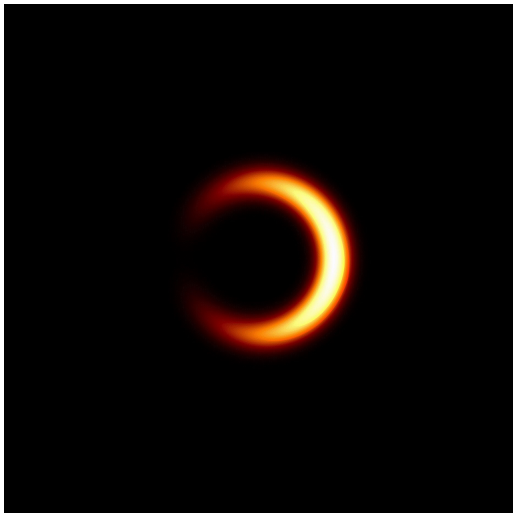
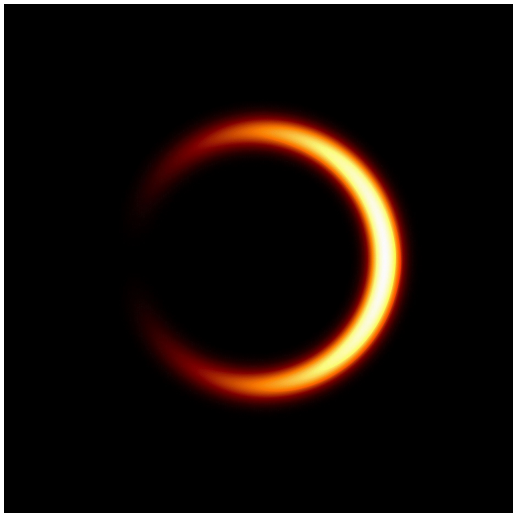
Towards Accelerated Inference for VLBI Analysis

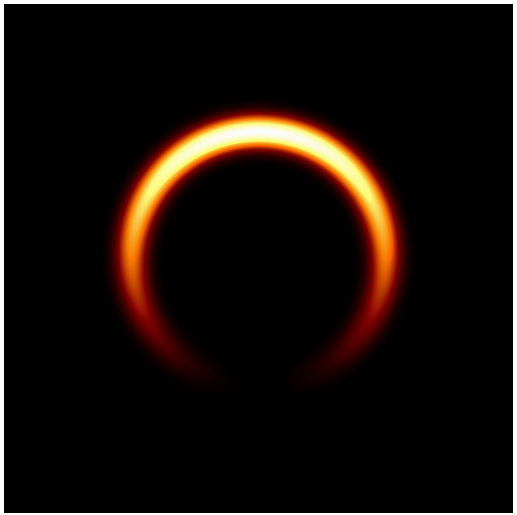
on GPUs using JAX and Julia

Peter Brin
pbrin@cfa.harvard.edu

Geometric Model Fitting







What is JAX?

JAX is Autograd and XLA, brought together for high-performance machine learning research. Open source project, developed by Google.

It provides composable transformations of Python+NumPy programs: differentiate, vectorize, parallelize, Just-In-Time compile to GPU/TPU, and more.

Same familiar numpy API

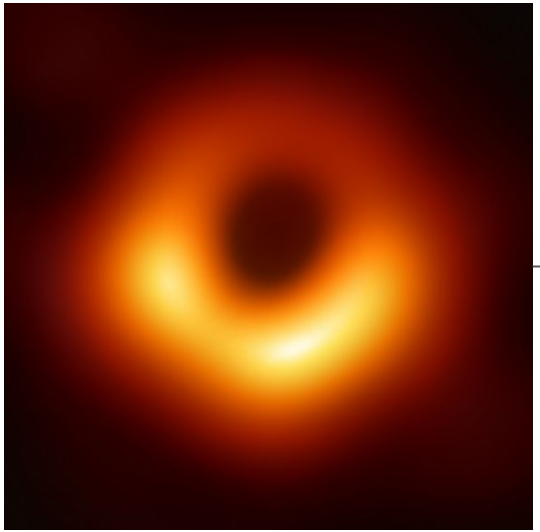
Numpy

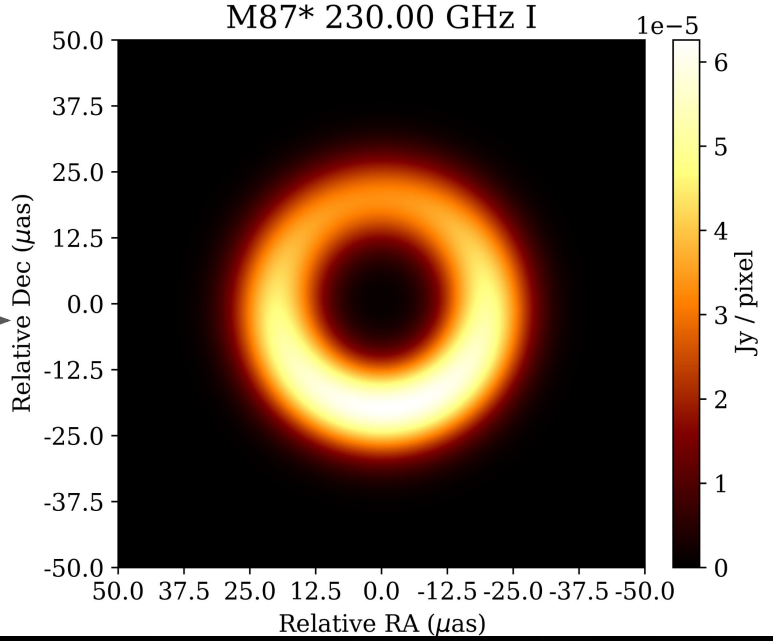
```
def circ_gauss_sample_uv(u, v):
    val = (params['F0']
           * np.exp(-np.pi**2/(4.*np.log(2.)) * (u**2 + v**2) * params['FWHM']**2)
           * np.exp(1j * 2.0 * np.pi * (u * params['x0'] + v * params['y0'])))

    return val
```

JAX code for CPUs (and GPUs!)

```
def circ_gauss_sample_uv(u, v):
    val = (params['F0']
           * jnp.exp(-jnp.pi**2/(4.*jnp.log(2.)) * (u**2 + v**2) * params['FWHM']**2)
           * jnp.exp(1j * 2.0 * jnp.pi * (u * params['x0'] + v * params['y0'])))

    return val
```

Model fitting M87* 2017 April 11 Stokes I visibility amplitudes and closure phases to an m-ring model (m=1)





	mean	std	median	5.0%	95.0%	n_eff	r_hat
F0	0.59	0.00	0.59	0.59	0.59	998.75	1.00
abs(b1)	0.16	0.00	0.16	0.16	0.16	569.52	1.00
angle(b1)	3.03	0.01	3.03	3.02	3.04	587.46	1.00
d	41.38	0.02	41.38	41.35	41.40	2288.56	1.00
w	13.79	0.02	13.79	13.76	13.81	662.17	1.00

