Reducing Complexity in Scalable Numerical Computing with cuNumeric.jl

David Krasowska Second Year Fellow

Introduction

With the end of Moore's Law, demand for specialized accelerators has surged to meet modern performance needs. Scientific computing often relies on large high-performance computing (HPC) clusters, which increasingly include many-core and heterogeneous systems. However, programming such systems introduces complexity in **memory management**, **parallelism**, and **performance portability**.

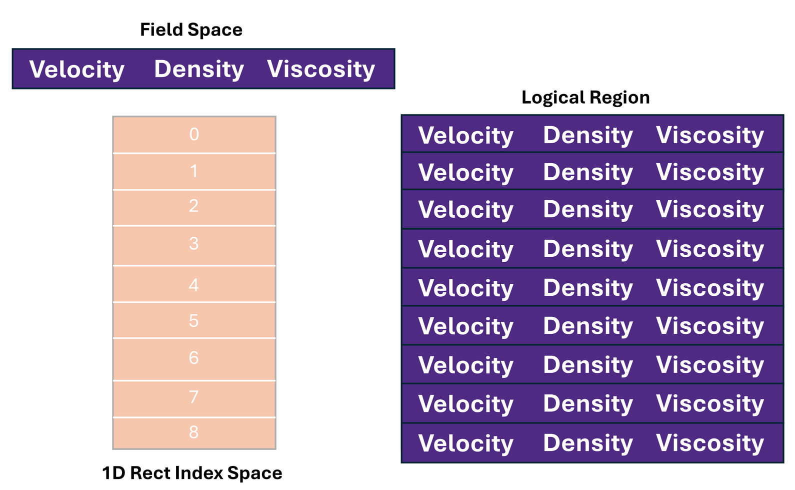
Claim: cuNumeric.jl will allow scientific developers to harness the power of GPU-accelerated HPC systems with minimal effort and maximum portability.

Legion

Achieving high performance requires the programming system understand the structure of program data to facilitate efficient placement and movement of data.

Legion [2]: runtime system and program model for distributed heterogeneous machines. Application data is structured by:

- Implicitly extract parallelism from the logical regions
- Data movement operations in accordance with the application data properties
- All tasks specify privileges and coherence that logical regions will access

