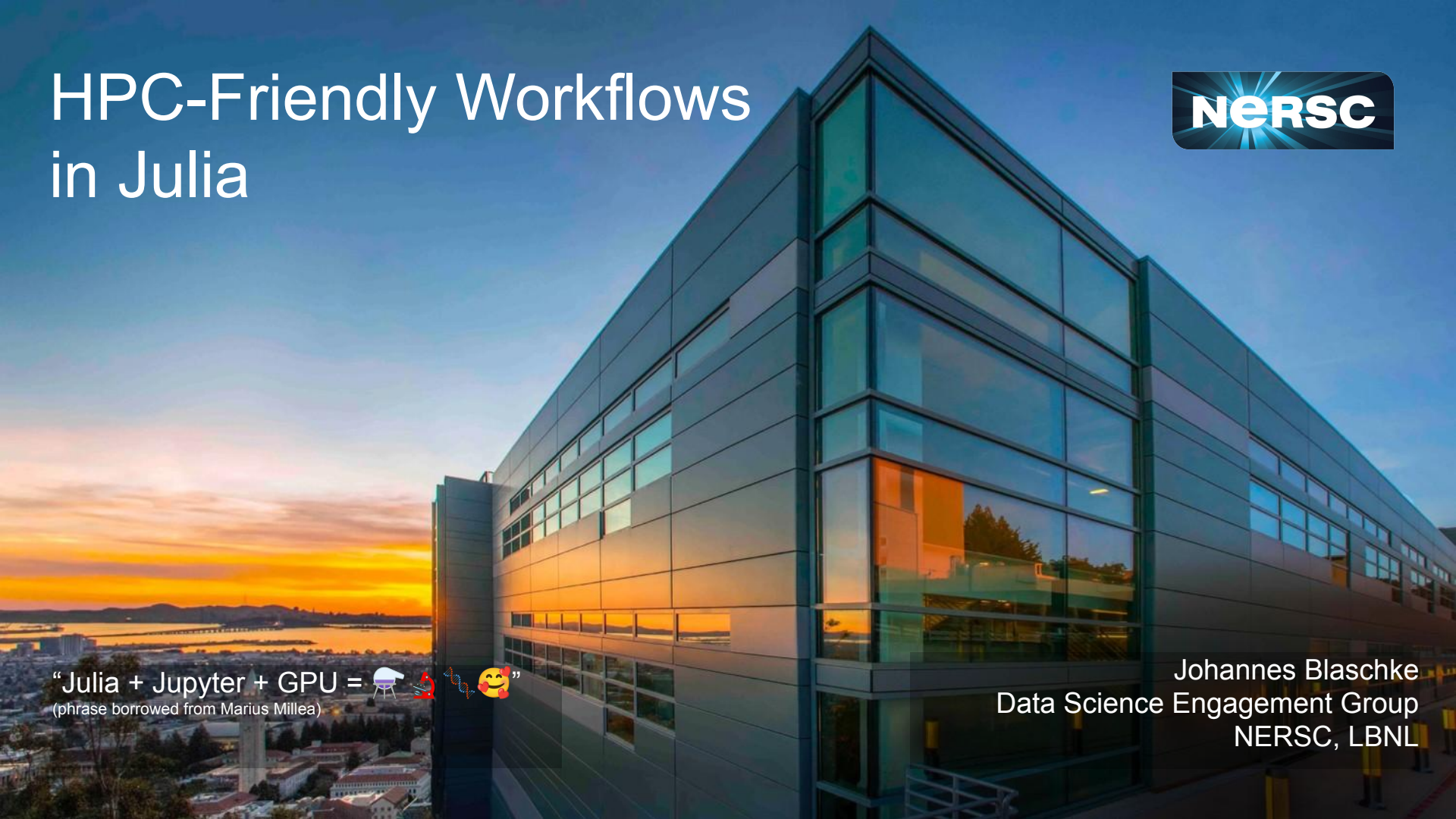
HPC-Friendly Workflows in Julia



"Julia + Jupyter + GPU = ⚗️ 🚬 🧬 🥰"
(phrase borrowed from Marius Millea)

Johannes Blaschke
Data Science Engagement Group
NERSC, LBNL

Credit and Disclaimers

None of this would be possible without:

- Tim Bersard, Valentin Churavy, Julian Samaroo (MIT Julia Lab) + Anton Smirnov (AMD) + Carsten Bauer (NHR, PC2)
  - Providing the Infrastructure
- Marius Millea (UC Davis) + Mark Hirsbrunner (LBNL*) + William Godoy, Pedro Valero Lara (OLCF)
  - Inspiring applications
- The Julia for HPC working group
  - https://github.com/JuliaParallel
  - Meets monthly on Zoom (cf. https://julialang.org/community/) and is very active on Discord
  - Julia for HPC BoF at SC and JuliaCon
- Soham Ghosh (NERSC)
  - Exploring AI applications: just-in-time AI, AI for science, UQ

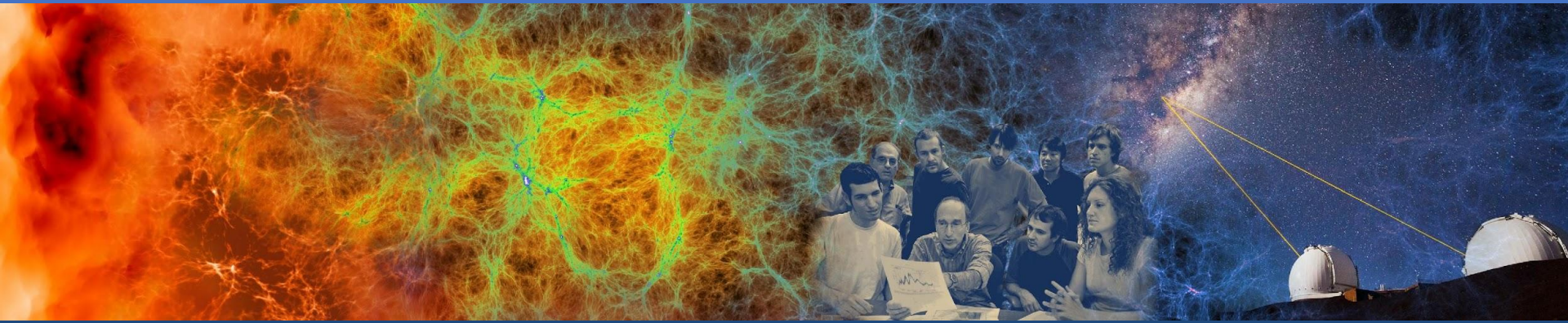
Disclaimer:

- I work on Perlmutter, but can be (and has been) easily applied to HPC more broadly.

Overview

1. Julia in 60s
2. Julia + Jupyter as an interactive workflow engine
3. Network Discovery
4. Programming GPUs
5. Inspiration: Particles in Potentials
6. Machine Learning using `Flux.jl`
7. Using `Dagger.jl` to parallelize your workflow

Julia in 60s

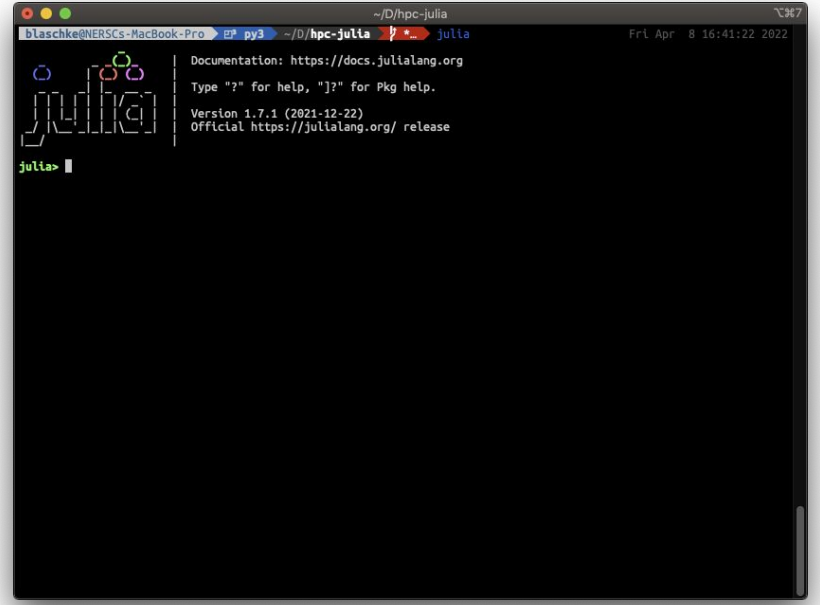


Julia is a High-Productivity Language

- It has all the modern HP features (rich stdlib, gc, …)

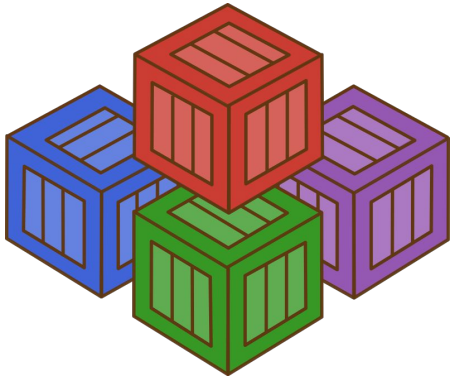
Julia is a High-Productivity Language

- It has all the modern HP features (rich stdlib, gc, …)
- + a powerful REPL



Julia is a High-Productivity Language

- It has all the modern HP features (rich stdlib, gc, …)
- + a powerful REPL
- + a comprehensive package manager (which integrates with system software)



`Project.toml`:

```
[extras]
MPIPreferences = "3da0fdf6-3ccc-4f1b-acd9-58baa6c99267"
CUDA_Runtime_jll = "76a88914-d11a-5bdc-97e0-2f5a05c973a2"
```

`LocalPreferences.toml`:

```
[MPIPreferences]
_format = "1.1"
abi = "MPICH"
binary = "system"
cclibs = ["cupti", "cudart", "cuda", "sci_gnu_82_mpi", "sci_gnu_82",
"dl", "dsmml", "xpmem"]
libmpi = "libmpi_gnu_91.so"
mpiexec = "srun"
preloads = ["libmpi_gtl_cuda.so"]
preloads_env_switch = "MPICH_GPU_SUPPORT_ENABLED"

[CUDA_Runtime_jll]
local = "true"
version = "11.7"
```

Julia has LLVM under the Hood

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

[10]:
```
f() = @ccall clock()::Cint
```

[10]:
```
f (generic function with 1 method)
```

[11]:
```
f()
```

[11]:
```
8698048
```

[12]:
```
@code_llvm(f())
```

```
;  @ In[10]:1 within `f`
define i32 @julia_f_1782() #0 {
top:
  %0 = call i32 inttoptr (i64 139954008063022 to i32 ()*)()
  ret i32 %0
}
```

[ ]:
```
```

Julia has LLVM under the Hood

Julia data types are binary-compatible with C

`@ccall` equivalent to c function call

SC23.ipynb (auto-x) - JupyterLab

```
[10]: 1 f() = @ccall clock()::Cint

[10]: f (generic function with 1 method)

[11]: 1 f()

[11]: 8698048
```

```
; @ In[10]:1 within `f`
define i32 @julia_f_1782() #0 {
top:
  %0 = call i32 inttoptr (i64 139954008063022 to i32 ()*)()
  ret i32 %0
}
```

```
[ ]: 1
```

Julia has LLVM under the Hood

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

```
[10]: f() = @ccall clock()::Cint
[10]: f (generic function with 1 method)
[11]: f()
[11]: 8698048
```

```
[12]: @code_llvm(f())
; @ In[10]:1 within `f`
define i32 @julia_f_1782() #0 {
top:
  %0 = call i32 inttoptr (i64 139954008063022 to i32 ()*)()
  ret i32 %0
}
```

`@code_llvm` exposes the LLVM IR for debug purposes

Julia has a Powerful Type System

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

[1]:
```julia
struct MyType{T<:Number}
    a::T
    b::T
    c::Int64
end
```

[2]:
```julia
x = MyType(1,2,3)
```

[2]:
```
MyType{Int64}(1, 2, 3)
```

[3]:
```julia
x = MyType(1.,2.,3)
```

[3]:
```
MyType{Float64}(1.0, 2.0, 3)
```

[5]:
```julia
f(a::Int) = "Hi"
f(b::Float64) = "Ho"
```

[5]:
```
f (generic function with 2 methods)
```

[6]:
```julia
f(1)
```

[6]:
```
"Hi"
```

[7]:
```julia
f(1.)
```

[7]:
```
"Ho"
```

Julia has a Powerful Type System

Structured data types are also compatible with C

`{T<:Number}` represents a type template for all types inheriting from `Number`

SC23.ipynb (auto-x) - JupyterLab

[1]:
```julia
struct MyType{T<:Number}
    a::T
    b::T
    c::Int64
end
```

[2]: `MyType{Int64}(1, 2, 3)`

[3]:
```julia
x = MyType(1.,2.,3)
```

[3]: `MyType{Float64}(1.0, 2.0, 3)`

[5]:
```julia
f(a::Int) = "Hi"
f(b::Float64) = "Ho"
```

[5]: `f (generic function with 2 methods)`

[6]:
```julia
f(1)
```

[6]: `"Hi"`

[7]:
```julia
f(1.)
```

[7]: `"Ho"`

Julia has a Powerful Type System

`{T<:Number}` represents a type template for all types inheriting from `Number`

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

```
[1]: struct MyType{T<:Number}
     a::T
```

```
[2]: x = MyType(1,2,3)

[2]: MyType{Int64}(1, 2, 3)

[3]: x = MyType(1.,2.,3)

[3]: MyType{Float64}(1.0, 2.0, 3)
```

```
f(b::Float64) = "Ho"

[5]: f (generic function with 2 methods)

[6]: f(1)

[6]: "Hi"

[7]: f(1.)

[7]: "Ho"
```

Julia has a Powerful Type System

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

[1]:
```julia
struct MyType{T<:Number}
    a::T
    b::T
    c::Int64
end
```

[2]:
```julia
x = MyType(1,2,3)
```

[2]:
```
MyType{Int64}(1, 2, 3)
```

[5]:
```julia
f(a::Int) = "Hi"
f(b::Float64) = "Ho"
```

[5]:
```
f (generic function with 2 methods)
```

[6]:
```julia
f(1)
```

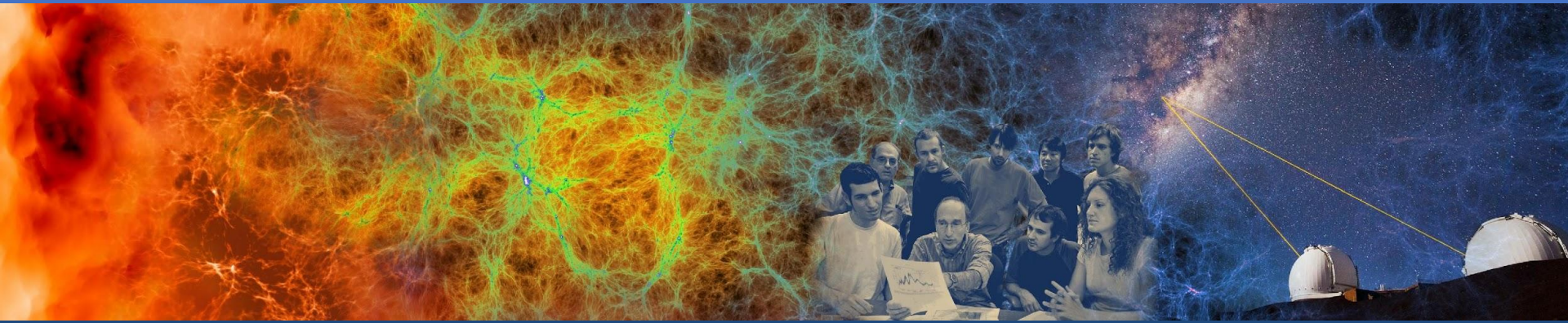
[6]:
```
"Hi"
```

[7]:
```julia
f(1.)
```

[7]:
```
"Ho"
```

Julia has multiple dispatch: a function can have several implementations (methods) depending on the input types

Julia + Jupyter as an interactive workflow engine



DOE SC User Requirements Are Evolving

IRI Science Patterns (3)

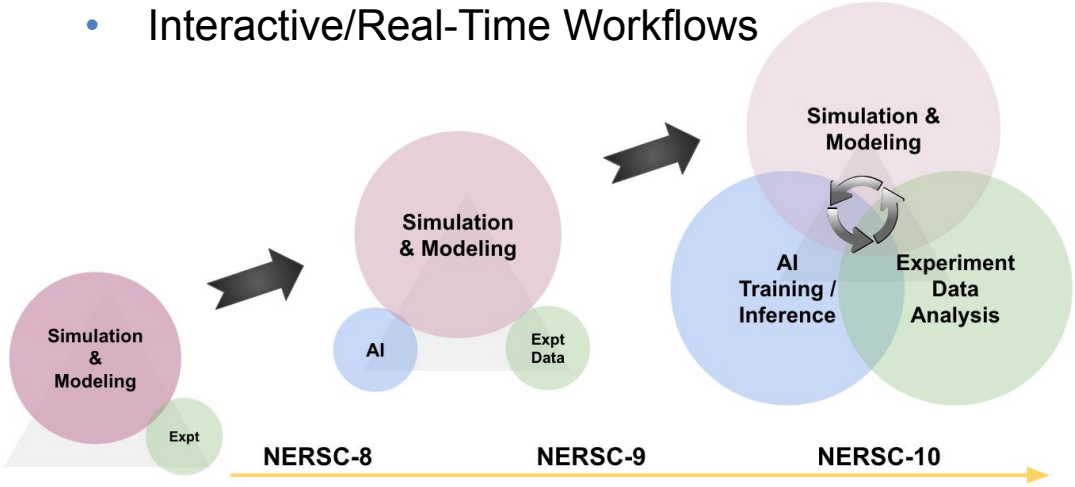
Time-sensitive pattern has *urgency*, requiring real-time or end-to-end performance with high reliability, e.g., for timely decision-making, experiment steering, and virtual proximity.

Data integration-intensive pattern requires combining and analyzing data from multiple sources, e.g., sites, experiments, and/or computational runs.

Long-term campaign pattern requires sustained access to resources over a long period to accomplish a well-defined objective.

Users require support for

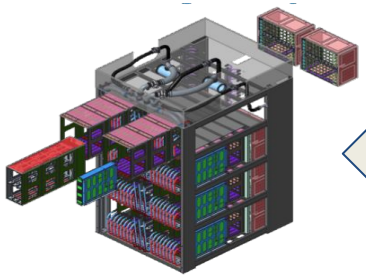
- End-to-end DOE SC Workflows involving multiple facilities
- New modes of scientific discovery through the integration of simulation & modeling, AI and experiment.
- Interactive/Real-Time Workflows



Why NERSC Cares about Julia Workflows

Function signature	Pybind11		ccall		speedup
`int fn0()`	132	±14.9	2.34	±1.24	56×
`int fn1(int)`	217	±20.9	2.35	±1.33	92×
`double fn2(int, double)`	232	±11.7	2.32	±0.189	100×
`char* fn3(int, double, char*)`	267	±28.9	6.27	±0.396	42×