Running time comparison of CPU vs GPU model fitting

VM: eht-gpu-test

CPU: Intel Xeon @ 2.2 GHz 2 cores

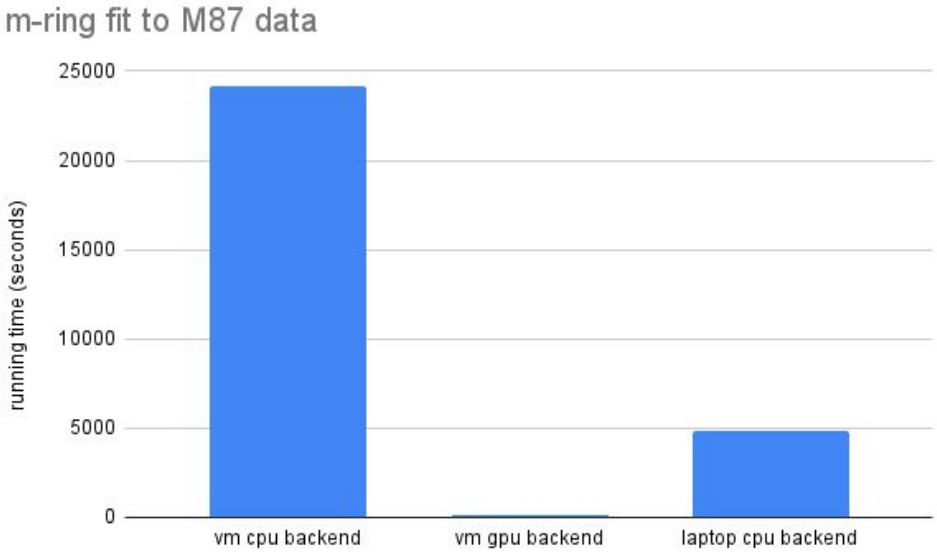
GPU: Nvidia Tesla P4

CPU model fitting time: 06:42:50

GPU model fitting time: 00:02:23

laptop model fitting time: 01:21:11

~34x speedup



Comrade.jl

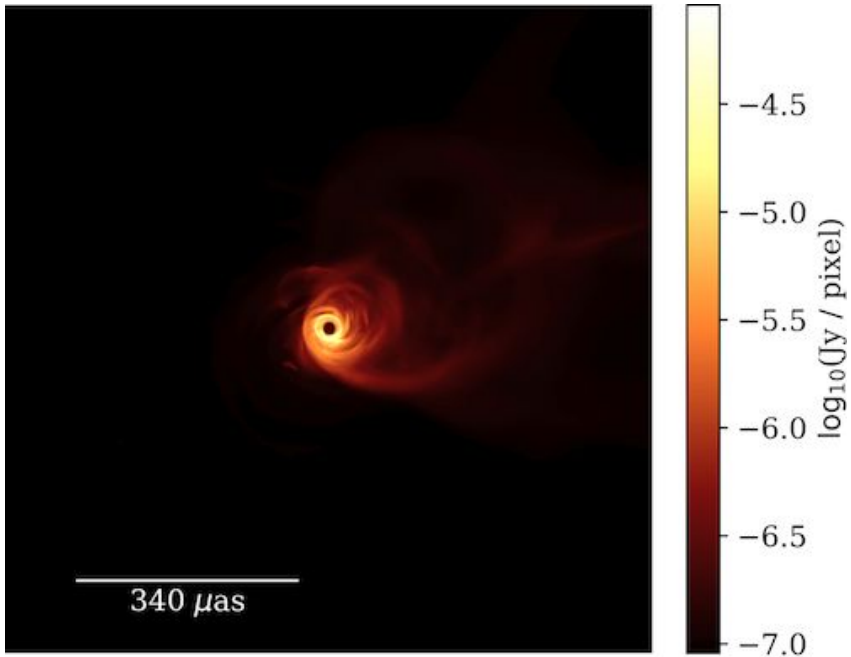
Comrade is a Bayesian differentiable modular modeling framework for use with very long baseline interferometry

Designed with performance in mind:

	Comrade (micro sec)	eht-imaging (micro sec)	Themis (micro sec)
posterior eval (min)	31	445	55
posterior eval (mean)	36	476	60
grad posterior eval (min)	105 (ForwardDiff)	1898	1809
grad posterior eval (mean)	119 (ForwardDiff)	1971	1866

source: Comrade.jl documentation

Analyzing ngEHT Challenge 1 data with Comrade.jl



MAD GRMHD frame from a rapid spinning black hole a=0.94 with electron thermodynamics from reconnection heating (see Mizuno et al. 2021 for details https://ui.adsabs.harvard.edu/abs/2021arXiv210609272M/abstract). The GRMHD simulation was performed with the BHAC code (Porth et al. 2017) using three levels of AMR in logarithm Kerr-Schild coordinates. The numerical grid covers 384x192x192 cells in radial, azimuthal and theta direction and is extending up to 2500 M in radial direction. The mass accretion rate and MAD parameter (see Tchekhovskoy 2012) is monitored and after obtaining a steady state we perform the general relativistic radiative transfer calculations (GRRT).

