master's project notes

Contents

1 Fast poisson logpmf

1.1 The implementations

The poisson logpmf is something aaaa

We have four different implementations performing (poisson logpmf?) ... :

The first implementation we consider is the implementation provided by the SciPy [1]. This implementation runs on the CPU.

```
from scipy.stats import poisson

def poisson_logpmf(k, r):
    return poisson.logpmf(k, r)
```

The second implementation is an implementation created by I&K using NumPy [2] and SciPy. This implementation also runs on the CPU, but is nearly twice as fast as the one provided by the SciPy library when the k and r matrices have 40 million elements each.

```
import numpy as np
import scipy

def poisson_logpmf(k, r):
    return k * np.log(r) - r - scipy.special.gammaln(k+1)
```

The third implementation piggybacks off of the previous NumPy implementation, but is ported to CuPy [3]. This implementation achieves large speed increases over the NumPy equivalent for large matrices.

```
import cupy as cp

def poisson_logpmf(k, r):
    return k * cp.log(r) - r - cupyx.scipy.special.gammaln(k+1)
```

The CUDA implementation (just the kernel).

```
__global__ void poisson_logpmf_kernel(const int *k, const float *r,
    float *out)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  out[i] = k[i] * logf(r[i]) - r[i] - lgammaf(k[i]+1);
}
```

Both the CuPy and the CUDA implementations allow for all, some or none of the input matrices to be located in the GPU's global memory. Any input not located in GPU memory will be copied to the device in order for the computation to be performed by the GPU. Only when both inputs are located on the host will the returned results also be located on the host, meaning that the input matrices will be copied from the host to the device for computation, before the result is copied from the device back to host. This introduces some extra variations for the CuPy and CUDA implementations, resulting in the following set of variations for both of them:

- When both of the input matrices are located on the host, and the returning data is located on the host: hh2h
- When one of the input matrices are located on the host and the other on the device, and the returning data is located on the device: dh2d
- When both of the input matrices are located on the device, and the returning data is located on the device: dd2d

We omit the variation where both the inputs are located on the host and the output is located on the device.

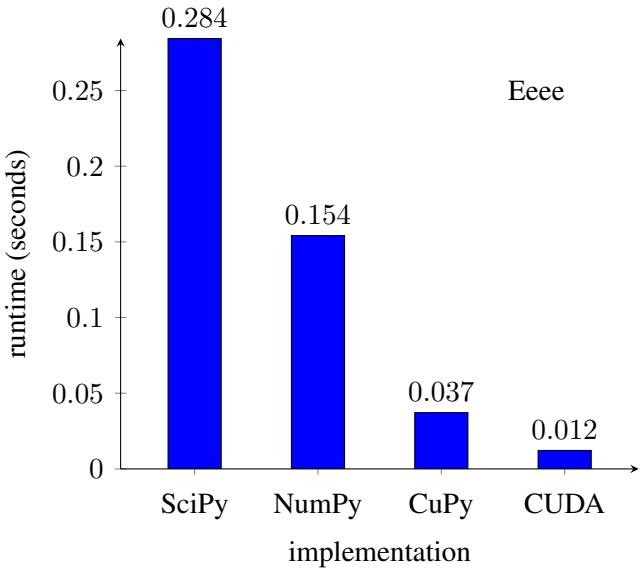

Figure 1: Runtime in seconds averaged over 100 function calls where len(k) = len(r) = 40,000,000

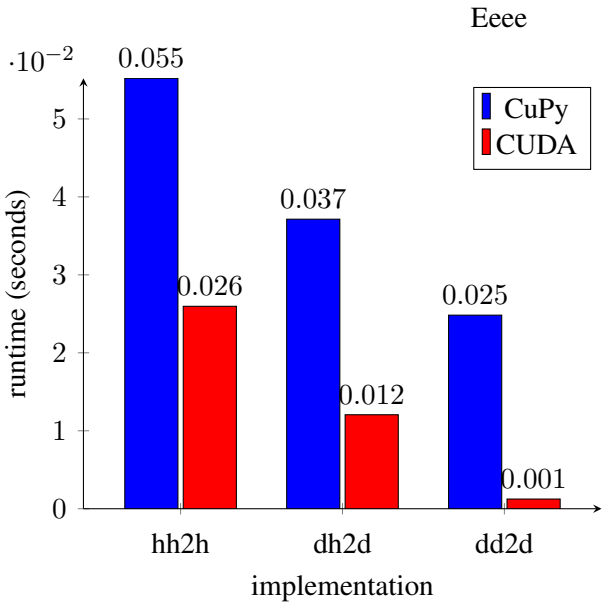

Figure 2: Runtime in seconds averaged over 100 function calls where len(k) = len(r) = 40,000,000

References

[1] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[2] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[3] Ryosuke Okuta et al. “CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations”. In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf.