Oak Ridge National Laboratory

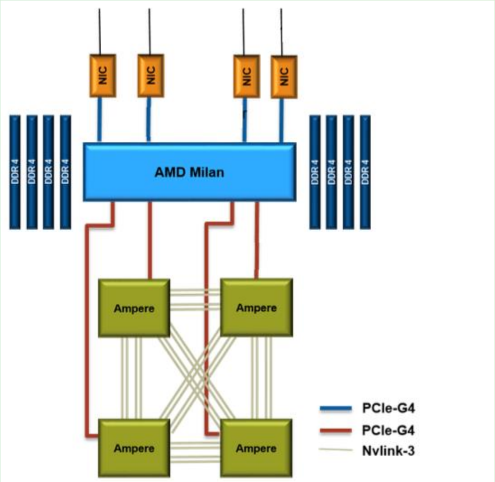
SC23 Denver, CO | i am hpc.

JACC.jl: On-node Performance Portable Programming Model in Julia

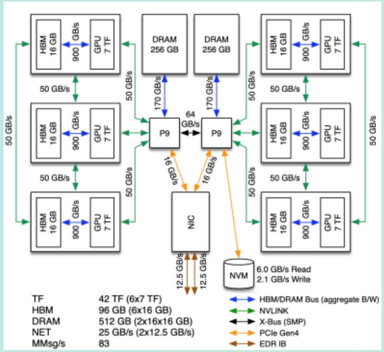
PhD. Pedro Valero-Lara,

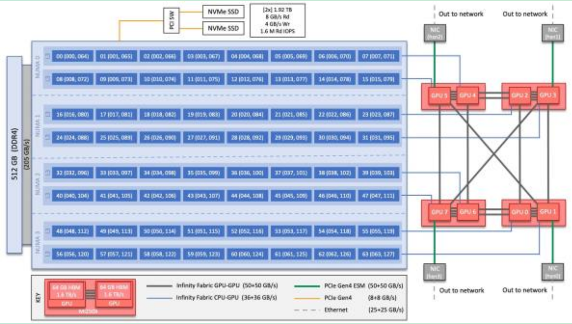
Computer Scientist at Programming Systems Group
valerolarap@ornl.gov

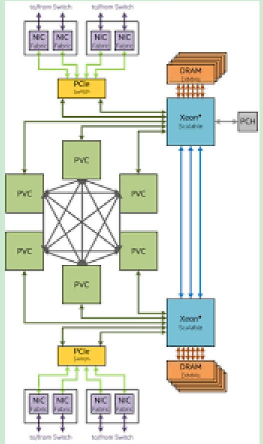
Motivation -> Performance Portability



- Performance portability is an important problem for the US Department of Energy (DOE)
- The DOE leadership computing facilities are "heterogeneous" GPU-accelerated systems
- Performant portable layers are a requirement in DOE: e.g. Kokkos, Raja, OpenMP, OpenACC
- Program "once" and deploy "many times"