Parameters

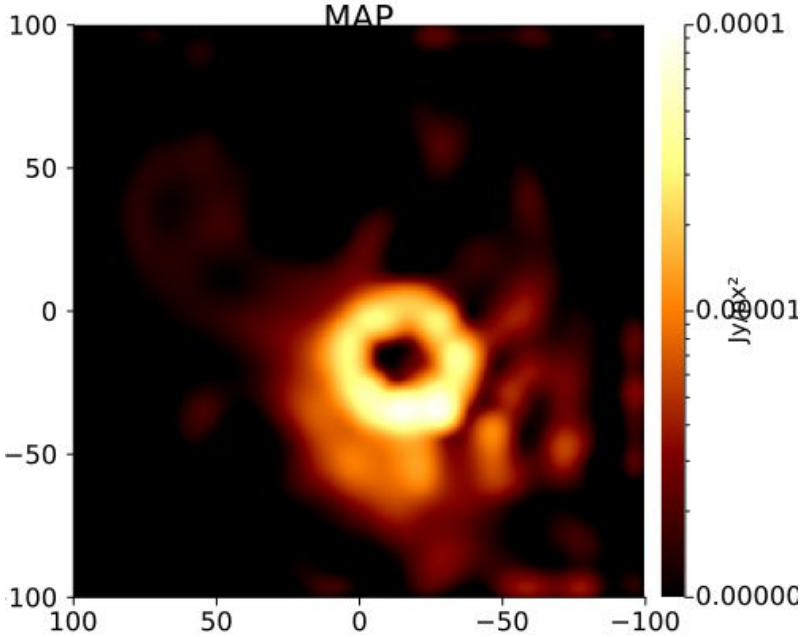
M87_eht2022_230_thnoise.uvfits

Fitting visibility amplitudes & closure phases

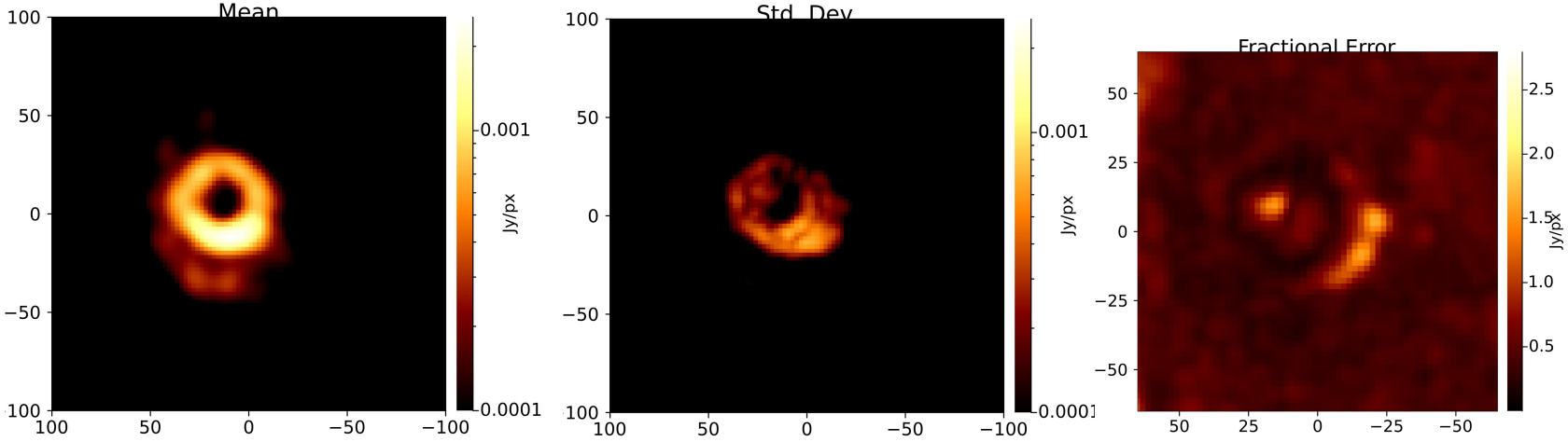
200 μas fov, 32x32 image size

5000 HMC steps for sampling from the posterior

Running time: 3:59:27



Sampling - Results



NVIDIA GPU computing on Julia

Porting Comrade.jl to execute on the GPU

Stripped version of Bayesian imaging core of Comrade (credit: Paul Tiede)

Benchmark: Likelihood computation using random data

Simplest version of GPU execution: uses Julia's multiple dispatch

NVIDIA GPU computing on Julia

CPU version:

```
nvis = data_size
u = randn(Float32, nvis)
v = randn(Float32, nvis)

sigma = fill(0.01, nvis)
vis = complex.(exp.(-2π^2. *(u.^2 .+ v.^2))) .+ sigma.*randn(nvis)

post = TestPost(u, v, vis, sigma, 10.0, im_size)
data = rand(Float32, im_size, im_size)
benchmark_results[i, j] = @benchmark $(post)($data)
```

GPU version:

```
nvis = data_size
u = randn(Float32, nvis)
