Relativistic Ray Tracing in Julia

Ryan McKinnon

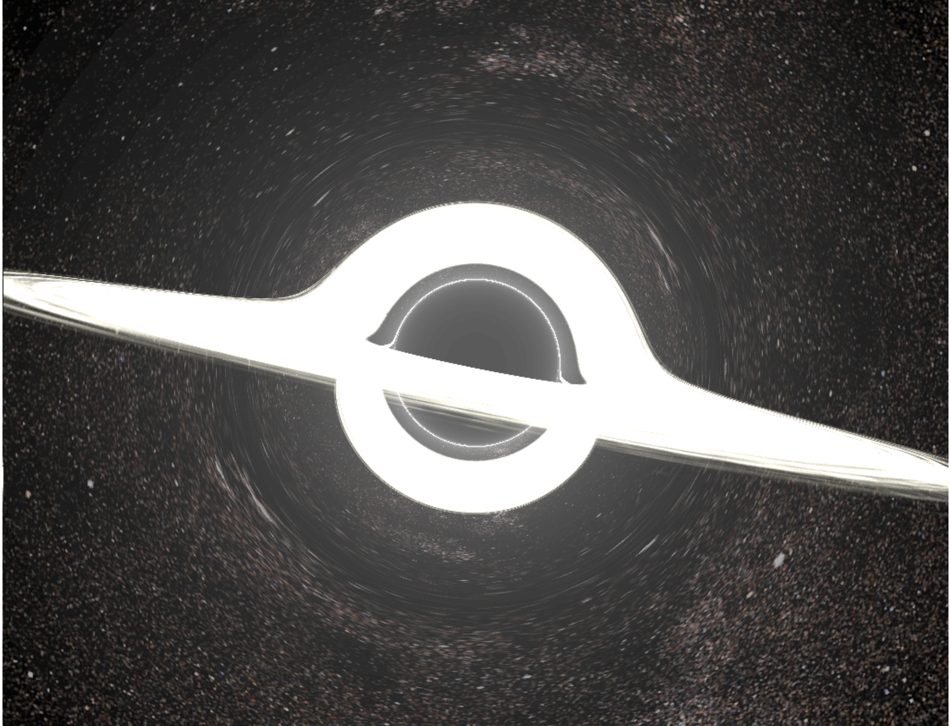
November 30, 2015

Introduction

Ray tracing is used in many scientific fields to visualize 3D data

Relativistic ray tracing is needed to model a black hole's warped spacetime under general relativity

This project implements and optimizes a relativistic ray tracer in Julia based on the STARLESS Python code



Methods

SharedArrays for shared memory array access

ConfParser.jl for reading configuration files at runtime

Using Harvard's Odyssey supercomputer with up to 64 cores per node for a test image of size 1024 by 786; could imagine rendering much bigger scenes

Timing comparisons below include parallelization overhead (e.g. assignment of regions of the image to different threads) and ray tracing time, but ignore loading of configuration files, etc.

Ease of Julia Parallelization

```
colour_shared = SharedArray(Float64, (numPixels, 3))
@sync begin
    for (i, wpid) in enumerate(workers())
        @async begin
            remotecall_wait(wpid, raytrace_func, i,
               schedules[i], colour_shared)
        end
    end
end
```

Much simpler than in Python, where the multiprocessing module requires you to use a separate array type that you must manually convert to a numpy array!

Optimization Efforts

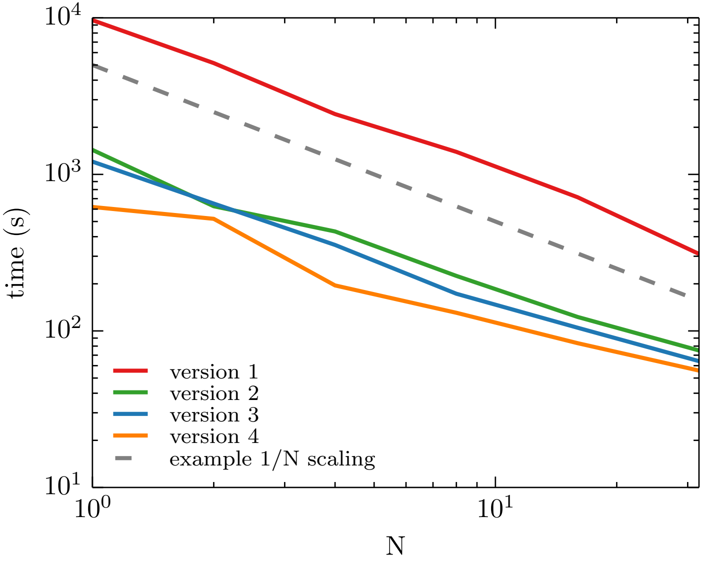
Ray tracing involves lots of vector math and norm computations; use of `sumabs2!` to compute the norm squared improved performance by **5.8x**

Another common routine was the Runge-Kutta integrator, which was rewritten inline to eliminate some temporary arrays, speedup of **1.2x**

Modifying the `CHUNKSIZE` parameter controlling the number of pixels worked on simultaneously by a thread also helped, speedup of **1.3x**

In total, improved initial Julia ray tracer by **8.6x**

Parallelization Efficiency



Other Opportunities for Improvement

MPI or distributed array programming is a natural extension

Perhaps an even more efficient way to compute norms?

More advanced integration routines that adaptively determine number of integration steps (rather than some fixed number)

Summary

Implemented a relativistic ray tracer in Julia based on STARLESS Python code

Parallelization (threading, SharedArrays) and optimization tweaks significantly improved performance

Parallelization easier to implement in Julia than Python!