`pip install …`
Rapid scripting



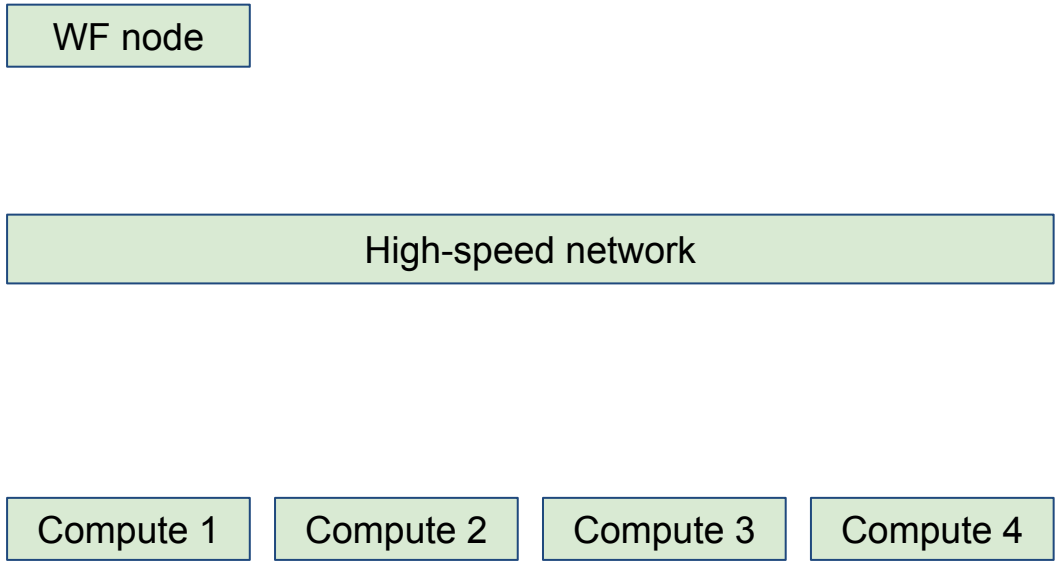
C/C++

Python

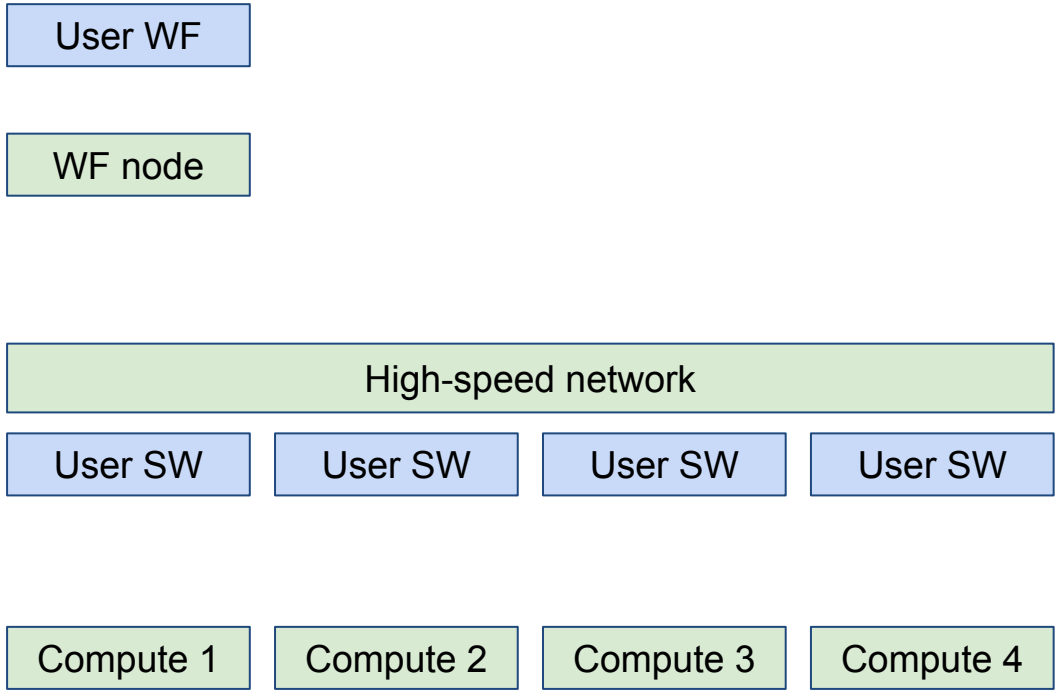
Julia

Workflows

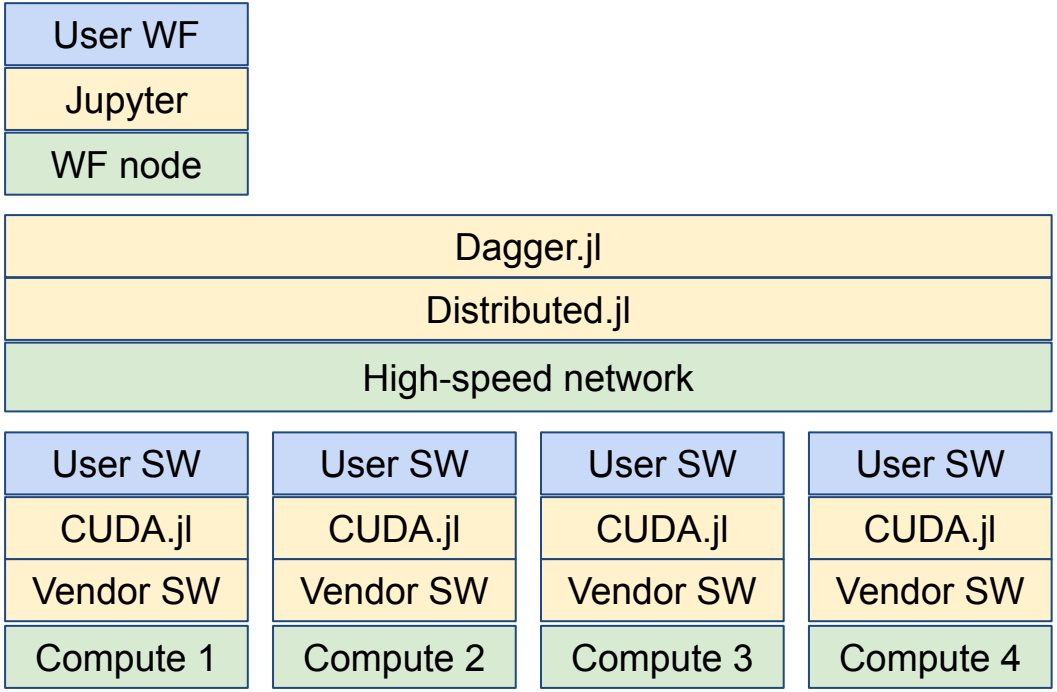
Building a Distributed Julia Application (without MPI)



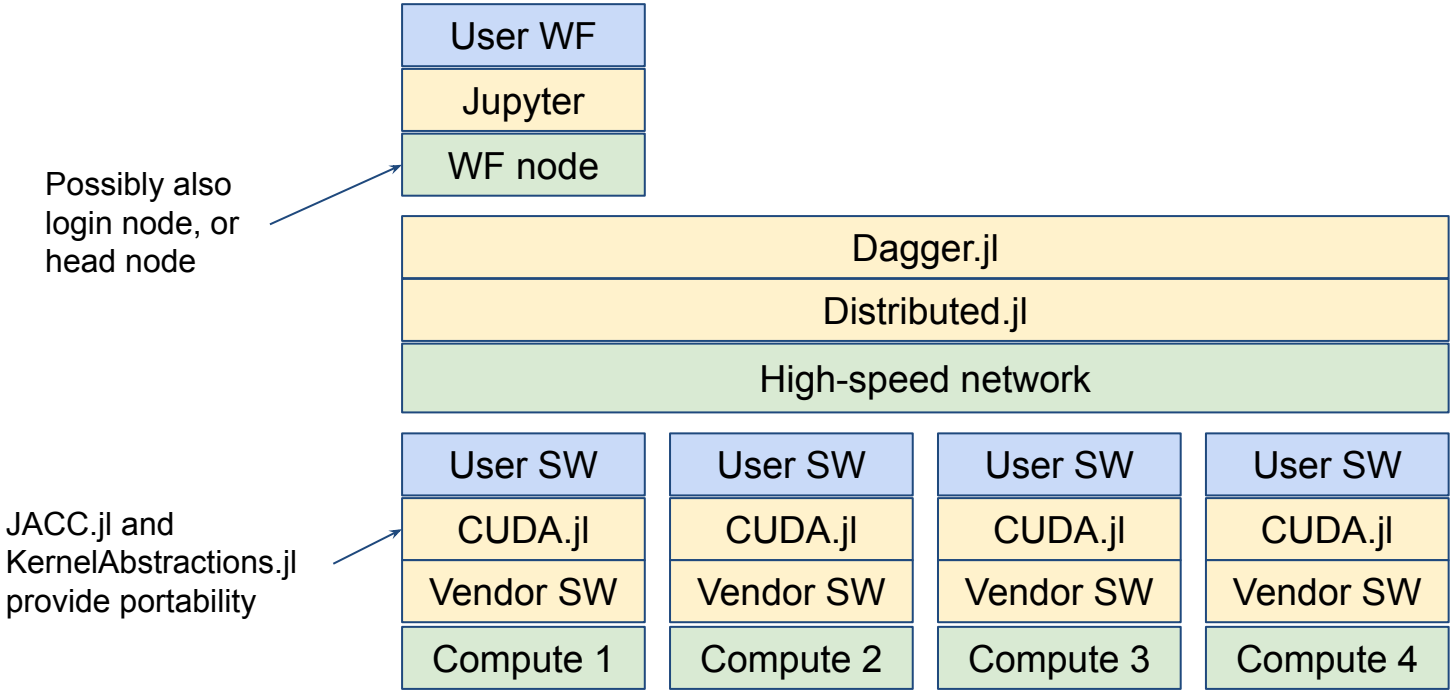
Building a Distributed Julia Application (without MPI)



Building a Distributed Julia Application (without MPI)



Building a Distributed Julia Application (without MPI)



Cluster Managers: Interaction with Slurm

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

[25]:
```
using Distributed, ClusterManagers
```

[26]:
```
addprocs(SlurmManager(2), N="2", constrain="cpu", qos="debug", time="00:5:00")
```

```
connecting to worker 1 out of 2
connecting to worker 2 out of 2
srun: job 18107782 queued and waiting for resources
srun: job 18107782 has been allocated resources
```

[26]:
```
2-element Vector{Int64}:
 2
 3
```

[27]:
```
hosts = []
pids = []
for i in workers()
    host, pid = fetch(@spawnat i (gethostname(), getpid()))
    push!(hosts, host)
    push!(pids, pid)
end
```

[28]:
```
hosts
```

[28]:
```
2-element Vector{Any}:
 "nid004709"
 "nid004953"
```

[29]:
```
pids
```

[29]:
```
2-element Vector{Any}:
 1605084
 1781811
```

[ ]:
```
```

NERSC

Cluster Managers: Interaction with Slurm

Request two cpu nodes

```
[26]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="debug", time="00:5:00")
```

```
connecting to worker 1 out of 2
connecting to worker 2 out of 2
srun: job 18107782 queued and waiting for resources
srun: job 18107782 has been allocated resources
```

`ElasticManager` can be used instead of `SlurmManager` to manually manage workers from *within* an allocation

```
[27]: hosts = []
pids = []
for i in workers()
    host, pid = fetch(@spawnat i (gethostname(), getpid()))
    push!(hosts, host)
    push!(pids, pid)
end
```

```
[28]: hosts
```

```
[28]: 2-element Vector{Any}:
 "nid004709"
 "nid004953"
```

```
[29]: pids
```

```
[29]: 2-element Vector{Any}:
 1605084
 1781811
```

```
[ ]:
```

Cluster Managers: Interaction with Slurm

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

[25]:
```
using Distributed, ClusterManagers
```

[26]:
```
addprocs(SlurmManager(2), N="2", constrain="cpu", qos="debug", time="00:5:00")
```

```
connecting to worker 1 out of 2
connecting to worker 2 out of 2
```

Get hostnames and pids from each worker

[27]:
```
hosts = []
pids = []
for i in workers()
    host, pid = fetch(@spawnat i (gethostname(), getpid()))
    push!(hosts, host)
    push!(pids, pid)
end
```

[28]:
```
2-element Vector{Any}:
 "nid004709"
 "nid004953"
```

[29]:
```
pids
```

[29]:
```
2-element Vector{Any}:
 1605084
 1781811
```

[ ]:
```
```

NERSC

Cluster Managers: Interaction with Slurm

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

[25]:
```
using Distributed, ClusterManagers
```

[26]:
```
addprocs(SlurmManager(2), N="2", constrain="cpu", qos="debug", time="00:5:00")
```

```
connecting to worker 1 out of 2
connecting to worker 2 out of 2
srun: job 18107782 queued and waiting for resources
srun: job 18107782 has been allocated resources
```

[26]:
```
2-element Vector{Int64}:
 2
 3
```

[28]:
```
hosts
```

[28]:
```
2-element Vector{Any}:
 "nid004709"
 "nid004953"
```

[29]:
```
pids
```

[29]:
```
2-element Vector{Any}:
 1605084
 1781811
```

`Distributed.jl` supports basic workflows

Basic task: count number of heads from 2x10^8 fair coin tosses.

- Serial implementation:
  `for` loop + increment counter
- Parallel (distributed) implementation:
  `@distributed for` loop + reduction (summation) on counter

- Most Julia data types are trivial to serialize and communicate over network (the users doesn't have to do anything "special" to enable this)

```julia
function n_heads()
    n = 0
    for i = 1:200000000
        n += Int(rand(Bool))
    end
    n
end

function n_heads_parallel()
    nheads = @distributed (+) for i = 1:200000000
        Int(rand(Bool))
    end
    nheads
end
```

Performance Gains from Distributing Work

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

[33]:
```
using BenchmarkTools
```

[31]:
```
@benchmark(n_heads())
```

[31]:
```
BenchmarkTools.Trial: 21 samples with 1 evaluation.
 Range (min … max):  245.963 ms … 246.808 ms  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     245.978 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   246.023 ms ± 181.318 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%

  246 ms           Histogram: frequency by time          247 ms <

 Memory estimate: 0 bytes, allocs estimate: 0.
```

[32]:
```
@benchmark(n_heads_parallel())
```

[32]:
```
BenchmarkTools.Trial: 51 samples with 1 evaluation.
 Range (min … max):  99.827 ms … 100.605 ms  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     99.855 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   99.871 ms ± 106.362 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%

  99.8 ms          Histogram: frequency by time         99.9 ms <

 Memory estimate: 6.62 KiB, allocs estimate: 155.
```

[ ]:
```
```

Performance Gains from Distributing Work

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

```
[31]: 1 @benchmark(n_heads())

[31]: BenchmarkTools.Trial: 21 samples with 1 evaluation.
 Range (min … max):  245.963 ms … 246.808 ms  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     245.978 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   246.023 ms ± 181.318 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%

 246 ms           Histogram: frequency by time          247 ms <

 Memory estimate: 0 bytes, allocs estimate: 0.
```

```
Time  (mean ± σ):   99.871 ms ± 106.362 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%

 99.8 ms          Histogram: frequency by time          99.9 ms <

 Memory estimate: 6.62 KiB, allocs estimate: 155.

[ ]: 1
```

Performance Gains from Distributing Work

Distributing work over 2 nodes results in a 2x performance increase

SC23.ipynb (auto-x) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.8.0-beta1 GPU

```
[33]: 1 using BenchmarkTools

[31]: 1 @benchmark(n_heads())

[31]: BenchmarkTools.Trial: 21 samples with 1 evaluation.
```

```
[32]: 1 @benchmark(n_heads_parallel())

[32]: BenchmarkTools.Trial: 51 samples with 1 evaluation.
 Range (min … max):  99.827 ms … 100.605 ms  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     99.855 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   99.871 ms ± 106.362 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%

 99.8 ms         Histogram: frequency by time         99.9 ms <

 Memory estimate: 6.62 KiB, allocs estimate: 155.
```

Tangent: Hybrid CPU/GPU Jobs

SC23.ipynb (11) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[2]:
```
using Distributed, ClusterManagers
```

[3]:
```
addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")
```

```
connecting to worker 1 out of 2
srun: job 18134163 queued and waiting for resources
srun: job 18134163 has been allocated resources
connecting to worker 2 out of 2
```

[3]:
```
2-element Vector{Int64}:
 2
 3
```

[4]:
```
addprocs(SlurmManager(2), N="2", constrain="gpu", qos="interactive", time="00:15:00", A="nstaff_g")
```

```
connecting to worker 1 out of 2
srun: job 18134168 queued and waiting for resources
srun: job 18134168 has been allocated resources
connecting to worker 2 out of 2
```

[4]:
```
2-element Vector{Int64}:
 4
 5
```

•••

[48]:
```
features = []
for i in workers()
    scontrol = fetch(@spawnat i get_features())
    push!(features, scontrol)
end
```

[49]:
```
features
```

[49]:
```
4-element Vector{Any}:
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"
```

[ ]:

Tangent: Hybrid CPU/GPU Jobs

SC23.ipynb (11) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

```
[3]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")

connecting to worker 1 out of 2
srun: job 18134163 queued and waiting for resources
srun: job 18134163 has been allocated resources
connecting to worker 2 out of 2

[3]: 2-element Vector{Int64}:
 2
 3
```

```
connecting to worker 1 out of 2
srun: job 18134168 queued and waiting for resources
srun: job 18134168 has been allocated resources
connecting to worker 2 out of 2

[4]: 2-element Vector{Int64}:
 4
 5

...

[48]: features = []
for i in workers()
    scontrol = fetch(@spawnat i get_features())
    push!(features, scontrol)
end

[49]: features

[49]: 4-element Vector{Any}:
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"

[ ]:
```

NERSC

Tangent: Hybrid CPU/GPU Jobs

SC23.ipynb (11) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[2]: using Distributed, ClusterManagers

[3]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")

connecting to worker 1 out of 2
srun: job 18134163 queued and waiting for resources
```

```
[4]: addprocs(SlurmManager(2), N="2", constrain="gpu", qos="interactive", time="00:15:00", A="nstaff_g")

connecting to worker 1 out of 2
srun: job 18134168 queued and waiting for resources
srun: job 18134168 has been allocated resources
connecting to worker 2 out of 2

[4]: 2-element Vector{Int64}:
 4
 5
```

```
[48]: features = []
      for i in workers()
          scontrol = fetch(@spawnat i get_features())
          push!(features, scontrol)
      end

[49]: features

[49]: 4-element Vector{Any}:
       "ActiveFeatures=cpu,milan,ss11"
       "ActiveFeatures=cpu,milan,ss11"
       "ActiveFeatures=gpu,ss11,a100,hbm40g"
       "ActiveFeatures=gpu,ss11,a100,hbm40g"

[ ]:
```

Tangent: Hybrid CPU/GPU Jobs

SC23.ipynb (11) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[2]: using Distributed, ClusterManagers

[3]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")

connecting to worker 1 out of 2
srun: job 18134163 queued and waiting for resources
srun: job 18134163 has been allocated resources
connecting to worker 2 out of 2

[3]: 2-element Vector{Int64}:
 2
 3

[4]: addprocs(SlurmManager(2), N="2", constrain="gpu", qos="interactive", time="00:15:00", A="nstaff_g")

connecting to worker 1 out of 2
srun: job 18134168 queued and waiting for resources
srun: job 18134168 has been allocated resources
connecting to worker 2 out of 2
```

```
[48]: features = []
for i in workers()
    scontrol = fetch(@spawnat i get_features())
    push!(features, scontrol)
end
```

```
[49]: 4-element Vector{Any}:
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"

[ ]:
```

NERSC

Tangent: Hybrid CPU/GPU Jobs

SC23.ipynb (11) - JupyterLab

SC23.ipynb

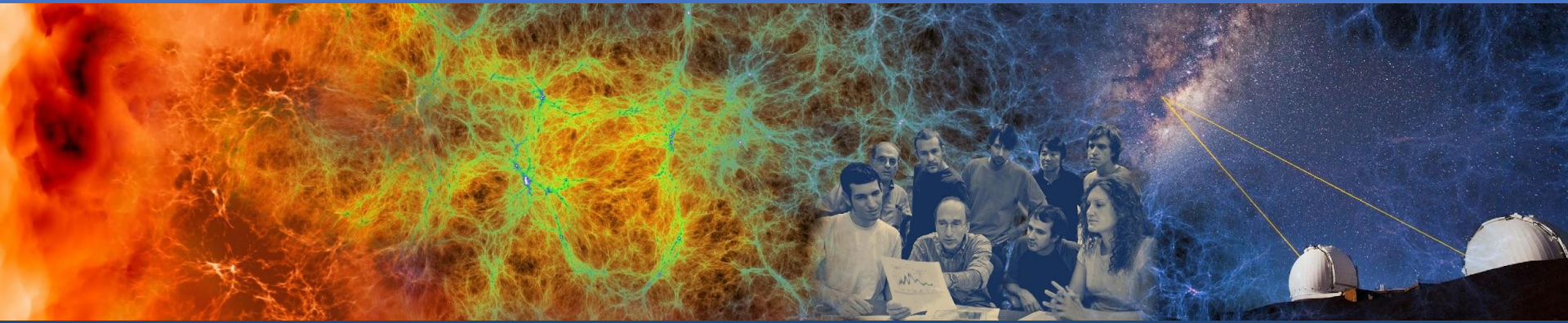
File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[2]: using Distributed, ClusterManagers

[3]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")

connecting to worker 1 out of 2
srun: job 18134163 queued and waiting for resources
srun: job 18134163 has been allocated resources
connecting to worker 2 out of 2

[3]: 2-element Vector{Int64}:
 2
 3

[4]: addprocs(SlurmManager(2), N="2", constrain="gpu", qos="interactive", time="00:15:00", A="nstaff_g")

connecting to worker 1 out of 2
srun: job 18134168 queued and waiting for resources
srun: job 18134168 has been allocated resources
connecting to worker 2 out of 2

[4]: 2-element Vector{Int64}:
 4
 5
```

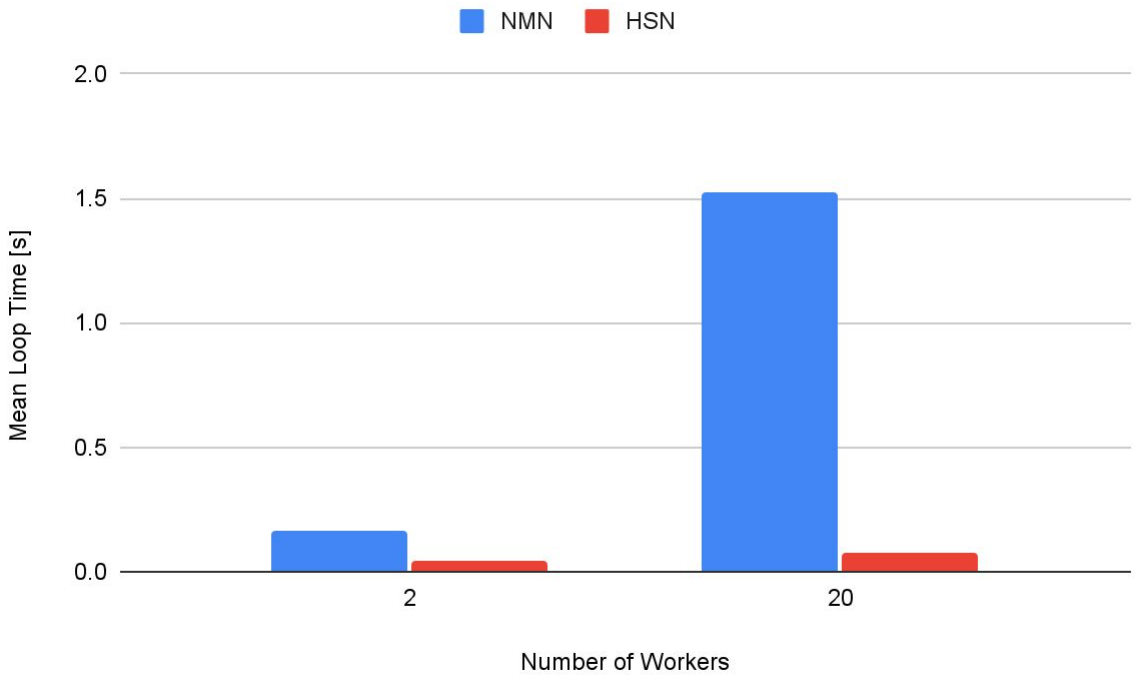
```
[49]: features

