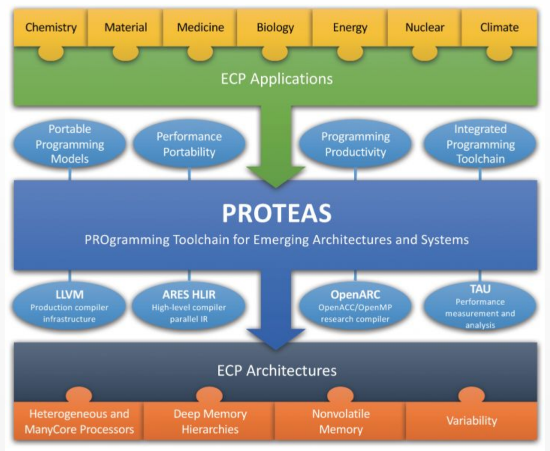
Exploring Julia as a unifying end-to-end workflow language for HPC on Frontier

Caira Anderson[1,2*]

Collaborators: William F. Godoy[2]; Pedro Valero-Lara[2]; Katrina Lee[3,2]; Ana Gainaru[2]; Rafael Ferreira da Silva[2]; Jeffrey S. Vetter[2]

[1]Cornell University; [2]Oak Ridge National Laboratory; [3]University of Texas at Dallas

*Contact: cairaanderson1@gmail.com



Abstract

- Objective: Explore the performance and scaling of Julia as an alternative language for high-performance computing (HPC) workflow component development on Frontier.
- Approach: We evaluated a 7-point stencil reaction-diffusion solver, GrayScott.jl using Julia's HPC stack: MPI.jl, AMDGPU.jl, and ADIOS2.jl.

Background

- Two-language problem: Scientists prototype solvers for various problems in high-productivity languages (e.g., Python), but need a solver in a traditional HPC language (e.g., C++, Fortran) for larger computations and speed.
- Julia: Meant to solve the two language problem by combining features of high-productivity languages with HPC capabilities.
  - High-productivity: Scientific/mathematical syntax, packaging environment, mathematical libraries, interactive, flexible
  - High-performance: MPI.jl, GPU programming model packages

Simulation

- Reaction-diffusion Gray-Scott 2-variable model:
  - *U, V* output concentrations of reacting and diffusing chemicals.

System of Equations	Discretization of Laplacian Term
$\frac{\partial U}{\partial t} = D_U \nabla^2 U - UV^2 + F(1-U) + nr$ $\frac{\partial V}{\partial t} = D_V \nabla^2 V + UV^2 + -(F+k)V$	$\nabla^2 U^t_{i,j,k} = -U^t_{i,j,k} + \frac{1}{6}\left[U^t_{i-1,j,k} + U^t_{i+1,j,k} + U^t_{i,j-1,k} + U^t_{i,j+1,k} + U^t_{i,j,k-1} + U^t_{i,j,k+1}\right]$

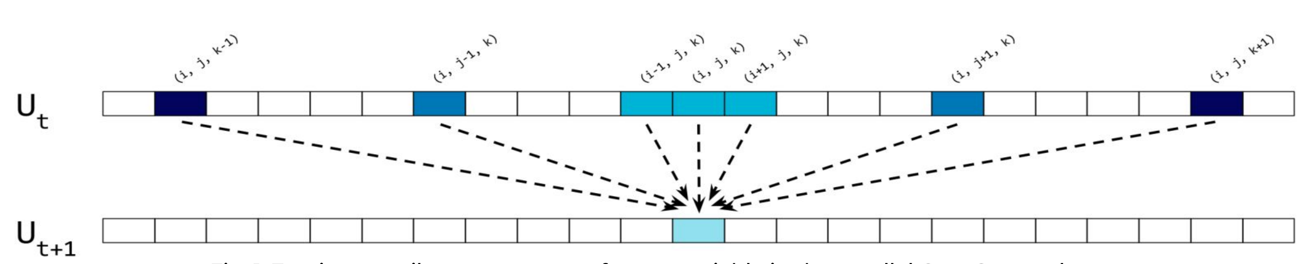


Fig. 1: 7-point stencil memory access for one variable in the parallel Gray-Scott solver.

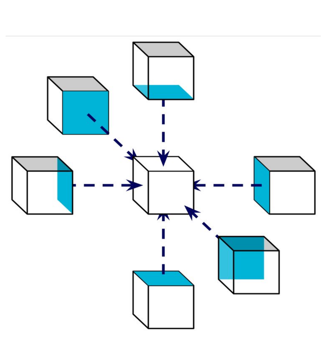


Fig. 2: MPI_Send/Recv memory patterns in Cartesian communicator decomposition used in simulation.

