julia structs to compound HDF5 datatypes.

```julia
julia> struct Point3(T)
    x::T
    y::T
    z::T
end
julia> datatype(Point3(Float64))
HDF5.datatypes = HST.COMPARABLE {
    HST_EXP64 "x" := 0;
    HST_EXP64 "y" := 0;
    HST_EXP64 "z" := 16;
}
```
Final Thoughts and Future Work

• HDF5 and Julia are for science 😊
• HDF5.jl is a key component for Julia adoption where HDF5 is crucial (e.g. facilities)
• Community is everything!
• Julia has a lot of potential for HPC use: proxy app, rapid prototyping, unifying ecosystem, productivity, AI workflows ➔ data
• High-performance + high-productivity options based on LLVM (e.g. Numba, Mojo, Julia)
• Efforts around Julia and Generative AI using large language models - LLMs

Possible on Frontier, gaps to be closed

ICPP23 P2S2

Evaluation of OpenAI Codex for HPC Parallel Programming Models Kernel Generation
William F. Godoy, Pedro Valero-Lara, Keita Teranishi, Prasanna Balaprakash, and Jeffrey S. Vetter
Oak Ridge National Laboratory
Oak Ridge, TN, USA
{godoywf, valerolara, teranishi, pbalaprash, jsvetter}@ornl.gov

Julia for HPC Birds of a Feather at SC23 (Thursday) and Research Poster

Julia for HPC
The "Julia for HPC" Birds of a Feather (BoF) session provides a place for the high-performance computing (HPC) community with interests in the Julia programming language. Julia provides an integrated development environment and co-design models as a LLVM front-end for scientists to close the gap between high-productivity languages and the shared performance of traditional, compiled languages on extreme heterogeneous systems.

https://juliaornl.github.io/TutorialJuliaHPC/

LCPC23

Comparing Llama-2 and GPT-3 LLMs for HPC kernels generation
Pedro Valero-Lara, William F. Godoy, Keita Teranishi, Prasanna Balaprakash, and Jeffrey S. Vetter
Oak Ridge National Laboratory
Oak Ridge, TN, USA
{valerolara, godoywf, teranishi, pbalaprash, jsvetter}@ornl.gov
ACKNOWLEDGMENTS

The Exascale Computing Project, PROTEAS-TUNE sub-project.

The ASCR Bluestone Project

The OMNI Technology Alliance and Sustainable Research Pathways (SRP) Internship Programs

Julia packages stewards: Julian Samaroo, Valentin Churavy, Tim Besard, Simon Byrne, Lucas Wilcox, Erik Schnetter and the Julia for HPC community members

SC23 activities

D-HPC 2023
The First International Workshop on Democratizing High-Performance Computing
held in conjunction with SC23, The International Conference on High-Performance Computing, Networking, Storage, and Analysis
Denver, Colorado
November 19th-22nd, 2023
Workshop date: Sunday November 12th, 2:00pm-3:30pm MST - Room 601

Presentation
Julia for HPC
The "Julia for HPC" Birds-of-a-Feather (BoF) session provides a place for the high-performance computing (HPC) community with interests in the Julia programming language. Julia proposes an integrated development and end-to-end code model as a LLVM front-end for science to close the gap between high-productivity languages and the desired performance of traditional compiled languages on extreme heterogenous systems.

Best paper award at SC23@WORKS

Thanks to the audience and HDF5 BoF

Julia for HPC BoF Thursday 12:15pm

SC23 National Laboratory