[49]: 4-element Vector{Any}:
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=cpu,milan,ss11"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"
 "ActiveFeatures=gpu,ss11,a100,hbm40g"
```

Tangent: Network Discovery



Workflow Support Story

- Unexpected poor performance and scaling
- User application 100x slower on Perlmutter

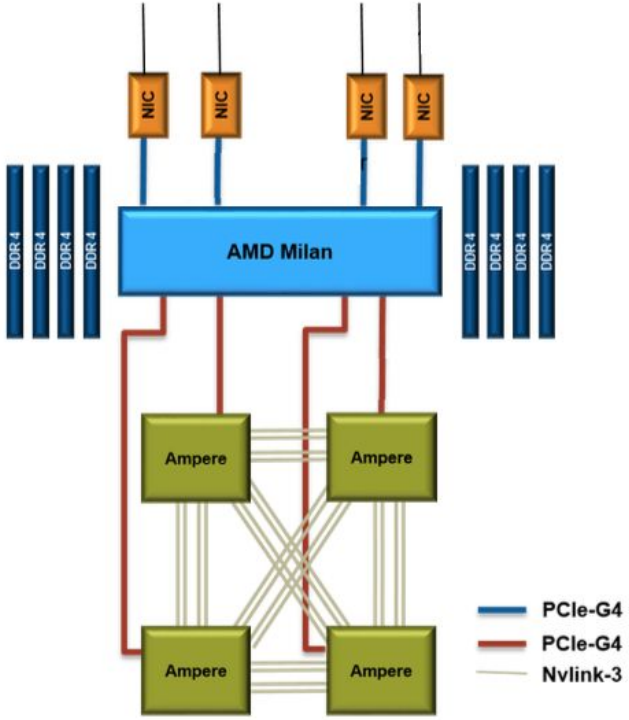


Perlmutter is a Heterogeneous System

Partition	Nodes	CPU	RAM	GPU	NIC
GPU	1536	1x AMD EPYC 7763	256GB	4x NVIDIA A100 (40GB)	4x HPE Slingshot 11
	256	1x AMD EPYC 7763	256GB	4x NVIDIA A100 (80GB)	4x HPE Slingshot 11
CPU	3072	2x AMD EPYC 7763	512GB	–	1x HPE Slingshot 11
Login	40	1x AMD EPYC 7713	512GB	4x NVIDIA A100 (40GB)	–
Large Memory	4	1x AMD EPYC 7713	1TB	4x NVIDIA A100 (40GB)	1x HPE Slingshot 11

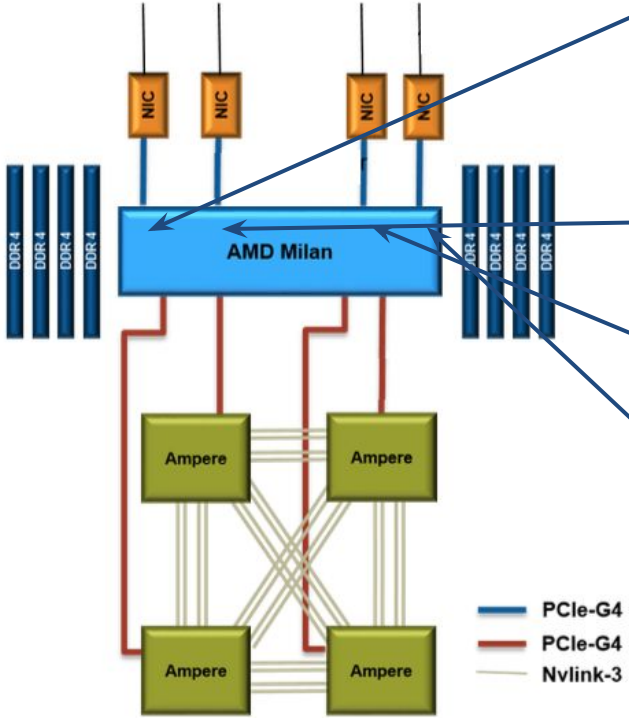
- Each GPU node has 4 NICs
  - 1 NIC and 1 GPU per host bridge
- Each CPU node has 1 NIC

Eg. GPU Node Topology




```
Hwloc.Object: Machine
└─ Hwloc.Object: Package [L#0 P#0]
   ├─ Hwloc.Object: Group
   │  ├─ Hwloc.Object: NUMANode
   │  └─ Hwloc.Object: Bridge [HostBridge]
   │     ├─ Hwloc.Object: Bridge [PCIBridge]
   │     │  └─ Hwloc.Object: PCI_Device [c2:00.0 (Ethernet)]
   │     │     └─ Hwloc.Object: OS_Device [Net "hsn0"]
   │     ├─ Hwloc.Object: Bridge [PCIBridge]
   │     │  └─ Hwloc.Object: PCI_Device [c3:00.0 (Ethernet)]
   │     │     └─ Hwloc.Object: OS_Device [Net "nmn0"]
…
   ├─ Hwloc.Object: Group
   │  ├─ Hwloc.Object: NUMANode
   │  └─ Hwloc.Object: Bridge [HostBridge]
   │     ├─ Hwloc.Object: Bridge [PCIBridge]
   │     │  └─ Hwloc.Object: PCI_Device [81:00.0 (Ethernet)]
   │     │     └─ Hwloc.Object: OS_Device [Net "hsn1"]
…
   ├─ Hwloc.Object: Group
   │  ├─ Hwloc.Object: NUMANode
   │  └─ Hwloc.Object: Bridge [HostBridge]
   │     ├─ Hwloc.Object: Bridge [PCIBridge]
   │     │  └─ Hwloc.Object: PCI_Device [42:00.0 (Ethernet)]
   │     │     └─ Hwloc.Object: OS_Device [Net "hsn2"]
…
   └─ Hwloc.Object: Group
      ├─ Hwloc.Object: NUMANode
      └─ Hwloc.Object: Bridge [HostBridge]
         ├─ Hwloc.Object: Bridge [PCIBridge]
         │  └─ Hwloc.Object: PCI_Device [01:00.0 (Ethernet)]
         │     └─ Hwloc.Object: OS_Device [Net "hsn3"]
…
```

Eg. GPU Node Topology




```
Hwloc.Object: Machine
└ Hwloc.Object: Package [L#0 P#0]
  ├ Hwloc.Object: Group
  │ ├ Hwloc.Object: NUMANode
  │ └ Hwloc.Object: Bridge [HostBridge]
  │   ├ Hwloc.Object: Bridge [PCIBridge]
  │   │ └ Hwloc.Object: PCI_Device [c2:00.0 (Ethernet)]
  │   │   └ Hwloc.Object: OS_Device [Net "hsn0"]
  │   ├ Hwloc.Object: Bridge [PCIBridge]
  │   │ └ Hwloc.Object: PCI_Device [c3:00.0 (Ethernet)]
  │   │   └ Hwloc.Object: OS_Device [Net "nmn0"]
...
  ├ Hwloc.Object: Group
  │ ├ Hwloc.Object: NUMANode
  │ └ Hwloc.Object: Bridge [HostBridge]
  │   ├ Hwloc.Object: Bridge [PCIBridge]
  │   │ └ Hwloc.Object: PCI_Device [81:00.0 (Ethernet)]
  │   │   └ Hwloc.Object: OS_Device [Net "hsn1"]
...
  ├ Hwloc.Object: Group
  │ ├ Hwloc.Object: NUMANode
  │ └ Hwloc.Object: Bridge [HostBridge]
  │   ├ Hwloc.Object: Bridge [PCIBridge]
  │   │ └ Hwloc.Object: PCI_Device [42:00.0 (Ethernet)]
  │   │   └ Hwloc.Object: OS_Device [Net "hsn2"]
...
  └ Hwloc.Object: Group
    ├ Hwloc.Object: NUMANode
    └ Hwloc.Object: Bridge [HostBridge]
      ├ Hwloc.Object: Bridge [PCIBridge]
      │ └ Hwloc.Object: PCI_Device [01:00.0 (Ethernet)]
      │   └ Hwloc.Object: OS_Device [Net "hsn3"]
...
```

Topo distance to NIC

- Finding the right NIC is easy now: pick the (non-nmn) interface with lowest tree distance between your core and the PCI device

```
for net in collect(network_devs)
    found, dist = distance_to_core(hwloc_tree, net, cpu_id)
    print("$(dist): ")
    print_tree(net)
end
```

```
213: Hwloc.Object: PCI_Device [c2:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn0"]
213: Hwloc.Object: PCI_Device [c3:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "nmn0"]
47: Hwloc.Object: PCI_Device [81:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn1"]
379: Hwloc.Object: PCI_Device [42:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn2"]
379: Hwloc.Object: PCI_Device [01:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn3"]
```

Topo distance to NIC

- Finding the right NIC is easy now: pick the (non-nmn) interface with lowest tree distance between your core and the PCI device

```
for net in collect(network_devs)
    found, dist = distance_to_core(hwloc_tree, net, cpu_id)
    print("$(dist): ")
    print_tree(net)
end
```

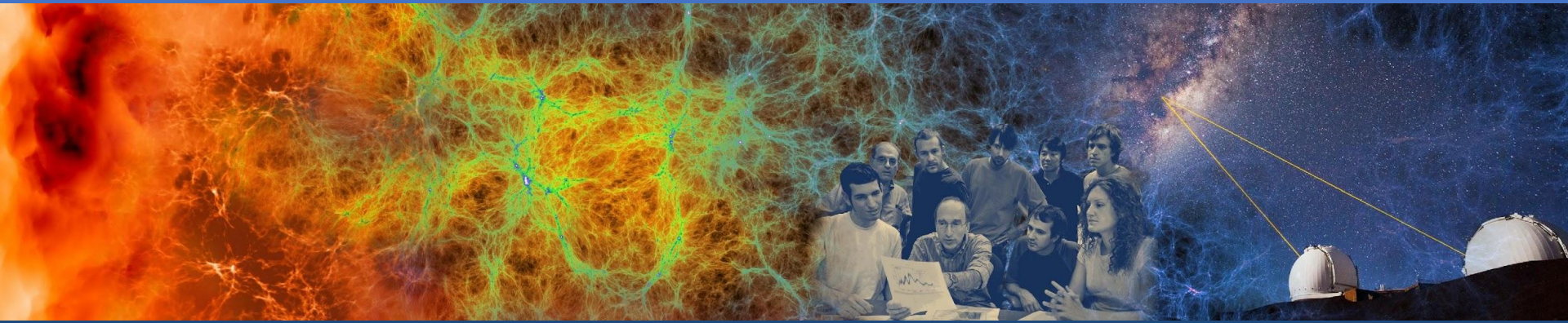
```
213: Hwloc.Object: PCI_Device [c2:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn0"]
213: Hwloc.Object: PCI_Device [c3:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "nmn0"]
47: Hwloc.Object: PCI_Device [81:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn1"]
379: Hwloc.Object: PCI_Device [42:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn2"]
379: Hwloc.Object: PCI_Device [01:00.0 (Ethernet)]
└─ Hwloc.Object: OS_Device [Net "hsn3"]
```

This one! →

Future

- This is pre-alpha, so far only deployed at NERSC
  - Looking for folks to test this at NERSC and on their favorite HPC systems
- `Distributed.jl` to use distance between NIC and Core (on Hwloc tree) to select preferred tree NIC
  - [JuliaParallel/NetworkInterfaceControllers.jl]

Programming GPUs



CUDA.jl: Interfacing with Nvidia GPUs

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[9]:
```
arr1 = rand(1_000, 1_000);
arr2 = rand(1_000, 1_000);
```

[10]:
```
@benchmark arr1 * arr2
```

[10]:
```
BenchmarkTools.Trial: 1493 samples with 1 evaluation.
 Range (min … max):  2.626 ms …   7.337 ms  ┊ GC (min … max): 0.00% … 2.79%
 Time  (median):     3.152 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   3.347 ms ± 490.147 µs  ┊ GC (mean ± σ):  0.90% ± 1.83%
```

```
 2.63 ms         Histogram: frequency by time        4.16 ms <

 Memory estimate: 7.63 MiB, allocs estimate: 2.
```

[11]:
```
using CUDA
```

[12]:
```
carr1 = cu(arr1);
carr2 = cu(arr2);
```

[13]:
```
@benchmark carr1 * carr2
```

[13]:
```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):   22.533 µs … 390.183 µs  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     121.205 µs               ┊ GC (median):    0.00%
 Time  (mean ± σ):   114.948 µs ±  24.119 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%
```

```
 22.5 µs         Histogram: log(frequency) by time        123 µs <

 Memory estimate: 1.09 KiB, allocs estimate: 44.
```

CUDA.jl: Interfacing with Nvidia GPUs

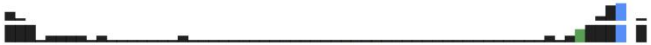
Basic example:
1000x1000 matmul
using OpenBLAS

SC23.ipynb (auto-L) - JupyterLab

```
[9]: arr1 = rand(1_000, 1_000);
     arr2 = rand(1_000, 1_000);

[10]: @benchmark arr1 * arr2

10]: BenchmarkTools.Trial: 1493 samples with 1 evaluation.
      Range (min … max):  2.626 ms …   7.337 ms  ┊ GC (min … max): 0.00% … 2.79%
      Time  (median):     3.152 ms               ┊ GC (median):    0.00%
      Time  (mean ± σ):   3.347 ms ± 490.147 µs  ┊ GC (mean ± σ):  0.90% ± 1.83%
```

```
      2.63 ms         Histogram: frequency by time        4.16 ms <

      Memory estimate: 7.63 MiB, allocs estimate: 2.
```

```
[12]: carr1 = cu(arr1);
      carr2 = cu(arr2);

[13]: @benchmark carr1 * carr2

[13]: BenchmarkTools.Trial: 10000 samples with 1 evaluation.
       Range (min … max):   22.533 µs … 390.183 µs  ┊ GC (min … max): 0.00% … 0.00%
       Time  (median):     121.205 µs               ┊ GC (median):    0.00%
       Time  (mean ± σ):   114.948 µs ±  24.119 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%
```



```
       22.5 µs        Histogram: log(frequency) by time        123 µs <

       Memory estimate: 1.09 KiB, allocs estimate: 44.
```

CUDA.jl: Interfacing with Nvidia GPUs

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[9]: arr1 = rand(1_000, 1_000);
     arr2 = rand(1_000, 1_000);

[10]: @benchmark arr1 * arr2

[10]: BenchmarkTools.Trial: 1493 samples with 1 evaluation.
       Range (min … max):  2.626 ms …   7.337 ms  ┊ GC (min … max): 0.00% … 2.79%
       Time  (median):     3.152 ms               ┊ GC (median):    0.00%
       Time  (mean ± σ):   3.347 ms ± 490.147 µs  ┊ GC (mean ± σ):  0.90% ± 1.83%

       2.63 ms         Histogram: frequency by time        4.16 ms <
```

```
[11]: using CUDA

[12]: carr1 = cu(arr1);
      carr2 = cu(arr2);
```

```
[13]: BenchmarkTools.Trial: 10000 samples with 1 evaluation.
       Range (min … max):  22.533 µs … 390.183 µs  ┊ GC (min … max): 0.00% … 0.00%
       Time  (median):    121.205 µs               ┊ GC (median):    0.00%
       Time  (mean ± σ):  114.948 µs ±  24.119 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%

       22.5 µs         Histogram: log(frequency) by time        123 µs <

       Memory estimate: 1.09 KiB, allocs estimate: 44.
```

Copy arrays to device

CUDA.jl: Interfacing with Nvidia GPUs

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[9]: arr1 = rand(1_000, 1_000);
     arr2 = rand(1_000, 1_000);

[10]: @benchmark arr1 * arr2

[10]: BenchmarkTools.Trial: 1493 samples with 1 evaluation.
       Range (min … max):  2.626 ms …   7.337 ms  ┊ GC (min … max): 0.00% … 2.79%
       Time  (median):     3.152 ms               ┊ GC (median):    0.00%
       Time  (mean ± σ):   3.347 ms ± 490.147 µs  ┊ GC (mean ± σ):  0.90% ± 1.83%

       2.63 ms         Histogram: frequency by time        4.16 ms <

       Memory estimate: 7.63 MiB, allocs estimate: 2.
```

```
[13]: @benchmark carr1 * carr2

[13]: BenchmarkTools.Trial: 10000 samples with 1 evaluation.
       Range (min … max):   22.533 µs … 390.183 µs  ┊ GC (min … max): 0.00% … 0.00%
       Time  (median):     121.205 µs               ┊ GC (median):    0.00%
       Time  (mean ± σ):   114.948 µs ±  24.119 µs  ┊ GC (mean ± σ):  0.00% ± 0.00%

       22.5 µs         Histogram: log(frequency) by time        123 µs <

       Memory estimate: 1.09 KiB, allocs estimate: 44.
```

Using cuBLAS to perform matmul decreases run time from 3.15ms to 121µs (26x)

Write Your Own CUDA Kernels in Julia

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[19]:
```julia
function my_kernel(carr_out, carr)
    start = (blockIdx().x - 1) * blockDim().x + threadIdx().x
    stride = blockDim().x * gridDim().x
    len = length(carr)
    for i = start:stride:len  # "grid-stride" loop
        carr_out[i] = sin(carr[i]) + 1
    end
    return
end
```

[19]:
```
my_kernel (generic function with 1 method)
```

[22]:
```julia
carr = cu(rand(10_000_000))
carr_out = similar(carr);

@cuda threads=256 my_kernel(carr_out, carr)
```

[22]:
```
CUDA.HostKernel for my_kernel(CuDeviceVector{Float32, 1}, CuDeviceVector{Float32, 1})
```

[23]:
```julia
carr_out
```

[23]:
```
10000000-element CuArray{Float32, 1, CUDA.Mem.DeviceBuffer}:
 1.1730438
 1.8287812
 1.5149846
 1.3774228
 1.606185
 1.431107
 1.4560564
 1.1032873
```

Write Your Own CUDA Kernels in Julia

Define kernels using Julia functions

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

[19]:
```julia
function my_kernel(carr_out, carr)
    start = (blockIdx().x - 1) * blockDim().x + threadIdx().x
    stride = blockDim().x * gridDim().x
    len = length(carr)
    for i = start:stride:len  # "grid-stride" loop
        carr_out[i] = sin(carr[i]) + 1
    end
    return
end
```

[22]:
```julia
carr = cu(rand(10_000_000))
carr_out = similar(carr);

@cuda threads=256 my_kernel(carr_out, carr)
```

[22]:
```
CUDA.HostKernel for my_kernel(CuDeviceVector{Float32, 1}, CuDeviceVector{Float32, 1})
```

[23]:
```julia
carr_out
```

[23]:
```
10000000-element CuArray{Float32, 1, CUDA.Mem.DeviceBuffer}:
 1.1730438
 1.8287812
 1.5149846
 1.3774228
 1.606185
 1.431107
 1.4560564
 1.1032873
```

Write Your Own CUDA Kernels in Julia

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[19]:
```julia
function my_kernel(carr_out, carr)
    start = (blockIdx().x - 1) * blockDim().x + threadIdx().x
    stride = blockDim().x * gridDim().x
    len = length(carr)
    for i = start:stride:len  # "grid-stride" loop
        carr_out[i] = sin(carr[i]) + 1
    end
    return
end
```

[19]:
```
my_kernel (generic function with 1 method)
```

Launch kernel using the `@cuda` macro

[22]:
```julia
carr = cu(rand(10_000_000))
carr_out = similar(carr);

@cuda threads=256 my_kernel(carr_out, carr)
```

[22]:
```
CUDA.HostKernel for my_kernel(CuDeviceVector{Float32, 1}, CuDeviceVector{Float32, 1})
```

[23]:
```
10000000-element CuArray{Float32, 1, CUDA.Mem.DeviceBuffer}:
 1.1730438
 1.8287812
 1.5149846
 1.3774228
 1.606185
 1.431107
 1.4560564
 1.1032873
```

CUDA.jl provides detailed profiling interface

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[14]:
```
CUDA.@profile carr1 * carr2
```

[14]:
```
Profiler ran for 2.23 ms, capturing 23 events.