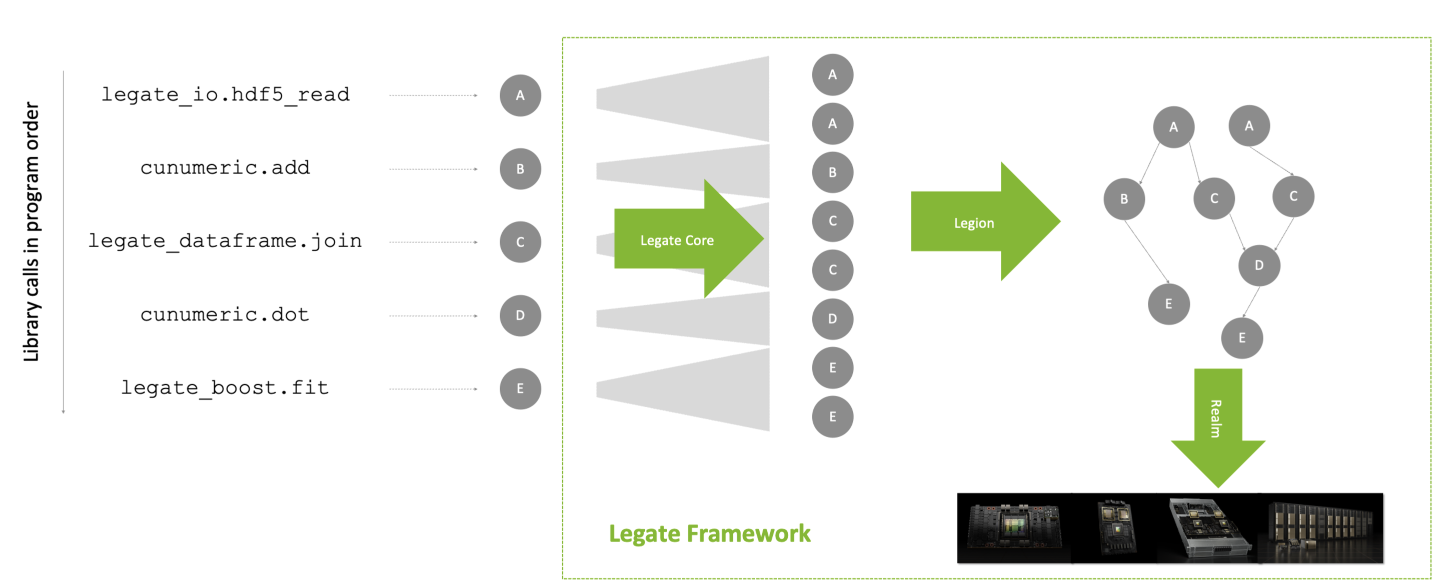


Legion utilizes large task graphs to minimize kernel invocation and communication latencies.

Legate and cuPyNumeric

Legate [1]: Provides an abstraction over the Legion runtime through a unified library interface, enabling efficient implmenetation of complex APIs that scale. cuPyNumeric is a Legate library that provides a Python Numpy API replacement for scalable execution.

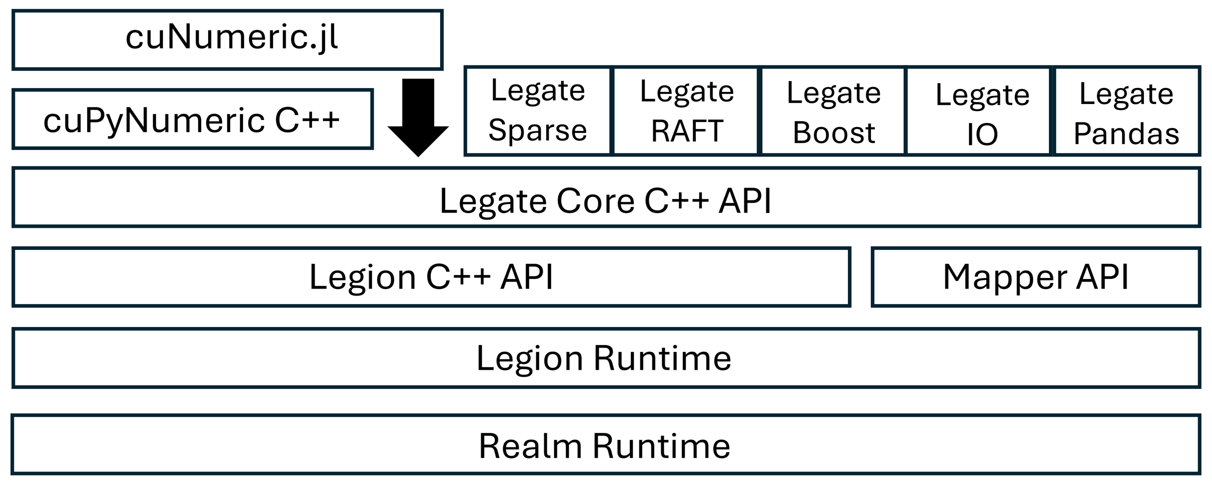
Motivation: "Code that runs on a single CPU should also be able to run on multiple cores, multiple GPUs and across multiple nodes"



cuNumeric.jl

Leverages existing infrastructure for cuPyNumeric to bring seamless distributed computing to Julia.

- Legion is used to schedule Julia tasks, enabling implicit parallelism
- Matches high level array abstractions within Julia
- Behavior a Julia programmer would "expect"



