# cuNumeric.jl : Automating Distributed Numerical Computing

**David Krasowska**[1], Ethan Meitz[2], Wonchan Lee[3]

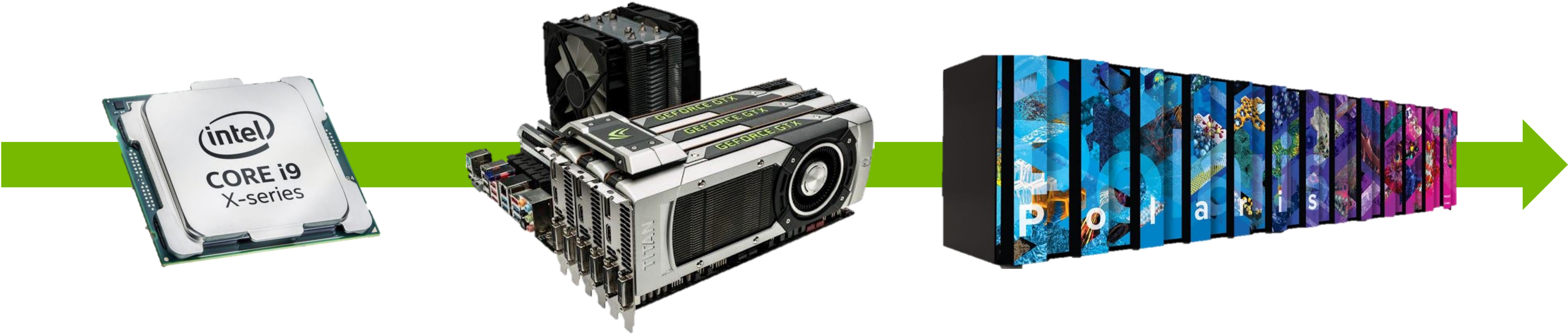
[1]Northwestern University
[2]Carnegie Mellon University
[3]NVIDIA

SC25 St. Louis, MO | hpc ignites.

Julia for HPC BoF

# The Goal: Scale with Zero Code Changes

- “Easy” to implement the correct physics in a high-level language like Python, Julia, or MATLAB
- Time consuming to modify code to scale across multiple CPUs/GPUs
  - Need to learn and debug new technologies like MPI, CUDA etc.




**Code that runs on a single CPU core should also be able to run on multiple cores, multiple GPUs and across multiple nodes**

# Legate Enables Composable and Distributed Libraries

By providing data and task management abstractions, this enables the efficient implementation of complex library APIs.

NumPy · SciPy · pandas · scikit-learn · dmlc XGBoost · HDF · ...

Today's scope

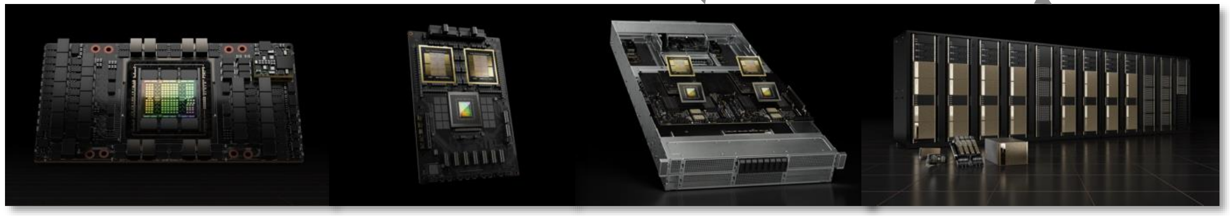
| cuPyNumeric | Legate Sparse | Legate Dataframe | Legate RAFT | Legate Boost | Legate IO |
|---|---|---|---|---|---|