Host-side activity: calling CUDA APIs took 1.96 ms (87.81% of the trace)
```

Time (%)	Time	Calls	Avg time	Min time	Max time	Name
29.29%	653.51 µs	1	653.51 µs	653.51 µs	653.51 µs	**cuMemAllo**
18.15%	404.83 µs	1	404.83 µs	404.83 µs	404.83 µs	**cudaEvent**
1.12%	25.03 µs	1	25.03 µs	25.03 µs	25.03 µs	cudaLaunc
0.82%	18.36 µs	1	18.36 µs	18.36 µs	18.36 µs	cudaMemse
0.44%	9.78 µs	2	4.89 µs	953.67 ns	8.82 µs	cudaOccup
0.28%	6.2 µs	3	2.07 µs	476.84 ns	4.77 µs	cudaStrea
0.26%	5.72 µs	1	5.72 µs	5.72 µs	5.72 µs	cudaEvent
0.00%	0.0 ns	1	0.0 ns	0.0 ns	0.0 ns	cudaGetLa

1 column omitted

```
Device-side activity: GPU was busy for 160.22 µs (7.18% of the trace)
```

Time (%)	Time	Calls	Avg time	Min time	Max time	Name
7.09%	158.07 µs	1	158.07 µs	158.07 µs	158.07 µs	**ampere_sg**
0.10%	2.15 µs	1	2.15 µs	2.15 µs	2.15 µs	[set devi

1 column omitted

(advanced) LLVM + Julia

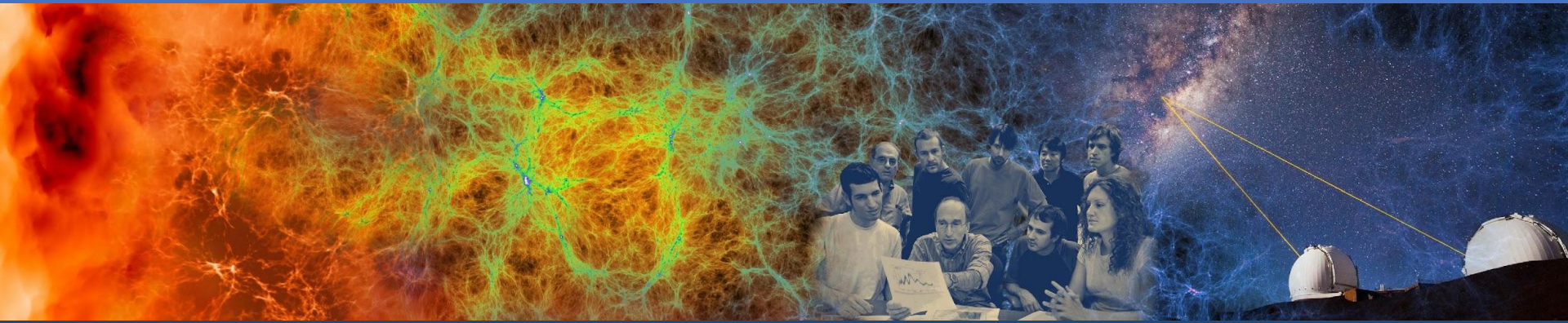
Julia provides interfaces to the LLVM backend.

Eg.:

- `loopinfo`
- `llvmcall`

```
[16]: macro unroll(expr)
          expr = loopinfo("@unroll", expr, (Symbol("llvm.loop.unroll.full"),))
          return esc(expr)
      end

      for (jlf, f) in zip((:+, :*, :-), (:add, :mul, :sub))
          for (T, llvmT) in ((:Float32, "float"), (:Float64, "double"))
              ir = """
                  %x = f$f contract nsz $llvmT %0, %1
                  ret $llvmT %x
              """
              @eval begin
                  # the @pure is necessary so that we can constant propagate.
                  @inline Base.@pure function $jlf(a::$T, b::$T)
                      Base.llvmcall($ir, $T, Tuple{$T, $T}, a, b)
                  end
              end
          end
          @eval function $jlf(args...)
              Base.$jlf(args...)
          end
      end
```

Inspiration: Particles in Potentials



Transition Rates between Potential Minima

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[1]:
```
using Plots
```

•••

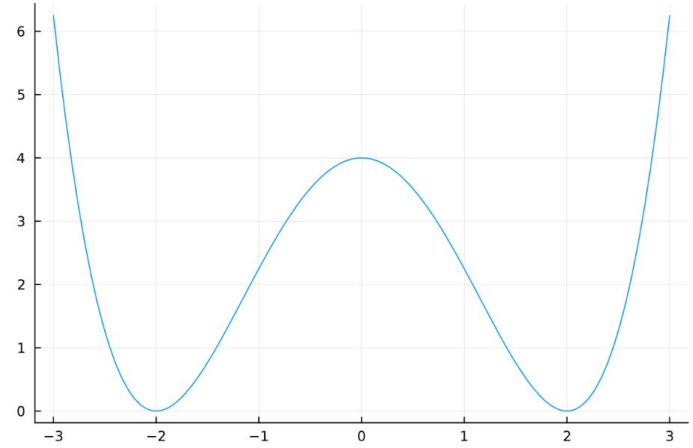
[4]:
```
energy(x; a=2, k=1) = @. 0.25*k*((x-a)^2) * ((x+a)^2)
force(x; a=2, k=1) = @. -k*x*(x-a)*(x+a)
```

[4]:
```
force (generic function with 1 method)
```

•••

[9]:
```
x = range(-3, 3, 100)
e = energy(x)
plot(x, e, legend=false)
```

[9]:



Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.64 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Transition Rates between Potential Minima

Julia ML Exp... (auto-p : 4) - JupyterLab

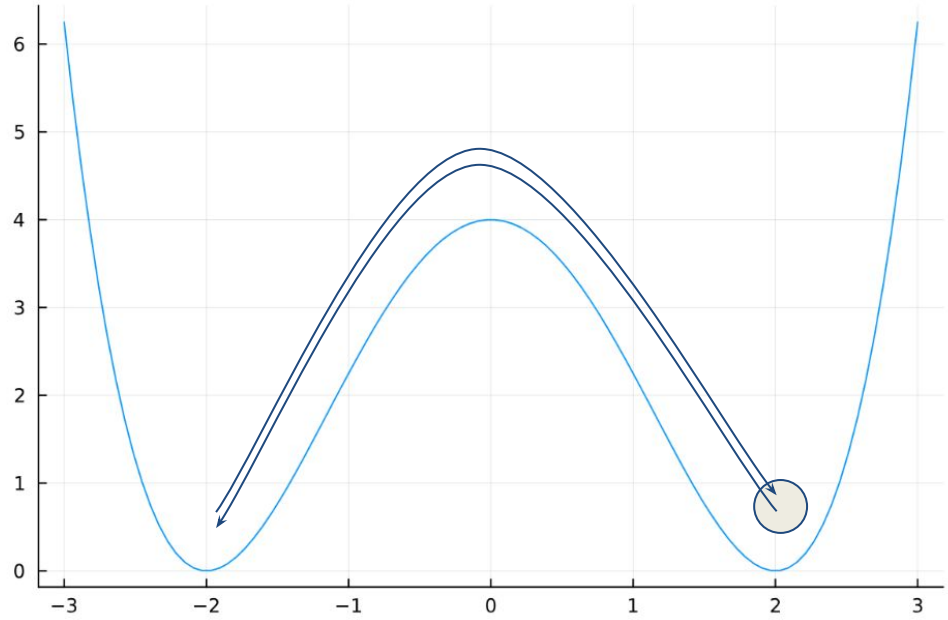
Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[9]:
```
x = range(-3, 3, 100)
e = energy(x)
plot(x, e, legend=false)
```

[9]:



This is a highly-simplified (1D) model for loads of interesting science: chemical reactions; protein conformations; etc

Integrating SDEs

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[10]:
```julia
using Random, Distributions
```

[11]:
```julia
Random.seed!(123)
```

[11]:
```
TaskLocalRNG()
```

[12]:
```julia
#this is step A
position_update(x, v, dt) = @. x + v*dt/2.

#this is step B
velocity_update(v,F,dt) = @. v + F*dt/2.

function random_velocity_update(
        v::AbstractVector, gamma::Number, kBT::Number, dt::Number;
        d=Normal(0, 1)
    )
    R = rand(d, length(v))
    c1 = @. exp(-gamma*dt)
    c2 = @. sqrt(1-c1*c1)*sqrt(kBT)
    v_new = @. c1*v + R*c2
    return v_new
end

random_velocity_update(
    v::Number, gamma::Number, kBT::Number, dt::Number; d=Normal(0, 1)
) = only(random_velocity_update([v], gamma, kBT, dt; d))
```

[12]:
```
random_velocity_update (generic function with 2 methods)
```

[13]:
```julia
function baoab(
        energy::Function, force::Function, max_time, dt,
        gamma, kBT, initial_position::T, initial_velocity::T;
        save_frequency = 3, kwargs...
    ) where T <: Number

    x = initial_position
    v = initial_velocity
    t = 0
```

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.66 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Integrating SDEs

Julia makes RNG easy!

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

```
[10]: using Random, Distributions

[11]: Random.seed!(123)

[11]: TaskLocalRNG()
```

```
#this is step B
velocity_update(v,F,dt) = @. v + F*dt/2.

function random_velocity_update(
        v::AbstractVector, gamma::Number, kBT::Number, dt::Number;
        d=Normal(0, 1)
    )
    R = rand(d, length(v))
    c1 = @. exp(-gamma*dt)
    c2 = @. sqrt(1-c1*c1)*sqrt(kBT)
    v_new = @. c1*v + R*c2
    return v_new
end

random_velocity_update(
    v::Number, gamma::Number, kBT::Number, dt::Number; d=Normal(0, 1)
) = only(random_velocity_update([v], gamma, kBT, dt; d))
```

```
[12]: random_velocity_update (generic function with 2 methods)
```

```
[13]: function baoab(
        energy::Function, force::Function, max_time, dt,
        gamma, kBT, initial_position::T, initial_velocity::T;
        save_frequency = 3, kwargs...
    ) where T <: Number

    x = initial_position
    v = initial_velocity
    t = 0
```

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.66 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Integrating SDEs

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

```
[10]: using Random, Distributions

[11]: Random.seed!(123)

[11]: TaskLocalRNG()

[12]: #this is step A
```

```
[12]: #this is step A
position_update(x, v, dt) = @. x + v*dt/2.

#this is step B
velocity_update(v,F,dt) = @. v + F*dt/2.

function random_velocity_update(
        v::AbstractVector, gamma::Number, kBT::Number, dt::Number;
        d=Normal(0, 1)
    )
    R = rand(d, length(v))
    c1 = @. exp(-gamma*dt)
    c2 = @. sqrt(1-c1*c1)*sqrt(kBT)
    v_new = @. c1*v + R*c2
    return v_new
end

random_velocity_update(
    v::Number, gamma::Number, kBT::Number, dt::Number; d=Normal(0, 1)
) = only(random_velocity_update([v], gamma, kBT, dt; d))
```

Define SDE algorithm

Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.66 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Integrating SDEs

SDE algorithms can be long and complex!

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[13]:
```julia
function baoab(
        energy::Function, force::Function, max_time, dt,
        gamma, kBT, initial_position::T, initial_velocity::T;
        save_frequency = 3, kwargs...
    ) where T <: Number

    x = initial_position
    v = initial_velocity
    t = 0
    step_number = 0
    positions = T[]
    velocities = T[]
    total_energies = T[]
    save_times = T[]

    while t < max_time
        # B
        potential_force = force(x; kwargs...)
        v = velocity_update(v, potential_force, dt)
        # A
        x = position_update(x, v, dt)
        # O
        v = random_velocity_update(v, gamma, kBT, dt)
        # A
        x = position_update(x, v, dt)
        # B
        potential_force = force(x; kwargs...)
        v = velocity_update(v, potential_force, dt)

        potential_energy = energy(x; kwargs...)
        if step_number%save_frequency == 0 && step_number>0
            e_total = @. .5*v*v + potential_energy

            push!(positions, x)
            push!(velocities, v)
            push!(total_energies, e_total)
            push!(save_times, t)
        end
```

Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.66 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

Transitions are Rare!

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

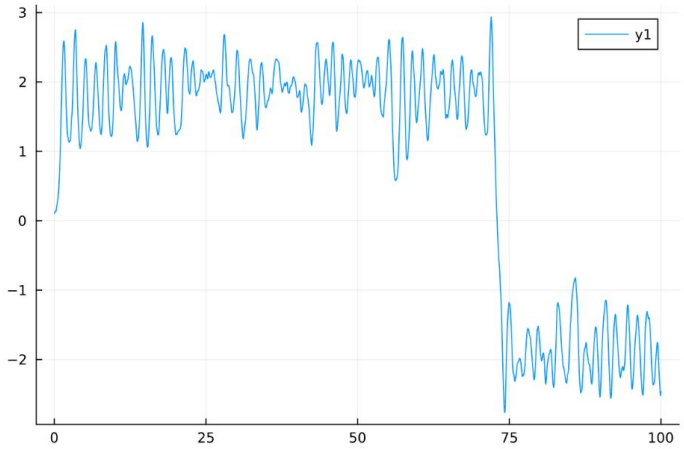
File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

```
initial_position = .1
initial_velocity = .5

my_gamma=0.5
my_kBT=3.0
my_dt=0.01

times, positions, velocities, total_energies = baoab(
    energy, force, my_max_time, my_dt,
    my_gamma, my_kBT,
    initial_position, initial_velocity;
    k=my_k
);
```

[11]:
```
plot(times, positions)
```

[11]:



Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.63 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

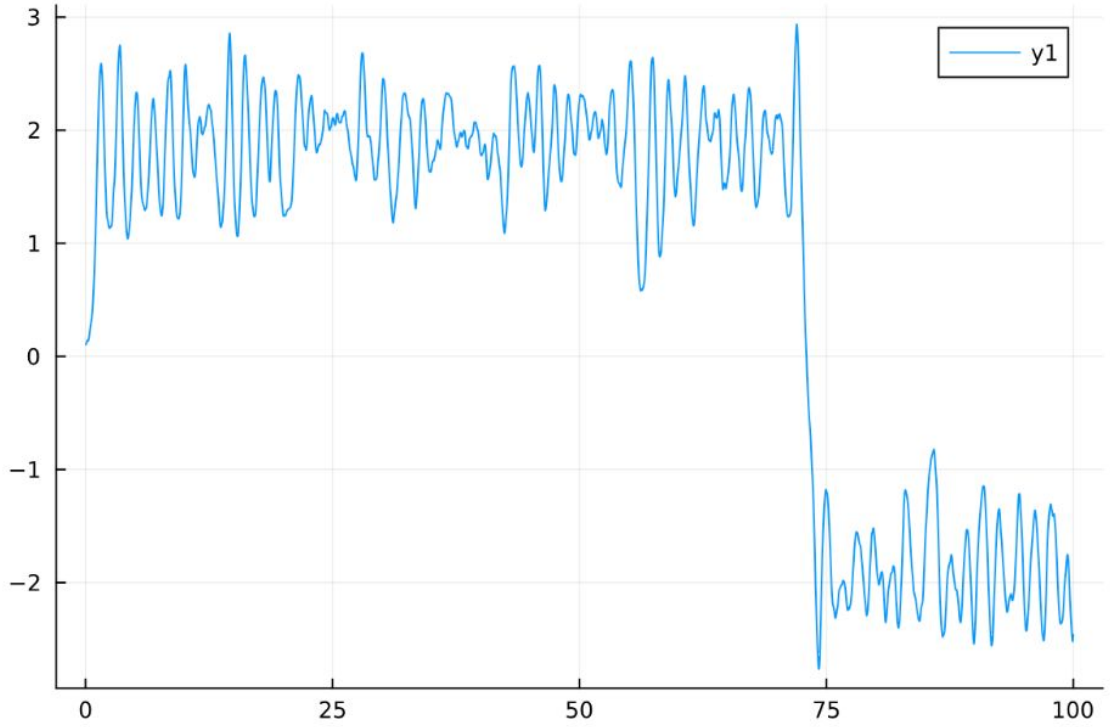
Transitions are Rare!

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

[11]:
```
plot(times, positions)
```

[11]:



Often you will 100s of millions of data points in order to collect a few thousand transitions

Solution Strategy: Local Monte-Carlo Sampling

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[13]:
```julia
function baoab_ensemble(
        energy::Function, force::Function, max_time, dt,
        gamma, kBT, initial_position::T, initial_velocity::T;
        save_frequency = 3, ensemble = (len=10, d=Normal(0, 0.01), n=10, st=10),
        kwargs...
    ) where T <: Number

    times, positions, velocities, total_energies = baoab(
        energy, force, max_time, dt,
        gamma, kBT, initial_position, initial_velocity;
        save_frequency=save_frequency, kwargs...
    )

    ensemble_t = Array{T, 1}[]
    ensemble_p = Array{T, 1}[]
    ensemble_v = Array{T, 1}[]
    for (t, p, v) in zip(
            times[1:ensemble.st:end],
            positions[1:ensemble.st:end],
            velocities[1:ensemble.st:end]
        )
        p_r = rand(ensemble.d, ensemble.n)
        v_r = rand(ensemble.d, ensemble.n)

        for i in 1:ensemble.n
            t_i, p_i, v_i, _ = baoab(
                energy, force, ensemble.len*dt, dt,
                gamma, kBT, p + p_r[i], v + v_r[i];
                save_frequency=save_frequency, kwargs...
            )
            push!(ensemble_t, t .+ t_i)
            push!(ensemble_p, p_i)
            push!(ensemble_v, v_i)

        end
    end

    return times, positions, velocities, total_energies, (t=ensemble_t, p=ensemble_p, v=ensemble_v)
end
```

Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.64 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

Solution Strategy: Local Monte-Carlo Sampling

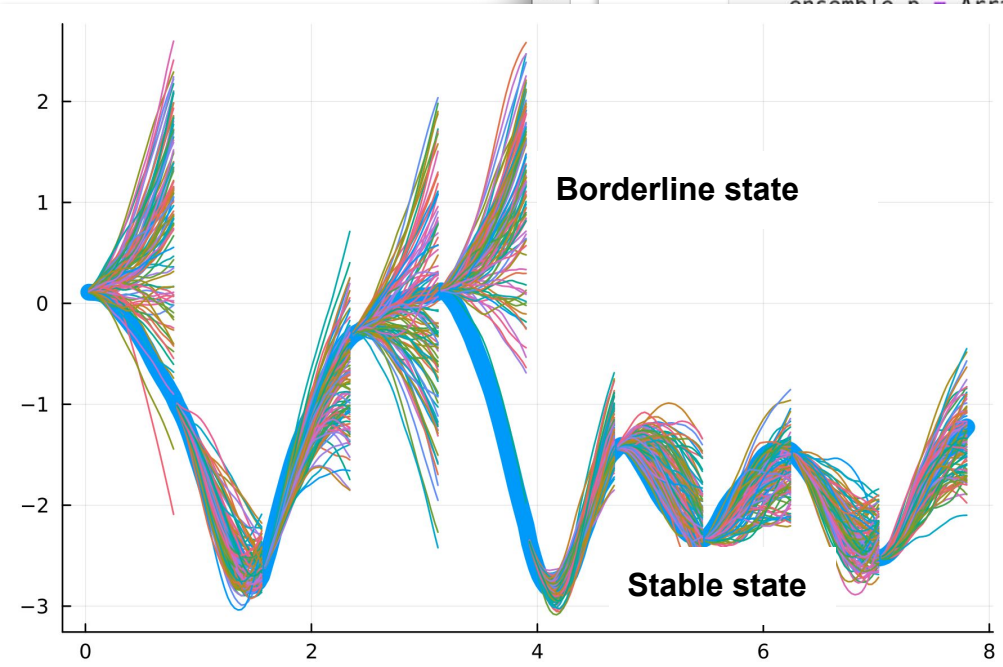
```julia
times, positions, velocities, total_energies = baoab(
    energy, force, max_time, dt,
    gamma, kBT, initial_position, initial_velocity;
    save_frequency=save_frequency, kwargs...
)

ensemble_t = Array{T, 1}[]
ensemble_p = Array{T, 1}[]
ensemble_v = Array{T, 1}[]
for (t, p, v) in zip(
        times[1:ensemble.st:end],
        positions[1:ensemble.st:end],
        velocities[1:ensemble.st:end]
    )
    p_r = rand(ensemble.d, ensemble.n)
    v_r = rand(ensemble.d, ensemble.n)

    for i in 1:ensemble.n
        t_i, p_i, v_i, _ = baoab(
            energy, force, ensemble.len*dt, dt,
            gamma, kBT, p + p_r[i], v + v_r[i];
            save_frequency=save_frequency, kwargs...
        )
        push!(ensemble_t, t .+ t_i)
        push!(ensemble_p, p_i)
        push!(ensemble_v, v_i)

    end
end
```

Solution Strategy: Local Monte-Carlo Sampling

```julia
times, positions, velocities, total_energies = baoab(
    energy, force, max_time, dt,
    gamma, kBT, initial_position, initial_velocity;
    save_frequency=save_frequency, kwargs...
)

ensemble_t = Array{T, 1}[]
ensemble_p = Array{T, 1}[]
ensemble_v = Array{T, 1}[]
for (t, p, v) in zip(
        times[1:ensemble.st:end],
        positions[1:ensemble.st:end],
        velocities[1:ensemble.st:end]
    )
    p_r = rand(ensemble.d, ensemble.n)
    v_r = rand(ensemble.d, ensemble.n)

    for i in 1:ensemble.n
        t_i, p_i, v_i, _ = baoab(
            energy, force, ensemble.len*dt, dt,
            gamma, kBT, p + p_r[i], v + v_r[i];
            save_frequency=save_frequency, kwargs...
        )
        push!(ensemble_t, t .+ t_i)
        push!(ensemble_p, p_i)
        push!(ensemble_v, v_i)

    end
end
```

For every time step, run a short simulation with slightly different starting conditions

Solution Strategy: Local Monte-Carlo Sampling




```
times, positions, velocities, total_energies = baoab(
    energy, force, max_time, dt,
    gamma, kBT, initial_position, initial_velocity;
    save_frequency=save_frequency, kwargs...
)

ensemble_t = Array{T, 1}[]
y{T, 1}[]
y{T, 1}[]
zip(
nsemble.st:end],
[1:ensemble.st:end],
s[1:ensemble.st:end]

semble.d, ensemble.n)
semble.d, ensemble.n)

semble.n
 v_i, _ = baoab(
y, force, ensemble.len*dt, dt,
, kBT, p + p_r[i], v + v_r[i];
frequency=save_frequency, kwargs...

emble_t, t .+ t_i)
emble_p, p_i)
emble_v, v_i)
```

Solution Strategy: Adaptive Algorithms

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

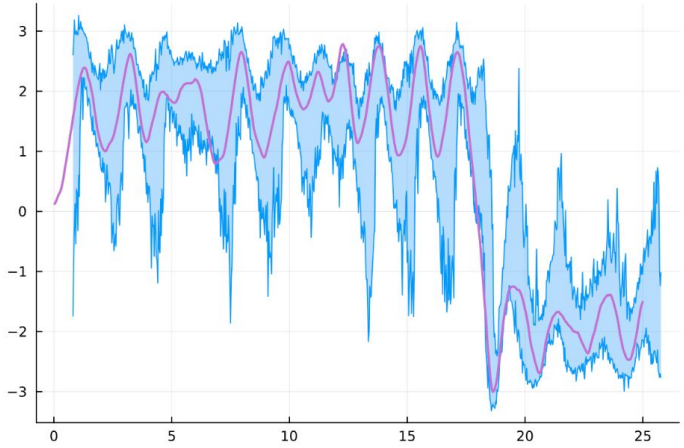
Code git Julia 1.9.4 CUNN-Enabled

```
    ensemble_range_p[k] = (minimum(s), maximum(s))
end

ensemble_t  = sort(collect(keys(ensemble_range_p)))
ensemble_lo = [ensemble_range_p[t][1] for t in ensemble_t]
ensemble_hi = [ensemble_range_p[t][2] for t in ensemble_t]

delta_t = ensemble_t[1] - times[1]
```

[17]: 0.7800000000000005

[18]:
```
plot(ensemble_t, ensemble_hi, c = 1)
plot!(ensemble_t, ensemble_lo, fillrange = ensemble_hi, fillalpha = 0.35, c = 1)
plot!(times, positions; linewidth=2, c=4, legend=false)
```

[18]:



Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 5.70 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

Solution Strategy: Adaptive Algorithms

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

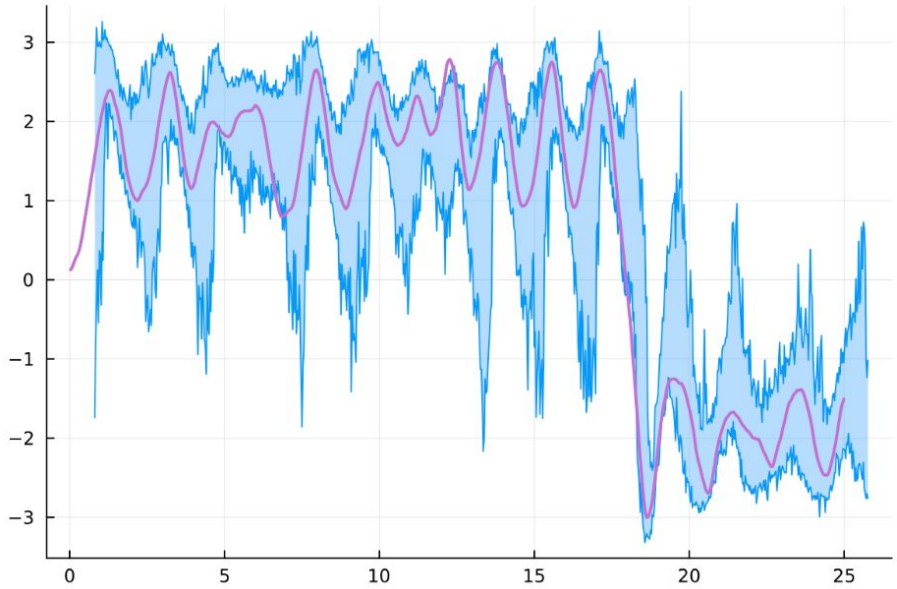
File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