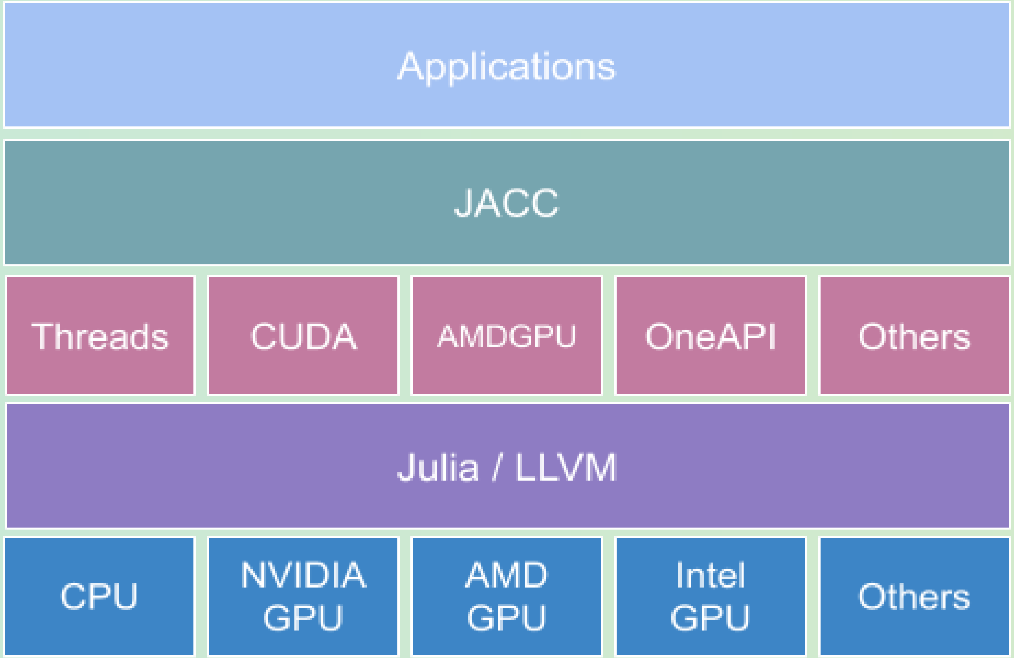


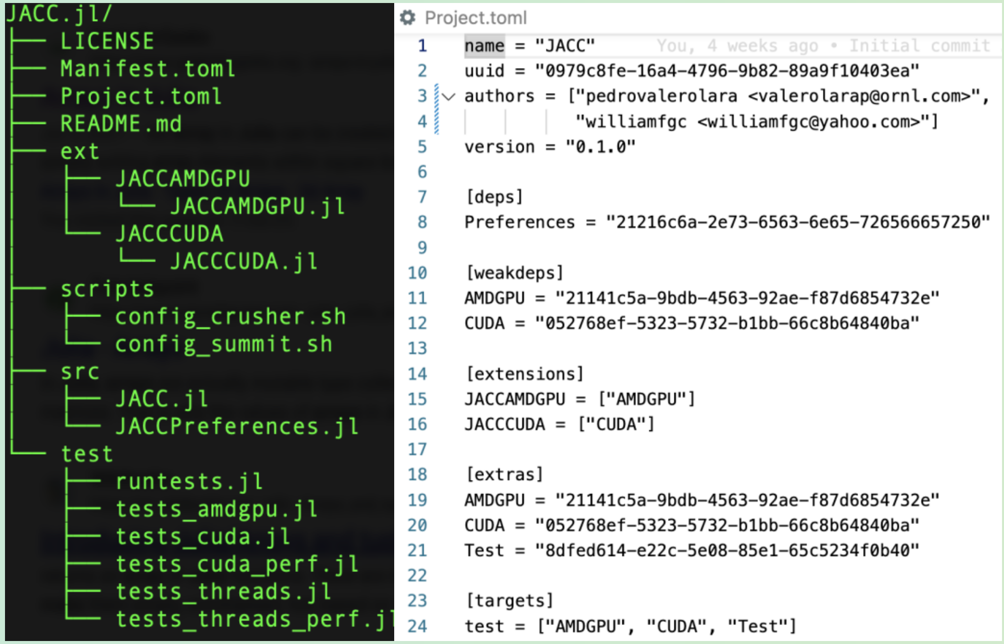




JACC.jl Model

- Uses a “functional” approach.
  Think of Kokkos and passing “kernel” functions
- Portable code front end between CPU and GPU
- We don’t reinvent the wheel:
  - Back ends: CPU Threads, and GPU CUDA.jl and AMDGPU.jl….OneAPI? (Intel GPUs?)
  - Preferences.jl to select a back end (we don’t mix back ends)
  - Julia v1.9 [weakdeps]
  - Coarse and fine granularity included





```
JACC.jl/
├── LICENSE
├── Manifest.toml
├── Project.toml
├── README.md
├── ext
│   ├── JACCAMDGPU
│   │   └── JACCAMDGPU.jl
│   └── JACCCUDA
│       └── JACCCUDA.jl
├── scripts
│   ├── config_crusher.sh
│   └── config_summit.sh
├── src
│   ├── JACC.jl
│   └── JACCPreferences.jl
└── test
    ├── runtests.jl
    ├── tests_amdgpu.jl
    ├── tests_cuda.jl
    ├── tests_cuda_perf.jl
    ├── tests_threads.jl
    └── tests_threads_perf.jl
```

