A user's perspective to Parallel FFT library

YU FENG, University of California Berkeley

About :

- Yu Feng
- Github: @rainwoodman
- FFT User / language binding
- Experience
  - Particle-mesh / cosmology simulation / data analysis
    - pfft-python, pmesh → nbodykit (~ 10K ranks)
    - pfft → FastPM MP-Gadget (~100K ranks)
  - Tools and development process from the Python data-science scene;
  - Python and MPI

Performance != Productivity

- Performance translates to productivity
  - Faster code → higher throughput → productivity
  - But not the full story.
- What about exploratory data analysis :
  - Designing and implementing models takes time too!
    - linking errors; dependency; compiler compatibility ...
    - connecting the FFT API to other analysis code.
    - “I’ll work on this tomorrow because the code looked so scary.” Tomorrow never comes.
  - Pipeline is longer than FFT:
    - I/O, short-range interaction, feature extraction, ...
    - In many cases it is only 10% to 50% of the computing
  - In real research activities, performance is still relevant, but

Why was PFFT chosen?

- Portable:
  - Special tools may yield more performance; at the cost of portability
  - Interpretive languages prefer shared libraries;
  - Unfortunately C is still the only language in the industry that is truly portable.
- Modular:
  - Do one thing and do it really well. Look at FFTW!
  - A giant library that contains many domain specific jargons scares user away.
- Higher level API that focuses on feature, not implementation:
  - Distribution of the FFT array across ranks (parallel); strides

Path Forward

- A Higher level API for Parallel FFT
  - Probably the easiest place for everyone to agree on;
  - Can we design a higher level parallel FFT wrapper library that can switch back-ends?
  - Can we design a higher level API that each of the current library can implement and expose?
  - Non-blocking / async FFT can be part of this API.

A Python example


```
def longrange(x, delta_k, split, factor):
    f = numpy.empty_like(x)

    pot_k = delta_k.apply(FKN.laplace) \
                   .apply(FKN.longrange(split), out=Ellipsis)

    for d in range(x.shape[1]):
        force_d = pot_k.apply(FKN.gradient(d)) \
                       .c2r(out=Ellipsis)
        force_d.readout(x, out=f[..., d])

    f[...] *= factor

    return f
```

- BTW: Anyone heard of distributed FFT on TensorFlow / Torch ?