JULIA FOR HPC ON OLCF SYSTEMS

William F Godoy, Pedro Valero-Lara, Philip Fackler

Computer Science and Mathematics Division
Advanced Computing Systems Research Section

Prepared for: SEPTEMBER 2022 OLCF USER CONFERENCE CALL 09/27/2022
Contents

• Julia’s value proposition for HPC: LLVM + Coordinated Ecosystem

• Community Efforts in HPC

• Running on OLCF System, preliminary results, opportunities

• Resources: where to get started?

• Final thoughts and acknowledgments
Contents

• Julia’s value proposition for HPC: LLVM + Coordinated Ecosystem

• Community Efforts in HPC

• Running on OLCF System, preliminary results, opportunities

• Resources: where to get started?

• Final thoughts and acknowledgments
Landscape of computing: The Tower of Babel

- Scientific software:
  - Performance languages: C, C++, Fortran
  - High-level productivity languages: Python, Matlab, R

- Plethora of programming models for heterogeneous computing. Standard, vendor-specific, third party

- Major shift: vendor compiler convergence around LLVM

- Slowdown in Moore's Law cadence puts more focus on massively parallel, vectorized computing: ARM, NVIDIA/AMD GPU, Intel’s Xeon, KNL, Sapphire Rapids, AVX-512

- AI/ML + traditional HPC requires powerful reproducible programming abstractions for computation, communication and data

- Choosing a programming model is not always a technical decision

LLVM: a game changer  


LLVM Compiler Infrastructure  
[LLVM: a game changer](https://llvm.org/)  
[LLVM Compiler Infrastructure](http://www.aosabook.org/en/llvm.html)  

LLVM Typed Static Single Assignment (SSA) Intermediate Representation (IR) aka LLVM-IR: [https://patshaughnessy.net/2022/2/19/llvm-ir-the-esperanto-of-computer-languages](https://patshaughnessy.net/2022/2/19/llvm-ir-the-esperanto-of-computer-languages)

%57 = call %"Array(Int32)** @""Array(Int32)@Array(T)::unsafe_build:Array(Int32)"(i32 610, i32 2), !dbg !89
LLVM: Vendor and programming model adoption

Intel

ARM

NVIDIA

Apple

AMD

Intel C/C++ compilers complete adoption of LLVM

The next generation Intel C/C++ compilers are even better because they use the LLVM open source infrastructure.

ARM Compiler Builds on Open Source LLVM Technology
April 08, 2014

Velocity of open source Clang and LLVM combined with the stability of commercial products improve code quality, performance and power efficiency on ARM processors

NVIDIA CUDA 4.1 Compiler Now Built on LLVM

By Chris Lattner
Dec 19, 2011

CUDA LLVM Compiler

NVIDIA's CUDA Compiler (NVCC) is based on the widely used LLVM open source compiler infrastructure. Developers can create or extend programming languages with support for GPU acceleration using the NVIDIA Compiler SDK.

Fast and powerful

From its earliest conception, Swift was built to be fast. Using the incredibly high-performance LLVM compiler technology, Swift code is transformed into optimized machine code that gets

AMD Updates ROCm™ For Heterogenous Software Support

Community support continues to grow for AMD Radeon Open eCosystem (ROCm™). AMD's open source foundation for heterogeneous compute. Major development milestones in the latest update include:

- The HIP-Clang compiler is now up-streamed and reviewed by the LLVM® community, providing a better open source experience for the developer.
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 ECP 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?” from Reimagining Codesign for Advanced Scientific Computing: Unlocking Transformational Opportunities for Future Computing Systems for Science. DOE Report https://doi.org/10.2172/1822198
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 = { conda, pip, pybind11, cython, Python, C, Fortran, C++, OpenMP, OpenACC, CUDA, HIP, CMake, numpy, scipy, matplotlib, Jupyter, …}

---

https://juliadatascience.io/


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

https://juliadatascience.io/

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

- History: started at MIT in the early 2010s (predates Python Numba)
  
  https://julialang.org/blog/2022/02/10years/

- Julia Computing is a major contributor:
  
  https://juliacomputing.com/case-studies

- First stable release v1.0 in 2018, v1.8 as of 2022
  
  https://julialang.org/

- Open-source GitHub-hosted packages and ecosystem with MIT permissive license:
  
  https://github.com/JuliaLang/julia

- Community: annual JuliaCon conference (next week):
  
  https://juliacon.org/2022/
  
  https://live.juliacon.org/agenda/2022-07-19
Julia Brief Walkthrough

- Reproducibility is in the core of the language:
  - Interactive: Jupyter, Pluto.jl
  - Packaging Pkg.jl
  - Environment Project.toml
  - Testing Test.jl

