OAK RIDGE National Laboratory

Julia and JACC.jl for easy CPU/GPU programming

William F Godoy, Pedro Valero-Lara, Het Mankad, **Keita Teranishi**, and Jeffery Vetter

Computer Science and Mathematics Division

Advanced Computing Systems Research Section

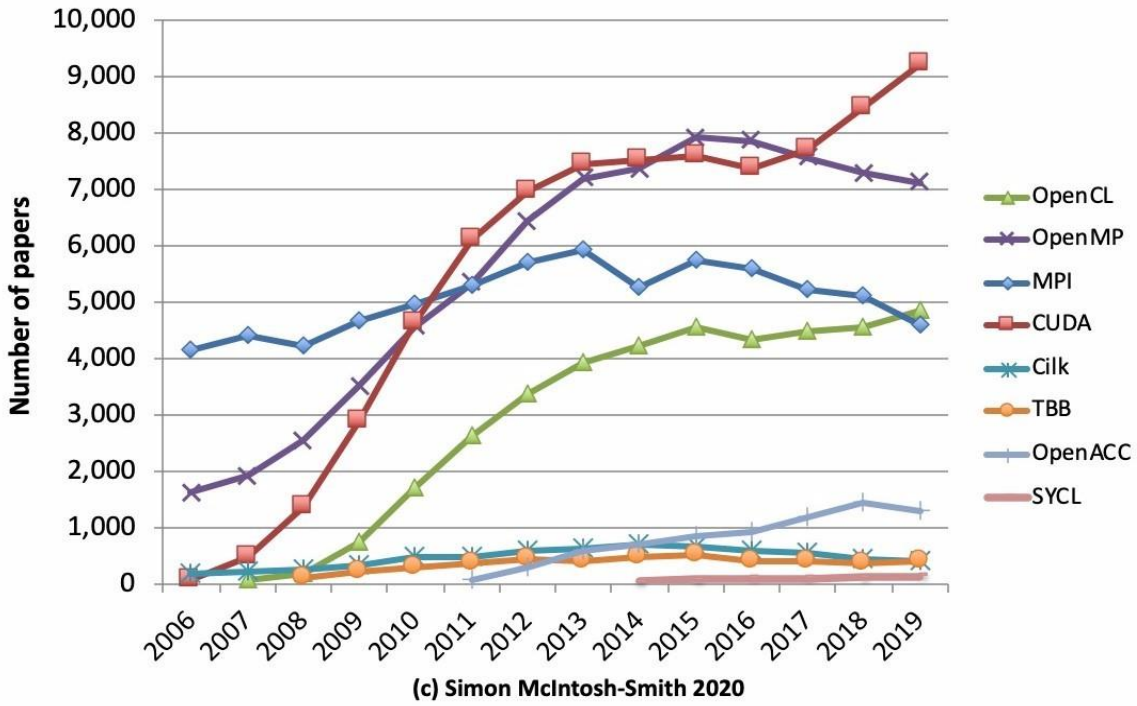
July 30th 2024,

Contents

Contents

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



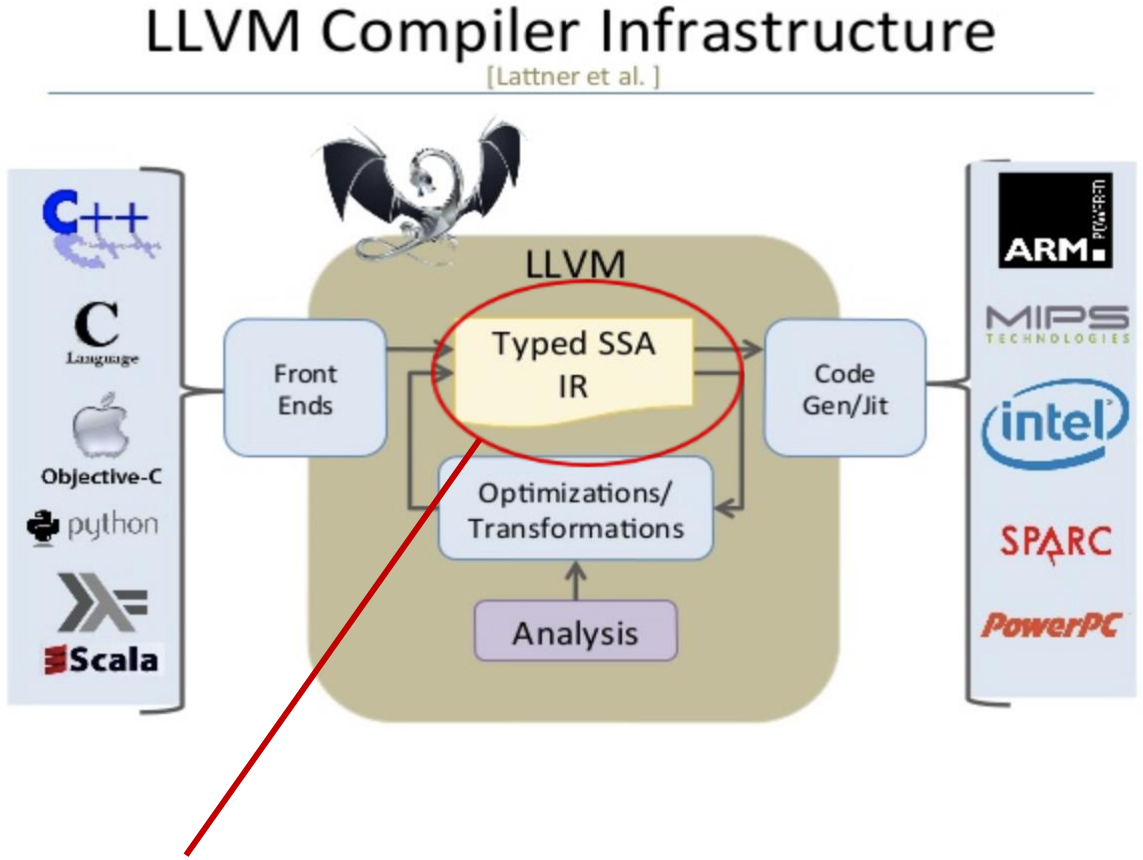
https://www.nextplatform.com/2020/04/28/programming-in-the-parallel-universe/

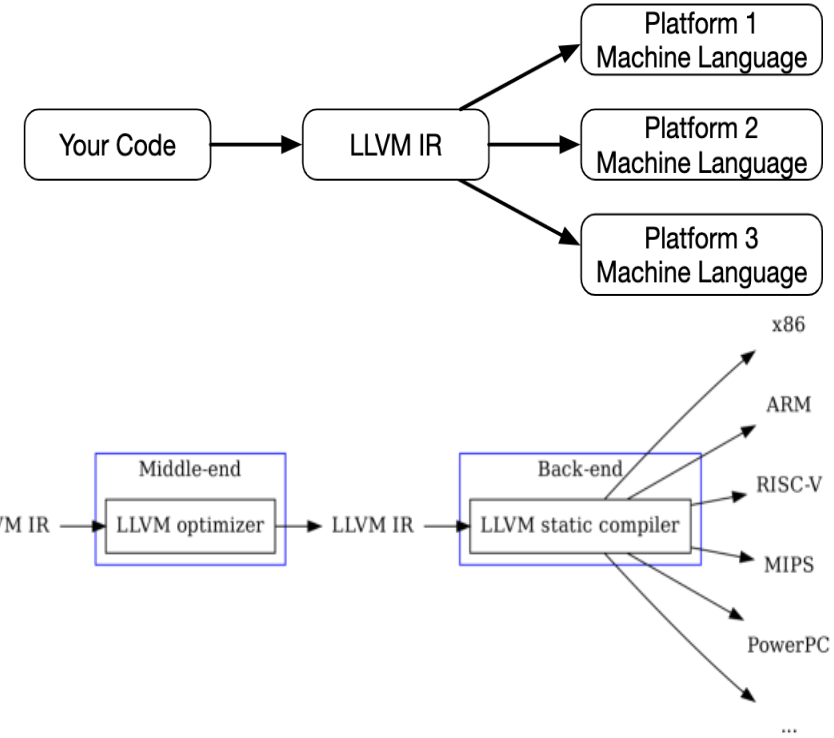
LLVM: a game changer

https://llvm.org/

http://www.aosabook.org/en/llvm.html

C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis & transformation," *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004, pp. 75-86, https://doi.org/10.1109/CGO.2004.1281665 .





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

%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

Developer / Tools / oneAPI / Components / Intel® oneAPI DPC++/C++ Compiler / Intel® Compilers Adopt LLVM

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.

oneAPI

SYCL™

https://www.ornl.gov/project/proteas-tune

ARM

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

arm

OpenACC
Directives for Accelerators

https://csmd.ornl.gov/project/clacc

OpenMP

https://openmp.llvm.org

NVIDIA

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.

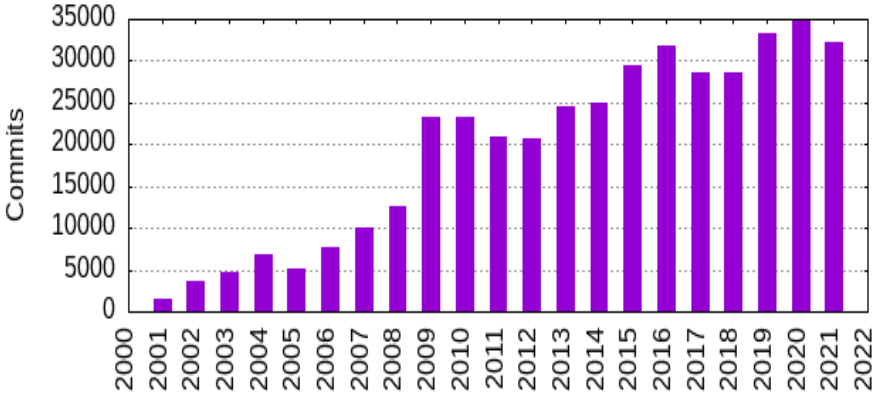
NVIDIA CUDA

LLVM Commits

Apple

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

AMD Updates ROCm™ For Heterogenous Software Support

Community support continues to grow for AMD Radeon Open eCosystem (ROCm™), AMD's open source foundation for heterogenous 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,

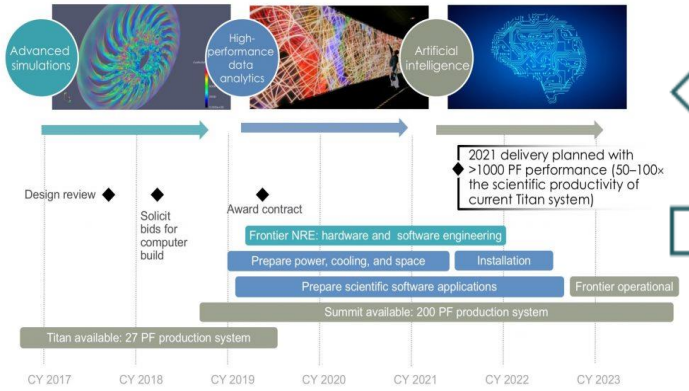
AMD ROCm

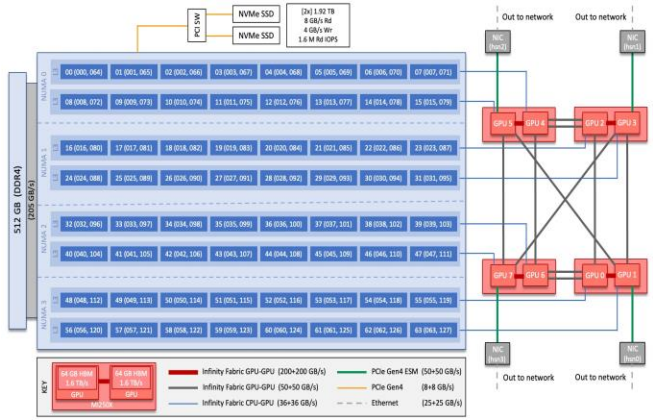
OAK RIDGE National Laboratory

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)

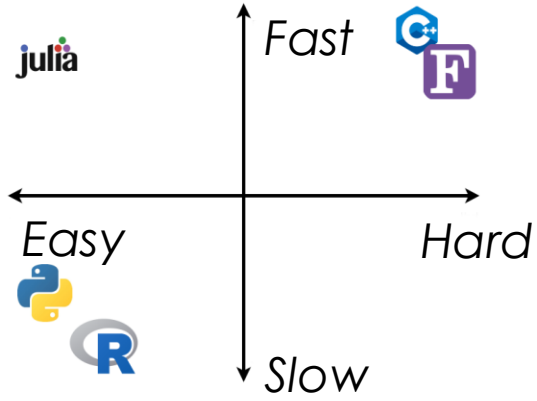




Key question: “What novel approaches to software design and implementation can be developed to provide performance portability for applications across radically diverse computing architectures?” from Reimagining Codesign for Advanced Scientific Computing: Unlocking Transformational Opportunities for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

