Discrete Fourier Transform Speed in SPNG

Brett Viren

September 30, 2025

Topics

- Discrete Fourier Transform (DFT) speed vs size.
- Code to select a “faster” size.

Intro to DFT speed theory

Original FFT (radix-2 Cooley-Tukey) algorithm

- Only supports array sizes that are powers of 2.
- FFT complexity scales with $\mathcal{O}(N \log N)$.

Mixed-radix algorithms extend FFT to sizes with other prime factors.

- Special, compile-time support optimizes for small prime factors (3, 5, 7).
- Special, run-time (slower) algorithms handle sizes with larger prime factors.

Need luck to hit a fast size but luck is fairly uniform

- Huge speed ups often possible with small increase in size.
- A simple heuristic can find good candidates for “special”.
  - First larger size with prime factors less than some value (eg 7).
  - But sometimes next or next-next with small primes is faster.
- Measurements needed to find the truly faster but problematic.
  - Very dependent on the device (CPU/GPU), amount of and makeup memory (L1/L2/L3 cache) and parallelism.
  - Measuring produces a large table that needs significant time for lookup.

Measurements

Test environment and plots

- Using wcgpu0 Ryzen 7970X, RTX 4090
- Look duration at `torch::fft::fft()` on all sizes of 1D tensor
  - Given a focus and color to just small-prime factor sizes.
- Look with and without CPU multi-threading.

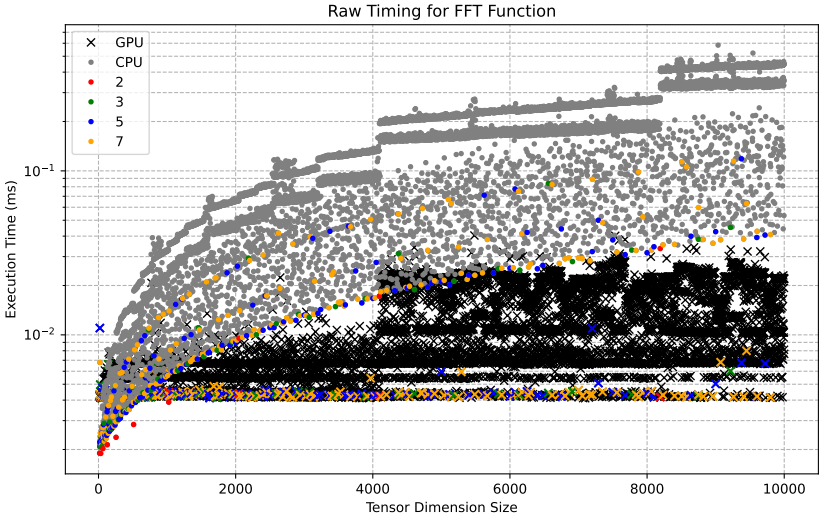
Testing procedure:

- Duration is measured with `<chrono>`
- Each test point repeats to get around 10ms total duration.
- A GPU test point loop is async followed by a sync to assure completion.

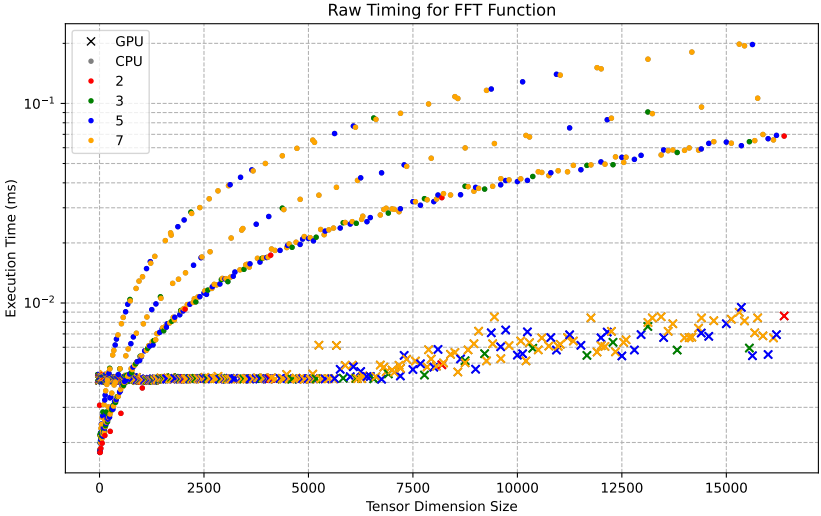
Code to reproduce is in `spng/test/check_dft_measure.{cxx,py}`.