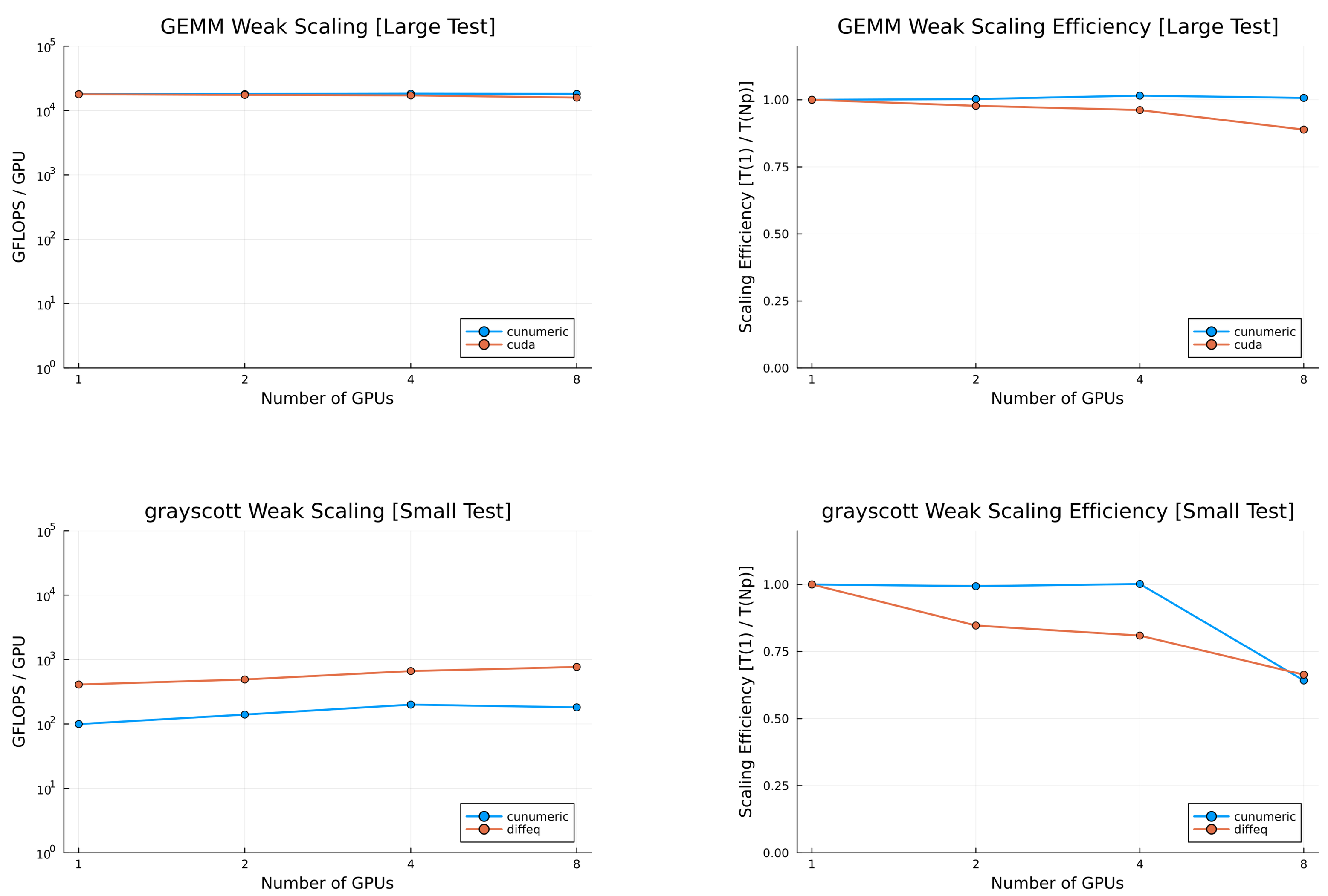
API examples

```julia
# found in examples/integrate.jl
using cuNumeric

integrand = (x) -> exp(-square(x))

N = 1_000_000

x_max = 5.0
domain = [-x_max, x_max]
Ω = domain[2] - domain[1]

samples = Ω*cuNumeric.rand(NDArray, N) - x_max
estimate = (Ω/N) * sum(integrand(samples))
```

```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)
```

More details are found on our repo:
github.com/JuliaLegate/cuNumeric.jl

Results



These results are provided using [1:8] NVIDIA A100 GPUs.

1. Good weak scaling on GEMM. Outperforms CUDA.jl multiGPU execution
2. ImplicitGlobalGrid.jl performs better on Gray-Scott 2D heat diffusion, particularly in GFLOPS

Conclusions and Future Work

- Minimal code changes that allow users to scale code across large heterogeneous distributed systems
- Good weak scaling efficiency on a variety of applications
- Ability to register custom CUDA kernels

As this is a work in progress, we have next steps

- Support a wider range of custom CUDA kernels
- Robust and accessible package installation
- More integration with the Julia Abstraction interface + Legate

Acknowledgments

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. We would like to thank the Legion team and NVIDIA for their support and feedback during this work in progress.

[1] Michael Bauer and Michael Garland. Legate numpy: accelerated and distributed array computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: Expressing locality and independence with logical regions. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2012.