Julia's value proposition for HPC

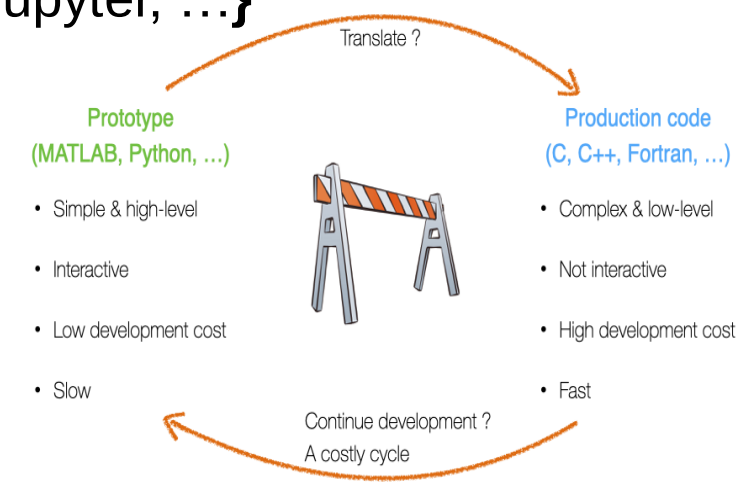
- Designed for "scientific computing" (Fortran-like) and "data science" (Python-like) with **performant kernel code via LLVM compilation**
- Lightweight **interoperability with existing Fortran and C libraries**
- Julia is a **unifying workflow language** with a **coordinated ecosystem**



https://juliadatascience.io/

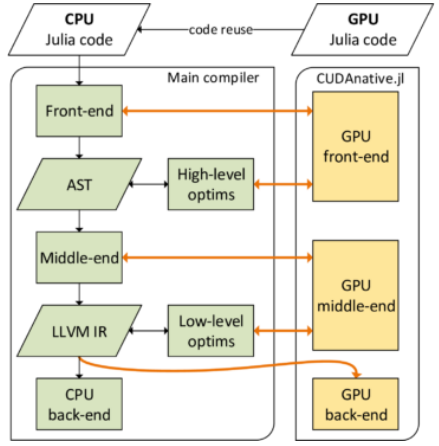
"Julia **does not** replace Python, but the costly workflow process around **Fortran+Python+X**, **C+X**, **Python+X** or **Fortran+X** (e.g. GPUs)"

where X = { conda, pip, pybind11, cython, Python, C, Fortran, C++, OpenMP, OpenACC, CUDA, HIP, CMake, numpy, scipy, matplotlib, Jupyter, …**}**



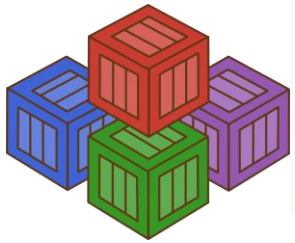
https://pde-on-gpu.vaw.ethz.ch/lecture7

LLVM

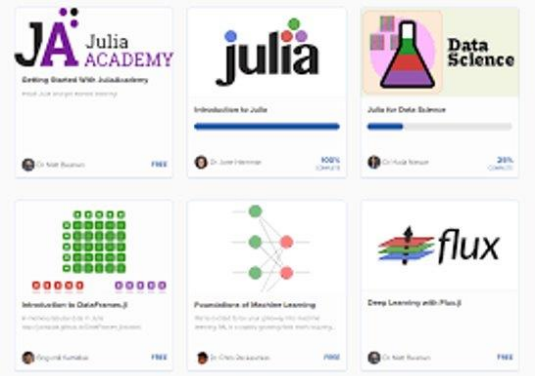


https://developer.nvidia.com/blog/gpu-computing-julia-programming-language/

Pkg.jl



Rich data science ecosystem

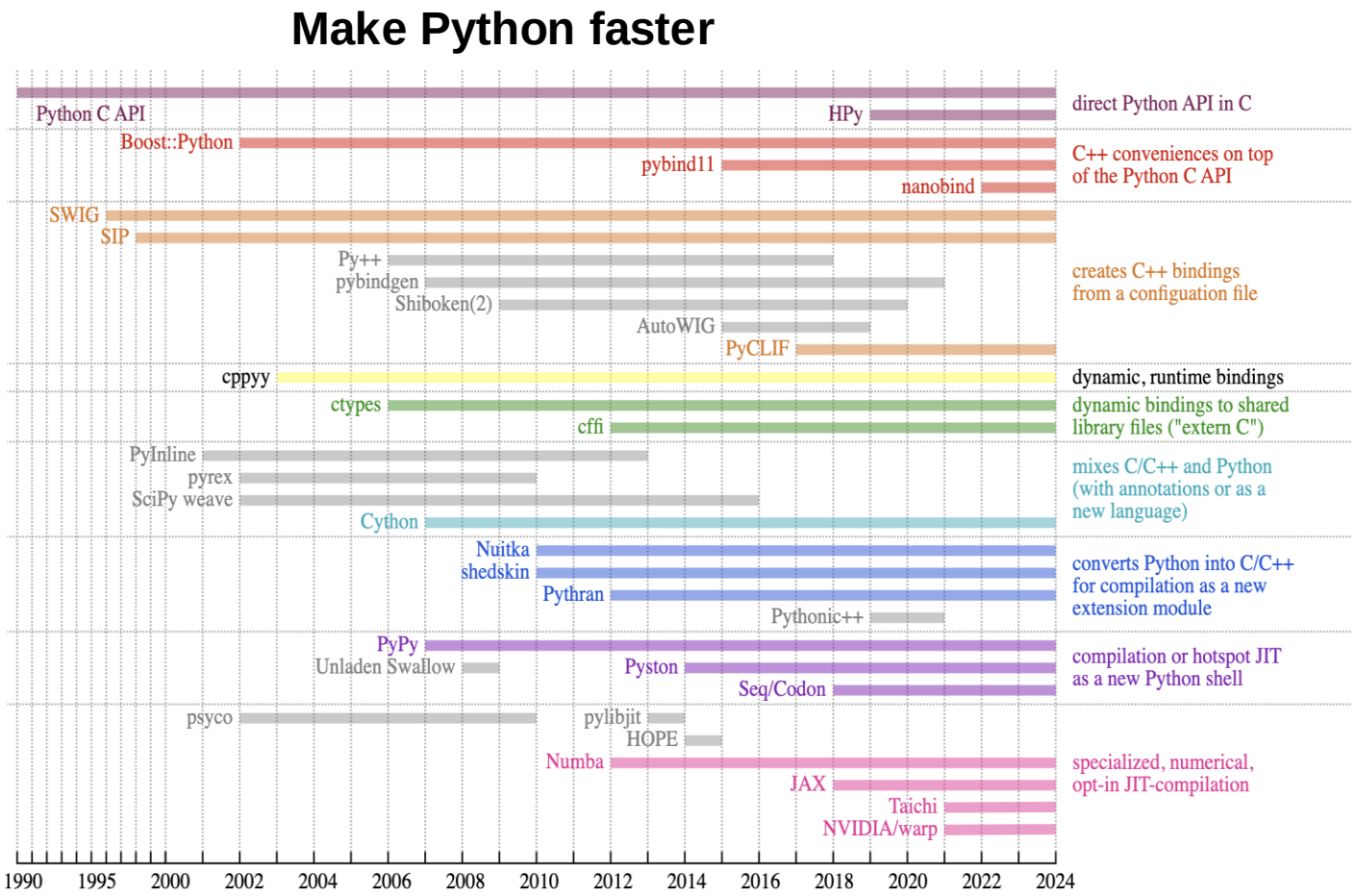


https://quantumzeitgeist.com/learning-the-julia-programming-language-for-free/

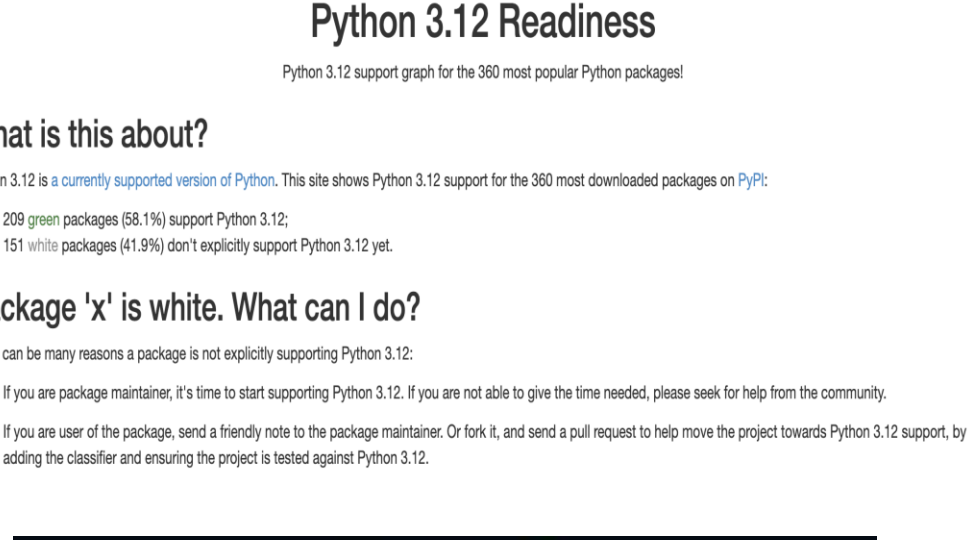
OAK RIDGE National Laboratory

Python is not enough…and is a fragmented world

https://pyreadiness.org/3.12/



https://raw.githubusercontent.com/jpivarski-talks/2023-05-01-hsf-india-tutorial/main/img/history-of-bindings-2.svg



Python 3.12 Readiness

Python 3.12 support graph for the 360 most popular Python packages!

What is this about?

Python 3.12 is a currently supported version of Python. This site shows Python 3.12 support for the 360 most downloaded packages on PyPI:

1. 209 green packages (58.1%) support Python 3.12;
2. 151 white packages (41.9%) don't explicitly support Python 3.12 yet.

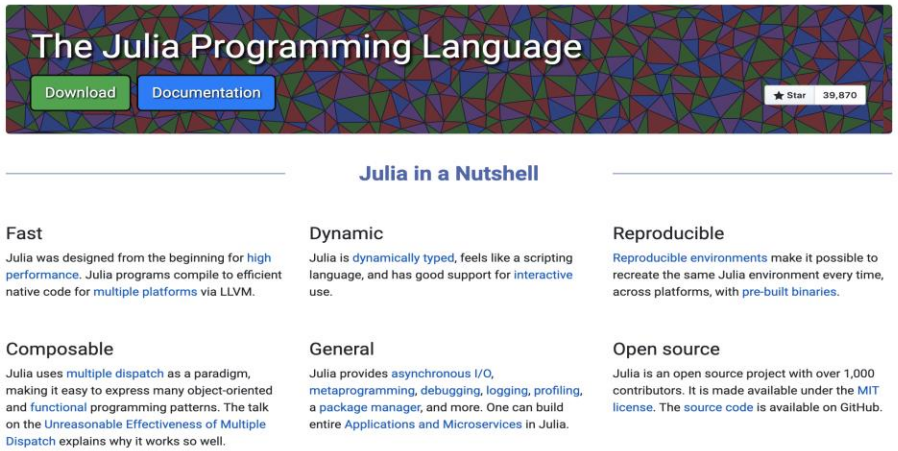
Package 'x' is white. What can I do?

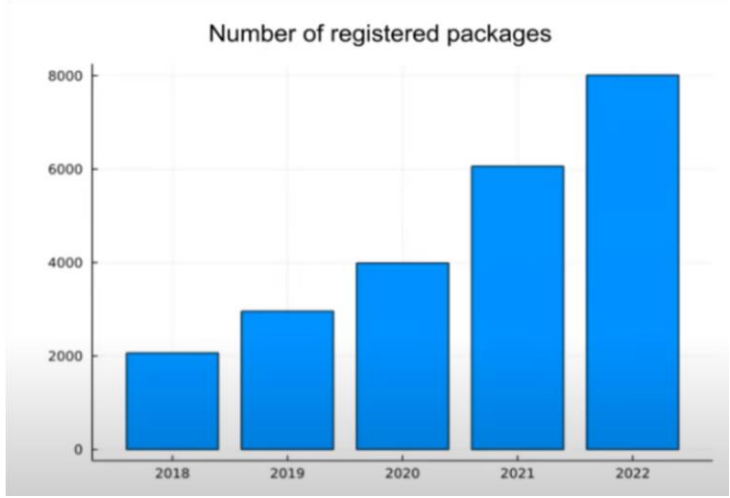
There can be many reasons a package is not explicitly supporting Python 3.12:

- If you are package maintainer, it's time to start supporting Python 3.12. If you are not able to give the time needed, please seek for help from the community.
- If you are user of the package, send a friendly note to the package maintainer. Or fork it, and send a pull request to help move the project towards Python 3.12 support, by adding the classifier and ensuring the project is tested against Python 3.12.