v = randn(Float32, nvis)

# Make some synthetic data
# Also should move to gpu array
sigma = fill(0.01, nvis)
vis = complex.(exp.(-2π^2. *(u.^2 .+ v.^2))) .+ sigma.*randn(nvis)

post = TestPost(u, v, cu(vis), cu(sigma), 10.0, im_size)
data = rand(Float32, im_size, im_size)
data = CuArray(data)
benchmark_results[i, j] = @benchmark $(post)($data)
```

NVIDIA GPU computing on Julia

Code where expensive computation takes place is unchanged:

```
function (m::TestPost)(I)
    vmodel = dft(m.dft_mat, I)
    # Now let's do a likelihood
    return -sum(abs2, (vmodel .- m.vis)./m.sigma)/2
end
```

```
function dft(dft_matrix::AbstractMatrix, img::AbstractMatrix)
    dft_matrix*reshape(img, :)
end
```

NVIDIA GPU computing on Julia

Porting Comrade.jl to execute on the GPU

Stripped version of Bayesian imaging core of Comrade (credit: Paul Tiede)

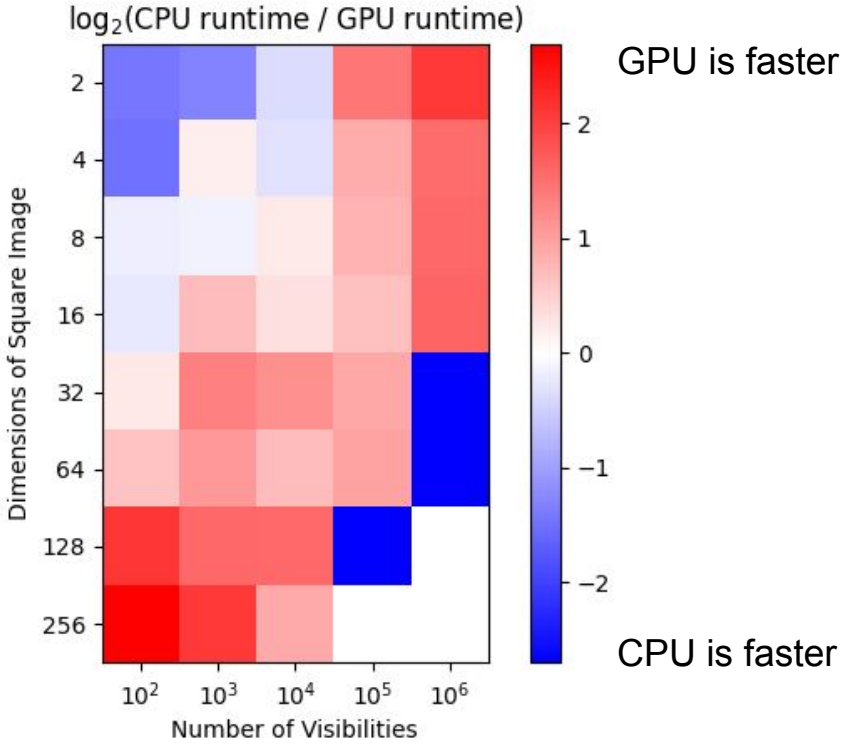
Simplest version of GPU execution: uses Julia's multiple dispatch