**Legate Framework**

**Legate Core** - Productivity & Composability Layer

**Legion** - Implicit Parallelism Layer

**Realm** - Runtime for Scalable and Portable Execution

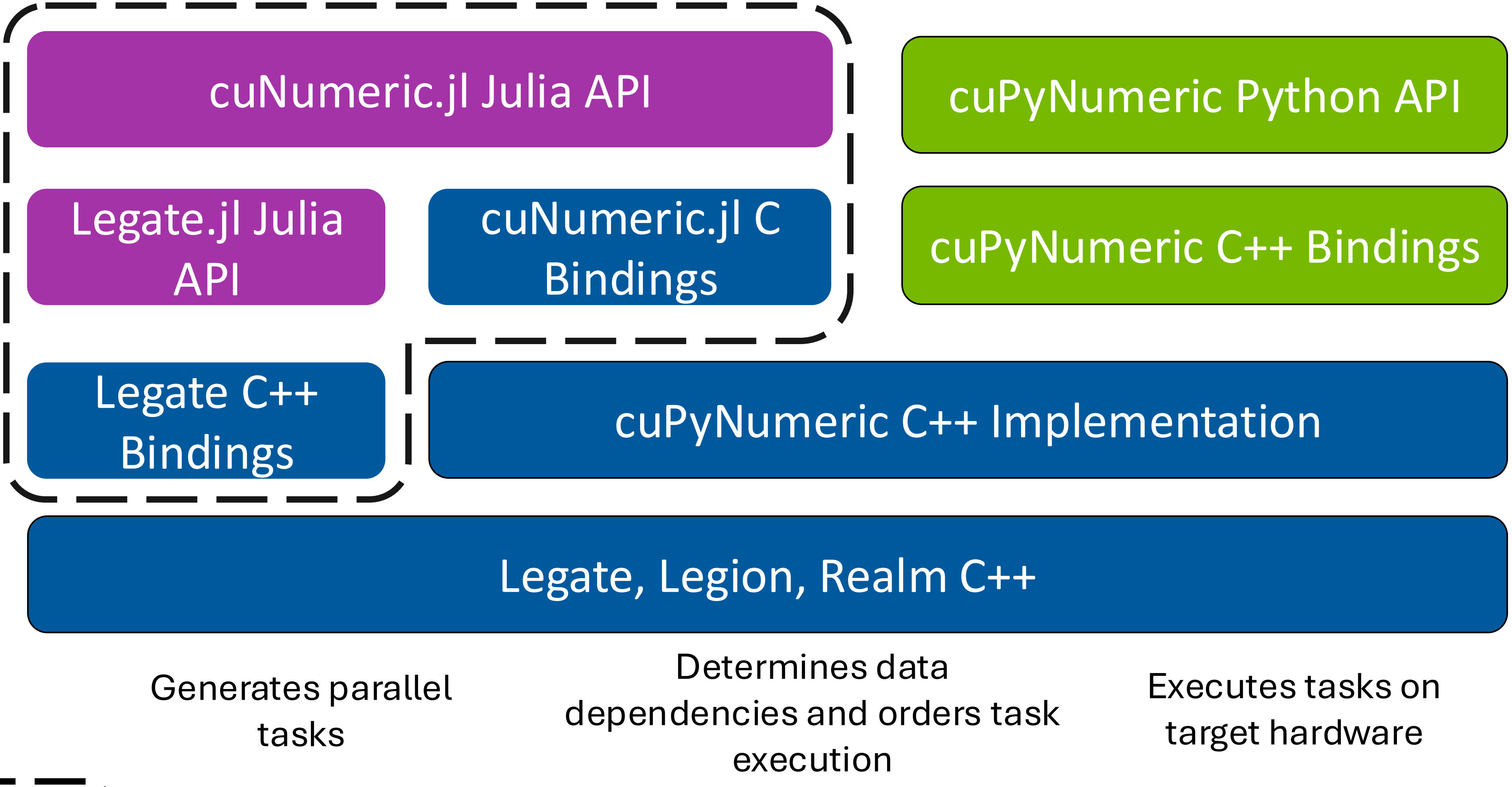


Single GPU · Mixed CPU/GPU · Single-Node Multi-GPU · Multi-Node Multi-GPU Cloud & Supercomputer