Julia Brief Walkthrough

- History: started at MIT in the early 2010s (predates Python Numba) https://julialang.org/blog/2022/02/10years/
- JuliaHub (formely Julia Computing) and MIT are major contributors: https://juliacomputing.com/case-studies
- First stable release v1.0 in 2018, v1.10 as of 2024 https://julialang.org/
- Open-source GitHub-hosted packages and ecosystem with MIT permissive license: https://github.com/JuliaLang/julia
- Community: annual JuliaCon summer conference: https://juliacon.org/2024/

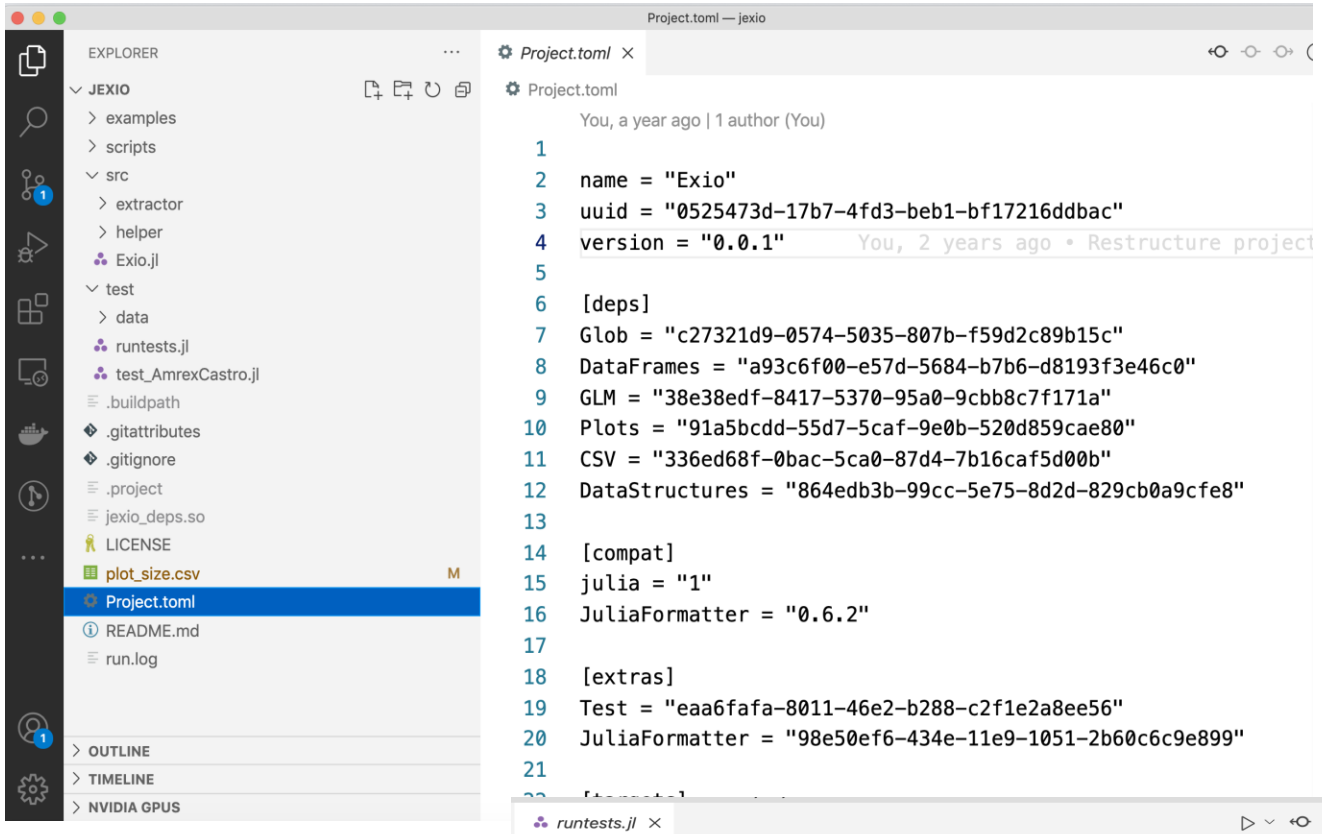


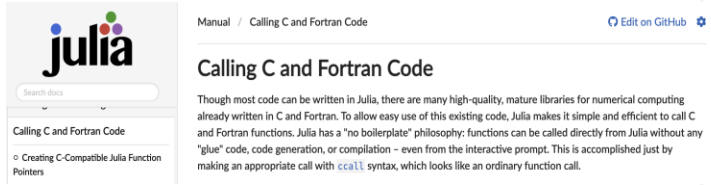


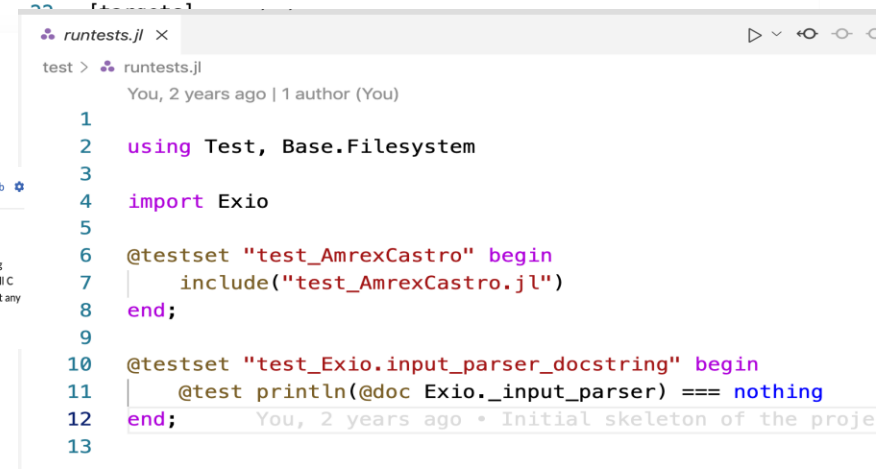
95% of Julia packages in the registry had some form of CI (youtube.com/watch?v=9YWwiFbaRx8)

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







OAK RIDGE National Laboratory

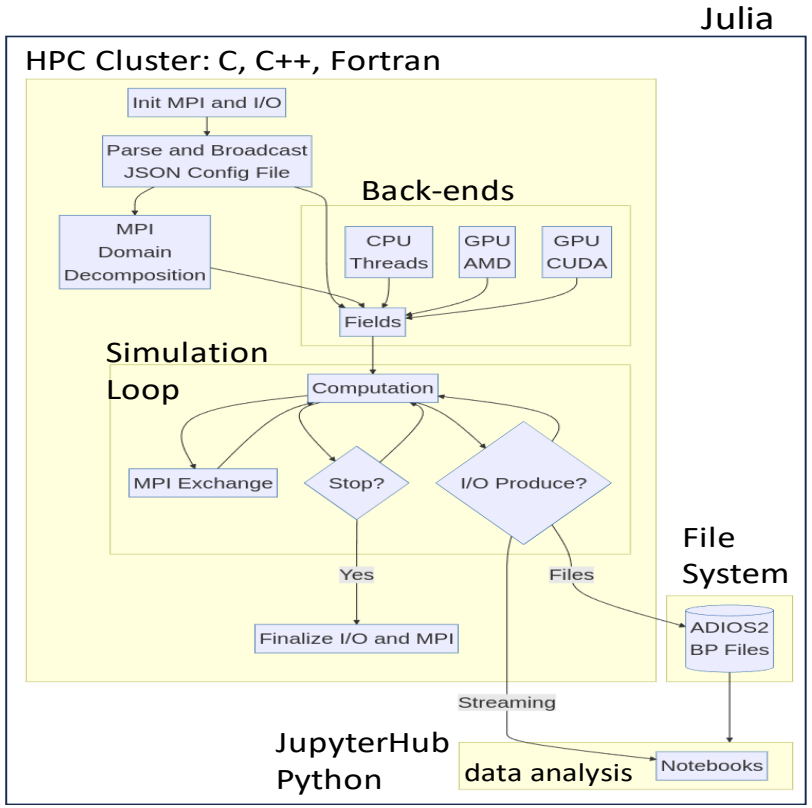
https://github.com/ornl-training/julia-basics

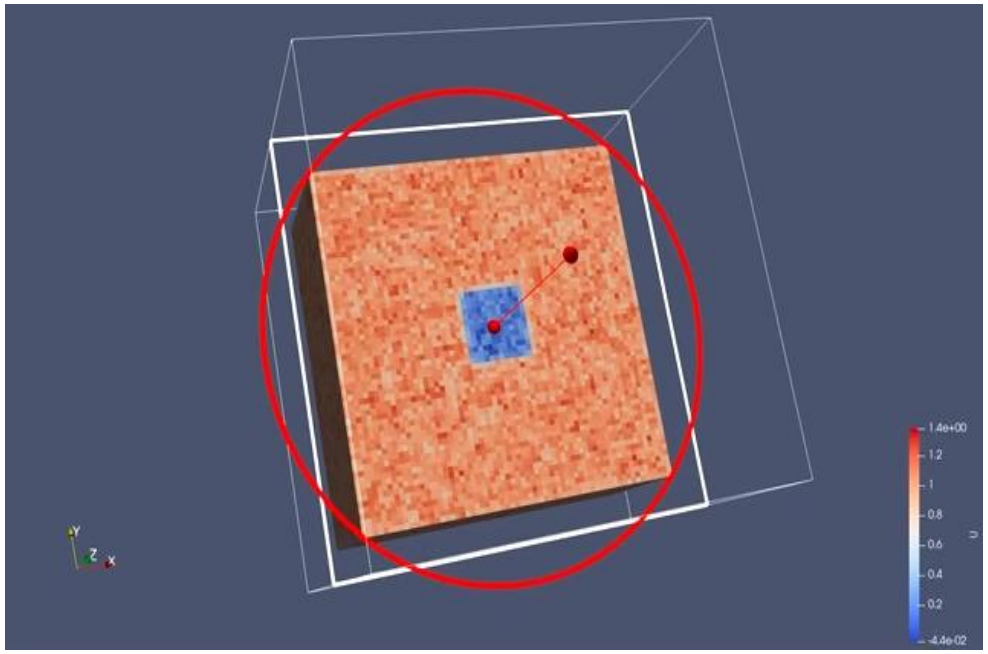
Contents

Gray-Scott miniapp: https://github.com/JuliaORNL/GrayScott.jl

Simple 3D 2-variable diffusion-reaction solver

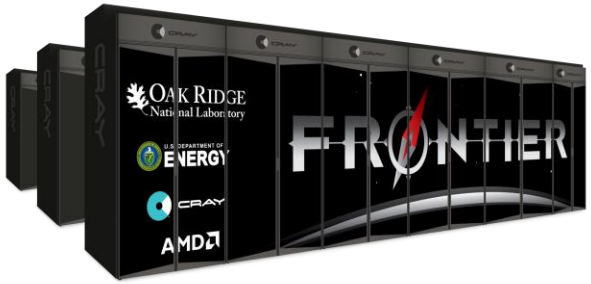
- CPU Threads, CUDA.jl and **AMDGPU.jl** backends using multiple dispatch
- Parallel I/O ADIOS2, can be visualized with ParaView
- MPI.jl for communication
- Configuration and job scripts for Frontier, Crusher and Summit under ./scripts/
- Data analysis on JupyterHub





$$\frac{\partial U}{\partial t} = D_U \nabla^2 U - UV^2 + F(1-U) + nr \qquad (1a)$$

$$\frac{\partial V}{\partial t} = D_V \nabla^2 V + UV^2 + -(F+k)V \qquad (1b)$$



Research question: Can I write an entire HPC workflow in Julia?

Best paper at SC23 WORKS

RESEARCH-ARTICLE

Julia as a unifying end-to-end workflow language on the Frontier exascale system

Authors: William F. Godoy, Pedro Valero-Lara, Caira Anderson, Katrina W. Lee, Ana Gainaru, Rafael Ferreira Da Silva, Jeffrey S. Vetter Authors Info & Claims

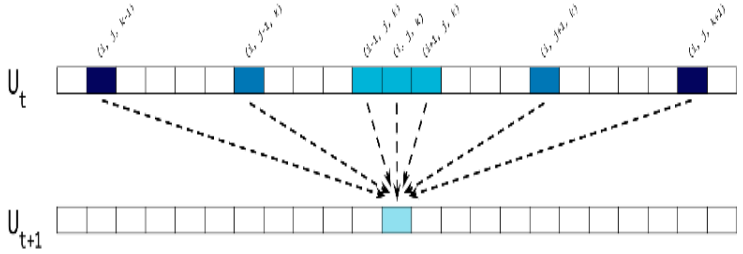
SC-W '23: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis • November 2023 • Pages 1989–1999 • https://doi.org/10.1145/3624062.3624278

https://doi.org/10.1145/3624062.3624278

https://www.nextplatform.com/2023/09/26/julia-still-not-grown-up-enough-to-ride-exascale-train/

Frontier on-node scalability using AMDGPU.jl

7-point stencil



$$fetch\_size_{effective} = \left[L^3 - 8 - 12\,(L-2)\right] \cdot sizeof(T) \tag{4a}$$

$$write\_size_{effective} = (L-2)^3 \cdot sizeof(T) \tag{4b}$$

$$bandwidth_{effective} = \frac{(fetch\_size + write\_size)_{effective}}{kernel\_time} \tag{5a}$$

$$bandwidth_{total} = \frac{(FETCH\_SIZE + WRITE\_SIZE)_{rocprof}}{kernel\_time} \tag{5b}$$

Table 2: Average bandwidth comparison of different stencil implementations on a single GPU.

Kernel	Bandwidth (GB/s) Effective	Total
Julia GrayScott.jl		
- 2-variable (application)	312	570
- 1-variable no random	312	625
HIP single variable	599	1,163
Theoretical peak MI250x	1,600	

Table 3: rocprof outputs for HIP 1-variable and Julia Gray-Scott.jl implementations

kernel metric	HIP 1-var	GrayScott.jl 1-variable no random	 2-variable (application)
wgr	256	512	512
lds	0	29,184	29,184
scr	0	8,192	8,192
FETCH_SIZE (GB)	25.08	25.40	50.80
WRITE_SIZE (GB)	8.35	8.38	16.78
TCC_HIT (M)	9.14	10.80	24.60
TCC_MISS (M)	8.36	8.69	17.19
Avg Duration (ms)	28.74	54.03	111.07

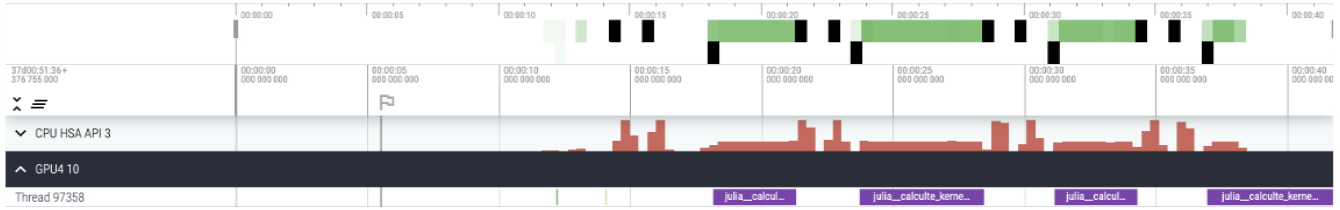


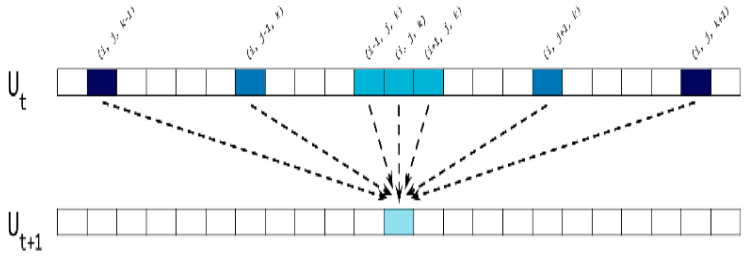
Figure 5: Gray-Scott simulation trace obtained with rocprof showing computational load on GPU and memory transfer to CPU for communication.