Project.toml

```
name = "JACC"        You, 4 weeks ago • Initial commit
uuid = "0979c8fe-16a4-4796-9b82-89a9f10403ea"
authors = ["pedrovalerolara <valerolarap@ornl.com>",
           "williamfgc <williamfgc@yahoo.com>"]
version = "0.1.0"

[deps]
Preferences = "21216c6a-2e73-6563-6e65-726566657250"

[weakdeps]
AMDGPU = "21141c5a-9bdb-4563-92ae-f87d6854732e"
CUDA = "052768ef-5323-5732-b1bb-66c8b64840ba"

[extensions]
JACCAMDGPU = ["AMDGPU"]
JACCCUDA = ["CUDA"]

[extras]
AMDGPU = "21141c5a-9bdb-4563-92ae-f87d6854732e"
CUDA = "052768ef-5323-5732-b1bb-66c8b64840ba"
Test = "8dfed614-e22c-5e08-85e1-65c5234f0b40"

[targets]
test = ["AMDGPU", "CUDA", "Test"]
```

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

JACC.jl Description

- Descriptive, not prescriptive
- Run simple portable kernels on CPU and GPU
- Components:
  - JACC.Array: alias to the corresponding back end
  - JACC.parallel_for (N, f, x…)
  - f: function “argument”
  - x… variadic

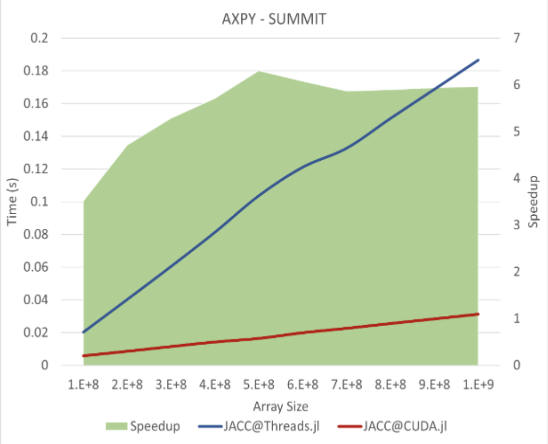
```
import JACC
function axpy (i , alpha , x , y )
  x[i] += alpha * y[i]
end

SIZE = 1000000
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)
```