- (written in “collaboration” with Gemini)

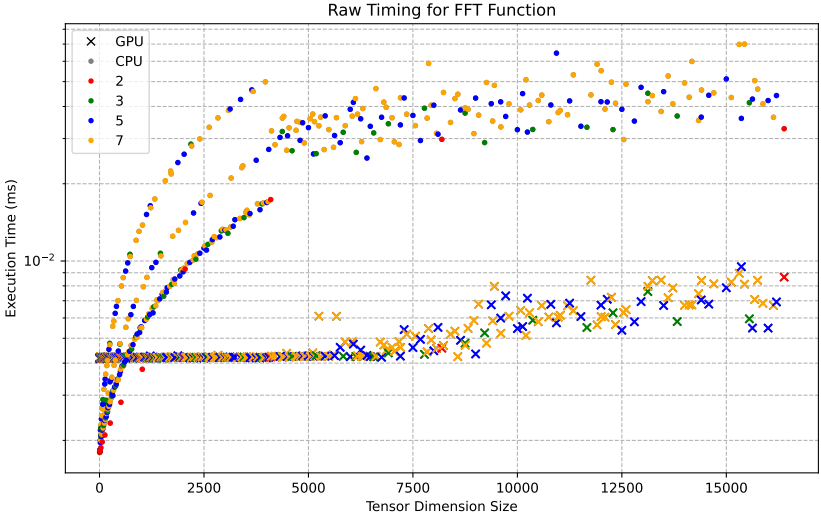
Measurements - all sizes, single-core CPU



Measurements - small-prime factor sizes, single-core CPU



Measurements - small-prime factor sizes, multi-core CPU



Some observations

- Powers of 2 win **but only up to 1024**.
  - 2048 and above is slower, maybe because of Ln CPU cache misses.
- Most points have a **nearby faster point** that has small-prime factors.
  - There is a spread in small-prime factor speeds.
  - Spread is in bands, actually, and I don't understand their origin.
- `libtorch` kicks in CPU-parallelism after 4096.
  - Increases throughput $\approx 2\times$ but uses ≈ 20 cores, a **high cost**!
- **CPU beats GPU for small sizes** below about 512!
  - Almost certainly due to GPU transfer overheads compared to fast CPU memory access.
- **GPU time largely constant** as function of size for moderate sizes.
  - I expect this is due to massive parallelization and fixed overheads.
- Results are very **device dependent**.
  - Testing other CPUs/GPUs is needed.

Reproducing

Environment for single-threaded tests:

```
$ export OMP_NUM_THREADS=1
$ export MKL_NUM_THREADS=1
```

Measure

```
$ ./build/spng/check_dft_measure -s range -m 16 -M 10000 benchmark-10k.csv
$ ./build/spng/check_dft_measure benchmark-st.csv
```

Unset the *_NUM_THREADS and repeat to get a MT data.

Plot

```
$ ./spng/test/check_dft_measure.py benchmark-10k.csv benchmark-10k.pdf
```

Etc for the others.

Picking the best size

WCT mainline already has an `fft_best_length()`, why not just use that?

Hard-coded lookup based on some undocumented measurement procedure on an unknown device using unknown software done at least seven years ago. SPNG can do better!

High-level API

```
#include "WireCellSpng/DFT.h"

// Init in single-thread context with default policy.
FasterDftSize fds;

// Optional customized policy.
fds.reset(make_faster_dft_size_measured(...));

// Use in MT context okay
int64_t new_size = fds(old_size);
```

Design

This design is driven by the need to be thread-safe inside `INode::operator()` while also handling file loading, caching, etc.

More API

Mid-level - construct with some available knobs

```
inline const std::string faster_dft_size_default_measurement_file =
    "torch_dft_measured.json";

FasterDftSizeBase::pointer make_faster_dft_size_measured(
    torch::Device device = torch::kCPU,
    const std::string& filename = faster_dft_size_default_measurement_file);

FasterDftSizeBase::pointer make_faster_dft_size_primes(int64_t max_prime=7);
```

Low-level - actual measure lookup or heuristic calculation

```
int64_t faster_dft_size(const std::vector<int64_t>& sizes,
                        const std::vector<float>& durations,
                        int64_t orig_size);

int64_t faster_dft_size(int64_t orig_size, int64_t max_prime=7);
```

Open questions:

- Is there significant variation across different devices?
- Is there significant variation across different DFT functions?
  - How does 2D play a role? Does `rfft()` have speeds from `fft()`?
- Can/should we exploit caching and reusing DFT plans?
- Does any of this matter compared to other operations?