Listing 2: Julia AMDGPU.jl Gray-Scott kernel

```
using AMDGPU
using Distributions

function _laplacian(i, j, k, var)
  l =   var[i - 1, j, k] + var[i + 1, j, k]
      + var[i, j - 1, k] + var[i, j + 1, k]
      + var[i, j, k - 1] + var[i, j, k + 1]
      - 6.0 * var[i, j, k]
  return l / 6.0
end

function _kernel_amdgpu!(u, v, u_temp, v_temp,
                         sizes, Du, Dv, F, K,
                         noise, dt)

  k = (workgroupIdx().x - 1) * workgroupDim().x
      + workitemIdx().x
  j = (workgroupIdx().y - 1) * workgroupDim().y
      + workitemIdx().y
  i = (workgroupIdx().z - 1) * workgroupDim().z
      + workitemIdx().z

  if k == 1 || k >= sizes[3] ||
     j == 1 || j >= sizes[2] ||
     i == 1 || i >= sizes[1]
    return
  end

  @inbounds begin
    u_ijk = u[i, j, k]
    v_ijk = v[i, j, k]

    du = Du * _laplacian(i, j, k, u)
         - u_ijk * v_ijk^2 + F * (1.0 - u_ijk)
         + noise * rand(Uniform(-1, 1))

    dv = Dv * _laplacian(i, j, k, v)
         + u_ijk * v_ijk^2 - (F + K) * v_ijk

    u_temp[i, j, k] = u_ijk + du * dt
    v_temp[i, j, k] = v_ijk + dv * dt
  end

  return nothing
end
```

Frontier on-node scalability using AMDGPU.jl

7-point stencil



$$fetch\_size_{effective} = \left[L^3 - 8 - 12\,(L-2)\right] \cdot sizeof(T) \quad (4a)$$

$$write\_size_{effective} = (L-2)^3 \cdot sizeof(T) \quad (4b)$$

$$bandwidth_{effective} = \frac{(fetch\_size + write\_size)_{effective}}{kernel\_time} \quad (5a)$$

$$bandwidth_{total} = \frac{(FETCH\_SIZE + WRITE\_SIZE)_{rocprof}}{kernel\_time} \quad (5b)$$

Table 2: Average bandwidth comparison of different stencil implementations on a single GPU.

Kernel	Bandwidth (GB/s) Effective	Total
Julia GrayScott.jl		
- 2-variable (application)	312	570
- 1-variable no random	312	625
HIP single variable	599	1,163
Theoretical peak MI250x	1,600	

Table 3: rocprof outputs for HIP 1-variable and Julia Gray-Scott.jl implementations

kernel / metric	HIP 1-var	GrayScott.jl 1-variable no random	GrayScott.jl 2-variable (application)
wgr	256	512	512
lds	0	29,184	29,184
scr	0	8,192	8,192
FETCH_SIZE (GB)	25.08	25.40	50.80
WRITE_SIZE (GB)	8.35	8.38	16.78
TCC_HIT (M)	9.14	10.80	24.60
TCC_MISS (M)	8.36	8.69	17.19
Avg Duration (ms)	28.74	54.03	111.07

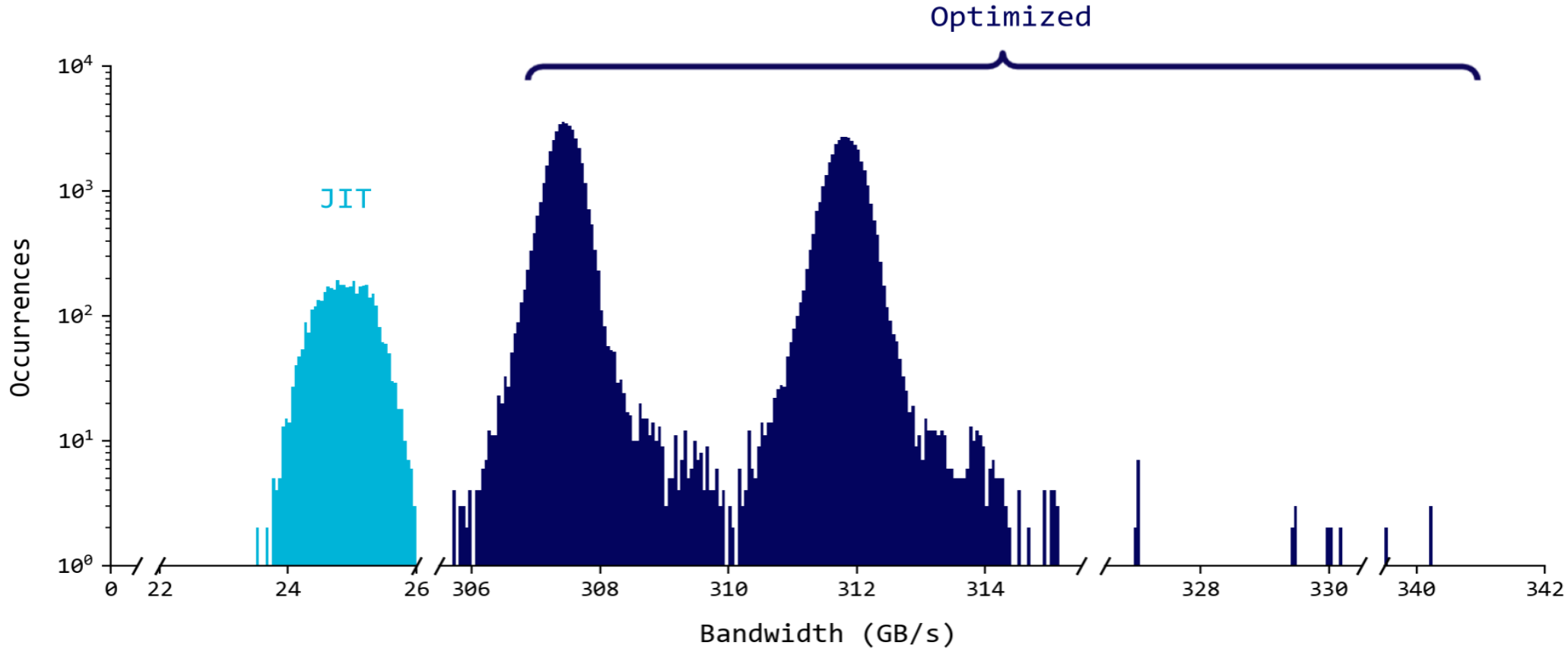
Listing 4: GrayScott.jl application kernel unique memory loads (14) and store (2) in LLVM-IR

```
  %94 = load double, double addrspace(1)*  %93, align 8
%103 = load double, double addrspace(1)* %102, align 8
%107 = load double, double addrspace(1)* %106, align 8
%110 = load double, double addrspace(1)* %109, align 8
%114 = load double, double addrspace(1)* %113, align 8
%117 = load double, double addrspace(1)* %116, align 8
%122 = load double, double addrspace(1)* %121, align 8
%126 = load double, double addrspace(1)* %125, align 8
%312 = load double, double addrspace(1)* %311, align 8
%315 = load double, double addrspace(1)* %314, align 8
%318 = load double, double addrspace(1)* %317, align 8
%321 = load double, double addrspace(1)* %320, align 8
%325 = load double, double addrspace(1)* %324, align 8
%329 = load double, double addrspace(1)* %328, align 8
...
store double %345, double addrspace(1)* %353, align 8
store double %355, double addrspace(1)* %363, align 8
```

Julia AMDGPU.jl reaches ~50% bandwidth (performance) of HIP
No surprises on: FETCH/WRITE SIZE, LLVM-IR
rocprof reports more activity “lds” on Julia

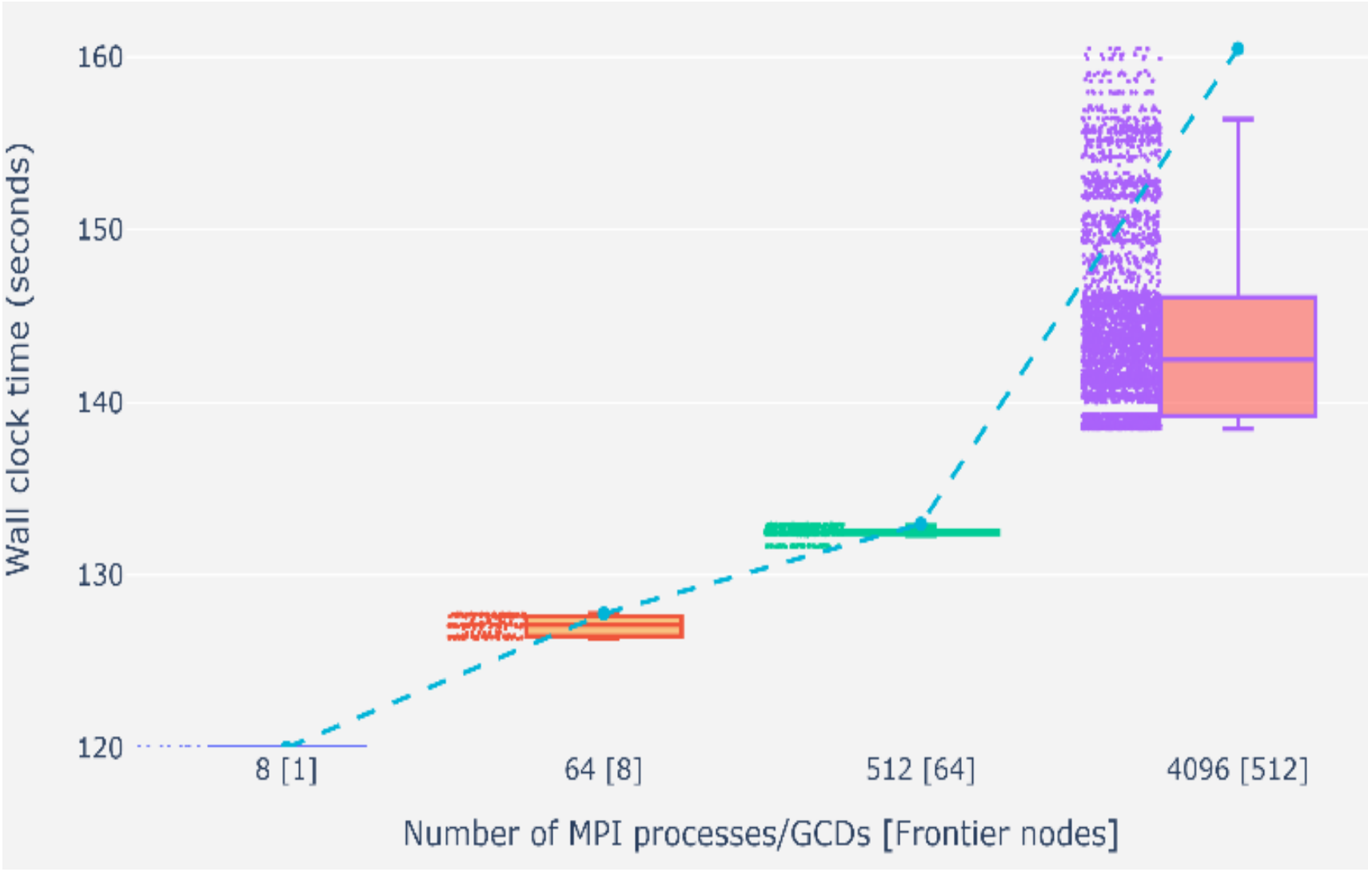
Frontier on-node scalability using AMDGPU.jl for several GPUs

Bandwidth distribution for 4,096 GCD (GPUs) and 20 timesteps. HIP ~ 600 GB/s (800 GB/s claimed on MI250x), Theoretical Peak on MI250x = 1,600 GB/s



