Advanced Topics in Numerical Analysis
High Performance Computing
Implementation of Fast Fourier Transform (FFT)

Yongyan Rao

Courant Institute, NYU

May 16, 2019

Introduction

The discrete Fourier transform (DFT) of a sequence of N complex numbers $\{x_0, x_1, \cdots, x_{N-1}\}$ is defined as

$$y_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}, k = 0, 1, \cdots, N-1. \quad (1)$$

- Complexity of $\mathcal{O}(N^2)$
- Underlying symmetry $\Rightarrow$ fast Fourier transform (FFT) of complexity $\mathcal{O}(N \log N)$

Carl Friedrich Gauss
1805, Interpolation of orbits of celestial bodies (vs. least squares)

Introduction

$$
\begin{aligned}
y_k \equiv y_{k,0,1} &= \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} = \sum_{\text{even } n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} + \sum_{\text{odd } n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-\frac{2\pi i}{N} k(2n)} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{2\pi i}{N} k(2n+1)} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-\frac{2\pi i}{N/2} kn} + e^{-\frac{2\pi i}{N} k} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{2\pi i}{N/2} kn} \\
&= y_{k,0,2} + e^{-\frac{2\pi i}{N} k} y_{k,1,2},
\end{aligned}
$$

$$
\begin{aligned}
y_{k,0,1} &= y_{k,0,2} + e^{-\frac{2\pi i}{N} k} y_{k,1,2} \\
&= y_{k,0,4} + e^{-\frac{2\pi i}{N} k} (y_{k,1,4} + y_{k,2,4}) + e^{-2\frac{2\pi i}{N} k} y_{k,3,4} = \cdots .
\end{aligned}
$$

Introduction

Listing 1: Iterative Cooley-Tukey algorithm for FFT

```
1  algorithm fft
2    //input: an array x of complex numbers of length $n$, with $n$ is a power of 2.
3    //output: an array y of complex numbers of length $n$, which is the FFT of x.
4
5    y = bit-reverse(x);
6
7    for(m = 2; m <= n; m *= 2)
8      theta = 2 * pi/m;
9      omega_m = exp(i * theta);
10     for(k = 0; k < n; k += m)
11       omega = 1;
12       for(j = 0; j < m/2; ++j)
13         t = omega * y[k + j + m/2];
14         u = y[k + j];
15         y[k + j] = u + t;
16         y[k + j + m/2] = u - t;
17         omega = omega * omega_m;
18
19   return y;
```

OpenMP Implementation of FFT

The correctness of the sequential implementation is verified by GNU GSL.
The OpenMP implementation was run on cuda2.cims, where 20 cores were utilized.