NEW! Give CPU version a fairer chance:

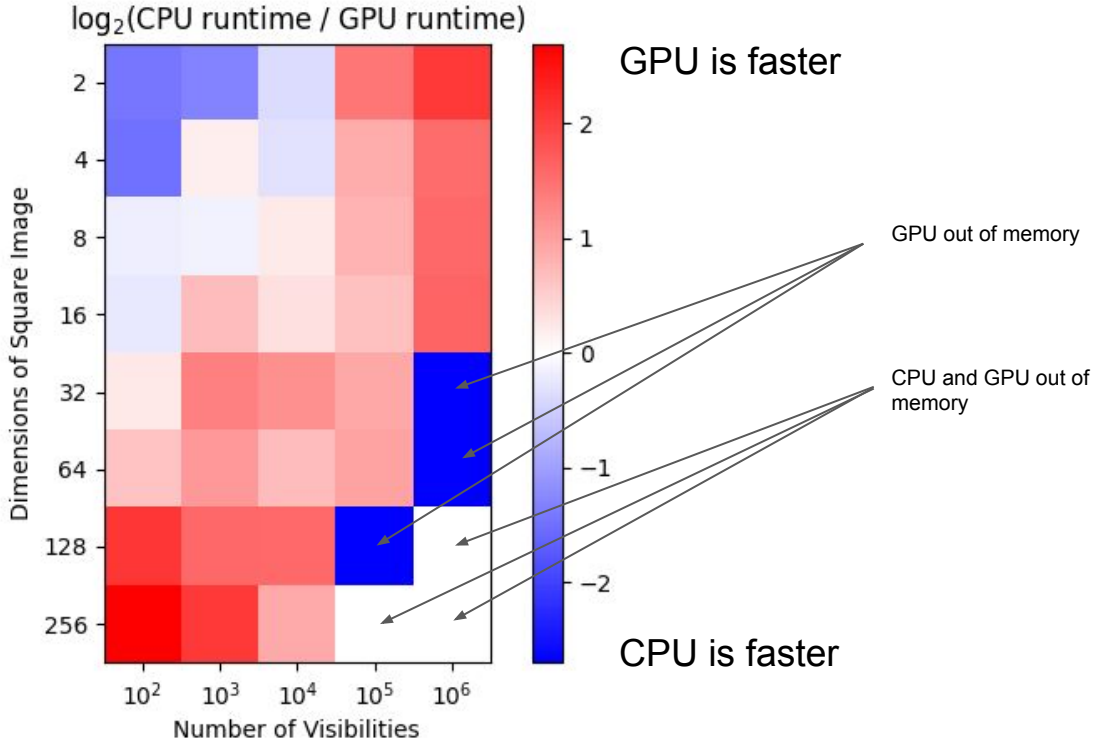
+ Upgrade from 2 cores 4 threads -> 8 cores 16 threads

+ (limited) multi-threading support via Polyester.jl

GPU computing on Julia - Benchmarks



GPU computing on Julia - Benchmarks



Next steps for Comrade.jl+CUDA

Currently bottlenecked by Zygote.jl, an autodiff library for Julia: issues with GPU differentiation of complex arguments; cannot differentiate through broadcasting

Possible solutions: rewrite Comrade, rewrite Zygote, wait for alternative autodiff library to mature?