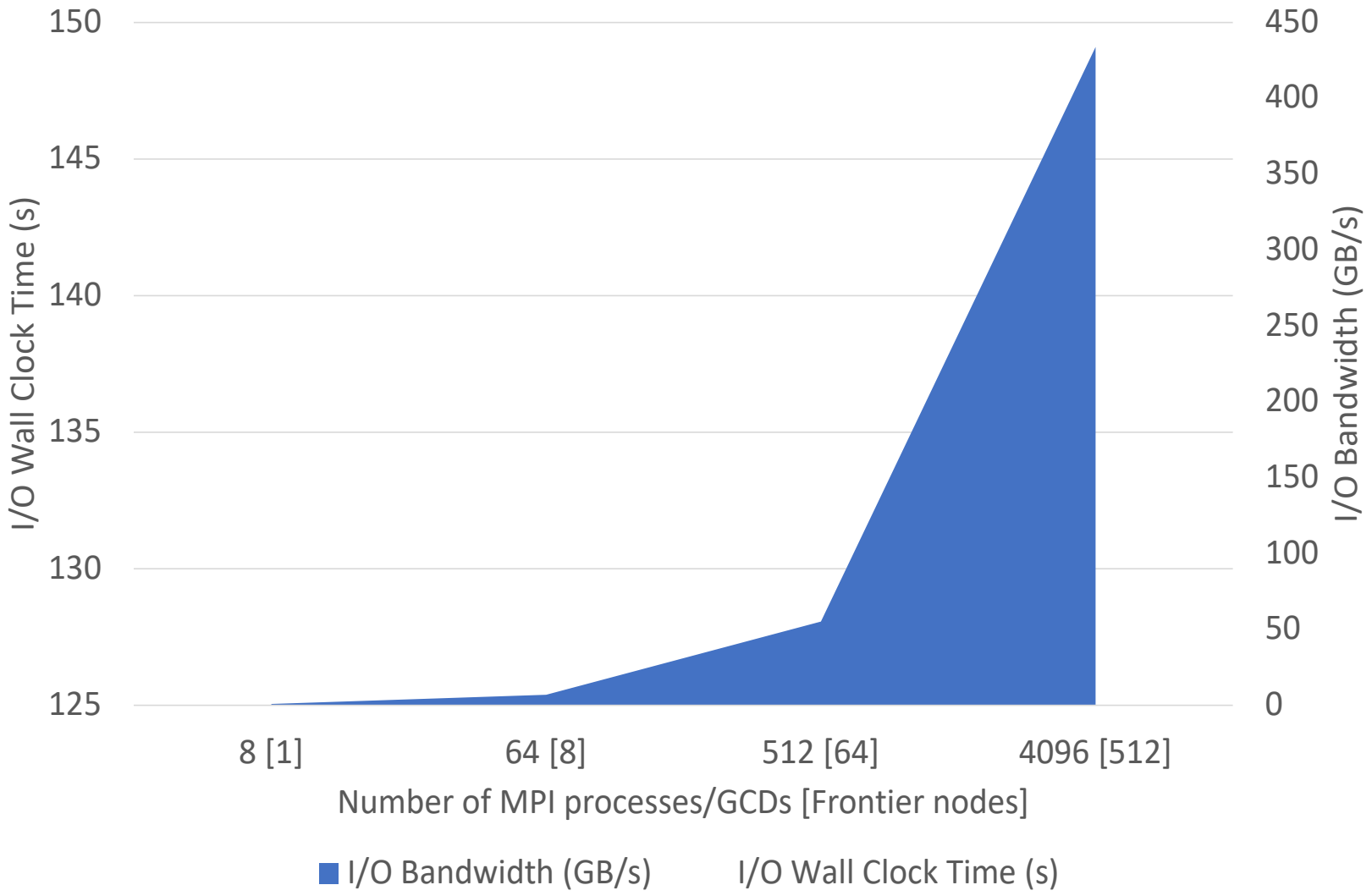
GrayScott.jl Weak Scaling on Frontier

- Tested successfully up to 512 nodes (5% of Frontier) 1 GCD/MPI proc using MPI.jl
- Tried 4K nodes (50% of Frontier) resulted in a libfabric error
- 2-3% variability up to 64 nodes
- 12-15% variability at 512 nodes



GrayScott.jl Parallel I/O Weak Scaling on Frontier file system

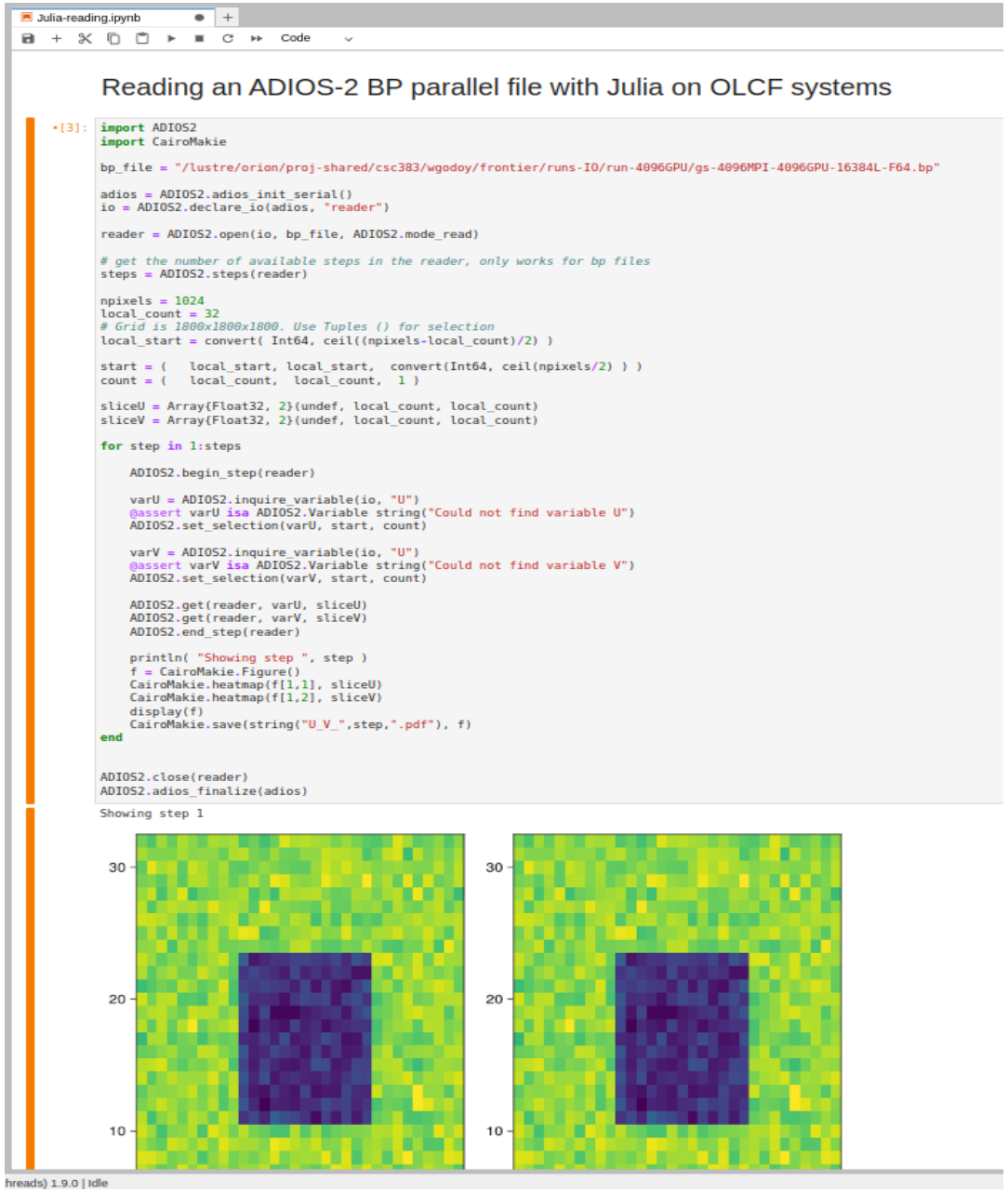
- We tested parallel I/O using Julia via the ADIOS2.jl Julia bindings
- ADIOS2 is a high-performance data I/O library funded by the US Department of Energy SC ASCR
- Results are shown for the Orion file system
- ADIOS2.jl shows negligible overhead wrt ADIOS2



Godoy, W. F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., ... & Klasky, S. (2020). Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, *12*, 100561. https://doi.org/10.1016/j.softx.2020.100561

Data analysis on JupyterHub at OLCF

- https://jupyter.olcf.ornl.gov/
- We launched a Julia kernel on JupyterHub to read and analyze data
- We read with ADIOS2.jl and visualize with Makie.jl
- JIT and TTFX (time to first plot) can be a nuance
- Pluto.jl?



Godoy, W. F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., ... & Klasky, S. (2020). Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, *12*, 100561. https://doi.org/10.1016/j.softx.2020.100561

Julia on Frontier

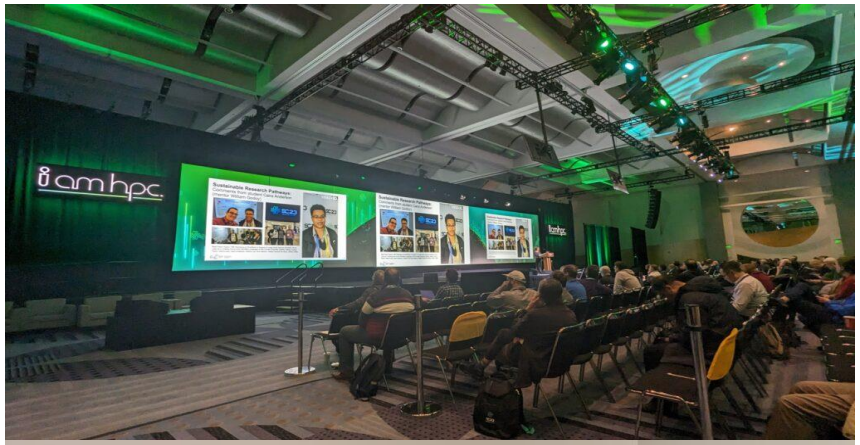
- Recent efforts:
  - Investigate new approaches in the post-ECP landscape: Julia for HPC, LLMs for HPC
  - Recent interest in the high-productivity + high-performance space
  - Enabled me to mentor in our internship program for the last 3 years
  - Build a "Julia for HPC" community https://juliaparallel.org/resources/

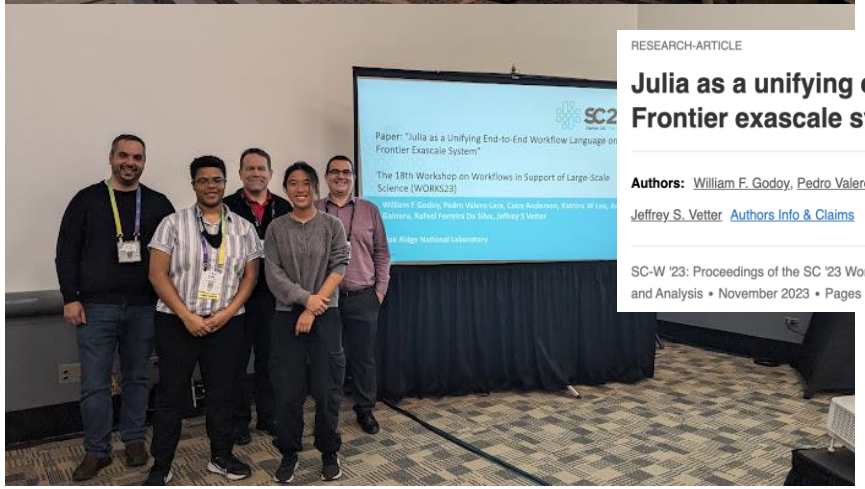
Evaluation of OpenAI Codex for HPC parallel programming models kernel generation

William F. Godoy, Pedro Valero-Lara, Keita Teranishi, Prasanna Balaprakash, and Jeffrey S. Vetter
Oak Ridge National Laboratory
Oak Ridge, TN, USA
{godoywf},{valerolarap},{teranishik},{pbalapra},{vetter}@ornl.gov

Evaluating performance and portability of high-level programming models: Julia, Python/Numba, and Kokkos on exascale nodes

William F. Godoy, Pedro Valero-Lara, T. Elise Dettling, Christian Trefftz, Ian Jorquera, Thomas Sheehy, Ross G. Miller, Marc Gonzalez-Tallada, Jeffrey S. Vetter
Oak Ridge National Laboratory
{godoywf}, {valerolarap}, {dettlingte}, {trefftzci}, {jorqueraid}, {sheehytb}, {rgmiller}, {gonzaleztalm}, {vetter}@ornl.gov
Valentin Churavy
Massachusetts Institute of Technology
vchuravy@mit.edu





RESEARCH-ARTICLE

Julia as a unifying end-to-end workflow language on the Frontier exascale system

Authors: William F. Godoy, Pedro Valero-Lara, Caira Anderson, Katrina W. Lee, Ana Gainaru, Rafael Ferreira Da Silva, Jeffrey S. Vetter Authors Info & Claims

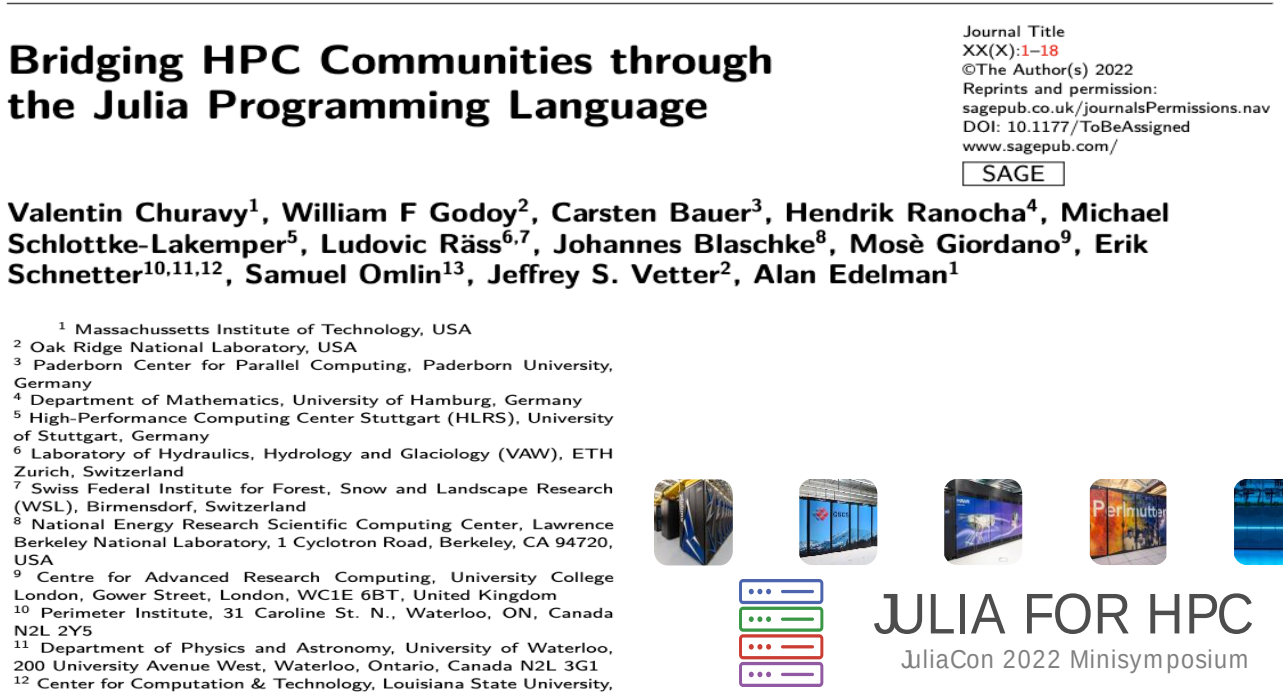
SC-W '23: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis • November 2023 • Pages 1989–1999 • https://doi.org/10.1145/3624062.3624278

