

Parallelizing DIPY model fits with Ray

Ariel Rokem

Asa Gilmore

1 Under construction

2 Abstract

3 Introduction

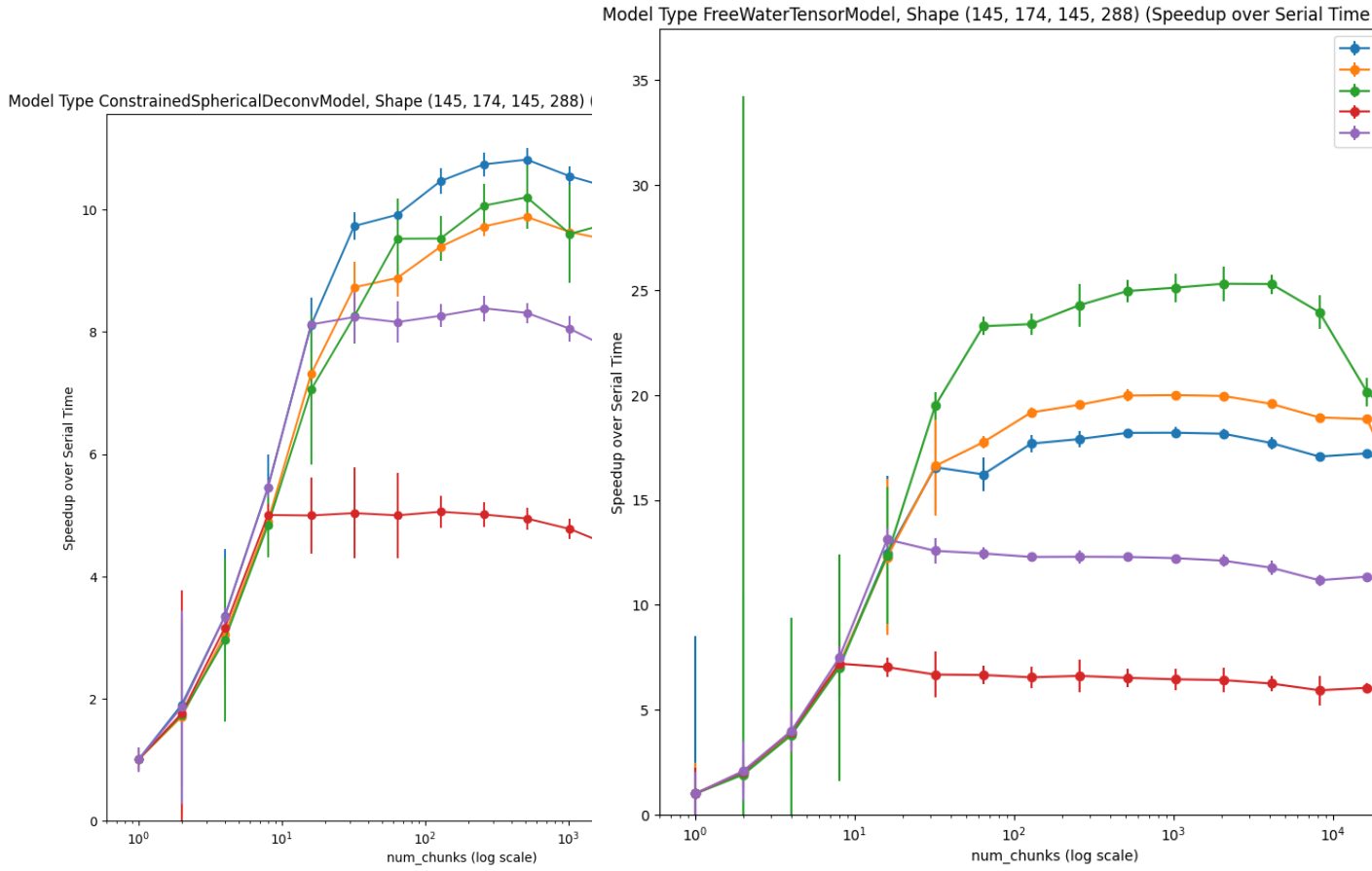
4 Methods

We ran both a constrained spherical deconvolution model and a free water diffusion tensor model through dipy on a subject from the human connectome project (add more about hcp). We created a docker image to encapsulate the test and allow for easy reproducibility of the tests. The testing program computes each model 5 times for each set of unique parameters. We then iterate across chunk sizes exponentially, from 1-15, where the number of chunks is 2^x (explain better). We ran the tests with the following arguments on docker instances with cpu counts, 8, 16, 32, 48, and 72:

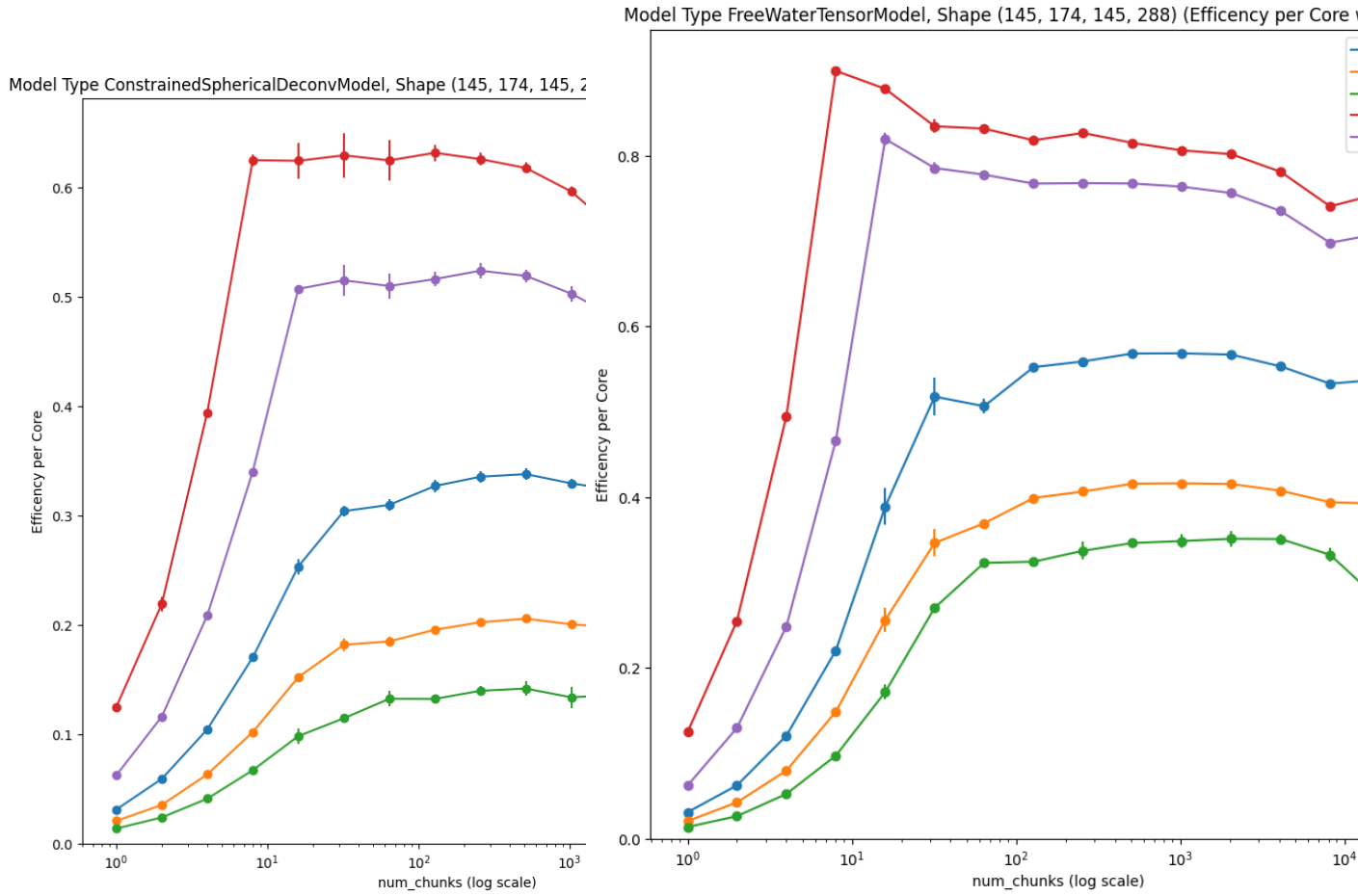
```
--models csdm fwddtm --min_chunks 1 --max_chunks 15 --num_runs 5
```

5 Results

Parallelization with `ray` provided considerable speedups over serial execution for both constrained spherical deconvolution models and free water models. We saw a much greater speedup for the free water model, which is possibly explained by the fact that it is much more computationally expensive per voxel. This would mean that the overhead from parallelizing the model would have a smaller effect on the runtime. Interestingly 48 and 72 core instances performed slightly worse than the 32 core instance on the csdm model, which may indicate that there is some increased overhead for each core, separate from the overhead for each task sent to ray.



Efficiency decreases as a function of number of CPUs, but is still rather high in many configurations. Efficiency is also considerably higher for the free water tensor model, which is consistent with our expectations given that it is more computationally expensive per voxel and therefore ray overhead would have less effect. The high efficiency of 8 core machines suggests that the most cost effective configuration for processing may be relatively cheap low core machines.



Ray tends to spill a large amount of data to disk and does not clean up afterwards. This can quickly become problematic when running multiple consecutive models. Withing just an hour or two of running ray could easily spill over 500gb to disk. We have implemented a fix for this within our model as follows:

```
if engine == "ray":
    if not has_ray:
        raise ray()

    if clean_spill:
        tmp_dir = tempfile.TemporaryDirectory()

        if not ray.is_initialized():
            ray.init(_system_config={
```

```

        "object_spilling_config": json.dumps(
            {"type": "filesystem", "params": {"directory_path":
                tmp_dir.name}},
        )
    },)

    func = ray.remote(func)
    results = ray.get([func.remote(ii, *func_args, **func_kwargs)
        for ii in in_list])

    if clean_spill:
        shutil.rmtree(tmp_dir.name)

```

6 Discussion

6.1 Acknowledgments

This work was funded through NIH grant EB027585 (PI: Eleftherios Garyfallidis) and a grant from the Chan Zuckerberg Initiative Essential Open Source Software program (PI: Serge Koudoro).