- Just-in-time or Ahead-of-time compilation with PackageCompiler.jl

- Powerful metaprogramming for code instrumentation: @profile, @time, @testset, @test, @code_llvm, @code_native, @inbounds,

- Interoperability is key: @ccall, @cxx, PyCall, CxxWrap.jl

https://github.com/ornl-training/julia-basics/tree/main/notebooks/MPI-Heat-Transfer-2D
<table>
<thead>
<tr>
<th></th>
<th>Fortran</th>
<th>C</th>
<th>C++</th>
<th>Python</th>
<th>Julia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package manager</td>
<td>fpm, Too many</td>
<td>Too many</td>
<td>Too many</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt; party: pip, conda</td>
<td>Pkg.jl</td>
</tr>
<tr>
<td>Environment Reproducibility</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt; party: pyproject.toml</td>
<td>Project.toml</td>
</tr>
<tr>
<td>Metaprogramming</td>
<td>fpp</td>
<td>macros #</td>
<td>Templates, macros #</td>
<td>decorators @</td>
<td>macros @</td>
</tr>
<tr>
<td>Performance</td>
<td>Compiled</td>
<td>Compiled</td>
<td>Compiled</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt; party: numba (LLVM), numpy, pybind11, Cython</td>
<td>JIT and/or AOT compiled: PackageCompile r.jl</td>
</tr>
<tr>
<td>Testing</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt; party</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt; party</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt; party</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt; party: pytest, unittest,</td>
<td>@testset</td>
</tr>
<tr>
<td>TIOBE index (Sep 2022)</td>
<td>15 (17)</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>21 (27)</td>
</tr>
</tbody>
</table>
Contents

• Julia’s value proposition for HPC: LLVM + Coordinated Ecosystem

• Community Efforts in HPC

  • Running on OLCF System, preliminary results, opportunities

  • Resources: where to get started?

  • Final thoughts and acknowledgments
CCSD efforts

• Projects:
  – **ECP PROTEAS-TUNE**: research performance on Exascale system of different programming models including Python Numba and Julia (W Godoy, J Vetter). SRP-HPC mentor
  – **ECP Proxy Apps**: evaluating Julia as a proxy to understand parallel I/O characteristics (W Godoy, P Fackler, G Watson) RIOPA.jl presented at JuliaCon 2022

• External Engagements with Julia HPC
  – Monthly meetings with stakeholders
  – JuliaCon 2022 HPC minisymposium
  – Julia HPC Position paper in the works
  – ECP BoF Session on Rapid Prototyping for HPC using Julia, Python Numba, Flang
  – SC22 BoF “Julia for HPC”

• Internal Engagements
  – Tutorials at ORNL Software and Data Expo
  – Julia Workshop for ORNL Science
Community Efforts in HPC

- Leverage HPC “backends”:
  - AMDGPU.jl
  - CUDA.jl
  - KernelAbstractions.jl
  - MPI.jl
  - Threads (part of Base)
  - ADIOS2, HDF5

- Monthly HPC Call (Valentin Churavy, MIT)

- Porting miniWeather App to Julia (Youngsung Kim, Hyun Kang, and Sarat Sreepathi, CSED)

- https://ptsolvers.github.io/GPU4GE/O/software/

- https://anxv.org/abs/2207.03711

- https://github.com/omlins/julia-gpu-course

- https://enccs.github.io/Julia-for-HPC

- https://docs.dftk.org/stable

- https://juliaastro.github.io/dev

- https://github.com/JuliaParallel

- Top15 most popular packages

- ECP ExaSDG on Summit

- Rapid Prototyping with Julia: From Mathematics to Fast Code
  By Michel Scekic, Valentin Churavy, Youngsung Kim, and Mikio Abresch
  Software development—a dominant expenditure for scientific projects—is often hindered by technical programming challenges, not mathematical insight. Here we share our experience with the Julia programming language in the context of the U.S. Department of Energy’s Exascale Computing Project (ECP) as part of ExaSDG, a power grid optimization application. Julia is a free and open-source language that has the potential for 1-2x performance improvement over C++.
Contents

• Julia’s value proposition for HPC: LLVM + Coordinated Ecosystem

• Community Efforts in HPC

• Running on OLCF System, preliminary results, opportunities

• Resources: where to get started?