**Wouldn't this be great in Julia?**

https://developer.nvidia.com/legate
https://legion.stanford.edu/

# Software Stack



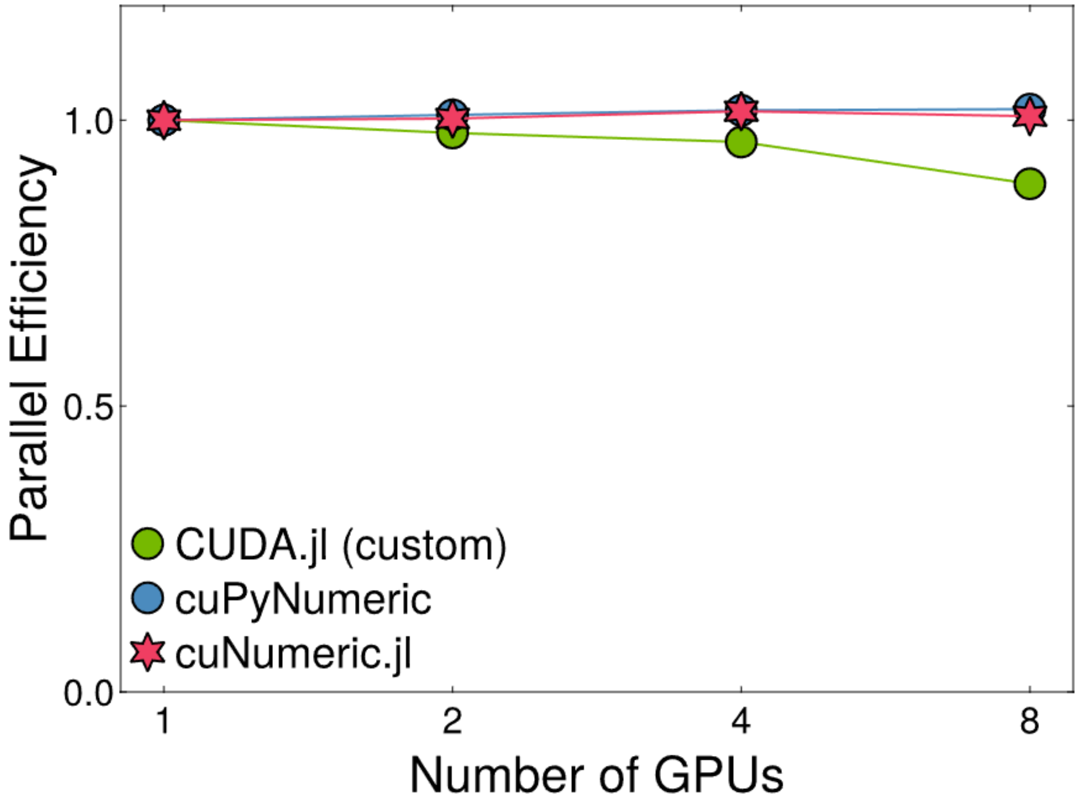

New Contributions

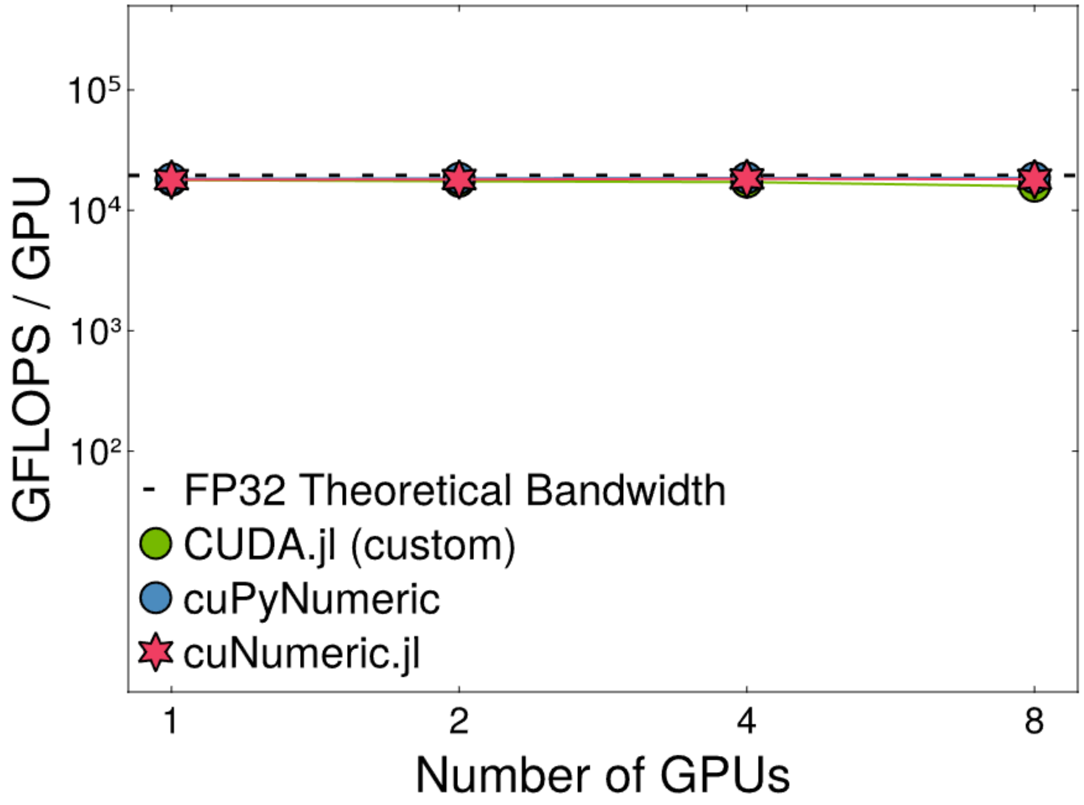
# Matrix Multiplication

## Syntax:

```
N = 10
A = cuNumeric.rand(Float32, N, N)
B = cuNumeric.rand(Float32, N, N)
C = cuNumeric.zeros(Float32, N, N)
mul!(C, A, B)
```

## Benchmark (8x A100):



# Running CUDA Kernels with cuNumeric.jl

```julia
using cuNumeric
using CUDA

function kernel_add(a, b, c, N)
    i = (blockIdx().x - 1i32) * blockDim().x + threadIdx().x
    if i <= N
        @inbounds c[i] = a[i] + b[i]
    end
    return nothing
end


N = 1024
threads = 256
blocks = cld(N, threads)

a = cuNumeric.full(N, 1.0f0)
b = cuNumeric.full(N, 2.0f0)
c = cuNumeric.ones(Float32, N)

task = cuNumeric.@cuda_task kernel_add(a, b, c, UInt32(1))

cuNumeric.@launch task=task threads=threads blocks=blocks \