Highlights: https://shinstitute.org/srp-mentor-spotlight

- Building Julia for HPC **research** and **community** activities: tutorials, papers, SC BoFs 2022/2023
- 2023 IPDPS HIPS workshop paper
- 2023 ICPP P2S2 paper on generative AI for HPC kernels
- **Best Paper** at the SC23 WORKS workshop
- D-HPC Workshop at SC23

Julia HPC Efforts

- Projects:
- External Engagements with Julia HPC
  - Monthly meetings with stakeholders
  - JuliaCon 2022/2023 HPC minisymposium
  - Julia HPC Position paper in the works
  - ECP BoF Session on Rapid Prototyping for HPC using Julia, Python Numba, Flang
  - SC22/23 BoF "Julia for HPC"
- Internal Engagements
  - Tutorials at ORNL Software and Data Expo
  - Julia Workshop for ORNL Science
  - OLCF User Call: Julia for HPC

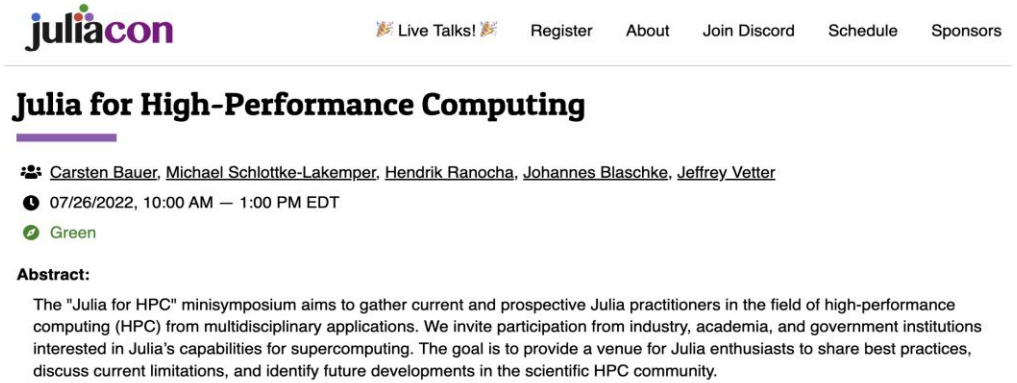


Bridging HPC Communities through the Julia Programming Language

Journal Title
XX(X):1–18
©The Author(s) 2022
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Valentin Churavy[1], William F Godoy[2], Carsten Bauer[3], Hendrik Ranocha[4], Michael Schlottke-Lakemper[5], Ludovic Räss[6,7], Johannes Blaschke[8], Mosè Giordano[9], Erik Schnetter[10,11,12], Samuel Omlin[13], Jeffrey S. Vetter[2], Alan Edelman[1]

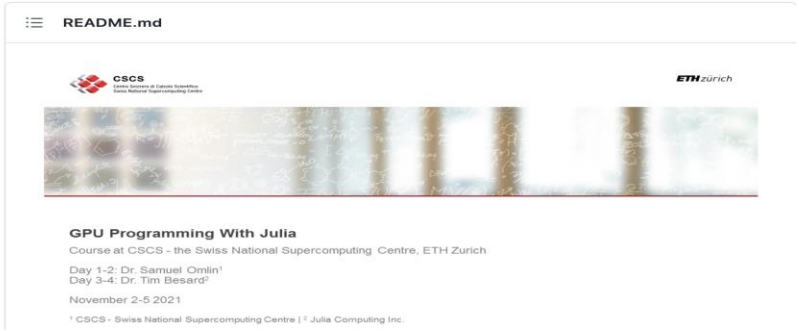
[1] Massachussetts Institute of Technology, USA
[2] Oak Ridge National Laboratory, USA
[3] Paderborn Center for Parallel Computing, Paderborn University, Germany
[4] Department of Mathematics, University of Hamburg, Germany
[5] High-Performance Computing Center Stuttgart (HLRS), University of Stuttgart, Germany
[6] Laboratory of Hydraulics, Hydrology and Glaciology (VAW), ETH Zurich, Switzerland
[7] Swiss Federal Institute for Forest, Snow and Landscape Research (WSL), Birmensdorf, Switzerland
[8] National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA
[9] Centre for Advanced Research Computing, University College London, Gower Street, London, WC1E 6BT, United Kingdom
[10] Perimeter Institute, 31 Caroline St. N., Waterloo, ON, Canada N2L 2Y5
[11] Department of Physics and Astronomy, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1
[12] Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803, USA
[13] Swiss National Supercomputing Centre (CSCS), ETH Zurich, Switzerland



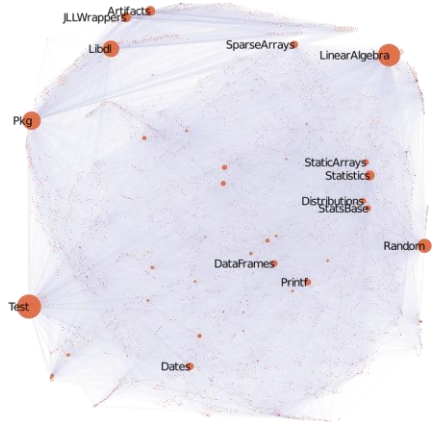
Community Efforts in HPC

- 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/GPU4GEO/software/
- https://arxiv.org/abs/2207.03711

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



Top15 most popular packages



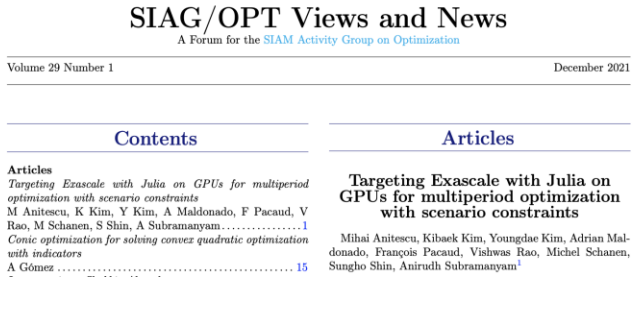
https://enccs.github.io/Julia-for-HPC

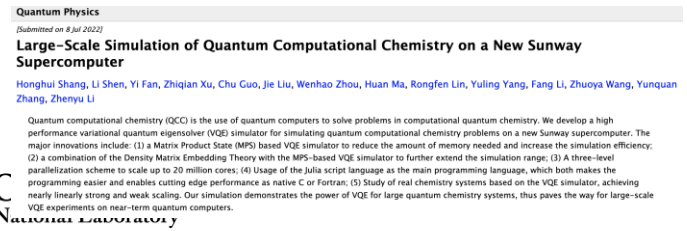
ECP ExaSDG on Summit

https://docs.dftk.org/stable

https://juliaastro.github.io/dev

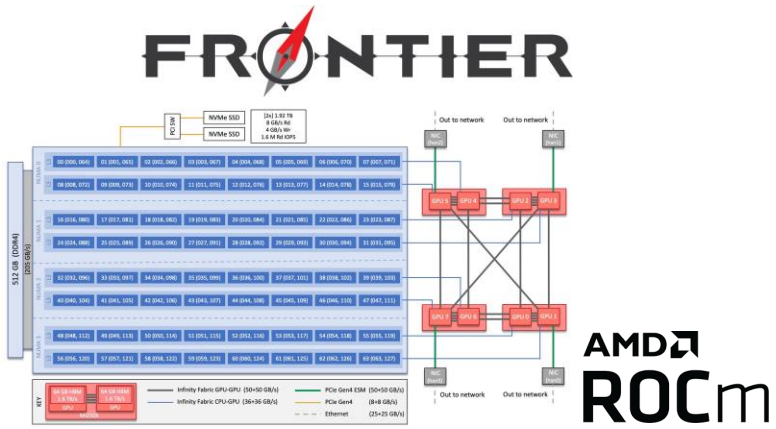
https://github.com/JuliaParallel

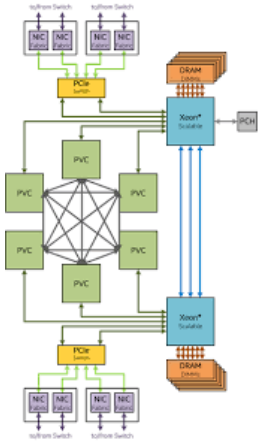


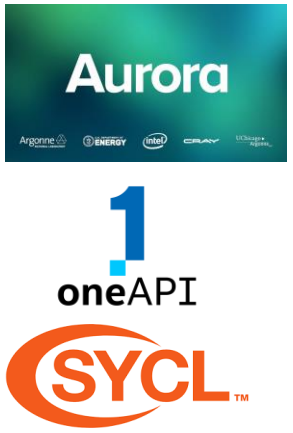


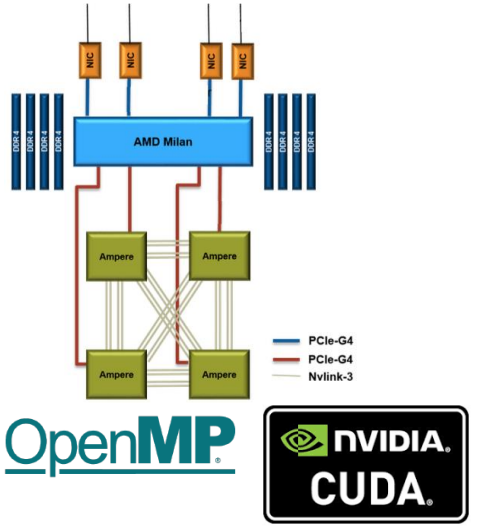
JACC: Performance Portability and Programming Productivity

- Performance portability is an important target for the DOE
- The DOE leadership computing facilities are GPU-accelerated systems
- Performant portable layers are a priority for DOE: e.g. Kokkos, Raja, OpenMP, OpenACC
- **Program "once" and deploy "everywhere"**







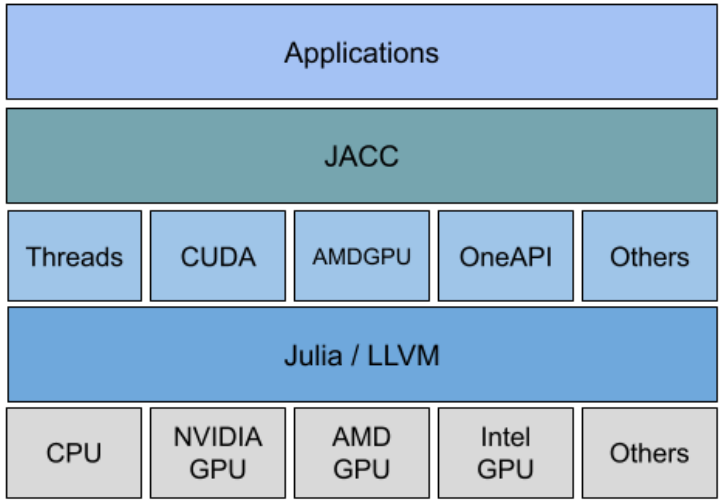


Contents

- Julia's value proposition for science: LLVM + Coordinated Ecosystem
- Efforts in HPC
- **JACC: Julia for Accelerators**
- Resources: where to get started?
- Acknowledgments

JACC (Julia ACCelerated), What is that??

- ***Think in Kokkos, but now imagine that it is easy to use***
- The metaprogramming and performance portability model of Julia
  - One code running everywhere
- JACC is a unified Julia front-end in top of multiple backends
  - Threads (CPUs), CUDA (NVIDIA GPUs), AMDGPU (AMD GPUs), and OneAPI (Intel GPUs)
- Hide low-level and device specific implementation
  - Memory, granularity, etc.
- Improve programming productivity for Science and HPC
- A growing community (family)
  - BNL(NERSC), Argonne, MIT, ETHZ, FI/CCQ, …
  - You are welcome to join (JACC meetings once a month)



https://github.com/JuliaORNL/JACC.jl

JACC model, How to use it??

- ***Descriptive, not prescriptive***
- Two main components
- JACC.Array
  - An alias to the corresponding backend memory management
- JACC.parallel_for and JACC.parallel_reduce
  - Kernel passed an argument
  - Uni-dimensional and bi-dimensional APIs
- JACC.ones or JACC.zeros
- JACC.shared
  - Exploit in-chip programable GPUs shared memory

```
# Unidimensional arrays
function axpy(i, alpha, x, y)
  x[i] += alpha * y[i]
end
function dot(i, x, y)
  return x[i] * y[i]
end
SIZE = 1_000_000
x = round.(rand(Float64, SIZE) * 100)
y = round.(rand(Float64, SIZE) * 100)
alpha = 2.5
dx = JACC.Array(x)
dy = JACC.Array(y)
JACC.parallel_for(SIZE, axpy, alpha, dx, dy)
res = JACC.parallel_reduce(SIZE, dot, dx, dy)
# Multidimensional arrays
function axpy(i, j, alpha, x, y)
  x[i,j] = x[i,j] + alpha * y[i,j]
end
function dot(i, j, x, y)
  return x[i,j] * y[i,j]
end
SIZE = 1_000
x = round.(rand(Float64, SIZE, SIZE) * 100)
y = round.(rand(Float64, SIZE, SIZE) * 100)
alpha = 2.5
dx = JACC.Array(x)
dy = JACC.Array(y)
JACC.parallel_for((SIZE,SIZE),axpy,alpha,dx,dy)
res = JACC.parallel_reduce((SIZE,SIZE),dot,dx,dy)
```

https://github.com/JuliaORNL/JACC.jl

How is JACC implemented??