```
    ensemble_range_p[k] = (minimum(s), maximum(s))
end
```

[18]:
```
plot(ensemble_t, ensemble_hi, c = 1)
plot!(ensemble_t, ensemble_lo, fillrange = ensemble_hi, fillalpha = 0.35, c = 1)
plot!(times, positions; linewidth=2, c=4, legend=false)
```

[18]:



Start Mapping a Phase Space

Julia ML Exp... (auto-p : 4) - JupyterLab

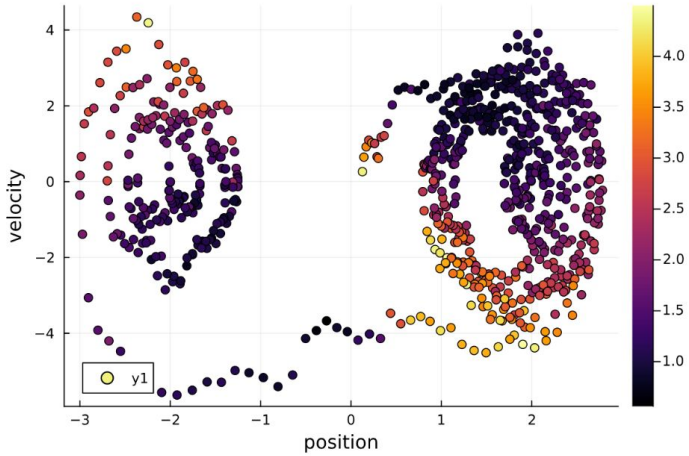
Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[20]:
```
scatter(positions, velocities, markershape=:circ, zcolor=ensemble_hi-ensemble_lo)
xlabel!("position")
ylabel!("velocity")
```

[20]:



[21]:
```
using Flux, CUDA, Statistics, ProgressMeter
```

[22]:
```
CUDA.versioninfo()
```

```
CUDA runtime 12.2, local installation
CUDA driver 12.3
NVIDIA driver 525.105.17, originally for CUDA 12.2

CUDA libraries:
- CUBLAS: 12.2.1
- CURAND: 10.3.3
```

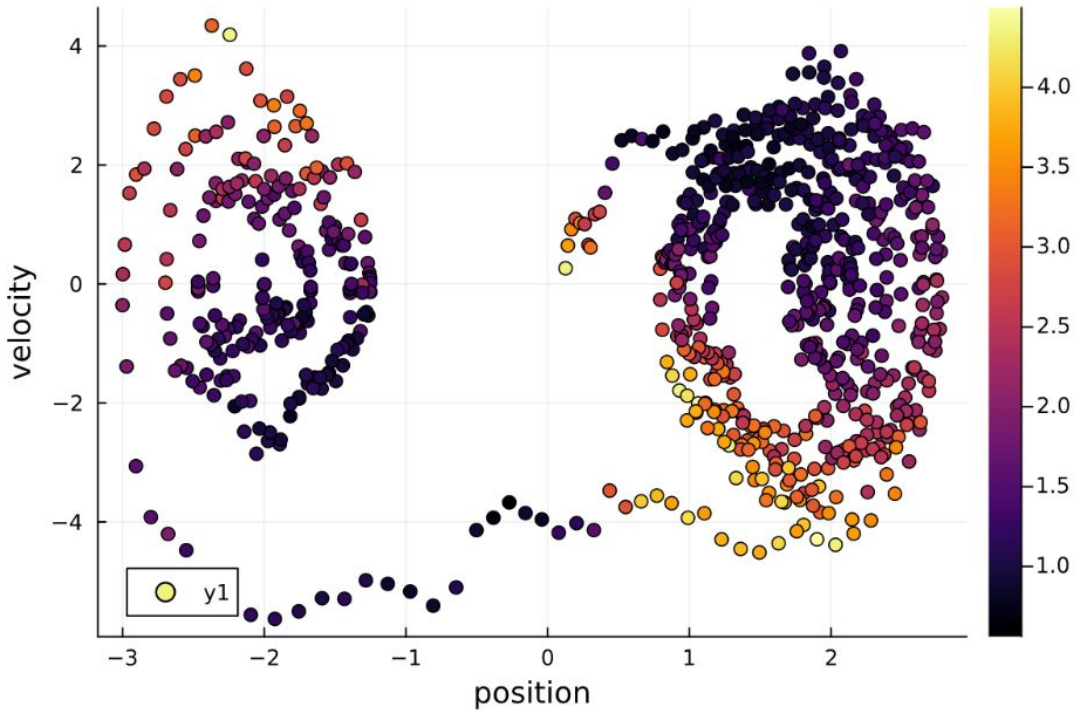
Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.40 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

Start Mapping a Phase Space

Julia ML Exp... (auto-p : 4) - JupyterLab

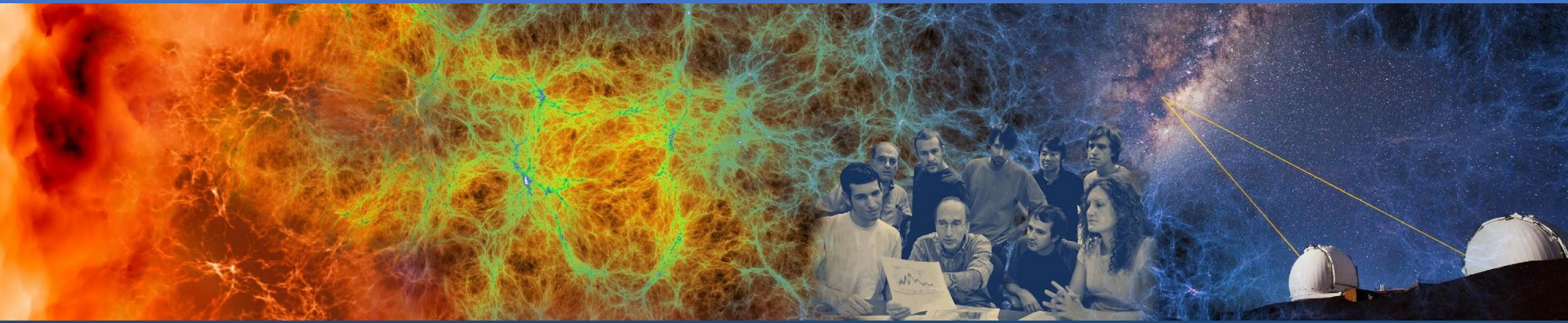
[20]:
```
scatter(positions, velocities, markershape=:circ, zcolor=ensemble_hi-ensemble_lo)
xlabel!("position")
ylabel!("velocity")
```

[20]:



Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.40 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

Machine learning using `Flux.jl`



Flux.jl Automatically Detects CUDA.jl

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[21]:
```julia
using Flux, CUDA, Statistics, ProgressMeter
```

[22]:
```julia
CUDA.versioninfo()
```

```
CUDA runtime 12.2, local installation
CUDA driver 12.3
NVIDIA driver 525.105.17, originally for CUDA 12.2

CUDA libraries:
- CUBLAS: 12.2.1
- CURAND: 10.3.3
- CUFFT: 11.0.8
- CUSOLVER: 11.5.0
- CUSPARSE: 12.1.1
- CUPTI: 20.0.0
- NVML: 12.0.0+525.105.17

Julia packages:
- CUDA: 5.2.0
- CUDA_Driver_jll: 0.7.0+1
- CUDA_Runtime_jll: 0.11.1+0
- CUDA_Runtime_Discovery: 0.2.3

Toolchain:
- Julia: 1.9.4
- LLVM: 14.0.6

Preferences:
- CUDA_Runtime_jll.version: 12.2
- CUDA_Runtime_jll.local: true

1 device:
  0: NVIDIA A100-PCIE-40GB (sm_80, 38.985 GiB / 40.000 GiB available)
```

[29]:
```julia
# Phase-space coordinate
pcoord = hcat(positions, velocities) |> transpose
# Phase-space class
lrate = ensemble_hi - ensemble_lo  # Lyapunov rate
pclass = Array{Int16, 1}(undef, size(lrate)...)
fill!(pclass, 0)
```

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.79 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Flux.jl Automatically Detects CUDA.jl

```
[21]: using Flux, CUDA, Statistics, ProgressMeter

[22]: CUDA.versioninfo()

CUDA runtime 12.2, local installation
CUDA driver 12.3
NVIDIA driver 525.105.17, originally for CUDA 12.2

CUDA libraries:
- CUBLAS: 12.2.1
- CURAND: 10.3.3
- CUFFT: 11.0.8
- CUSOLVER: 11.5.0
- CUSPARSE: 12.1.1
- CUPTI: 20.0.0
- NVML: 12.0.0+525.105.17

Julia packages:
- CUDA: 5.2.0
- CUDA_Driver_jll: 0.7.0+1
- CUDA_Runtime_jll: 0.11.1+0
- CUDA_Runtime_Discovery: 0.2.3

Toolchain:
- Julia: 1.9.4
- LLVM: 14.0.6

Preferences:
- CUDA_Runtime_jll.version: 12.2
- CUDA_Runtime_jll.local: true

1 device:
  0: NVIDIA A100-PCIE-40GB (sm_80, 38.985 GiB / 40.000 GiB available)
```

Julia introspection is a powerful tool to detect / confirm system configuration

At NERSC Julia is configured to automatically detect the system's CUDA runtime

Define Input Data

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[29]:
```
# Phase-space coordinate
pcoord = hcat(positions, velocities) |> transpose
# Phase-space class
lrate = ensemble_hi - ensemble_lo  # Lyapunov rate
pclass = Array{Int16, 1}(undef, size(lrate)...)
fill!(pclass, 0)
pclass[lrate .> 2] .= 1
pclass[lrate .> 3] .= 2;
```

[30]:
```
N_coord = 2
N_class = 3
N_neuron = 8

model = Chain(
    Dense(N_coord => N_neuron, tanh),
    BatchNorm(N_neuron),
    Dense(N_neuron => N_class),
    softmax
) |> gpu        # move model to GPU, if available
```

[30]:
```
Chain(
  Dense(2 => 8, tanh),                  # 24 parameters
  BatchNorm(8),                         # 16 parameters, plus 16
  Dense(8 => 3),                        # 27 parameters
  NNlib.softmax,
)         # Total: 6 trainable arrays, 67 parameters,
          # plus 2 non-trainable, 16 parameters, summarysize 1.266 KiB.
```

[31]:
```
# The model encapsulates parameters, randomly initialised. Its initial output is:
out1 = model(pcoord |> gpu) |> cpu     # 2×N Matrix{Float32}
```

[31]:
```
3×833 Matrix{Float32}:
 0.30174   0.292602  0.286916  0.283165  …  0.401954  0.436592  0.44925
 0.291435  0.260622  0.244021  0.235257     0.362114  0.384997  0.391649
 0.406825  0.446776  0.469063  0.481578     0.235932  0.17841   0.159102
```

[32]:
```
# To train the model, we use batches of 128 samples, and one-hot encoding:
target = Flux.onehotbatch(pclass, [0, 1, 2])                    # 2×N OneHotMatrix
loader = Flux.DataLoader((pcoord, target) |> gpu, batchsize=128, shuffle=true);
```

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.75 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Define Input Data

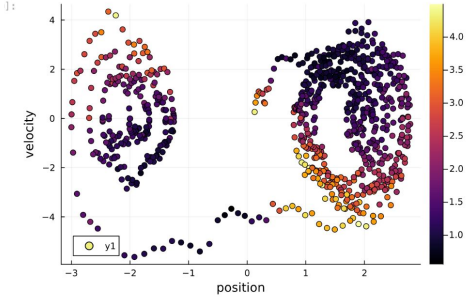
Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Classify point in phase space based on local MC sample's spread rate (~Lyapunov rate)

```
[29]: # Phase-space coordinate
pcoord = hcat(positions, velocities) |> transpose
# Phase-space class
lrate = ensemble_hi - ensemble_lo  # Lyapunov rate
pclass = Array{Int16, 1}(undef, size(lrate)...)
fill!(pclass, 0)
pclass[lrate .> 2] .= 1
pclass[lrate .> 3] .= 2;
```




```
model = Chain(
    Dense(N_coord => N_neuron, tanh),
    BatchNorm(N_neuron),
    Dense(N_neuron => N_class),
    softmax
) |> gpu        # move model to GPU, if available
```

```
[30]: Chain(
  Dense(2 => 8, tanh),                  # 24 parameters
  BatchNorm(8),                         # 16 parameters, plus 16
  Dense(8 => 3),                        # 27 parameters
  NNlib.softmax,
)         # Total: 6 trainable arrays, 67 parameters,
          # plus 2 non-trainable, 16 parameters, summarysize 1.266 KiB.
```

```
[31]: # The model encapsulates parameters, randomly initialised. Its initial output is:
out1 = model(pcoord |> gpu) |> cpu     # 2×N Matrix{Float32}
```

```
[31]: 3×833 Matrix{Float32}:
 0.30174   0.292602  0.286916  0.283165  …  0.401954  0.436592  0.44925
 0.291435  0.260622  0.244021  0.235257     0.362114  0.384997  0.391649
 0.406825  0.446776  0.469063  0.481578     0.235932  0.17841   0.159102
```

```
[32]: # To train the model, we use batches of 128 samples, and one-hot encoding:
target = Flux.onehotbatch(pclass, [0, 1, 2])                # 2×N OneHotMatrix
loader = Flux.DataLoader((pcoord, target) |> gpu, batchsize=128, shuffle=true);
```

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.75 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Define a (simple) Neural Network Model

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

```
[29]: # Phase-space coordinate
      pcoord = hcat(positions, velocities) |> transpose
      # Phase-space class
      lrate = ensemble_hi - ensemble_lo  # Lyapunov rate
```

```
[30]: N_coord = 2
      N_class = 3
      N_neuron = 8

      model = Chain(
          Dense(N_coord => N_neuron, tanh),
          BatchNorm(N_neuron),
          Dense(N_neuron => N_class),
          softmax
      ) |> gpu        # move model to GPU, if available
```

Chain conveniently chains together layers. Ingests a 2D (position, velocity) vector, outputs 3D class probability vector

```
[30]: Chain(
        Dense(2 => 8, tanh),                  # 24 parameters
        BatchNorm(8),                         # 16 parameters, plus 16
        Dense(8 => 3),                        # 27 parameters
        NNlib.softmax,
      )         # Total: 6 trainable arrays, 67 parameters,
                # plus 2 non-trainable, 16 parameters, summarysize 1.266 KiB.
```

```
[32]: # To train the model, we use batches of 128 samples, and one-hot encoding:
      target = Flux.onehotbatch(pclass, [0, 1, 2])                # 2×N OneHotMatrix
      loader = Flux.DataLoader((pcoord, target) |> gpu, batchsize=128, shuffle=true);
```

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.75 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Use `|> gpu` and `|> cpu` to Move Data

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[29]:
```
# Phase-space coordinate
pcoord = hcat(positions, velocities) |> transpose
# Phase-space class
lrate = ensemble_hi - ensemble_lo  # Lyapunov rate
pclass = Array{Int16, 1}(undef, size(lrate)...)
fill!(pclass, 0)
pclass[lrate .> 2] .= 1
pclass[lrate .> 3] .= 2;
```

[30]:
```
N_coord = 2
N_class = 3
N_neuron = 8

model = Chain(
    Dense(N_coord => N_neuron, tanh),
    BatchNorm(N_neuron),
    Dense(N_neuron => N_class),
    softmax
) |> gpu        # move model to GPU, if available
```

[30]:
```
Chain(
  Dense(2 => 8, tanh),                  # 24 parameters
  BatchNorm(8),                         # 16 parameters, plus 16
```

```
# The model encapsulates parameters, randomly initialised. Its initial output is:
out1 = model(pcoord |> gpu) |> cpu      # 2×N Matrix{Float32}
```

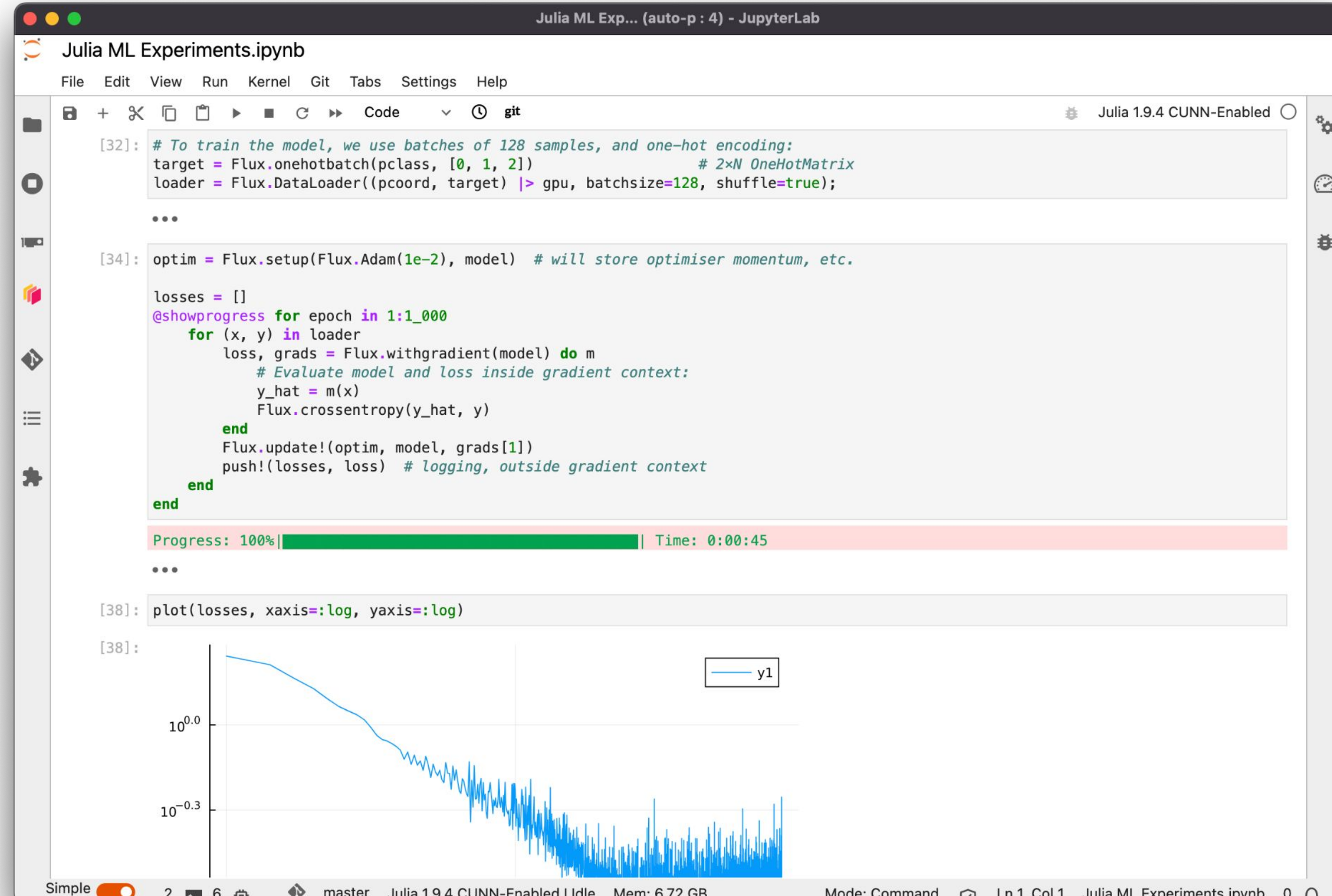
```
3×833 Matrix{Float32}:
 0.30174   0.292602  0.286916  0.283165  …  0.401954  0.436592  0.44925
 0.291435  0.260622  0.244021  0.235257     0.362114  0.384997  0.391649
 0.406825  0.446776  0.469063  0.481578     0.235932  0.17841   0.159102
```

```
target = Flux.onehotbatch(pclass, [0, 1, 2])                    # 2×N OneHotMatrix
```

Column-major inputs. Model outputs the likelihoods (columns) of each class

NERSC

Train the Model



[32]:

```
# To train the model, we use batches of 128 samples, and one-hot encoding:
target = Flux.onehotbatch(pclass, [0, 1, 2])                    # 2×N OneHotMatrix
loader = Flux.DataLoader((pcoord, target) |> gpu, batchsize=128, shuffle=true);
```

[34]:

```
optim = Flux.setup(Flux.Adam(1e-2), model)  # will store optimiser momentum, etc.

losses = []
@showprogress for epoch in 1:1_000
    for (x, y) in loader
        loss, grads = Flux.withgradient(model) do m
            # Evaluate model and loss inside gradient context:
            y_hat = m(x)
            Flux.crossentropy(y_hat, y)
        end
        Flux.update!(optim, model, grads[1])
        push!(losses, loss)  # logging, outside gradient context
    end
end
```

```
Progress: 100%|█████████████████████████████████████████| Time: 0:00:45
```

[38]:

```
plot(losses, xaxis=:log, yaxis=:log)
```

[38]:

Train the Model

`|> gpu` also works with complex data types (tuples, vectors, structs, tuples of structs, …)

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

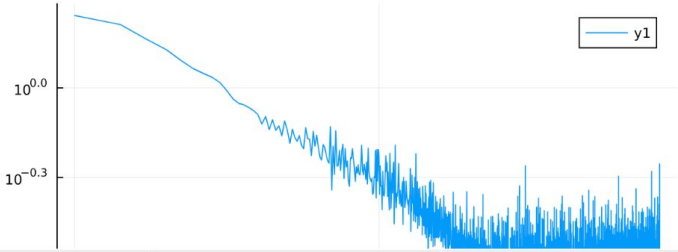
```julia
# To train the model, we use batches of 128 samples, and one-hot encoding:
target = Flux.onehotbatch(pclass, [0, 1, 2])                    # 2×N OneHotMatrix
loader = Flux.DataLoader((pcoord, target) |> gpu, batchsize=128, shuffle=true);
```

```julia
losses = []
@showprogress for epoch in 1:1_000
    for (x, y) in loader
        loss, grads = Flux.withgradient(model) do m
            # Evaluate model and loss inside gradient context:
            y_hat = m(x)
            Flux.crossentropy(y_hat, y)
        end
        Flux.update!(optim, model, grads[1])
        push!(losses, loss)  # logging, outside gradient context
    end
end
```

Progress: 100%| | Time: 0:00:45

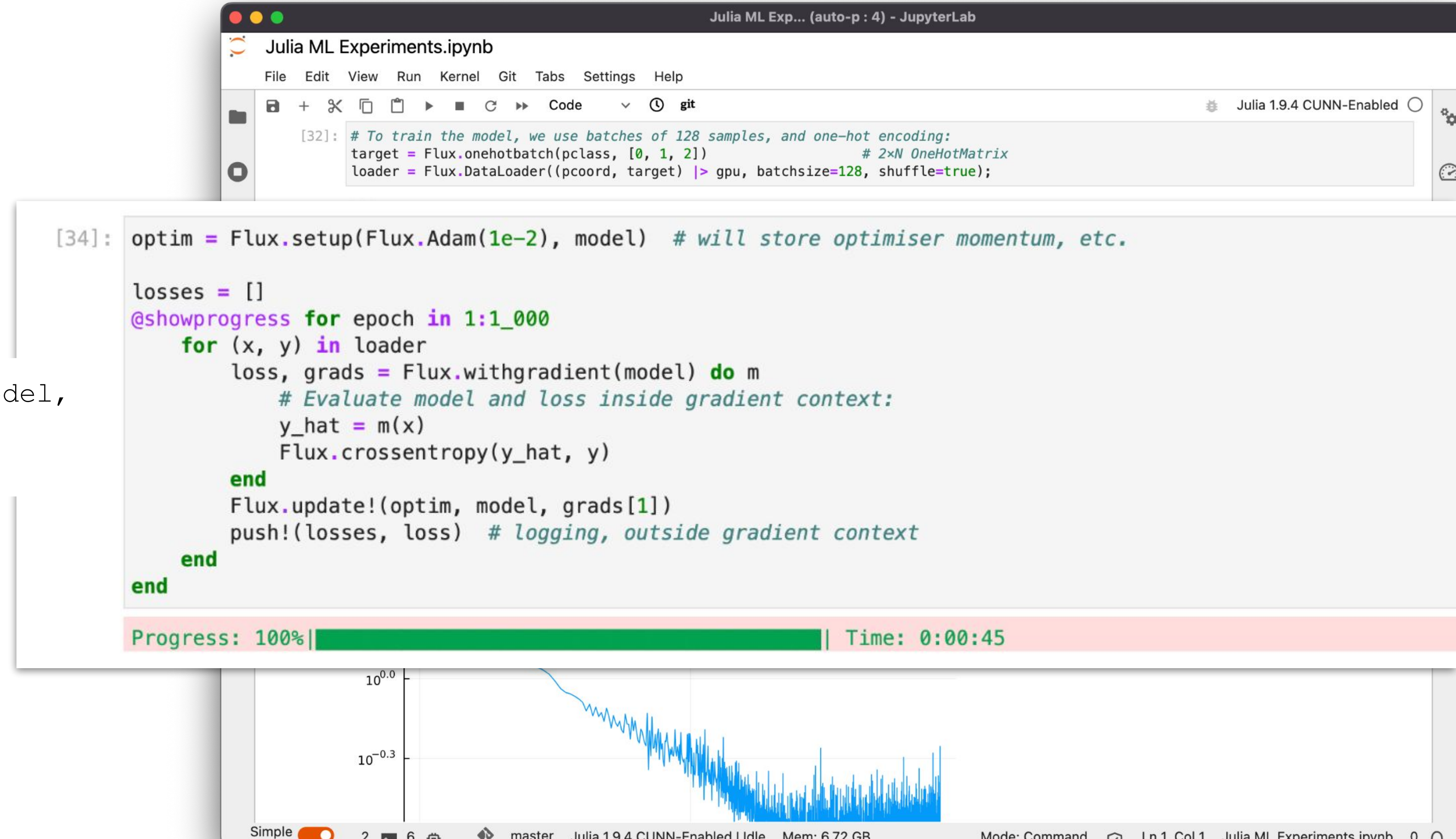
[38]:
```julia
plot(losses, xaxis=:log, yaxis=:log)
```

[38]:



Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.72 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

Train the Model



[32]:

```julia
# To train the model, we use batches of 128 samples, and one-hot encoding:
target = Flux.onehotbatch(pclass, [0, 1, 2])                  # 2×N OneHotMatrix
loader = Flux.DataLoader((pcoord, target) |> gpu, batchsize=128, shuffle=true);
```

[34]:

```julia
optim = Flux.setup(Flux.Adam(1e-2), model)  # will store optimiser momentum, etc.

losses = []
@showprogress for epoch in 1:1_000
    for (x, y) in loader
        loss, grads = Flux.withgradient(model) do m
            # Evaluate model and loss inside gradient context:
            y_hat = m(x)
            Flux.crossentropy(y_hat, y)
        end
        Flux.update!(optim, model, grads[1])
        push!(losses, loss)  # logging, outside gradient context
    end
end
```

```
Progress: 100%|██████████████████████████████████████| Time: 0:00:45
```

`Flux.train!(model, loader, optim)` does all of this

We've Finished Building our AI Classifier

Julia ML Exp... (auto-p : 4) - JupyterLab

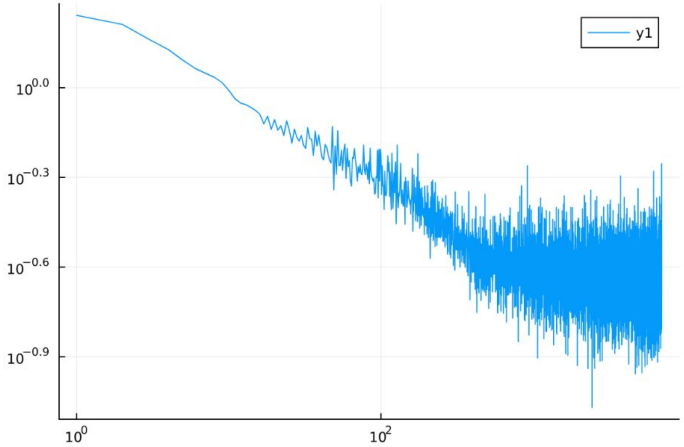
Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[38]:
```
plot(losses, xaxis=:log, yaxis=:log)
```

[38]:



[39]:
```
function classify(coords)
    prob_class = model(coords |> gpu) |> cpu
    return map(x->x[1], argmax(prob_class, dims=1))[:];
end
```

[39]:
```
classify (generic function with 1 method)
```

[40]:
```
rand_coord = 6 .* rand(Float32, 2, 10_000) .- 3
```

[40]:
```
2×10000 Matrix{Float32}:
  1.58479  -2.6759   -0.757071  -1.1845  …  -0.673121    0.585823   1.14873
 -1.77659  -2.31748   0.752013  -1.3579     -0.0657971  -2.94806   -1.41052
```

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.79 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

We've Finished Building our AI Classifier

Julia ML Exp... (auto-p : 4) - JupyterLab

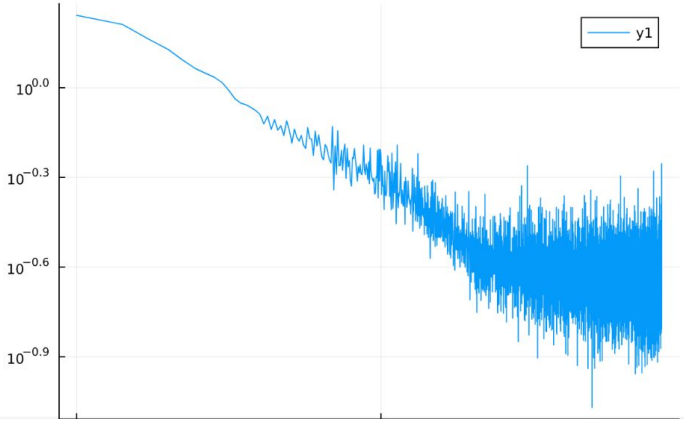
Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[38]:
```
plot(losses, xaxis=:log, yaxis=:log)
```

[38]:



[39]:
```
function classify(coords)
    prob_class = model(coords |> gpu) |> cpu
    return map(x->x[1], argmax(prob_class, dims=1))[:];
end
```

[40]:
```
rand_coord = 6 .* rand(Float32, 2, 10_000) .- 3
```

[40]:
```
2×10000 Matrix{Float32}:
  1.58479  -2.6759   -0.757071  -1.1845  …  -0.673121    0.585823   1.14873
 -1.77659  -2.31748   0.752013  -1.3579     -0.0657971  -2.94806   -1.41052
```

Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.79 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

ML Model Infers

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[39]:
```
function classify(coords)
    prob_class = model(coords |> gpu) |> cpu
    return map(x->x[1], argmax(prob_class, dims=1))[:];
end
```

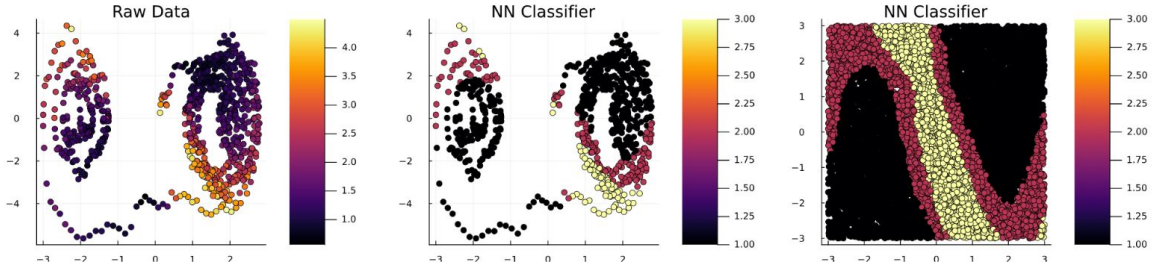
[39]:
```
classify (generic function with 1 method)
```

[40]:
```
rand_coord = 6 .* rand(Float32, 2, 10_000) .- 3
```

[40]:
```
2×10000 Matrix{Float32}:
  1.58479  -2.6759   -0.757071  -1.1845  …  -0.673121    0.585823   1.14873
 -1.77659  -2.31748   0.752013  -1.3579     -0.0657971  -2.94806   -1.41052
```

[41]:
```
p_true = scatter(pcoord[1, :],     pcoord[2,:],      zcolor=lrate,                title="Raw Data")
p_nn   = scatter(pcoord[1, :],     pcoord[2,:],      zcolor=classify(pcoord),     title="NN Classifier")
p_rand = scatter(rand_coord[1, :], rand_coord[2, :], zcolor=classify(rand_coord), title="NN Classifier")

plot(p_true, p_nn, p_rand, layout=(1,3), size=(1500,330), legend=false)
```

[41]:



[ ]:

Simple 2 6 master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.79 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb 0

Julia ML Exp... (auto-p : 4) - JupyterLab

Julia ML Experiments.ipynb

File Edit View Run Kernel Git Tabs Settings Help

```
[40]: rand_coord = 6 .* rand(Float32, 2, 10_000) .- 3
```

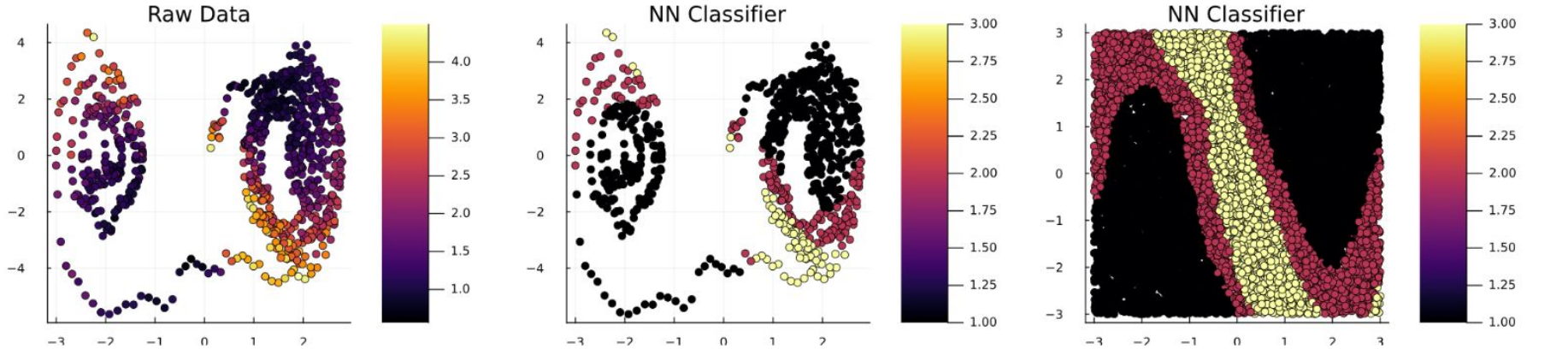
```
[40]: 2×10000 Matrix{Float32}:
  1.58479  -2.6759   -0.757071  -1.1845  …  -0.673121    0.585823   1.14873
 -1.77659  -2.31748   0.752013  -1.3579     -0.0657971  -2.94806   -1.41052
```


```
[40]: rand_coord = 6 .* rand(Float32, 2, 10_000) .- 3
```

```
[40]: 2×10000 Matrix{Float32}:
  1.58479  -2.6759   -0.757071  -1.1845  …  -0.673121    0.585823   1.14873
 -1.77659  -2.31748   0.752013  -1.3579     -0.0657971  -2.94806   -1.41052
```

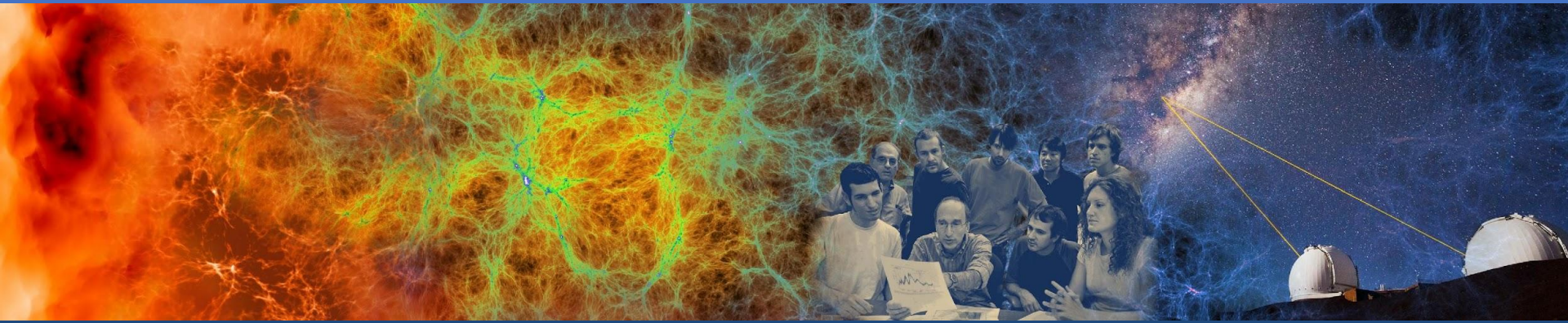

```
plot(p_true, p_nn, p_rand, layout=(1,3), size=(1500,330), legend=false)
```

[41]:



Simple master Julia 1.9.4 CUNN-Enabled Idle Mem: 6.79 GB Mode: Command Ln 1, Col 1 Julia ML Experiments.ipynb

Using `Dagger.jl` To parallelize your workflow



Dagger.jl: Easy work distribution

SC23.ipynb (11) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[1]:
```
using Distributed, ClusterManagers, BenchmarkTools
```

[2]:
```
addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")
```

```
connecting to worker 1 out of 2
srun: job 18135272 queued and waiting for resources
srun: job 18135272 has been allocated resources
connecting to worker 2 out of 2
```

[2]:
```
2-element Vector{Int64}:
 2
 3
```

[3]:
```
@everywhere using Dagger
```

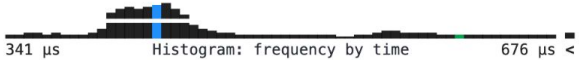
[11]:
```
darr1 = rand(Blocks(500, 500), 1_000, 1_000);
darr2 = rand(Blocks(500, 500), 1_000, 1_000);
```

[12]:
```
function test_mul()
    darr1 * darr2
    wait;
end
```

[12]:
```
test_mul (generic function with 1 method)
```

[13]:
```
@benchmark(test_mul())
```

[13]:
```
BenchmarkTools.Trial: 8128 samples with 1 evaluation.
 Range (min … max):  340.665 µs …   1.253 s  ┊ GC (min … max):  0.00% … 87.19%
 Time  (median):     432.180 µs              ┊ GC (median):     0.00%
 Time  (mean ± σ):   612.252 µs ± 13.935 ms  ┊ GC (mean ± σ):  21.96% ±  0.97%
```




```
 Memory estimate: 167.05 KiB, allocs estimate: 3113.
```

[ ]:

Dagger.jl: Easy work distribution

SC23.ipynb (11) - JupyterLab

SC23.ipynb

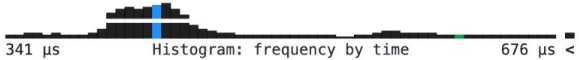
File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[1]: using Distributed, ClusterManagers, BenchmarkTools

[2]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")

connecting to worker 1 out of 2
srun: job 18135272 queued and waiting for resources
srun: job 18135272 has been allocated resources
```

```
[3]: @everywhere using Dagger

[11]: darr1 = rand(Blocks(500, 500), 1_000, 1_000);
      darr2 = rand(Blocks(500, 500), 1_000, 1_000);
```

```
    darr1 * darr2
    wait;
end

[12]: test_mul (generic function with 1 method)

[13]: @benchmark(test_mul())

[13]: BenchmarkTools.Trial: 8128 samples with 1 evaluation.
 Range (min … max):  340.665 μs …   1.253 s  ┊ GC (min … max):  0.00% … 87.19%
 Time  (median):     432.180 μs              ┊ GC (median):     0.00%
 Time  (mean ± σ):   612.252 μs ± 13.935 ms  ┊ GC (mean ± σ):  21.96% ±  0.97%
```




```
 Memory estimate: 167.05 KiB, allocs estimate: 3113.

[ ]:
```

Dagger.jl: Easy work distribution

SC23.ipynb (11) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[1]: using Distributed, ClusterManagers, BenchmarkTools

[2]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")

connecting to worker 1 out of 2
srun: job 18135272 queued and waiting for resources
srun: job 18135272 has been allocated resources
connecting to worker 2 out of 2

[2]: 2-element Vector{Int64}:
 2
 3

[3]: @everywhere using Dagger

[11]: darr1 = rand(Blocks(500, 500), 1_000, 1_000);

[12]: function test_mul()
          darr1 * darr2
          wait;
      end

[13]: BenchmarkTools.Trial: 8128 samples with 1 evaluation.
       Range (min … max):  340.665 μs …   1.253 s  ┊ GC (min … max):  0.00% … 87.19%
       Time  (median):     432.180 μs              ┊ GC (median):     0.00%
       Time  (mean ± σ):   612.252 μs ± 13.935 ms  ┊ GC (mean ± σ):  21.96% ±  0.97%

       341 μs           Histogram: frequency by time          676 μs <

       Memory estimate: 167.05 KiB, allocs estimate: 3113.