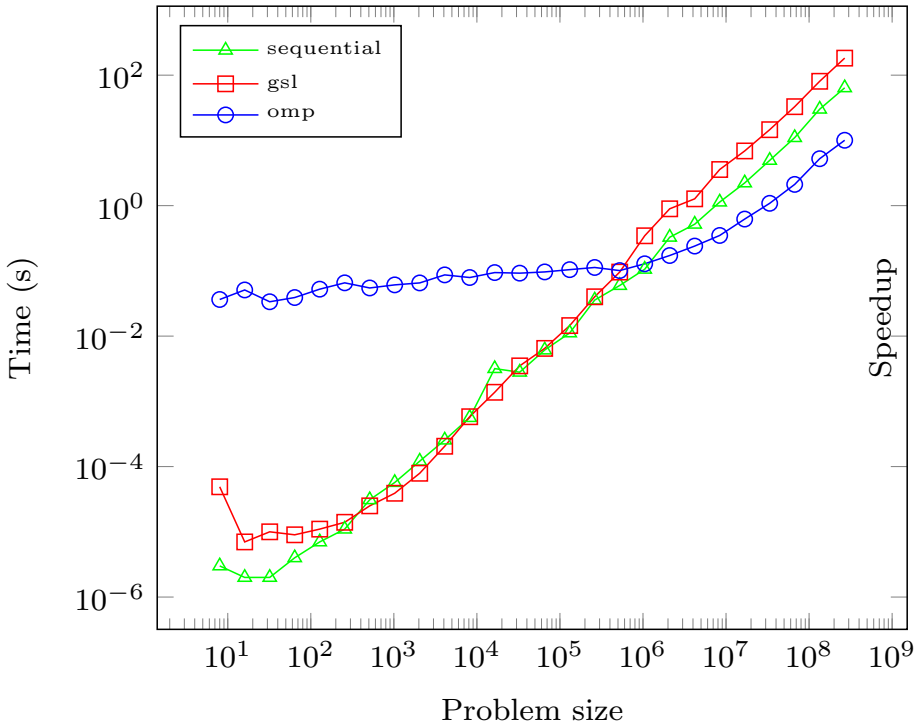

Figure 1: Running time of FFT implementations

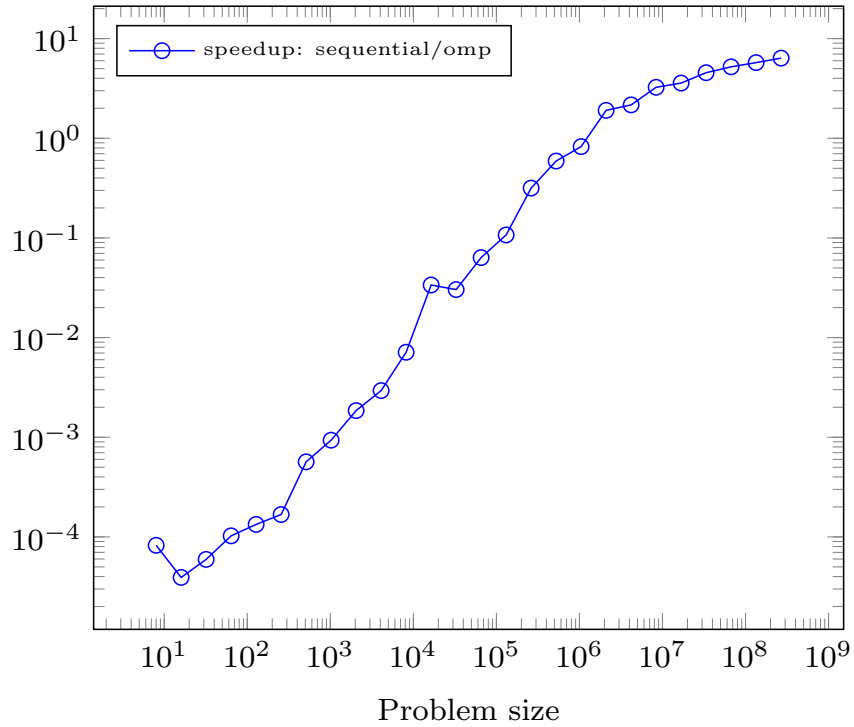

Figure 2: Speedup of OpenMP FFT

OpenMP Implementation of FFT Scalability

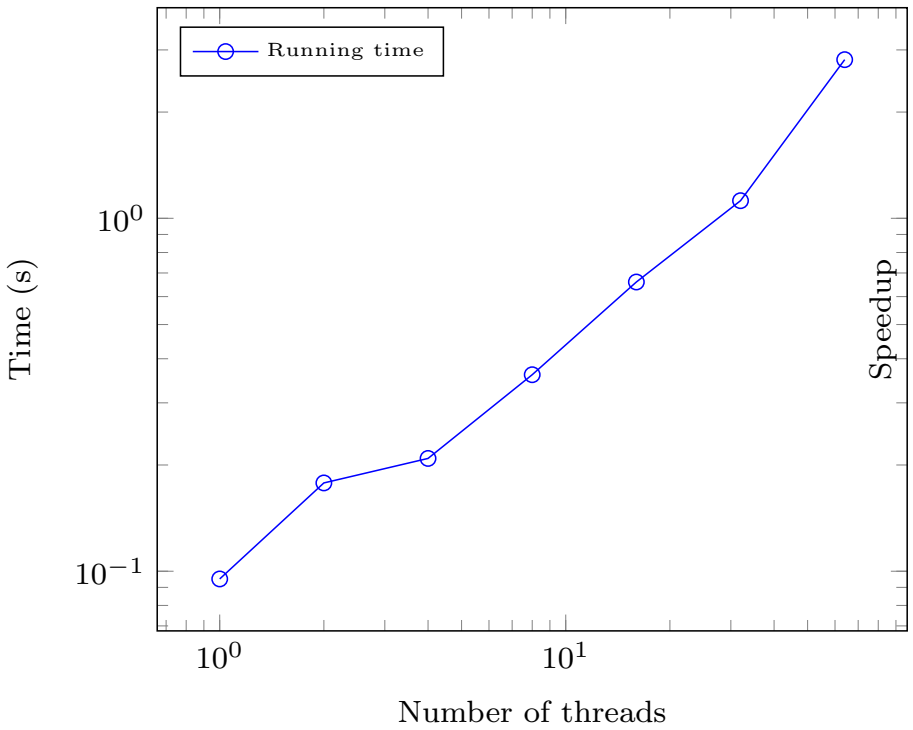

Figure 3: OpenMP FFT weak scaling study

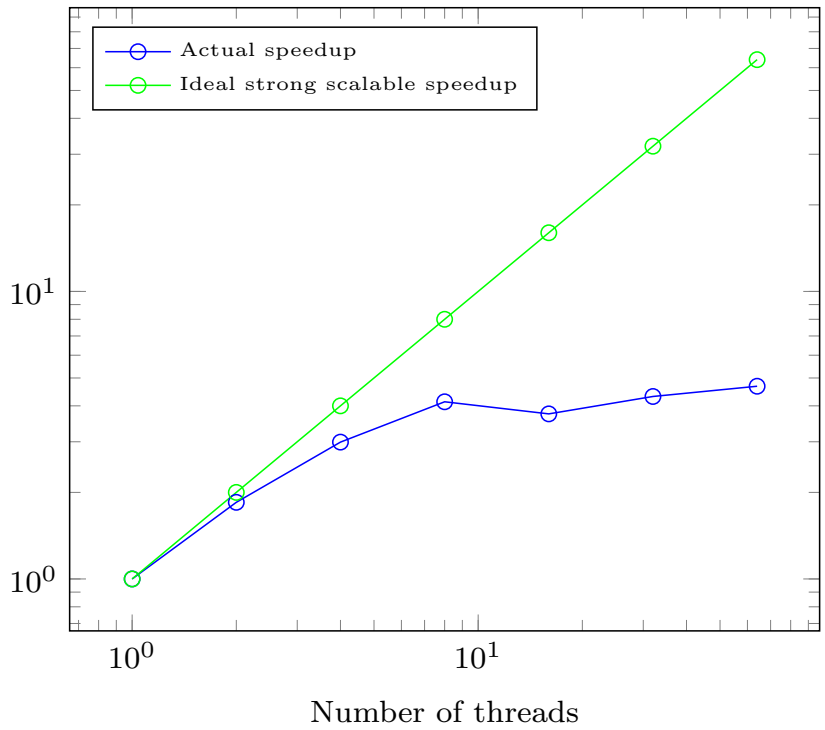

Figure 4: OpenMP FFT strong scaling study

- A program is weakly scalable, if when the number of processes/threads increases x times, the program completes an x times larger problem within the same amount of time.
- A program is strongly scalable, if when the number of processes/threads increases x times, the program completes a same-size problem x faster, which means the speedup of the program is proportional to the number of processes/threads.