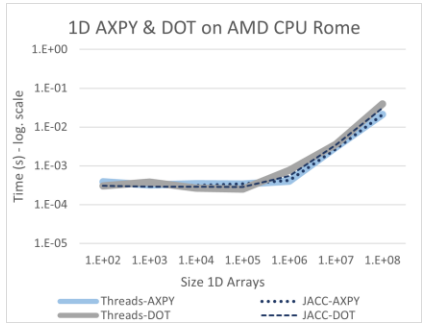
- ***The simpler the better, use everything that Julia can provide***
- Every JACC front-end capability has to be implemented many times
  - One implementation per backend

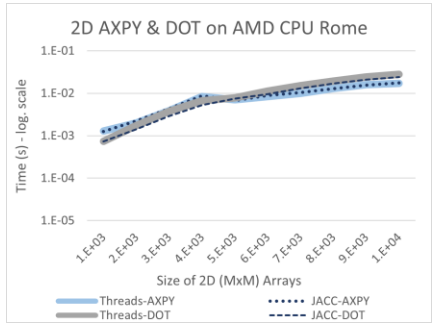
```
#JACC.Array and JACC.parallel_for on top of
    Threads
function __init__()
  const JACC.Array = Base.Array{T,N} where {T,N}
end
#Unidimensional
function parallel_for(N::I, f::F, x...) where {I<:
    Integer,F<:Function}
  Threads.@sync Threads.@threads for i in 1:N
    f(i, x...)
  end
end
#Multidimensional
function parallel_for((M, N)::Tuple{I,I}, f::F, x
    ...) where {I<:Integer,F<:Function}
  Threads.@sync Threads.@threads for j in 1:N
    for i in 1:M
      f(i, j, x...)
    end
  end
end
```

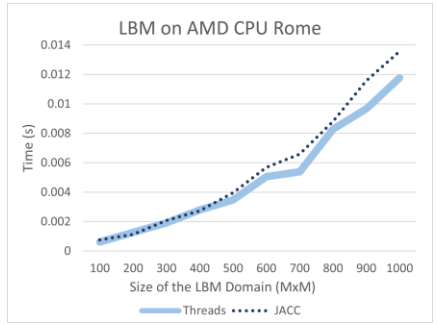
```
#JACC.Array and JACC.parallel_for on top of CUDA
function __init__()
  const JACC.Array = CUDA.CuArray{T,N} where {T,N}
end
#Unidimensional
function _parallel_for_cuda(f, x...)
  i = ( blockIdx().x - 1) * blockDim().x +
      threadIdx().x
  f(i, x...)
  return nothing
end
function JACC.parallel_for(N::I, f::F, x...) where
    {I<:Integer,F<:Function}
  maxPossibleThreads = attribute(device(),CUDA.
      DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X)
  cuda_threads = min(N, maxPossibleThreads)
  cuda_blocks = ceil(Int, N/cuda_threads)
  CUDA.@sync @cuda threads=cuda_threads blocks=
      cuda_blocks _parallel_for_cuda(N, f, x...)
end
#Multidimensional
function _parallel_for_cuda_MN(f,x...)
  i = ( blockIdx().x - 1) * blockDim().x +
      threadIdx().x
  j = ( blockIdx().y - 1) * blockDim().y +
      threadIdx().y
  f(i, j, x...)
  return nothing
end
function JACC.parallel_for((M, N)::Tuple{I,I}, f::
    F, x...) where {I<:Integer,F<:Function}
  numThreads = 16
  Mthreads = min(M, numThreads)
  Nthreads = min(N, numThreads)
  Mblocks = ceil(Int, M/Mthreads)
  Nblocks = ceil(Int, N/Nthreads)
  CUDA.@sync @cuda threads=(Mthreads, Nthreads)
      blocks=(Mblocks, Nblocks)
      _parallel_for_cuda_MN(f, x...)
end
```

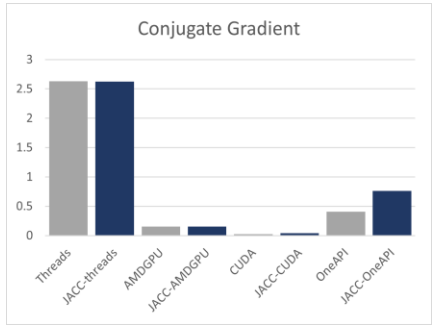
https://github.com/JuliaORNL/JACC.jl

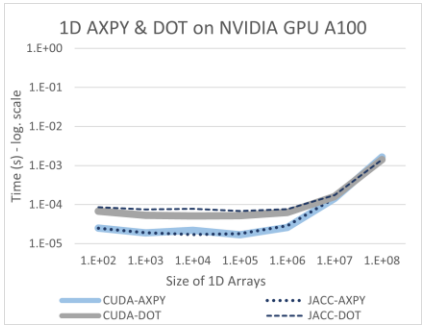
OK, but this is HPC, What about performance??

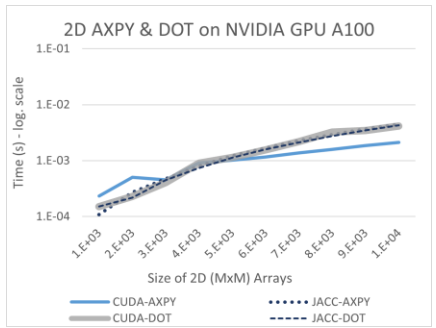


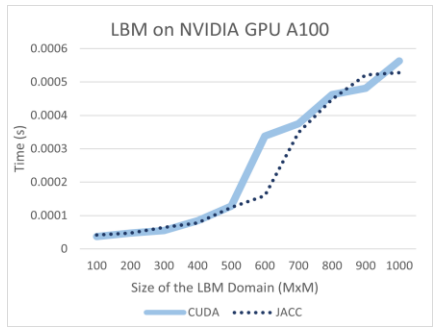


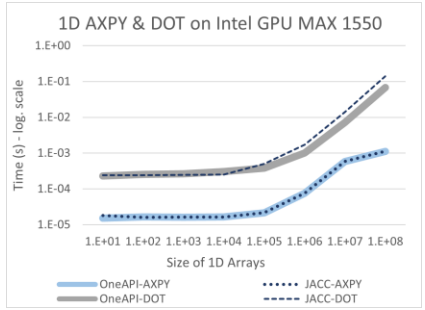


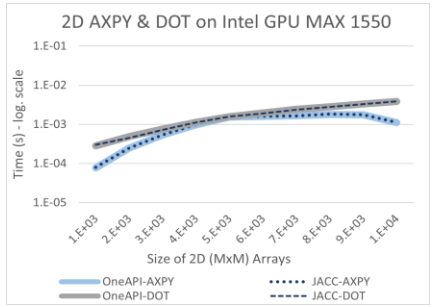


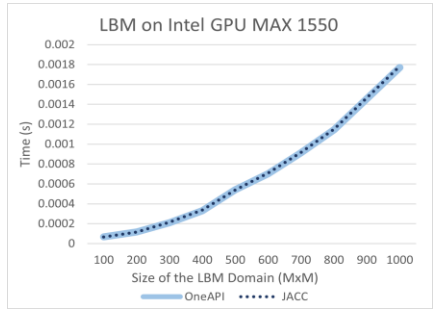


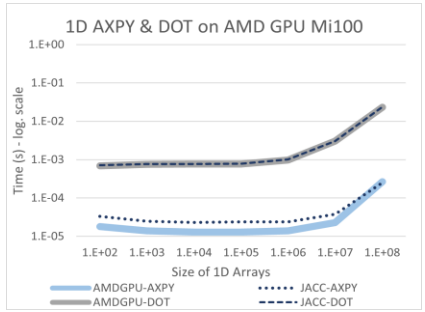


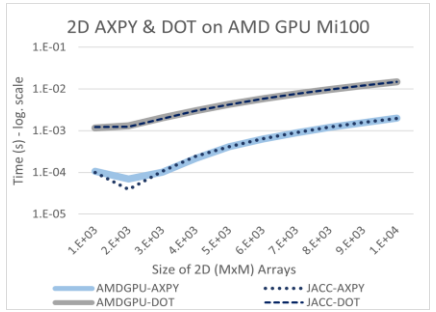


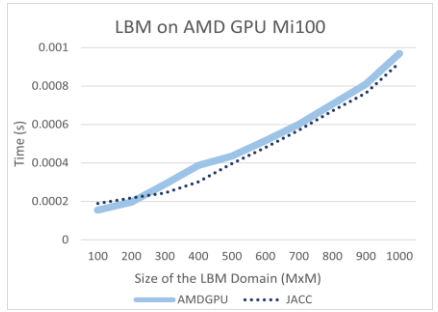












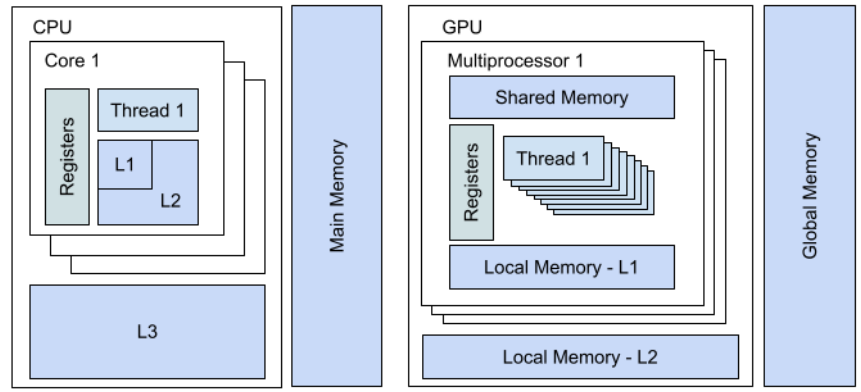
Julia as a unifying end-to-end workflow language on the Frontier exascale system. SC Workshops 2023: 1989-1999 https://arxiv.org/abs/2309.10292

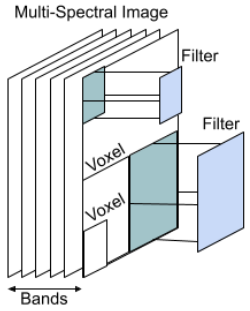
Evaluating performance and portability of high-level programming models: Julia, Python/Numba, and Kokkos on exascale nodes. IPDPS Workshops 2023: 373-382 https://arxiv.org/abs/2303.06195

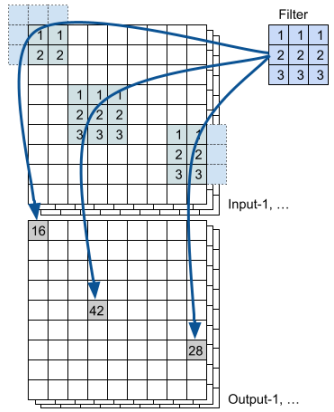
Ongoing efforts??

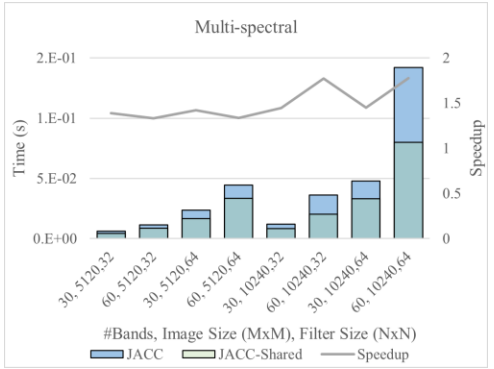
- JACC.Shared
  - Exploiting high-bandwidth programable in-chip GPUs memory

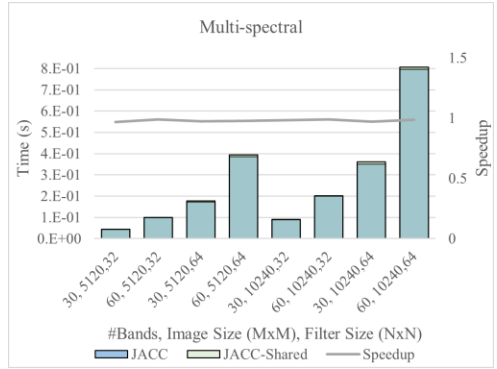
```
function spectral(i, j, image, filter, num_bands)
 for b in 1:bands
    @inbounds image[b, i, j] *= filter[j]
  end
end

function spectral_shared(i, j, image, filter,
    num_bands)
 #Shared memory initialization
 filter_shared = JACC.shared(filter)
 for b in 1:bands
  @inbounds image[b, i, j] *= filter_shared[j]
 end
end

num_bands = 60
num_voxel = 10_240
size_voxel = 64*64
image = init_image(Float32,
        num_bands, num_voxel, size_voxel)
filter = init_filter(Float32, size_voxel)
jimage = JACC.Array(image)
jimage_shared = JACC.Array(image)
jfilter = JACC.Array(filter)
JACC.parallel_for((num_voxel,size_voxel),
     spectral[_shared], jimage, jfilter, num_bands)
```

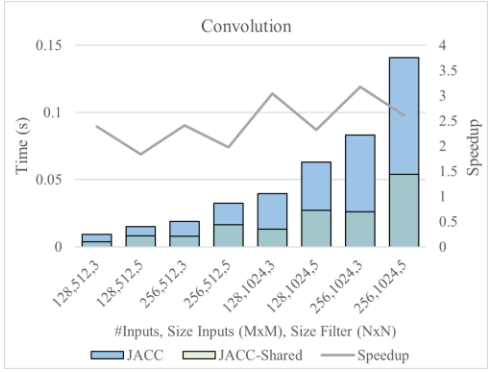


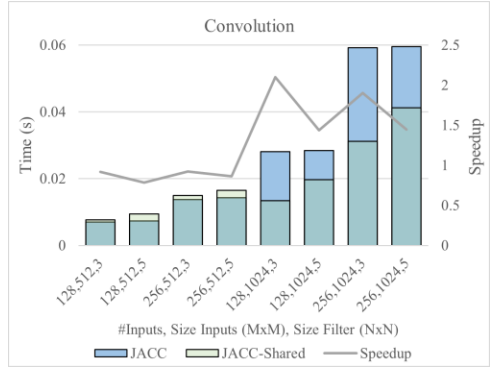












https://github.com/JuliaORNL/JACC.jl

Ongoing efforts??