[ ]:
```

NERSC

Dagger.jl: Easy work distribution

SC23.ipynb (11) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

```
[1]: using Distributed, ClusterManagers, BenchmarkTools

[2]: addprocs(SlurmManager(2), N="2", constrain="cpu", qos="interactive", time="00:15:00", A="nstaff")

connecting to worker 1 out of 2
srun: job 18135272 queued and waiting for resources
srun: job 18135272 has been allocated resources
connecting to worker 2 out of 2

[2]: 2-element Vector{Int64}:
 2
 3

[3]: @everywhere using Dagger

[11]: darr1 = rand(Blocks(500, 500), 1_000, 1_000);
      darr2 = rand(Blocks(500, 500), 1_000, 1_000);
```

```
[13]: @benchmark(test_mul())

[13]: BenchmarkTools.Trial: 8128 samples with 1 evaluation.
 Range (min … max):  340.665 μs …   1.253 s  ┊ GC (min … max):  0.00% … 87.19%
 Time  (median):     432.180 μs              ┊ GC (median):     0.00%
 Time  (mean ± σ):   612.252 μs ± 13.935 ms  ┊ GC (mean ± σ):  21.96% ±  0.97%
```

341 μs Histogram: frequency by time 676 μs <

For references: openBLAS (single node) = 3.15ms, cuBLAS = 121μs

Dagger Provides Async Task Spawning

DataDay2024.... (auto-p : 4) - JupyterLab

DataDay2024.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[10]:
```
th = Dagger.@spawn sleep(10)
while true
    st = isready(th)
    println("isready = $(st)")
    if ! st
        sleep(1)
    else
        break
    end
end
```

```
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = true
```

[11]:
```
@everywhere function f(dist, len, reps, σ)
    v = Vector{Float64}(undef, len) # avoiding allocations
    maximum(mean(rand!(dist, v)) for _ in 1:reps)/σ, myid()
end
```

[14]:
```
function experiments_dagger(dists, lens, K=1000)
    res = DataFrame()
    @sync for T in dists
        dist = T()
        σ = Dagger.@spawn std(dist)
        for L in lens
            z = Dagger.@spawn f(dist, L, K, σ)
            push!(res, (;T, σ, L, z))
        end
    end
```

Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 9.82 GB Mode: Command Ln 1, Col 1 DataDay2024.ipynb 0

Dagger Provides Async Task Spawning

Tasks start immediately.
Spawning is non-blocking.
This is called `EagerThunk`

DataDay2024.... (auto-p : 4) - JupyterLab

```
[10]: th = Dagger.@spawn sleep(10)
while true
    st = isready(th)
    println("isready = $(st)")
    if ! st
        sleep(1)
    else
        break
    end
end
```

```
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = true
```

```
[14]: function experiments_dagger(dists, lens, K=1000)
    res = DataFrame()
    @sync for T in dists
        dist = T()
        σ = Dagger.@spawn std(dist)
        for L in lens
            z = Dagger.@spawn f(dist, L, K, σ)
            push!(res, (;T, σ, L, z))
        end
    end
```

Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 9.82 GB Mode: Command Ln 1, Col 1 DataDay2024.ipynb 0

Dagger Provides Async Task Spawning

DataDay2024.... (auto-p : 4) - JupyterLab

DataDay2024.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[10]:
```
th = Dagger.@spawn sleep(10)
while true
    st = isready(th)
    println("isready = $(st)")
    if ! st
        sleep(1)
    else
        break
    end
end
```

```
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
```

Track where the function runs

Functions to be used by `@spawn` need to be defined with `@everywhere`

11]:
```
@everywhere function f(dist, len, reps, σ)
    v = Vector{Float64}(undef, len) # avoiding allocations
    maximum(mean(rand!(dist, v)) for _ in 1:reps)/σ, myid()
end
```

[14]:
```
function experiments_dagger(dists, lens, K=1000)
    res = DataFrame()
    @sync for T in dists
        dist = T()
        σ = Dagger.@spawn std(dist)
        for L in lens
            z = Dagger.@spawn f(dist, L, K, σ)
            push!(res, (;T, σ, L, z))
        end
    end
```

Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 9.82 GB Mode: Command Ln 1, Col 1 DataDay2024.ipynb 0

Use Loop to Submit Tasks to Workers

DataDay2024.... (auto-p : 4) - JupyterLab

DataDay2024.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[11]:
```
@everywhere function f(dist, len, reps, σ)
    v = Vector{Float64}(undef, len) # avoiding allocations
    maximum(mean(rand!(dist, v)) for _ in 1:reps)/σ, myid()
end
```

[14]:
```
function experiments_dagger(dists, lens, K=1000)
    res = DataFrame()
    @sync for T in dists
        dist = T()
        σ = Dagger.@spawn std(dist)
        for L in lens
            z = Dagger.@spawn f(dist, L, K, σ)
            push!(res, (;T, σ, L, z))
        end
    end
    res.z = fetch.(res.z)
    res.σ = fetch.(res.σ)
    res
end
```

[14]:
```
experiments_dagger (generic function with 2 methods)
```

[15]:
```
dists = [
    Cosine,
    Epanechnikov,
    Laplace,
    Logistic,
    Normal,
    NormalCanon,
    PGeneralizedGaussian,
    SkewNormal,
    SkewedExponentialPower,
    SymTriangularDist
]
lens = [10, 20, 50, 100, 200, 500]

x = experiments_dagger(dists, lens)
```

[15]: 60×4 DataFrame *35 rows omitted*

Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 9.82 GB Mode: Command Ln 1, Col 1 DataDay2024.ipynb 0

Use Loop to Submit Tasks to Workers

DataDay2024.... (auto-p : 4) - JupyterLab

DataDay2024.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[14]:
```
function experiments_dagger(dists, lens, K=1000)
    res = DataFrame()
    @sync for T in dists
        dist = T()
        σ = Dagger.@spawn std(dist)
        for L in lens
            z = Dagger.@spawn f(dist, L, K, σ)
            push!(res, (;T, σ, L, z))
        end
    end
    res.z = fetch.(res.z)
    res.σ = fetch.(res.σ)
    res
end
```

`@spawn` is non-blocking, `fetch` or using a variable are blocking

[15]:
```
dists = [
    Cosine,
    Epanechnikov,
    Laplace,
    Logistic,
    Normal,
    NormalCanon,
    PGeneralizedGaussian,
    SkewNormal,
    SkewedExponentialPower,
    SymTriangularDist
]
lens = [10, 20, 50, 100, 200, 500]

x = experiments_dagger(dists, lens)
```

[15]: 60×4 DataFrame *35 rows omitted*

Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 9.82 GB Mode: Command Ln 1, Col 1 DataDay2024.ipynb 0

Where Did All The Tasks Run?

DataDay2024.... (auto-p : 4) - JupyterLab

DataDay2024.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[15]:
```
dists = [
    Cosine,
    Epanechnikov,
    Laplace,
    Logistic,
    Normal,
    NormalCanon,
    PGeneralizedGaussian,
    SkewNormal,
    SkewedExponentialPower,
    SymTriangularDist
]
lens = [10, 20, 50, 100, 200, 500]

x = experiments_dagger(dists, lens)
```

[15]: 60×4 DataFrame *35 rows omitted*

Row	T	σ	L	z
	UnionAll	Float64	Int64	Tuple...
1	Cosine	0.361512	10	(0.879684, 2)
2	Cosine	0.361512	20	(0.738629, 3)
3	Cosine	0.361512	50	(0.420286, 7)
4	Cosine	0.361512	100	(0.331991, 9)
5	Cosine	0.361512	200	(0.199034, 11)
6	Cosine	0.361512	500	(0.151369, 6)
7	Epanechnikov	0.447214	10	(0.939707, 5)
8	Epanechnikov	0.447214	20	(0.664243, 10)
9	Epanechnikov	0.447214	50	(0.484392, 6)
10	Epanechnikov	0.447214	100	(0.287399, 8)
11	Epanechnikov	0.447214	200	(0.250568, 11)
12	Epanechnikov	0.447214	500	(0.148376, 2)

Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 9.82 GB Mode: Command Ln 1, Col 1 DataDay2024.ipynb 0

Where Did

[15]:
```
dists = [
    Cosine,
    Epanechnikov,
    Laplace,
    Logistic,
    Normal,
    NormalCanon,
    PGeneralizedGaussian,
    SkewNormal,
    SkewedExponentialPower,
    SymTriangularDist
]
lens = [10, 20, 50, 100, 200, 500]

x = experiments_dagger(dists, lens)
```

[15]: 60×4 DataFrame

Row	T	σ	L	z
	UnionAll	Float64	Int64	Tuple...
1	Cosine	0.361512	10	(0.879684, 2)
2	Cosine	0.361512	20	(0.738629, 3)
3	Cosine	0.361512	50	(0.420286, 7)
4	Cosine	0.361512	100	(0.331991, 9)
5	Cosine	0.361512	200	(0.199034, 11)
6	Cosine	0.361512	500	(0.151369, 6)
7	Epanechnikov	0.447214	10	(0.939707, 5)
8	Epanechnikov	0.447214	20	(0.664243, 10)
9	Epanechnikov	0.447214	50	(0.484392, 6)
10	Epanechnikov	0.447214	100	(0.287399, 8)
11	Epanechnikov	0.447214	200	(0.250568, 11)

All the tasks are distributed over the `Distributed.jl` processors

Time to Accelerate the MC Sampler

Serial version:

1. Simulate N steps
2. Simulate `ensemble.n` independent trajectories
3. Aggregate data

```
times, positions, velocities, total_energies = baoab(
    energy, force, N*dt, dt,
    gamma, kBT, initial_position, initial_velocity;
    save_frequency=save_frequency, kwargs...
)

p_r = rand(ensemble.d, ensemble.n)
v_r = rand(ensemble.d, ensemble.n)

dtasks = Any[]
for i in 1:ensemble.n
    p = initial_position + p_r[i]
    v = initial_velocity + v_r[i]

    dtask = baoab(
        energy, force, ensemble.len*dt, dt,
        gamma, kBT, p + p_r[i], v + v_r[i];
        save_frequency=save_frequency, kwargs...
    )

    push!(dtasks, dtask)
end

ensemble_t = Array{T, 1}[]
ensemble_p = Array{T, 1}[]
for task in dtasks
    t_i, p_i, v_i, _ = task
    push!(ensemble_t, t .+ t_i)
    push!(ensemble_p, p_i)
end
```

Time to Accelerate the MC Sampler

Naive parallelization:

1. Simulate N steps
2. Spawn `ensemble.n` independent trajectory simulation tasks
3. `Fetch` data

```
times, positions, velocities, total_energies = baoab(
    energy, force, N*dt, dt,
    gamma, kBT, initial_position, initial_velocity;
    save_frequency=save_frequency, kwargs...
)

p_r = rand(ensemble.d, ensemble.n)
v_r = rand(ensemble.d, ensemble.n)

dtasks = Dagger.EagerThunk[]
@sync for i in 1:ensemble.n
    p = initial_position + p_r[i]
    v = initial_velocity + v_r[i]

    dtask = Dagger.@spawn baoab(
        energy, force, ensemble.len*dt, dt,
        gamma, kBT, p + p_r[i], v + v_r[i];
        save_frequency=save_frequency, kwargs...
    )

    push!(dtasks, dtask)
end

ensemble_t = Array{T, 1}[]
ensemble_p = Array{T, 1}[]
for task in dtasks
    t_i, p_i, _, _ = fetch(task)
    push!(ensemble_t, t .+ t_i)
    push!(ensemble_p, p_i)
end
```

There is No Free Lunch

DataDay2024_... (auto-p : 4) - JupyterLab

DataDay2024_3.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[220]:
```
@benchmark integrate_ensemble(4, initial_position, initial_velocity)
```

[220]:
```
BenchmarkTools.Trial: 100 samples with 1 evaluation.
 Range (min … max):  46.908 ms … 58.275 ms  ┊ GC (min … max): 14.97% … 26.60%
 Time  (median):     49.374 ms              ┊ GC (median):    16.31%
 Time  (mean ± σ):   50.150 ms ±  2.361 ms  ┊ GC (mean ± σ):  16.60% ±  2.01%

  46.9 ms         Histogram: frequency by time        56.8 ms <

 Memory estimate: 55.27 MiB, allocs estimate: 861963.
```

[221]:
```
@benchmark integrate_ensemble_dagger_1(4, initial_position, initial_velocity)
```

[221]:
```
BenchmarkTools.Trial: 3 samples with 1 evaluation.
 Range (min … max):  2.131 s …   2.185 s  ┊ GC (min … max): 6.53% … 10.35%
 Time  (median):     2.131 s              ┊ GC (median):    8.76%
 Time  (mean ± σ):   2.149 s ± 31.165 ms  ┊ GC (mean ± σ):  8.56% ±  1.92%

  2.13 s          Histogram: frequency by time        2.18 s <

 Memory estimate: 264.10 MiB, allocs estimate: 4976193.
```

[222]:
```
@benchmark integrate_ensemble_dagger_2(4, initial_position, initial_velocity)
```

[222]:
```
BenchmarkTools.Trial: 164 samples with 1 evaluation.
 Range (min … max):  24.207 ms … 77.653 ms  ┊ GC (min … max): 0.00% … 33.55%
 Time  (median):     26.291 ms              ┊ GC (median):    0.00%
 Time  (mean ± σ):   30.775 ms ± 10.293 ms  ┊ GC (mean ± σ):  5.14% ±  9.23%

  24.2 ms       Histogram: log(frequency) by time       65.4 ms <

 Memory estimate: 3.21 MiB, allocs estimate: 62962.
```

Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 10.02 GB Mode: Command Ln 1, Col 2 DataDay2024_3.ipynb 0

There is No Free Lunch

DataDay2024_... (auto-p : 4) - JupyterLab

DataDay2024_3.ipynb

```
[220]: @benchmark integrate_ensemble(4, initial_position, initial_velocity)

[220]: BenchmarkTools.Trial: 100 samples with 1 evaluation.
 Range (min … max):  46.908 ms … 58.275 ms  ┊ GC (min … max): 14.97% … 26.60%
 Time  (median):     49.374 ms              ┊ GC (median):    16.31%
 Time  (mean ± σ):   50.150 ms ±  2.361 ms  ┊ GC (mean ± σ):  16.60% ±  2.01%

 46.9 ms         Histogram: frequency by time        56.8 ms <

 Memory estimate: 55.27 MiB, allocs estimate: 861963.
```

```
 Range (min … max):  2.131 s …   2.185 s  ┊ GC (min … max): 6.53% … 10.35%
 Time  (median):     2.131 s              ┊ GC (median):    8.76%
 Time  (mean ± σ):   2.149 s ± 31.165 ms  ┊ GC (mean ± σ):  8.56% ±  1.92%

 2.13 s          Histogram: frequency by time        2.18 s <

 Memory estimate: 264.10 MiB, allocs estimate: 4976193.

[222]: @benchmark integrate_ensemble_dagger_2(4, initial_position, initial_velocity)

[222]: BenchmarkTools.Trial: 164 samples with 1 evaluation.
 Range (min … max):  24.207 ms … 77.653 ms  ┊ GC (min … max): 0.00% … 33.55%
 Time  (median):     26.291 ms              ┊ GC (median):    0.00%
 Time  (mean ± σ):   30.775 ms ± 10.293 ms  ┊ GC (mean ± σ):  5.14% ±  9.23%

 24.2 ms       Histogram: log(frequency) by time      65.4 ms <

 Memory estimate: 3.21 MiB, allocs estimate: 62962.
```

Simple Julia 1.9.4 CUNN-Enabled Idle Mem: 10.02 GB Mode: Command Ln 1, Col 2 DataDay2024_3.ipynb

There is No Free Lunch

DataDay2024_... (auto-p : 4) - JupyterLab

DataDay2024_3.ipynb

```
[220]: @benchmark integrate_ensemble(4, initial_position, initial_velocity)

[220]: BenchmarkTools.Trial: 100 samples with 1 evaluation.
 Range (min … max):  46.908 ms … 58.275 ms  ┊ GC (min … max): 14.97% … 26.60%
 Time  (median):     49.374 ms              ┊ GC (median):    16.31%
 Time  (mean ± σ):   50.150 ms ±  2.361 ms  ┊ GC (mean ± σ):  16.60% ±  2.01%

 46.9 ms         Histogram: frequency by time        56.8 ms <

 Memory estimate: 55.27 MiB, allocs estimate: 861963.
```

```
Range (min … max):  2.131 s …   2.185 s  ┊ GC (min … max): 6.53% … 10.35%
Time  (median):     2.131 s              ┊ GC (median):    8.76%
```

```
[221]: @benchmark integrate_ensemble_dagger_1(4, initial_position, initial_velocity)

[221]: BenchmarkTools.Trial: 3 samples with 1 evaluation.
 Range (min … max):  2.131 s …   2.185 s  ┊ GC (min … max): 6.53% … 10.35%
 Time  (median):     2.131 s              ┊ GC (median):    8.76%
 Time  (mean ± σ):   2.149 s ± 31.165 ms  ┊ GC (mean ± σ):  8.56% ±  1.92%

 2.13 s          Histogram: frequency by time        2.18 s <

 Memory estimate: 264.10 MiB, allocs estimate: 4976193.
```

You're not keeping workers busy enough to amortize `@spawn` costs (+too much data movement)

Simple | Julia 1.9.4 CUNN-Enabled Idle | Mem: 10.02 GB | Mode: Command | Ln 1, Col 2 | DataDay2024_3.ipynb

Keep Workers Busy Enough

Unit of work: simulate several trajectories (or make each trajectory longer).

Don't transfer data unnecessarily: We're only interested in how much trajectories spread out

```
@everywhere function baoab_ensemble_batch(
        energy::Function, force::Function, t, dt, batch_size,
        gamma, kBT, initial_position::T, initial_velocity::T;
        save_frequency = 3, ensemble = (len=10, d=Normal(0, 0.01), n=10),
        kwargs...
    ) where T <: Number

    p_r = rand(ensemble.d, batch_size)
    v_r = rand(ensemble.d, batch_size)

    dtasks = Any[]
    for i in 1:batch_size
        p = initial_position + p_r[i]
        v = initial_velocity + v_r[i]

        dtask = baoab(
            energy, force, ensemble.len*dt, dt,
            gamma, kBT, p + p_r[i], v + v_r[i];
            save_frequency=save_frequency, kwargs...
        )

        push!(dtasks, dtask)
    end

    hi_p = T[]
    lo_p = T[]
    for task in dtasks
        _, p_i, _, _ = task
        push!(hi_p, maximum(p_i))
        push!(lo_p, minimum(p_i))
    end

    return minimum(lo_p), maximum(hi_p)
end
```

Keep Workers Busy Enough

Improved parallelization:

1. Simulate N steps
2. Spawn batches of trajectory simulation tasks
3. `Fetch` only needed data

```
function baoab_ensemble_step_dagger_2(
        energy::Function, force::Function, t, N, dt,
        gamma, kBT, initial_position::T, initial_velocity::T;
        save_frequency = 3, ensemble = (len=10, d=Normal(0, 0.01), n=10),
        kwargs...
    ) where T <: Number

    times, positions, velocities, total_energies = baoab(
        energy, force, N*dt, dt,
        gamma, kBT, initial_position, initial_velocity;
        save_frequency=save_frequency, kwargs...
    )

    dtasks = Dagger.EagerThunk[]
    for i in 1:ensemble.n/ensemble.batch
        dtask = Dagger.@spawn baoab_ensemble_batch(
            energy, force, ensemble.len*dt, dt, ensemble.batch,
            gamma, kBT, initial_position, initial_velocity;
            save_frequency=save_frequency, kwargs...
        )

        push!(dtasks, dtask)
    end

    hi_p = T[]
    lo_p = T[]
    for task in dtasks
        lo_p_i, hi_p_i = fetch(task)
        append!(hi_p, hi_p_i)
        append!(lo_p, lo_p_i)
    end

    return t .+ times, positions, velocities, total_energies, (;p_spread=maximum(hi_p)-minimum(lo_p))
end
```

Busy Workers Amortize Overhead

DataDay2024_... (auto-p : 4) - JupyterLab

DataDay2024_3.ipynb

```
[220]: @benchmark integrate_ensemble(4, initial_position, initial_velocity)

[220]: BenchmarkTools.Trial: 100 samples with 1 evaluation.
 Range (min … max):  46.908 ms … 58.275 ms  ┊ GC (min … max): 14.97% … 26.60%
 Time  (median):     49.374 ms              ┊ GC (median):    16.31%
 Time  (mean ± σ):   50.150 ms ±  2.361 ms  ┊ GC (mean ± σ):  16.60% ±  2.01%

 46.9 ms         Histogram: frequency by time        56.8 ms <

 Memory estimate: 55.27 MiB, allocs estimate: 861963.
```

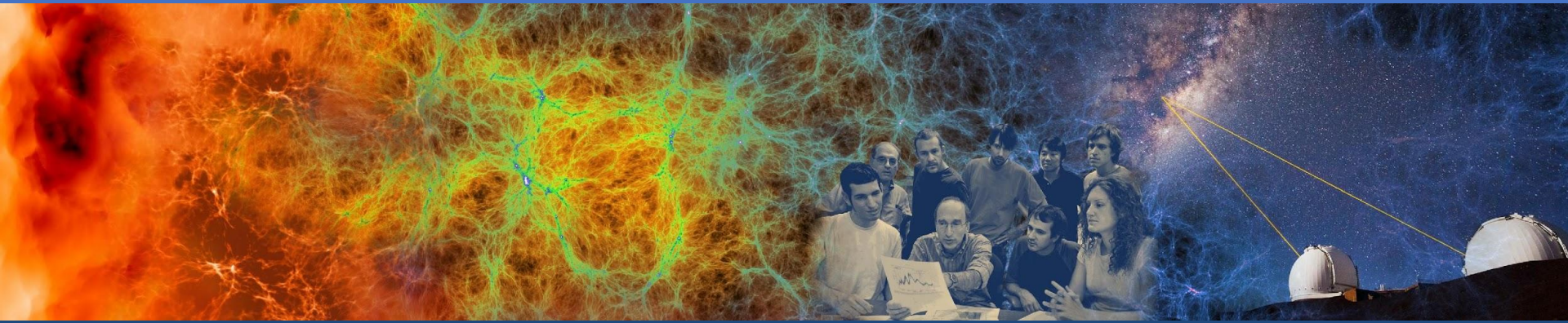
```
Range (min … max):  2.131 s …   2.185 s  ┊ GC (min … max): 6.53% … 10.35%
Time  (median):     2.131 s              ┊ GC (median):    8.76%
```

```
[222]: @benchmark integrate_ensemble_dagger_2(4, initial_position, initial_velocity)

[222]: BenchmarkTools.Trial: 164 samples with 1 evaluation.
 Range (min … max):  24.207 ms … 77.653 ms  ┊ GC (min … max): 0.00% … 33.55%
 Time  (median):     26.291 ms              ┊ GC (median):    0.00%
 Time  (mean ± σ):   30.775 ms ± 10.293 ms  ┊ GC (mean ± σ):  5.14% ±  9.23%