cuda Implementation of FFT

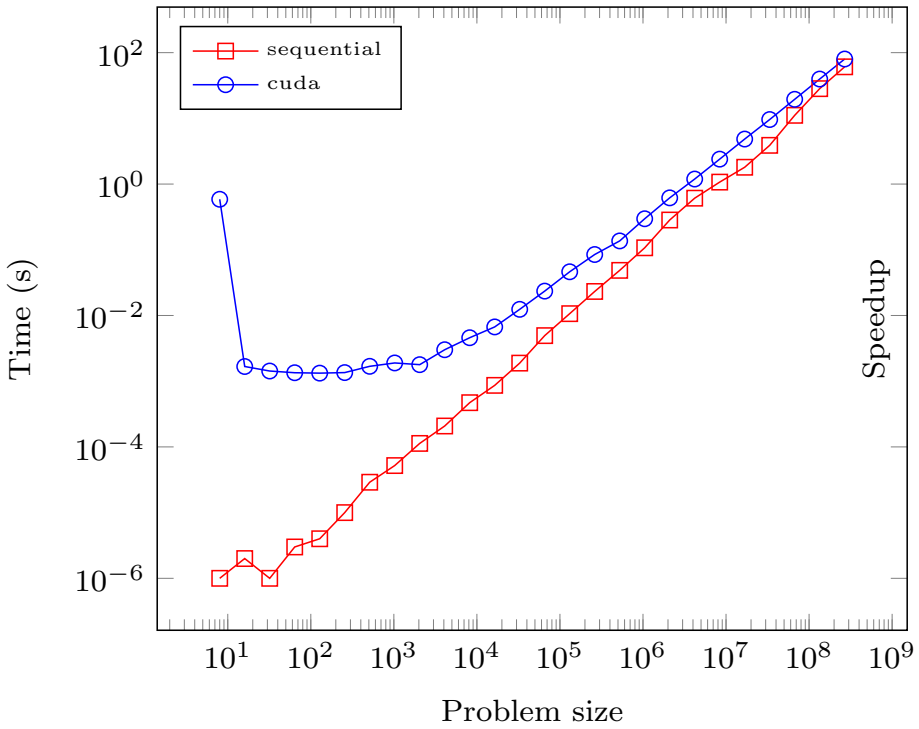

Figure 5: Running time of FFT implementations

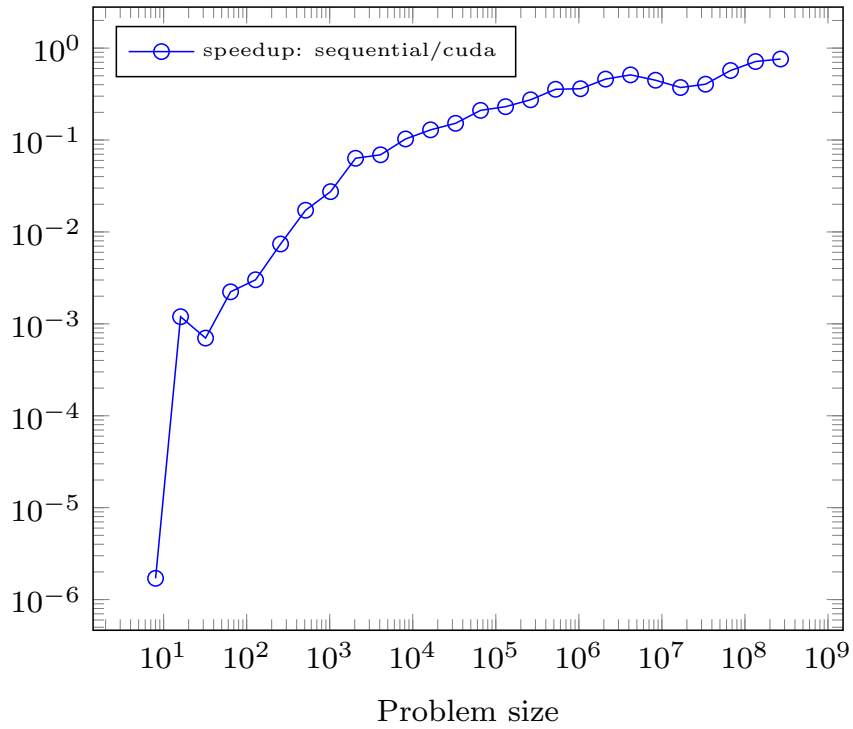

Figure 6: Speedup of cuda FFT

- The greatest problem size accepted was 268,435,456 (on cuda2.cims).
- Problem size $n \Rightarrow 8 \times 4 \times n$ byte device memory, 268,435,456 $\Rightarrow 8.6 \times 10^9$ byte, where the device's total memory was 11.5×10^9 byte and free memory was 9.2×10^9 byte.
- Poor memory access pattern: not fully coalesced memory access, no shared memory used.

Conclusion

- The self-implemented sequential FFT algorithm generally has very similar performance to the GSL FFT function, and for large problem size, the performance of self-implementation even exceeds GSL.
- The OpenMP implementation of FFT shows significant advantage when problem size is greater than 10^5. However, because of the existence of the sequential portion in the algorithm, the multithreaded FFT algorithm is not either weakly scalable or strongly scalable.
- The naive cuda implementation of FFT does not provide reasonable speedup due to multiple factors, including device memory limit, and inefficient device memory access. cuda provides optimized cuFFT API, which is the cuda Fast Fourier Transform library.

Mathematical Joke

Think as a mathematician.

- Q: Why cannot you grow corn in $\mathbb{Z}/5\mathbb{Z}$?
- A: Because it is an abstract mathematical object.

- Q: Why cannot you grow corn in $\mathbb{Z}/6\mathbb{Z}$?
- A: Because it is not a field.