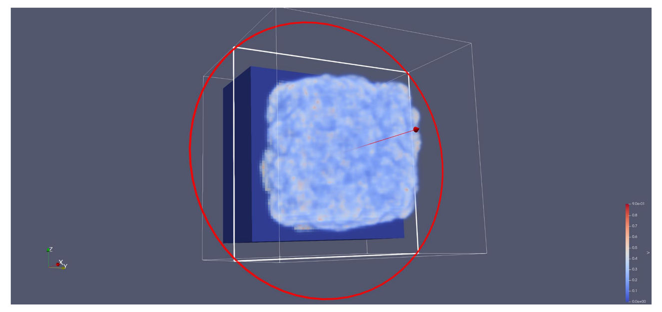
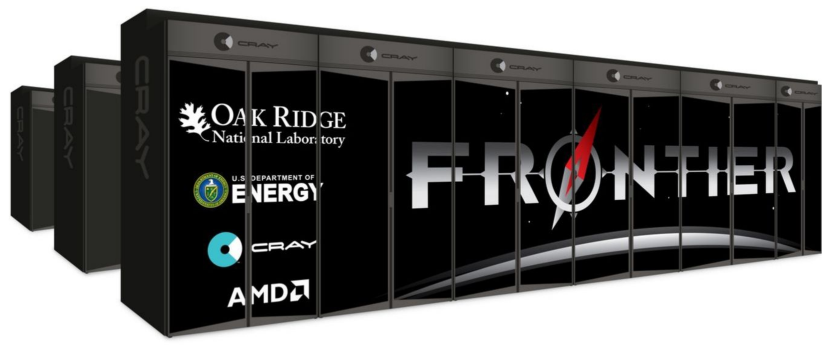


Fig. 3: *V* diffusing at t=109.

Frontier Exascale System

Frontier Characteristics	
Nodes	9,408
—GPU Architecture—	
GPU	4×AMD MI250X 8×GCDs
Memory Bandwidth	HBM2E 64 GB 1,600 GB/s per GCD



Implementation

- Running GrayScott.jl simulation in Julia on Frontier:

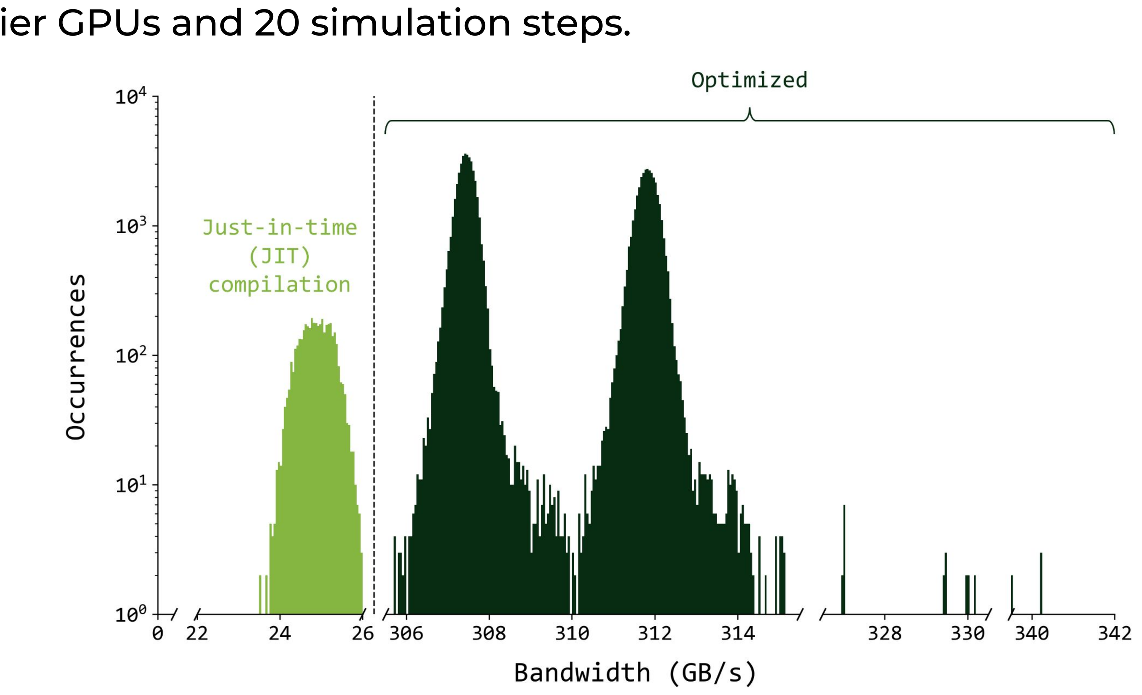
```
#1 Configure Frontier environment by loading modules.
#2 Set MPI backend; Tell MPIPreferences to use Cray's MPICH.
julia --project=$GS_DIR -e 'using Pkg; Pkg.add("MPIPreferences")'
julia --project=$GS_DIR -e 'using MPIPreferences; MPIPreferences.use_system_binary(;
library_names=["libmpi_cray"], mpiexec="srun")'
### Adds a custom branch in case the development version is needed (for devs to test new
features)
julia --project=$GS_DIR -e 'using Pkg;
Pkg.add(url="https://github.com/utkarsh530/AMDGPU.jl.git", rev="u/random")'
#3 Instantiate the project by installing packages listed in Project.toml.
julia --project=$GS_DIR -e 'using Pkg; Pkg.instantiate()'
### Verify the packages are installed correctly.
julia --project=$GS_DIR -e 'using Pkg; Pkg.build()'
julia --project=$GS_DIR -e 'using Pkg; Pkg.precompile()'
#4 Select number of MPI processes per number of GPU's when running the simulation.
srun -n 8 --ntasks-per-node=8 --gpus-per-node=8 --gpu-bind=closest julia --project=$GS_DIR
$GS_DIR/gray-scott.jl settings-files.json
    # Select GPU programming model (AMDGPU for Frontier) in settings-files.json.
    ###snippet from settings-files.json
     {
       "L": 2048,
        "output": "gs-8MPI-8GPU-2048L-F64.bp",
        "precision": "Float64",
       "backend": "AMDGPU"
     }
```