- JACC.experimental
  - A separate JACC module to explore new ideas
  - Once the ideas are well implemented, these can be “promoted”
  - JACC.shared
- JACC.BLAS
  - BLAS library on top of JACC
  - Extend the capabilities of JACC to math libraries
  - Working on BLAS level-1 routines
  - Het Mankad
- JACC.multi
  - Support for multi-device
- JACC.auto
  - Support for auto-tuning
- DAGGER.jl
  - Having JACC as a backend or/and having DAGGER as a JACC backend (JACC.task)

https://github.com/JuliaORNL/JACC.jl

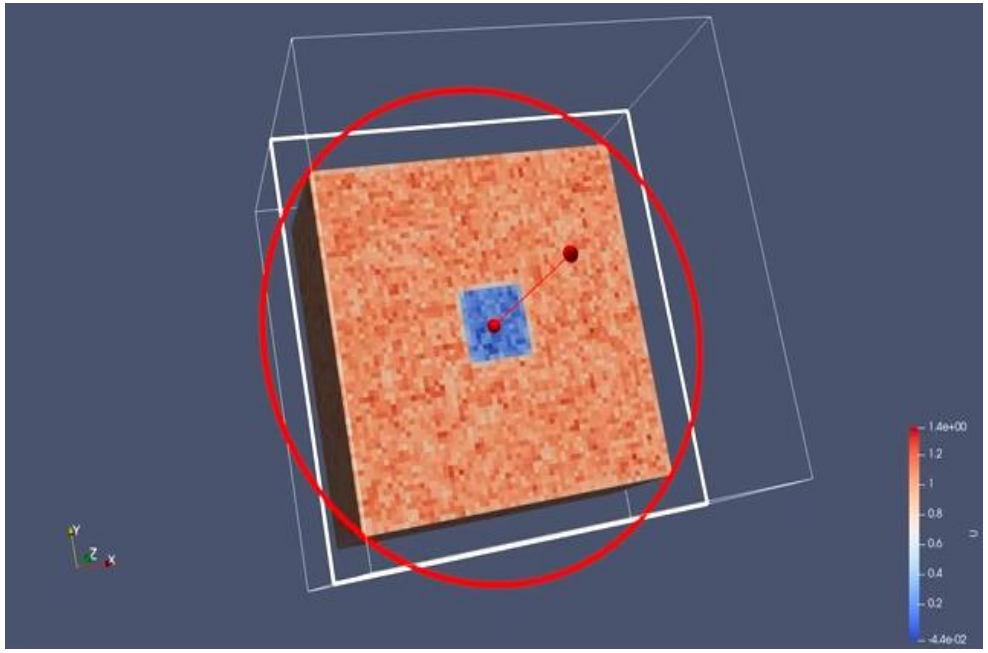
Contents

- Welcome
- Julia's value proposition for science: LLVM + Coordinated Ecosystem
- Update: Efforts in HPC
- **Resources: where to get started?**
- Acknowledgments

Contents

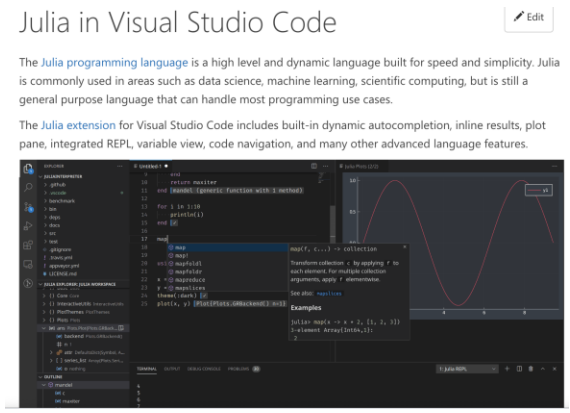
Tutorial Material

- https://github.com/JuliaORNL/GrayScott.jl
- Simple 3D 2-variable diffusion-reaction solver
  - Branch “GrayScott-JACC”
  - CPU Threads, CUDA.jl and AMDGPU.jl backends using JACC and [weakdependencies]
  - Parallel I/O ADIOS2, can be visualized with ParaView
  - MPI.jl for communication
  - Configuration and job scripts for Odo/Frontier under ./scripts/



Where to get started?

- Pick a gentle tutorial: https://techytok.com/from-zero-to-julia/
- https://github.com/ornl-training/julia-basics (training by WF Godoy & Philip Fackler) OLCF Tutorial: https://juliaornl.github.io/TutorialJuliaHPC/applications/GrayScott/01-Solver.html
- Use VS Code as the official IDE + debugger
- JuliaCon talks are available on YouTube
- https://discourse.julialang.org/ Stackoverflow might be outdated, https://julialang.slack.com/
- Julia docs and standard library: https://docs.julialang.org/en/v1/
- Learn: Project.toml, Testing.jl @testset @test, Pluto.jl , CUDA.jl/AMDGPU.jl , JACC.jl, KernelAbstractions.jl, LinearAlgebra.jl , Makie.jl , Plots.jl and Flux.jl (AI/ML), how to build a sysimage with PackageCompiler.jl
- **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.



OAK RIDGE National Laboratory
https://williamfgc.github.io/programming/scientific-computing/2020/07/22/first-project-julia-language.html
34

GitHub Copilot (GPT-3/Codex), Large Language Models + Julia

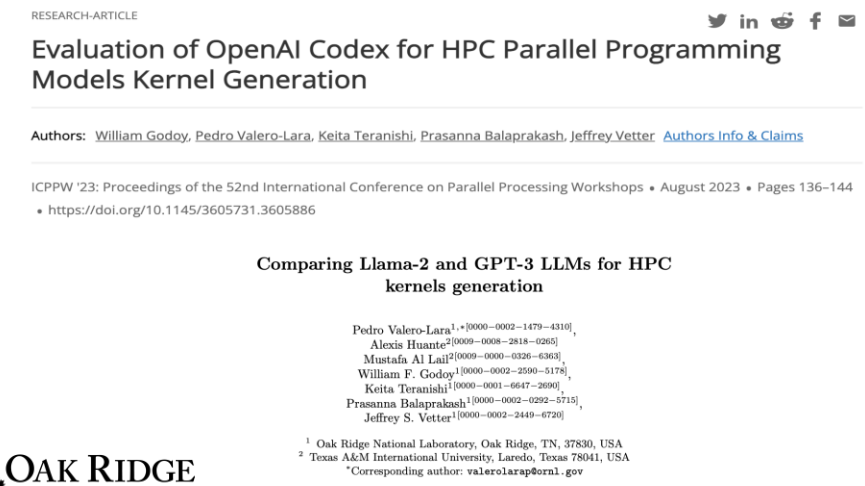
Julia is a more targeted language (like Fortran)

Julia + LLMs are a powerful combination for productivity

GitHub Copilot on VS Code $100/year/account

Compared several languages + parallel programming models

C++, Fortran, Python, Julia. Fortran and Julia scored high!



RESEARCH-ARTICLE

Evaluation of OpenAI Codex for HPC Parallel Programming Models Kernel Generation

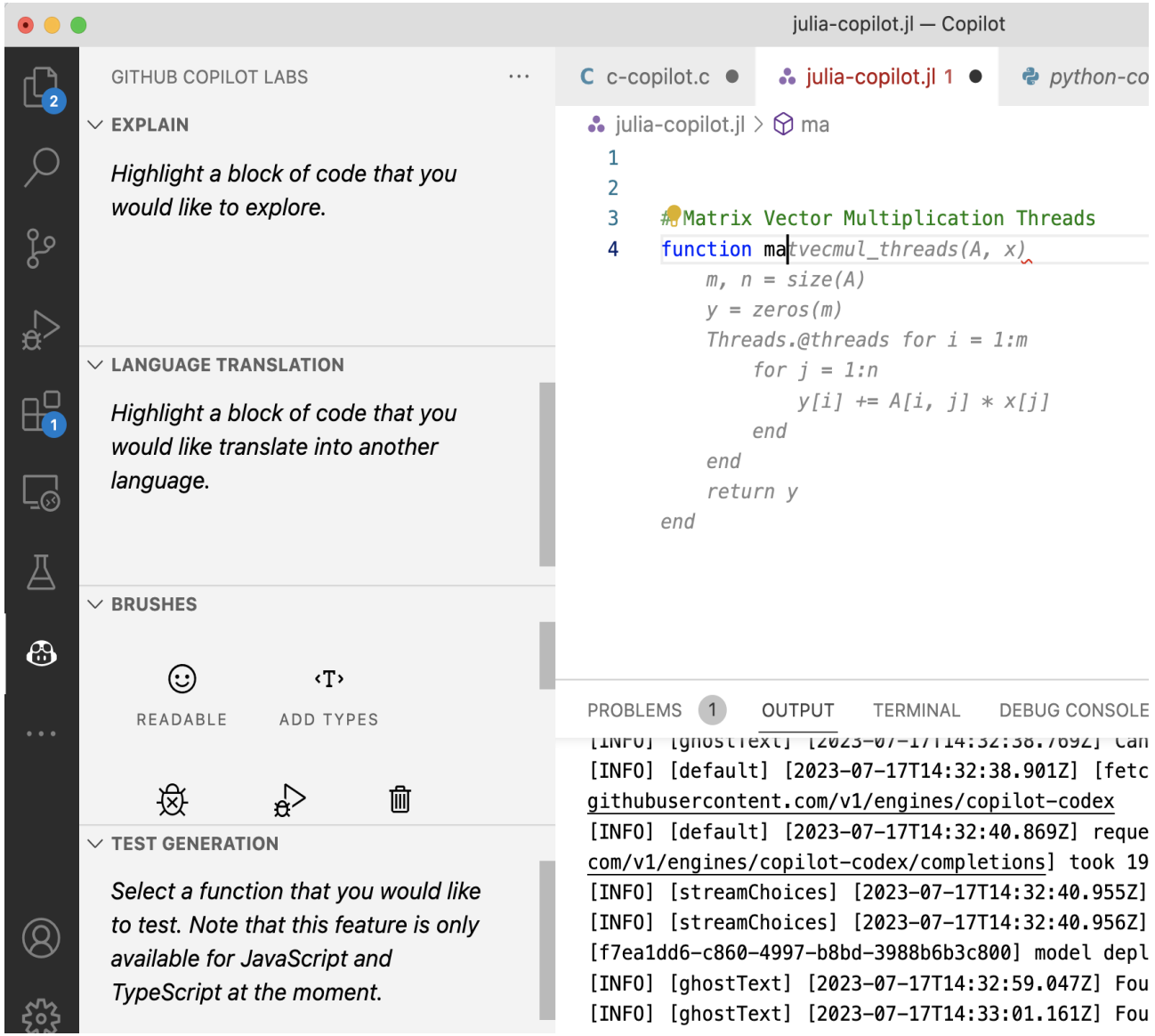
Authors: William Godoy, Pedro Valero-Lara, Keita Teranishi, Prasanna Balaprakash, Jeffrey Vetter Authors Info & Claims

ICPPW '23: Proceedings of the 52nd International Conference on Parallel Processing Workshops • August 2023 • Pages 136–144 • https://doi.org/10.1145/3605731.3605886

Comparing Llama-2 and GPT-3 LLMs for HPC kernels generation

Pedro Valero-Lara[1,*][0000−0002−1479−4310], Alexis Huante[2][0009−0008−2818−0265], Mustafa Al Lail[2][0009−0000−0326−6363], William F. Godoy[1][0000−0002−2590−5178], Keita Teranishi[1][0000−0001−6647−2690], Prasanna Balaprakash[1][0000−0002−0292−5715], Jeffrey S. Vetter[1][0000−0002−2449−6720]

[1] Oak Ridge National Laboratory, Oak Ridge, TN, 37830, USA
[2] Texas A&M International University, Laredo, Texas 78041, USA
*Corresponding author: valerolarap@ornl.gov



OAK RIDGE National Laboratory

S4PST: Stewardship of Programming Systems and Tools (2024-2029)

- PI: Keita Teranishi (ORNL), Co-PIs: Pedro Valero-Lara, William F Godoy
- 8 National Laboratories:
  - Oak Ridge National Laboratory
  - Argonne National Laboratory
  - Lawrence Livermore National Laboratory
  - Lawrence Berkeley National Laboratory
  - Sandia National Laboratories
  - Brookhaven National Laboratory
  - Los Alamos National Laboratory
  - SLAC National Accelerator Laboratory
- University Partners:
  - University of Delaware
  - Massachusetts Institute of Technology
- Collaborations:
  - Louisiana State University
  - Pacific Northwest National Laboratory
  - Carnegie Mellon University
  - University of Tennessee, Knoxville
  - Stanford University
  - Other 6 NSSGT projects

Future HPC Expert

Acknowledgements

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.

Co-authors:
- William F Godoy (ORNL-CSMD)
- Pedro Valero-Lara (ORNL-CSMD)
- Het Mankad (ORNL-CSMD)

Organizers: OSDX 2024

Sponsors:
The ASCR Bluestone Project
CASS PESO and S4PST projects
OLCF/NERSC

Thanks to the audience!