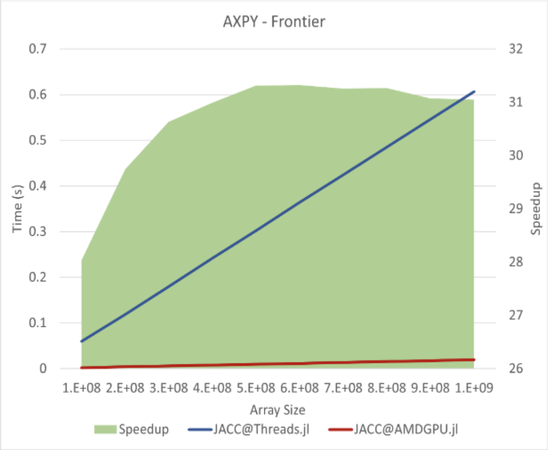
Performance

Summit/Frontier

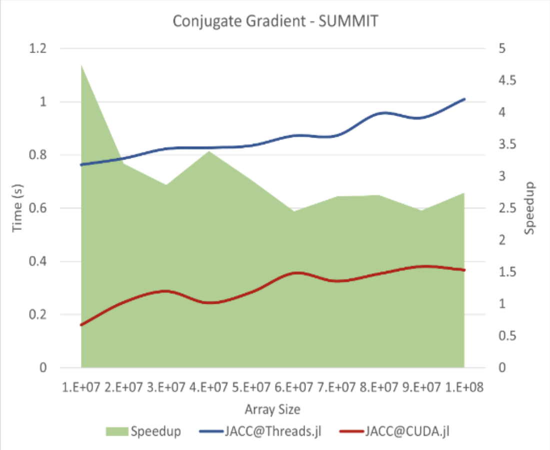
Architecture	Summit	Frontier
—CPU Architecture—		
CPU	IBM Power9	AMD EPYC 7A53
Cores	22	64
L1	32 KB	1,792 KB
L2 (unified)	512 KB	14 MB
L3 (unified)	10 MB	64 MB
Memory	DDR4 256 GB	DDR4 512 GB
Bandwidth	170 GB/s	205 GB/s
—GPU Architecture—		
GPU	6 × NVIDIA V100	4 × AMD MI250X
		8 × GCDs
Frequency	1,455 MHz	1,700 MHz
Cores	5,120	14,080
SM/CU Count	80	220
L1	up to 96 KB per SM	16 KB per CU
L2 (unified)	6,144 KB	8,192 KB per GCD
Memory	HBM2 16 GB	HBM2E 64 GB
Bandwidth	900 GB/s	3,276.8 GB/s
—Connectivity—		
GPU-to-GPU	NVLink 2.0	Infinity Fabric
Bandwidth	50 GB/s	50–100 GB/s
GPU-to-CPU	NVLink 2.0	Infinity Fabric
Bandwidth	50 GB/s	36 GB/s
—Compiler—		
	Julia 1.9.1	Julia 1.9.0
—Julia flags/env—		
	–threads 21	–threads 64
	JULIA_EXCLUSIVE=1	

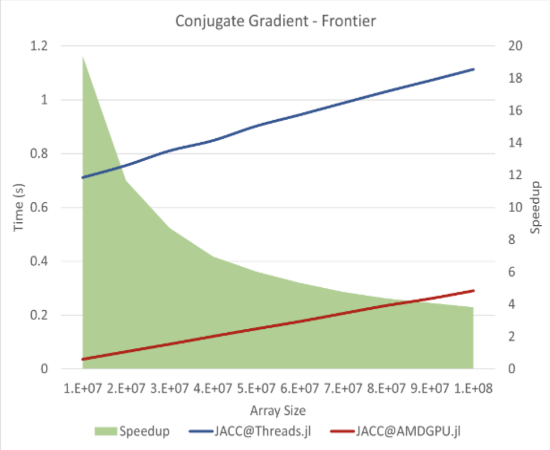
AXPY



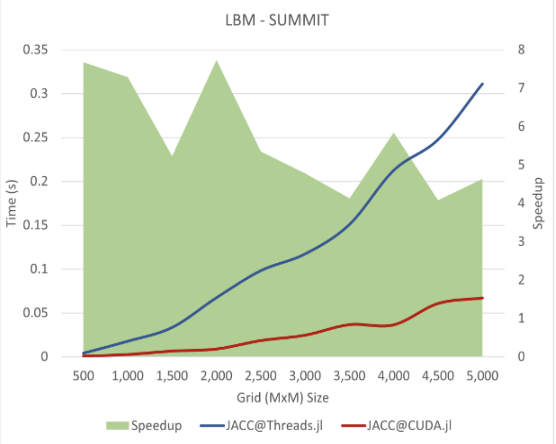


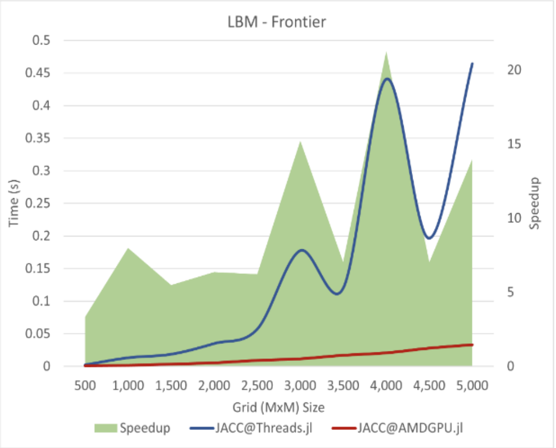
Conjugate Gradient





Lattice Boltzmann





Conclusions

- We present a “performance portable” programming model in Julia
- We try to keep programming in Julia **as simple as possible**
- Overhead is minimal thanks to Julia features: weakdeps, multiple dispatch, type alias, Preferences.jl
- Expand to more relevant kernels in DOE so people can try Julia in a “write once” descriptive style
- Feedback is welcome!

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

JACC.jl: On-node Performance Portable Programming Model in Julia

PhD. Pedro Valero-Lara,

Computer Scientist at Programming Systems Group
valerolarap@ornl.gov