Results

- We compared the speed of accessing the GPU allocated arrays u and v for 7 read (fetch/load) and 1 write operations of our GrayScott.jl kernel in Julia with an AMD provided HIP implementation of a Laplacian kernel.

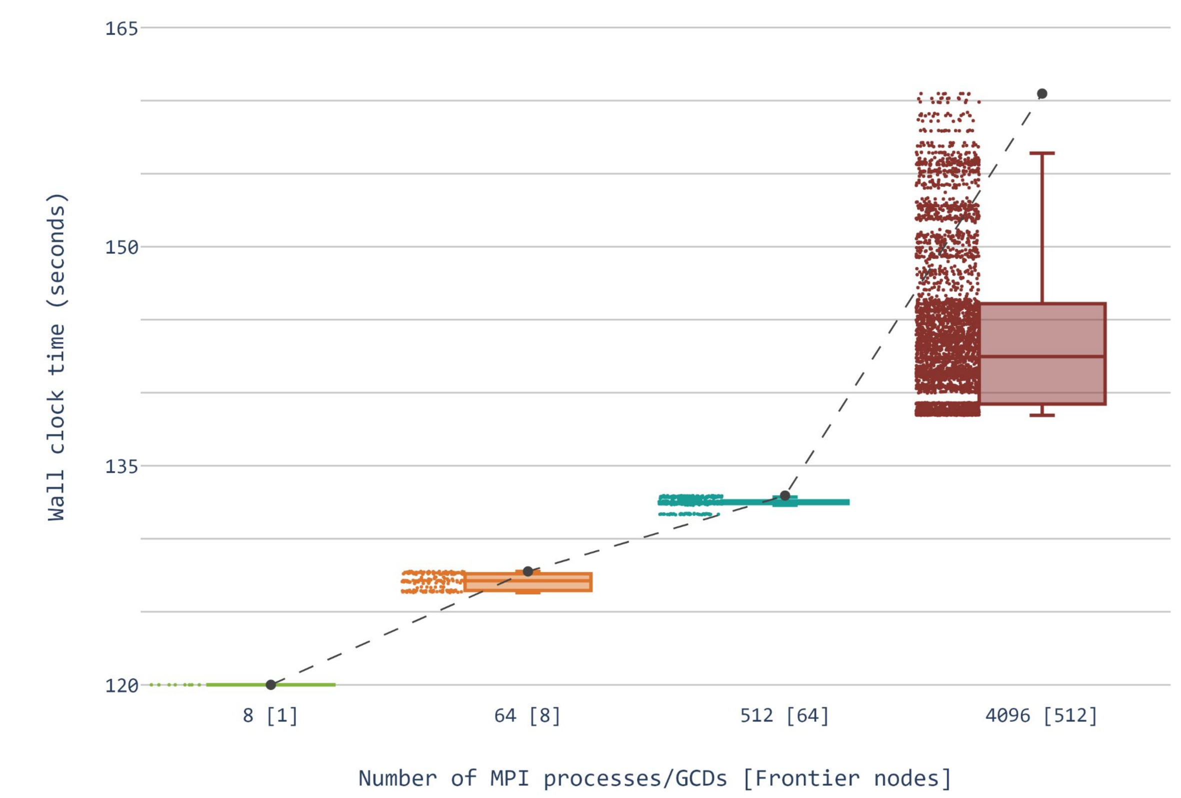
Kernel	Bandwidth (GB/s) Effective	Total
Julia GrayScott.jl		
- 2-variable (application)	312	570
- 1-variable no random	312	625
HIP single variable	599	1,163
Theoretical peak MI250x	1,600	