                    inputs=(a, b) outputs=c scalars=UInt32(N)

```

CUDA.jl kernel

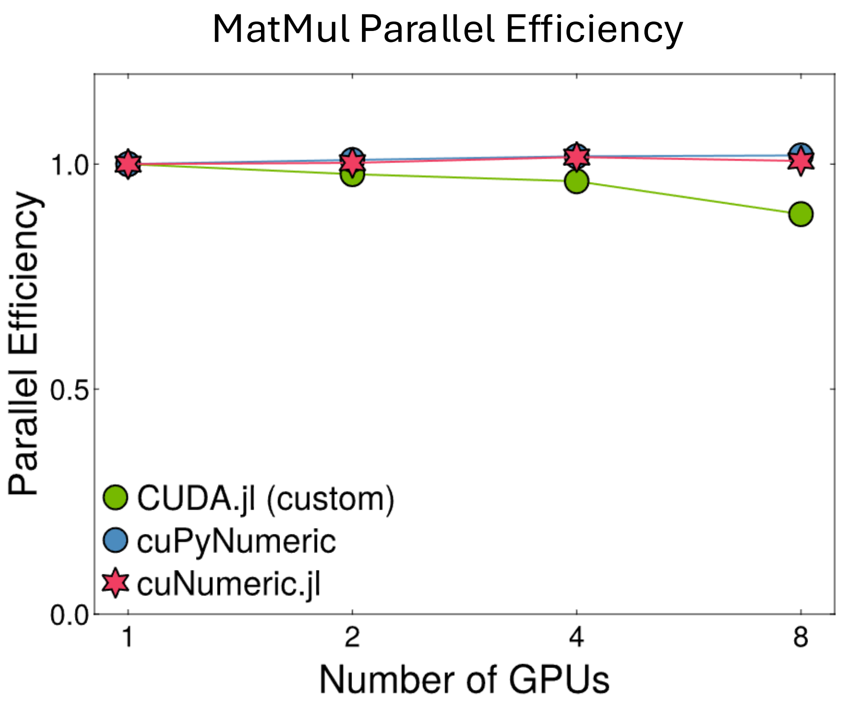
Initialize NDArrays

Compile

Launch

# Conclusions and Future Work

- Minimal code changes for scaling across large heterogeneous distributed systems
- Good weak scaling efficiency
- Ability to register custom CUDA kernels




**Disclaimers:**

1. Still in development, we will be releasing a public beta within the coming months

2. cuPyNumeric and Legate are currently in beta

**David Krasowska** krasow@u.northwestern.edu

Ethan Meitz emeitz@andrew.cmu.edu

Wonchan Lee wonchanl@nvidia.com

Check out our repo
https://github.com/JuliaLegate/cuNumeric.jl/

This material is based upon work supported by the U.S. Department of Energy, National Nuclear Security Administration, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award Number(s) DE-SC0024386.