• Final thoughts and acknowledgments
Performance results on GPU
Results for Matrix Multiplication

Wombat (ARM)
NVIDIA A100 GPU

Crusher (AMD)
MIX250X


https://github.com/luraess/ROCm-MPI

Julia and Python’s Numba kernel portability, performance, and productivity on heterogeneous exascale nodes

Oak Ridge National Laboratory
{godoywf}, {valerolarag}, {dettlinge}, {trefftz}, {jorquerai}, {sheehy}, {rmmiller}, {gonzalez}, {vetter}@ornl.gov
Interactive computing on JupyterHub with Julia

(v1.6) pkg> add IJulia
... (Or install IJulia any other way).
julia> using IJulia
julia> installkernel("Julia (16 threads)", env=Dict("LD_LIBRARY_PATH" => ",
"JULIA_NUM_THREADS"="16"))

...or use Pluto.jl on a terminal
Contents

• Julia’s value proposition for HPC: LLVM + Coordinated Ecosystem

• Community Efforts in HPC

• Running on OLCF System, preliminary results, opportunities

• Resources: where to get started?

• Final thoughts and acknowledgments
Where to get started?

- Pick a gentle tutorial: [https://techytok.com/from-zero-to-julia/](https://techytok.com/from-zero-to-julia/)
- [https://github.com/ornl-training/julia-basics](https://github.com/ornl-training/julia-basics) (training by WF Godoy Fackler)
- Use VS Code as the official IDE + debugger (not Juno)
- JuliaCon talks are available on YouTube
- [https://discourse.julialang.org/](https://discourse.julialang.org/) Stackoverflow might be outdated, [https://julialang.slack.com/](https://julialang.slack.com/)
- Julia docs and standard library: [https://docs.julialang.org/en/v1/](https://docs.julialang.org/en/v1/)
- Learn: Project.toml, Testing.jl @testset @test, Pluto.jl, CUDA.jl/AMDGPU.jl, LinearAlgebra.jl, Makie.jl, Plots.jl and Flux.jl (AI/ML), how to build a sysimage
- **Pick problems you care about! Let us know if you’re interested in a hackathon.**
- Patience and community reliance: learning a language is a big investment.

Contents

• Julia’s value proposition for HPC: LLVM + Coordinated Ecosystem

• Community Efforts in HPC

• Running on OLCF System, preliminary results, opportunities

• Resources: where to get started?

• Final thoughts and acknowledgments
Final Thoughts

- Results from registered participants at ORNL JuFOS workshop: [https://ornl.github.io/events/jufos2022/](https://ornl.github.io/events/jufos2022/)

**Why are you interested in the event?**

101 responses

- Learn more about Julia: 92 (91.1%)
- Seek alternatives to complex F...: 53 (52.5%)
- Find out more about performan...: 54 (53.5%)
- Connect with other ORNL com...: 41 (40.6%)
- Presenting at JuliaCon: 7 (6.9%)
- Rapid prototyping with high-lev...: 42 (41.6%)
- Simulating large complex quant...: 1 (1%)
- usage of Julia in HPC: 1 (1%)
- Testing the registration form: 1 (1%)
- Learn about the portability of J...: 1 (1%)
- I admin a cluster that has users...: 1 (1%)
Final Thoughts

• Can Julia (as a LLVM DSL for science) help scientists invest more time in science?

• Can Julia help bridging social barriers between AI, data science and HPC?

• Should DOE invest more in a DSL for science, as it did with Fortran, but for our needs in 2022?

• Can performance gaps: JIT, garbage collection, “time-to-first-plot” be leveraged?

• What’s the language and community roadmap, locally and externally?

• What’s the ROI for ORNL scientists adopting Julia?

• Personal reasons:
  
  • Julia is Fortran for arrays and math 😊 with a rich Standard Library: https://docs.julialang.org/en/v1/
  • LLVM is here to stay for performance. Threads, GPU….fast!
  • No object-oriented, no complex C++ templates, no “dependency hell”
  • I dedicate more time to write tests than figuring out language syntax and environment related bugs
  • Pluto.jl: no messing with conda environments or kernels for Jupyter
  • Software is more “to the point” as expected from a specialized piece of equipment
  • People: community is really active, helpful, and enthusiastic
Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Thanks to Ross Miller for enabling Julia on Wombat, thanks to Youngsung Kim, Hyun Kang, and Sarat Sreepathi from CSED for sharing their Julia experiences. The Julia for HPC community.

Suzanne Parete-Koon and Michael Sandoval for the invitation.

JuFOS organizers:
- William F Godoy (p)
- Pedro Valero-Lara (p)
- Philip Fackler (p)
- Greg Watson
- Jeff Vetter (p)
- Theresa Ahearn
- Donna Wilkerson

JuFOS Presenters:
- Youngsung Kim
- Ada Sedova
- Gavin Wiggins
- Jean-Luc Fattebert
- Elise Dettling
- Alexia Arthur
- Singanallur Venkatakrishnan
- Christian Trefftz
- John Gounley

JuFOS Sponsors:
The Exascale Computing Project, PROTEAS-TUNE, Proxy App and SRP-HPC sub-projects.
The ASCR Bluestone Project

Thanks to the audience!