 24.2 ms       Histogram: log(frequency) by time      65.4 ms <

 Memory estimate: 3.21 MiB, allocs estimate: 62962.
```

Also note: you can scale to many nodes

Simple Julia 1.9.4 CUNN-Enabled Idle Mem: 10.02 GB Mode: Command Ln 1, Col 2 DataDay2024_3.ipynb

Conclusions



Conclusion

- Julia provides a rich ecosystem to build performant distributed applications on HPC systems
  - Saw examples of Jupyter (`IJulia.jl`); Beginnings of sophisticated multi-node workflows (`Distributed.jl`, `Dagger.jl`); Programming GPUs (`CUDA.jl`); and AI (`Flux.jl`)
- Modern high-productivity design
- HPC vendor aware. Built on top of LLVM, with vendor backends (`CUDA.jl`, `AMDGPU.jl`, `oneAPI.jl`, etc)
- Provides interfaces to examine and manipulate what you're doing (including LLVM IR)

Noteworthy Julia Packages (for HPC)

- **JuliaIO**: https://github.com/JuliaIO
  JuliaData: https://github.com/JuliaData
  Collects many Julia packages around I/O and Data
- **JuliaParallel**: https://github.com/JuliaParallel
  Collects many Julia packages around distributed and parallel computing
- **JuliaGPU**: https://github.com/JuliaGPU
  Collects many Julia packages used for GPU computing

Noteworthy I/O Packages

- **Pidfile.jl**: Provides the linux/unix pidfile mechanism to hold mutex'es – useful for locking files
- **HDF5.jl**: HDF5-file support
- **Zarr.jl**: Julia Zarr (N-D array compressed data) support
- **JLD.jl** / **JLD2.jl**: Julia-native serialization support
- **Tables.jl** / **DTables.jl** / **DistributedArrays.jl**: arrays and tables build on distributed / **CSV.jl**: Tabular data support
- **JuliaDB.jl**: A distributed database for tables (implemented in pure Julia)

Noteworthy REST and Web Frameworks

- **HTTP.jl**: Send and receive HTTP requests
- **Mux.jl** / **Oxygen.jl**: Routing middleware for HTTP requests – Oxygen is newer and makes multithreading easier (considered an all-Julia replacement for FastAPI)
- **Genie.jl**: Fully-fledged web development framework (Julia's answer to Flask)

Noteworthy HPC Packages

"Traditional" HPC support:
(https://github.com/JuliaParallel)

- **MPI.jl**: no explanation needed (it is CUDA/ROCM-aware)
- **ClusterManagers.jl**: manager HPC resources on the fly (also note **SlurmClusterManager.jl** and **MPIClusterManagers.jl** for HPC clusters)
- **ImplicitGlobalGrid.jl** / **MPIArrays.jl**: implement a global address space (using the Array interface) built on MPI.jl

Noteworthy HPC Packages

Tasking (producer-consumer) style HPC support: (https://github.com/JuliaParallel)

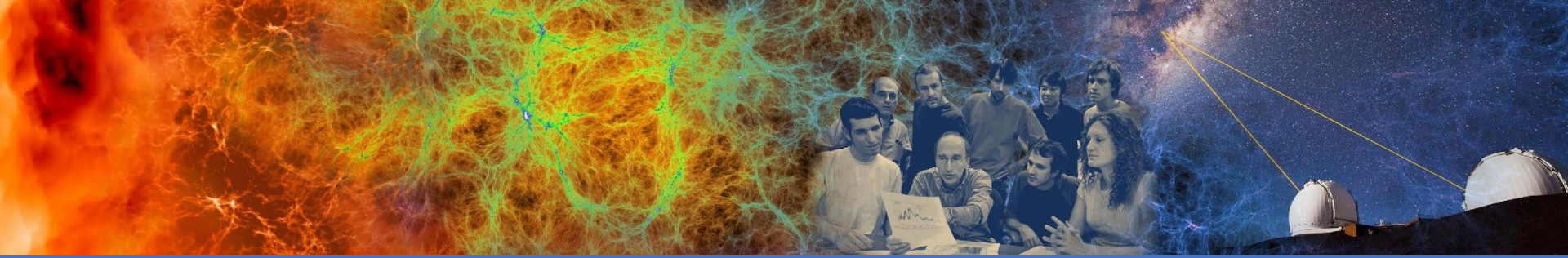
- **Distributed.jl** / **Dagger.jl**: task-based parallelism (like Dask and Ray)
- **DTables.jl** / **DistributedArrays.jl**: arrays and tables build on distributed

ML support: **Flux.jl** (like pytorch, but different)

Noteworthy HPC Packages

GPU Support:
(https://github.com/JuliaGPU)

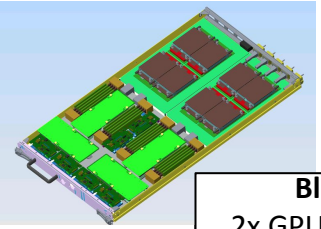
- **CUDA.jl** / **AMDGPU.jl** / **oneAPI.jl**: low-level GPU support (expose GPU Array interface + helper functions to manage GPU resources)
- **KernelAbstractions.jl**: lets you write portable code by writing portable kernels (a bit “like” Kokkos)
- + Many Many more



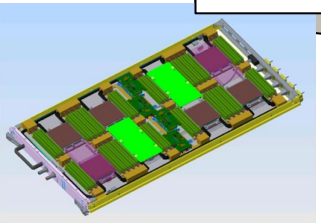
Extra Slides

Perlmutter system configuration

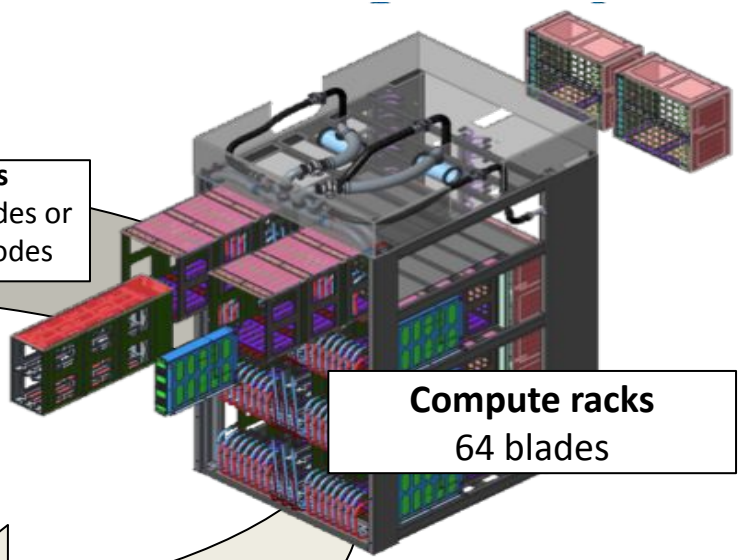
NVIDIA "Ampere" GPU Nodes
4x GPU + 1x CPU
40 GiB HBM + 256 GiB DDR
4x 200G "Slingshot" NICs



AMD "Milan" CPU Node
2x CPUs
> 256 GiB DDR4
1x 200G "Slingshot" NIC



Blades
2x GPU nodes or
4x CPU nodes



Compute racks
64 blades

Centers of Excellence
Network
Storage
App. Readiness
System SW

→

Perlmutter system
GPU racks
CPU racks
~6 MW

CUDA.jl provides detailed profiling interface

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[14]:
```
CUDA.@profile carr1 * carr2
```

[14]:
```
Profiler ran for 2.23 ms, capturing 23 events.

Host-side activity: calling CUDA APIs took 1.96 ms (87.81% of the trace)
```

Time (%)	Time	Calls	Avg time	Min time	Max time	Name
29.29%	653.51 µs	1	653.51 µs	653.51 µs	653.51 µs	**cuMemAllo**
18.15%	404.83 µs	1	404.83 µs	404.83 µs	404.83 µs	**cudaEvent**
1.12%	25.03 µs	1	25.03 µs	25.03 µs	25.03 µs	cudaLaunc
0.82%	18.36 µs	1	18.36 µs	18.36 µs	18.36 µs	cudaMemse
0.44%	9.78 µs	2	4.89 µs	953.67 ns	8.82 µs	cudaOccup
0.28%	6.2 µs	3	2.07 µs	476.84 ns	4.77 µs	cudaStrea
0.26%	5.72 µs	1	5.72 µs	5.72 µs	5.72 µs	cudaEvent
0.00%	0.0 ns	1	0.0 ns	0.0 ns	0.0 ns	cudaGetLa

1 column omitted

```
Device-side activity: GPU was busy for 160.22 µs (7.18% of the trace)
```

Time (%)	Time	Calls	Avg time	Min time	Max time	Name
7.09%	158.07 µs	1	158.07 µs	158.07 µs	158.07 µs	**ampere_sg**
0.10%	2.15 µs	1	2.15 µs	2.15 µs	2.15 µs	[set devi

1 column omitted

CUDA.jl is compatible with Structs

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git SC23 Julia 1.9.3

[17]:
```julia
struct Point{T}
    x :: T
    y :: T
end
```

[18]:
```julia
arr = Point.(rand(100), rand(100))
carr = cu(arr)
```

[18]:
```
100-element CuArray{Point{Float64}, 1, CUDA.Mem.DeviceBuffer}:
 Point{Float64}(0.8010859437783524, 0.8642616255324465)
 Point{Float64}(0.17750804805255238, 0.15240658172450294)
 Point{Float64}(0.4940416892271239, 0.604324864671999)
 Point{Float64}(0.8190761447346417, 0.24524461825461796)
 Point{Float64}(0.8136328436633238, 0.703444856961141)
 Point{Float64}(0.9014731786301712, 0.6976978604609408)
 Point{Float64}(0.4734214731060712, 0.546675832272095)
 Point{Float64}(0.9030369868126105, 0.5340128051286633)
 Point{Float64}(0.5317337783755941, 0.8033039913403202)
 Point{Float64}(0.6763381060145333, 0.6507417892793194)
 Point{Float64}(0.7484751505993789, 0.748566736019653)
 Point{Float64}(0.599307695857283, 0.7081606826204108)
 Point{Float64}(0.4096041416396996, 0.17860717723966113)
 ⋮
 Point{Float64}(0.8113003140580276, 0.8828241515817755)
 Point{Float64}(0.13536512021293456, 0.46910612940518404)
 Point{Float64}(0.9009776673421677, 0.047089614167290184)
 Point{Float64}(0.7299575952344476, 0.10684973792700014)
 Point{Float64}(0.0563314509491073, 0.6472313211625557)
 Point{Float64}(0.18604391157970612, 0.041993134446333125)
 Point{Float64}(0.9627585143646942, 0.38452121311513965)
```

CUDA.jl is compatible with Structs

Julia converts struct to cuda-compatible type

SC23.ipynb (auto-L) - JupyterLab

SC23.ipynb

File Edit View Run Kernel Git Tabs Settings Help

[17]:
```
struct Point{T}
    x :: T
    y :: T
end
```

[18]:
```
arr = Point.(rand(100), rand(100))
carr = cu(arr)
```

```
Point{Float64}(0.801085943778524, 0.8642616255324465)
Point{Float64}(0.17750804805255238, 0.15240658172450294)
Point{Float64}(0.4940416892271239, 0.604324864671999)
Point{Float64}(0.8190761447346417, 0.24524461825461796)
Point{Float64}(0.8136328436633238, 0.703444856961141)
Point{Float64}(0.9014731786301712, 0.6976978604609408)
Point{Float64}(0.4734214731060712, 0.546675832272095)
Point{Float64}(0.9030369868126105, 0.5340128051286633)
Point{Float64}(0.5317337783755941, 0.8033039913403202)
Point{Float64}(0.6763381060145333, 0.6507417892793194)
Point{Float64}(0.7484751505993789, 0.748566736019653)
Point{Float64}(0.599307695857283, 0.7081606826204108)
Point{Float64}(0.4096041416396996, 0.17860717723966113)
⋮
Point{Float64}(0.8113003140580276, 0.8828241515817755)
Point{Float64}(0.13536512021293456, 0.46910612940518404)
Point{Float64}(0.9009776673421677, 0.047089614167290184)
Point{Float64}(0.7299575952344476, 0.10684973792700014)
Point{Float64}(0.0563314509491073, 0.6472313211625557)
Point{Float64}(0.18604391157970612, 0.041993134446333125)
Point{Float64}(0.9627585143646942, 0.38452121311513965)
```

DataDay2024.... (auto-p : 4) - JupyterLab

DataDay2024.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Code git Julia 1.9.4 CUNN-Enabled

[1]:
```
using Distributed
```

[2]:
```
addprocs(10)
```

[2]:
```
10-element Vector{Int64}:
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
```

[3]:
```
@everywhere using Dagger, Random, Distributions, StatsBase, DataFrames
```

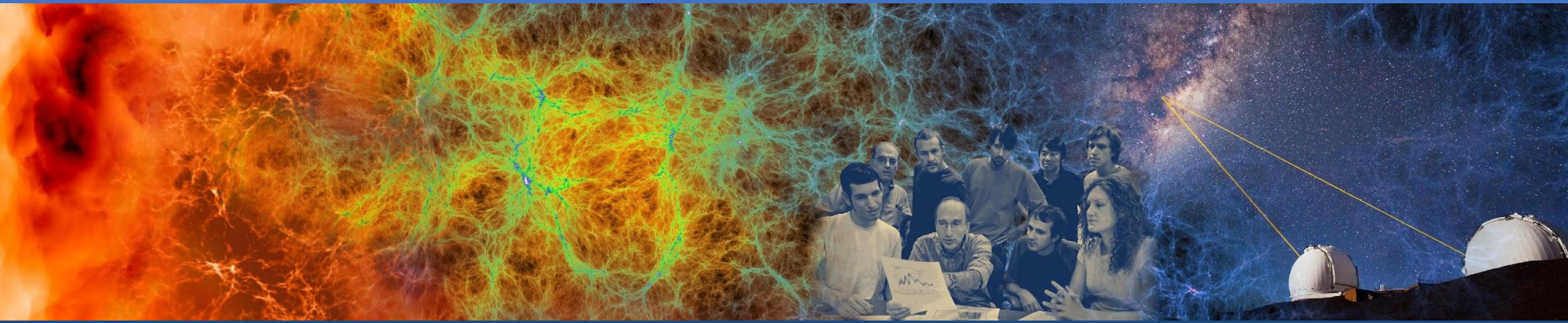
```
WARNING: using Dagger.In in module Main conflicts with an existing identifier.
WARNING: using Dagger.Out in module Main conflicts with an existing identifier.
```

[10]:
```
th = Dagger.@spawn sleep(10)
while true
    st = isready(th)
    println("isready = $(st)")
    if ! st
        sleep(1)
    else
        break
    end
end
```

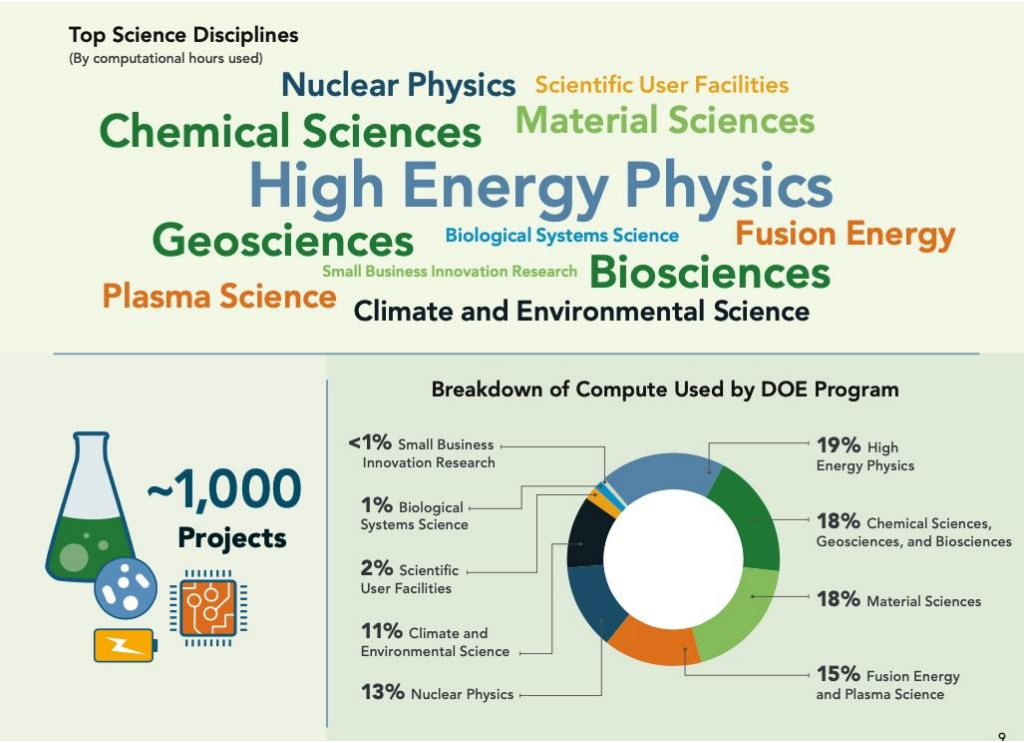
```
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
isready = false
```

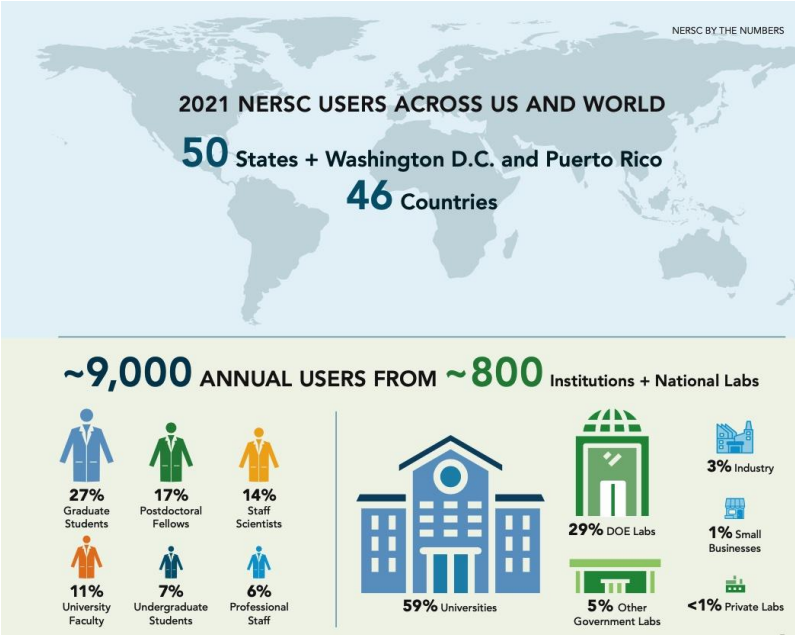
Simple 2 6 Julia 1.9.4 CUNN-Enabled Idle Mem: 9.82 GB Mode: Command Ln 1, Col 1 DataDay2024.ipynb 0

Why does NERSC care about Julia?



NERSC is the mission HPC and data facility for the U.S Department of Energy Office of Science

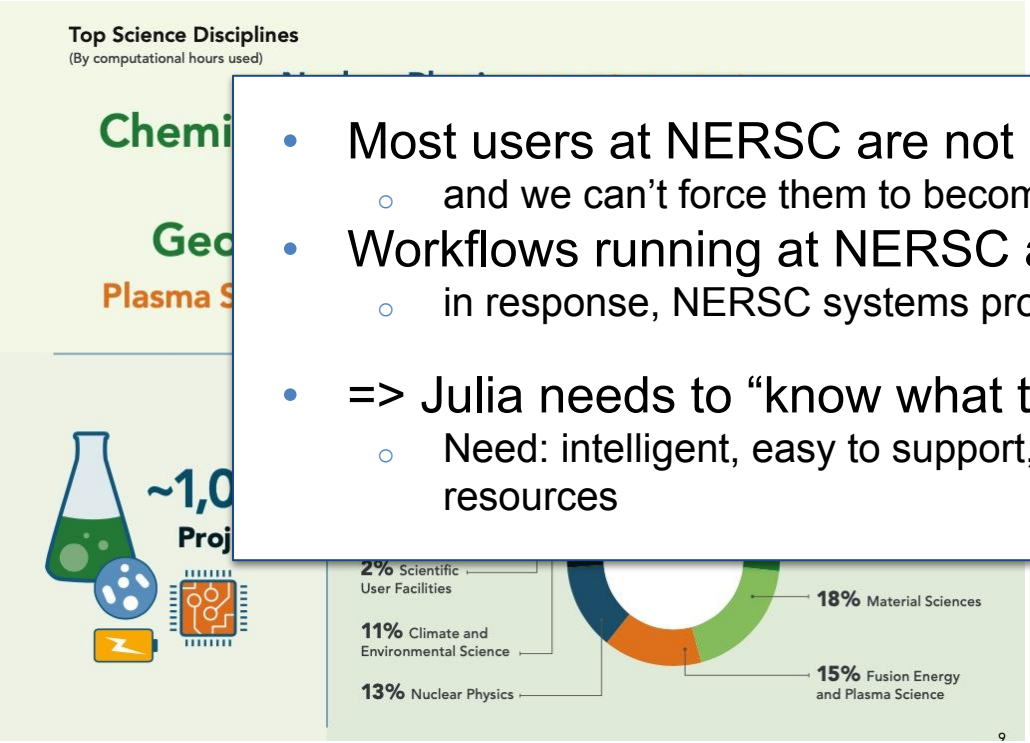


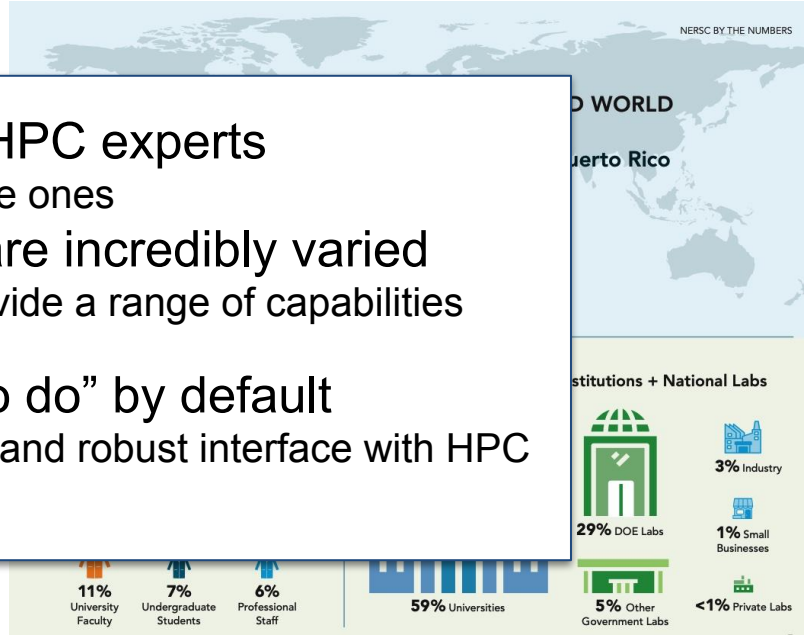


>2,000

Scientific Journal Articles per Year

NERSC is the mission HPC and data facility for the U.S Department of Energy Office of Science





- Most users at NERSC are not HPC experts
  - and we can't force them to become ones
- Workflows running at NERSC are incredibly varied
  - in response, NERSC systems provide a range of capabilities
- => Julia needs to "know what to do" by default
  - Need: intelligent, easy to support, and robust interface with HPC resources

>2,000

Scientific Journal Articles per Year