An Approach for Large-Scale Distributed FFT Framework on GPUs

1st Yichang Hu
School of Computer Science and Engineering,
South China University of Technology
Guangzhou, China
sxy05100415@126.com

2nd BinBin Zhou
School of Computer Science and Engineering,
South China University of Technology
Guangzhou, China
oudwwv@163.com

3rd Lu Lu*
Advanced Micro Devices Inc
eelu.lu@amd.com

***Abstract*—The fast Fourier Transforms (FFT), a reduced-complexity formulation of the Discrete Fourier Transform (DFT), dominate the computational cost in many areas of science and engineering. Due to the large-scale data, multi-node heterogeneous systems aspire to meet the increasing demands from parallel computing FFT in the field of High-Performance Computing (HPC). In this work, we present a highly efficient GPU-based distributed FFT framework by adapting the Cooley-Tukey recursive FFT algorithm. Two major types of optimizations, including automatical low-dimensional FFT kernel generation and asynchronous strategy for multi-GPUs, are presented to enhance the performance of our approach for large-scale distributed FFT, and numerical experiments demonstrate that our work achieves more than 40x speedup over CPU FFT libraries and about 2x speedup over heFFTe, currently available state-of-art research, on GPUs.**

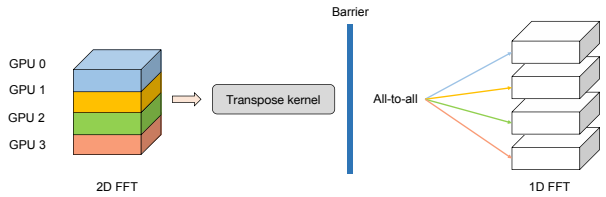
***Index Terms*—Distributed FFT, GPU, Kernel generation, Asynchronous strategy**

I. Introduction

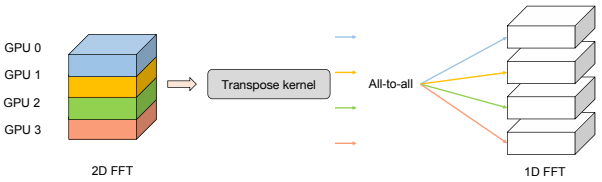
Frequency domain analysis and Fourier methods are extensively applied in a wide range of applications in various engineering and scientific fields. In the interim, with the wide use of the FFT algorithm and the tendency of scalability in the field of HPC, the design and optimization of FFT on heterogeneous systems remain an active area. In view of the poor scalability of the existing distributed FFT library and the limitation of the past work, the parallel performance of large-scale FFT should be enhanced with a strong motivation [1].

This work focuses on optimizing large-scale 3D-FFT for efficient execution on distributed systems and makes the following four primary, novel contributions in optimizing large-scale distributed FFT framework on GPUs:

- A low-dimensional FFT kernel generation that automatically generates assembly kernels, which efficiently utilize shared memory to complete remove global memory (DRAM) access within the butterfly operation [2].
- A hybrid MPI-OpenMP parallel programming model composed of intra-node communication and inter-node communication.
- An asynchronous strategy of packing and communication that overlaps packing/unpacking with MPI communication.



(a) Standard approach of packing and synchronous data transfer



(b) Asynchronous approach coupling packing and communication

Fig. 1. Comparison of the synchronous and asynchronous implementation.

- Specific kernels for highly efficient local transpose in non-strided FFT.

II. Design and Methodology

In this section we will dive into our overall algorithmic approach, which is divided into the following four stages.

(1) First, low-dimensional FFT kernels defined to calculate mixed-radices butterflies are generated. Small sequence butterflies are grouped for non-strided data access mode, while data is reshuffled during the algorithm with an additional buffer when the input sequence is large and all transformations are performed in place in Alg.1, compared with non-continuous data pattern which requires twice the size of shared memory. FFT kernels are compiled by *hipRTC* APIs, providing performance improvement compared with regular offline static compilation [3].

(2) A hybrid MPI-OpenMP programming model is employed to complete the distributed 3D FFT. Multiple threads are utilized for intra-node communication to control multiple devices, while GPU-aware MPI for inter-node communication. The hybrid strategy accelerates the intra-node data copy and reduces the overhead of MPI effectively.

