Parallel Two-dimensional Fast Fourier Transform

Sijing Guo Xinyang Wang

Courant Institute of Mathematical Sciences

Final Project of High Performace Computing

May 10, 2017

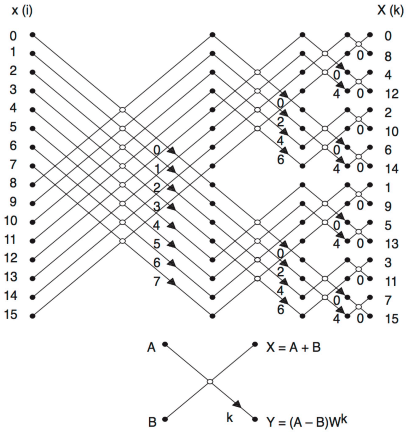
1D FFT

FFT computation is based on a repeated application of an elementary transform known as a “butterfly”.

Step 1: append zeroes to make the total length N a power of 2.

Step 2: repeat doing butterfly operations.

Step 3: adjust the order of outputs.



Fastest Fourier Transform in the West (FFTW) is known as the fastest free software implementation of the FFT algorithm. It supports a variety of algorithms and chooses the one it estimates to be preferable in the particular circumstances.

Sequential 2D Discrete Fourier Transform (DFT)

Suppose the 2D signals are stored in a $N_1 \times N_2$ matrix $\mathbf{x}$. The 2D DFT of $\mathbf{x}$ is defined as

$$X_{r_1,r_2} = \sum_{l_1=0}^{N_1-1} \sum_{l_2=0}^{N_2-1} x_{l_1,l_2} \exp(-\frac{2\pi i}{N_1} r_1 l_1) \exp(-\frac{2\pi i}{N_2} r_2 l_2)$$

$$r_1 = 0, 1, \cdots, N_1 - 1, \text{ and } r_2 = 0, 1, \cdots, N_2 - 1$$

Total computational cost: $O(N_1^2 N_2^2)$.

This can be reduced significantly by separating the 2D-DFT into a series of 1D-DFTs, which can each be implemented using a fast 1D-FFT algorithm.

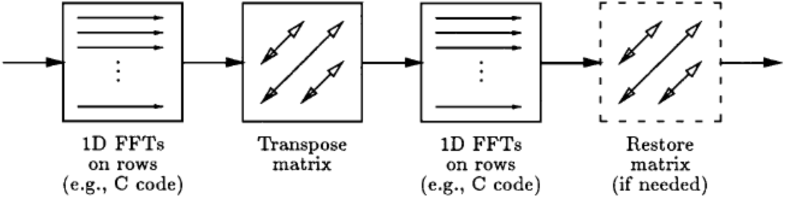
Sequential 2D FFT

$$
\begin{aligned}
X_{r_1,r_2} &= \sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1} x_{l_1,l_2}\exp(-\frac{2\pi i}{N_1}r_1 l_1)\exp(-\frac{2\pi i}{N_2}r_2 l_2) \\
&= \sum_{l_1=0}^{N_1-1}\exp(-\frac{2\pi i}{N_1}r_1 l_1)\left(\sum_{l_2=0}^{N_2-1} x_{l_1,l_2}\exp(-\frac{2\pi i}{N_2}r_2 l_2)\right) \\
&= \sum_{l_1=0}^{N_1-1}\exp(-\frac{2\pi i}{N_1}r_1 l_1)\tilde{X}_{l_1,r_2} \\
&= \sum_{l_1=0}^{N_1-1}\tilde{X}_{l_1,r_2}\exp(-\frac{2\pi i}{N_1}r_1 l_1)
\end{aligned}
$$

Do 1D FFTs on the N_1 rows (of length N_2), and then do 1D FFT2 on the N_2 columns (of length N_1). Total computational cost is $O(N_1 N_2 log_2(N_1 N_2))$.

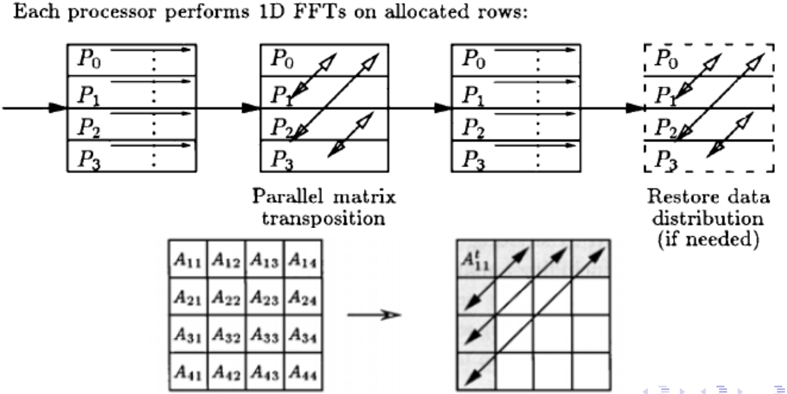
Row-Column Method

- Step 1: 1D FFTs on the rows (locally).
- Step 2: matrix transpose.
- Step 3: 1D FFTs on the rows again (locally).
- Step 4: matix transpose.



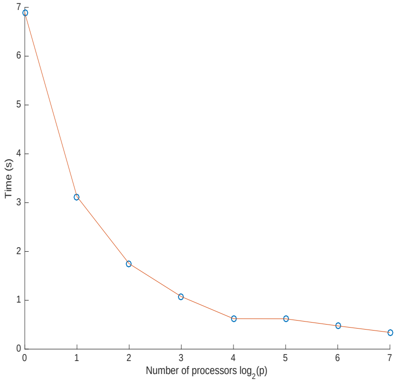
Parallel 2D FFT

- 1D FFT can be computed locally. Use FFTW library to do the 1D FFT.
- Communication is only needed during the transpose step (hardest). Imagine there are blocks in each processor and use MPI Alltoall function to exchange blocks of data with every other processor. Then do the transpose for each block of data.

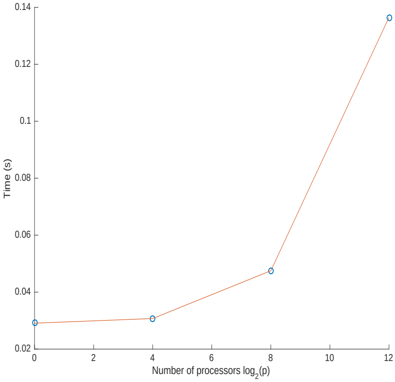


Scalability

- Strong scalability: square matrix N=8192, number of processors: 2^j, $j = 0, \cdots, 10$.
- Weak scalability: keep the number of data in each processor fixed 512×512.



(a) Strong scalability



(b) Weak scalability

- Strong and weak scalability are clear when p is small.
- Communication in the transpose step dominates when p is large.
- When p is too large, running time may increase.
- 2D version FFTW takes 10s to compute this problem when N=8192. Parallel version is much faster.
- Efficient transpose algorithm is needed to improve the performance.

Chu, Eleanor and Alan George
Inside the FFT black box: serial and parallel fast Fourier transform algorithms..
CRC Press, 1999.

Matteo Frigo and Steven G. Johnson
FFTW Manual

Piedra, Rose Marie.
Parallel 1-D FFT: Implementation with TMS320C4x DSPs.
Texas Instruments, 1992.