- We evaluated the bandwidth distribution of a run of GrayScott.jl across 4,096 Frontier GPUs and 20 simulation steps.

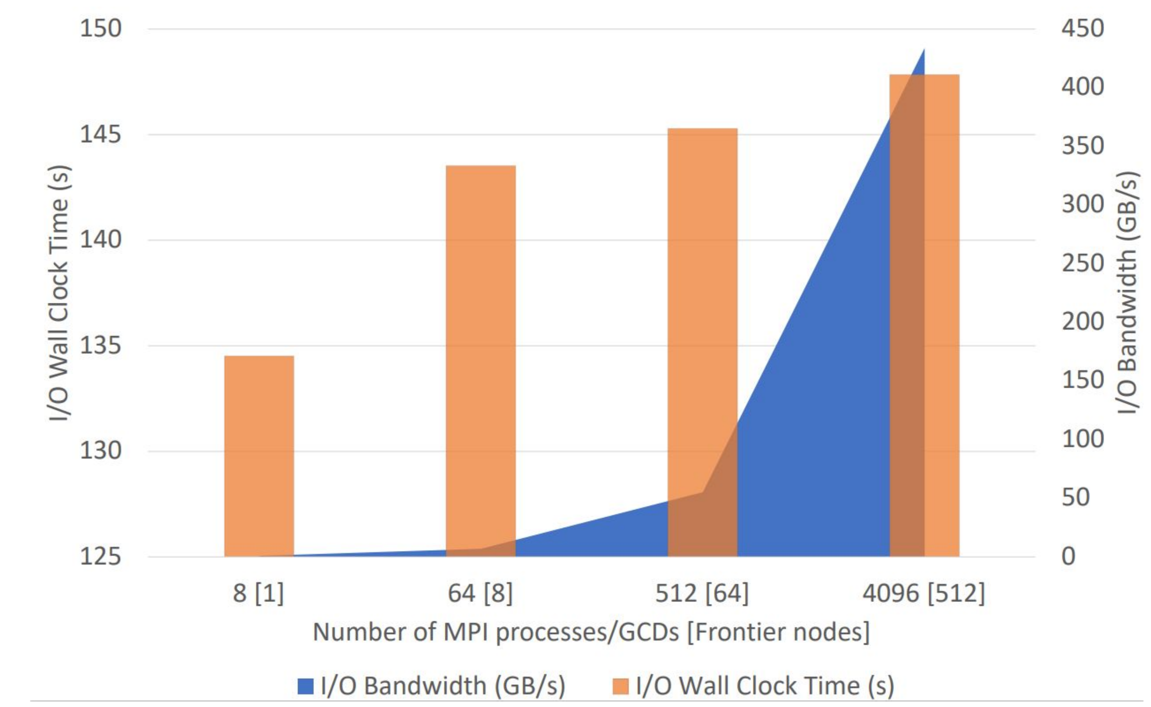


Results (cont.)

- We evaluated the weak scalability (without I/O) of the GrayScott.jl simulation on up to 4,096 GPUs or 512 nodes of Frontier.



- We evaluated the weak scalability of the parallel I/O component, ADIOS2.jl, saving only one output step.



Conclusion

- Julia achieved ~50% of the bandwidth of AMD's HIP implementation of a similar kernel.
- Julia's JIT causes performance to suffer initially.
- Weak scalability with and without I/O was somewhat linear, can indicate that Julia's bindings are lightweight layers on MPI, ADIOS2.
- Julia was able to complete simulation on up to ~50% of the Frontier System.
- Next Steps:
  - Further evaluate other metrics of performance: Strong scalability, compare with C++ version of GrayScott.jl.
  - Explore the capabilities of Julia's tools for data visualization and interactive computing with this HPC workload.

Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the US Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources of the Oak Ridge Leadership Computing Facility and the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC05-00OR22725.