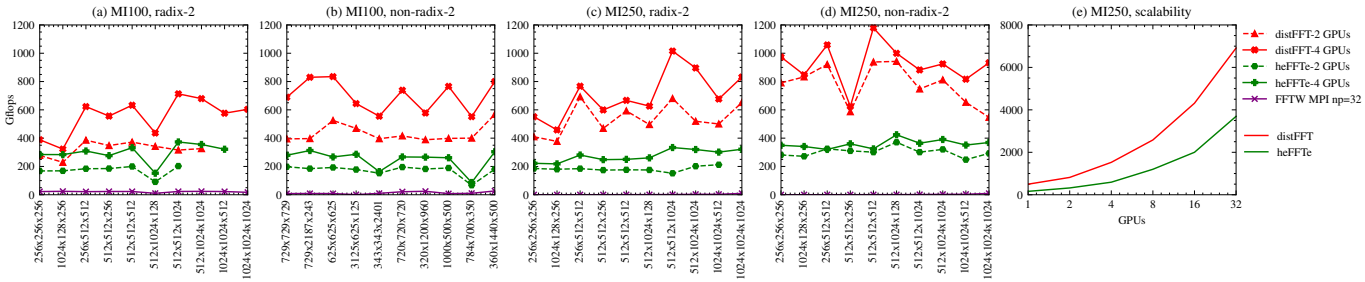


Fig. 2. Performance Evaluation.

(3) Our work adopts a novel efficient asynchronous-management of packing and communication, as shown in Fig.1, which demonstrates the classical packing process and a novel approach in our design, overlaping packing/unpacking with MPI communication.

(4) Two specific transpose kernels are designed. The first transpose kernel is to rearrange the data in the x dimension into a continuous buffer after transforming the y and z dimension, allowing only one communication across every two GPUs. A 3D point on (i, j, k) can be mapped into the $(i \times Y \times Z + j \times Z + k)^{th}$ element, and the destination location after transposition is

$$X \times \lceil Y/D \rceil \times Z \times (j/\lceil Y/D \rceil) + i \times H \times Z + j mod \lceil Y/d \rceil \times Z + k \quad (1)$$

where D denotes the number of devices. The second kernel is to transpose data from (x, y, z) to (y, z, x) so that a striding FFT is performed in the x dimension after the all-to-all communication. 3D data is splitted into small 2D tiles, which are padded in LDS to avoid bank conflicts, and a diagonal block is employed to avoid partitioning camping.

Algorithm 1: FFT_kernel(in, out, twiddleTable, s, l)

Input: in[]: input; tw[]: twiddles; s: stride size; l: length
Output: out[]: output

```
foreach Thread do
    Load inputs from in[] according to s;
    Bufferfly s-point DFT on in[];
    MULTIPLY(in[], tw[]);
    __syncthreads();
    Bufferfly l/s-point DFT on in[];
    MULTIPLY(in[], tw[]);
    __syncthreads();
end
Store outputs to output out[];
```

III. Evaluation

This section assesses the exhibition of our work on a heterogeneous system furnished with AMD GPUs and compare with heFFTe [4] on multiple devices, which are the state-of-art distributed FFT libraries. Fig.2 shows the line graph for our results, where distFFT denotes our work, displays the results of distributed 3D FFT in a single node utilizing two and four GPUs. For the first four graph several generic FFT size is evaluated on AMD MI100 and MI250 GPUs [5], including radix-2 and non-radix 2 scenario, and scalability of our work is in the last graph. It is clear that our implementations consistently outperforms heffte, while not compromising on communication overhead. And this advantage turns out to be more evident with the increase in the input length. This is in that each GPU spends little time on calculation and the communication time accounts for a huge part when the input length is excessively small.

IV. Conclusion

Heterogeneous parallel system will continue to evolve in the near furture and pose new challenge on high performance algorithm development. This work proposes an apporach for large-scale distributed FFT framework on GPUs that makes following three primary novel contributions:

- Novel data reshuffling and memory transfer pattern.
- Asynchronous management for packing and communication.
- Unit-stride data layout and specific transpose kernel for high efficient memory operations.

Our work demonstrates its effectiveness through some practical case studies and achieves more than 40x speedup over CPU FFT libraries and about 2x speedup over heFFTe. Future work will focus on enhancing the algorithm's scalability and other reductions of the computational resources that can accomplish efficient communication, like using non-blocking collectives or dispersing the data to both CPU and GPU.

References

[1] D. T. Popovici, T. M. Low, and F. Franchetti, "Large bandwidth-efficient ffts on multicore and multi-socket systems," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018.

[2] Z. Li, H. Jia, Y. Zhang, T. Chen, L. Yuan, L. Cao, and X. Wang, "Autofft: A template-based fft codes auto-generation framework for arm and x86 cpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, 2019.

[3] AMD, *rocFFT library*, 2017. [Online]. Available: https://github.com/ROCmSoftwarePlatform/rocFFT

[4] A. Ayala, S. Tomov, A. Haidar, and J. Dongarra, "heffte: Highly efficient fft for exascale," in *Computational Science – ICCS 2020*, 2020.

[5] AMD, *CDNA architecture*, 2020. [Online]. Available: https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf