Research on heterogeneous acceleration platform based on FPGA

Yuan Meng*, and *Jun* Yang

School of Information Science and Engineering, Yunnan University, Kunming 650500, China

Abstract. In the context of today's artificial intelligence, the volume of data is exploding. Although scaling distributed clusters horizontally to cope with the increasing demands on computing power for massive data processing is feasible. But the unlimited addition of nodes will lead to bloated cluster size. Most of the transistors in CPUs are used to build cache memory and control units, which are not efficient for computing operations of massive data processing. Currently, academia uses hardware devices such as GPU (Graphics Processing Unit), ASIC (Application Specific Integrated Circuit), FPGA (Field Programmable Gate Array) to accelerate deep learning, image processing, which require massive computational operations. The paper first discussed the advantages and technical requirements of FPGA acceleration based on the characteristics of the Spark cluster. Then the paper proposed the design of the FPGA-CPU heterogeneous acceleration platform, and introduced the base-two-FFT algorithm. Finally, the paper present and compared the computation time of the base-two-FFT algorithm before and after the acceleration. The results show that the heterogeneous cluster has a speedup ratio of about 1.79 times compared to the CPU cluster.

1 Introduction

With the rapid growth of data volume in recent years, the design of computing platforms needs to consider both high performance and acceptable power consumption. Spark is an open-source big data processing engine that supports the performance computation of RDD and provides reusability, fault tolerance, and real-time stream processing [1]. Spark's application tasks are executed only on the CPU. Low parallelism and low power efficiency may limit the performance and scalability of Spark clusters. Heterogeneous accelerators such as FPGA, GPU and MIC show better performance than general processors in the field of big data processing. If we integrate these heterogeneous accelerators into the original Spark framework, we can significantly improve the performance of each node. GPU expands its performance through the number of cores and SIMD/ SIMT parallelism. It has the best performance when it repeatedly performs several tasks and simultaneously performs the same operation, but it is inferior to FPGA when it performs different tasks [2]. Based on the above considerations, this paper uses FPGA as a heterogeneous accelerator. Spark applications can be developed in Scala, Java, and Python programming languages. Python has an extensive

* Corresponding author: 519207213@qq.com

library of scientific computation and data processing and is fast to develop [3]. Therefore, this article mainly uses PySpark to develop CPU programs.

The rest of this paper is organized as follows: Section 2 introduces the technical background and related research; The third section introduces the design of integrating FPGA acceleration core into Spark cluster. The fourth section introduces the implementation and simulation of the FPGA acceleration core based on the base-two-FFT algorithm. The fifth section presents the test and analysis of computing time and power consumption of the heterogeneous system.

2 Technical background and related research

2.1 FFT Algorithm

The FFT (Fast Fourier Transform) is widely used in speech processing, graphics processing and software radio [4], [5] and is an efficient implementation of the DFT (Discrete Fourier Transform). In the DFT, the sequence *X(k)* (where k = 0, 1, ... *N-1*) is computed from the input sequence *X[n]* (where n = 0, 1, ... *N-1*) computed from the input sequence *X[n]* (where n = 0, 1, ... *N-1*), which is defined in Equation 1.

$$X(k) = \sum_{n=0}^{N-1} \left(x[n] W_N^{nk}\right) \tag{1}$$

The phase factor W_N^{nk}, also known as the rotation factor, is defined in equation 2.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}} x(2n+1) W_{N/2}^{2nk} \tag{2}$$

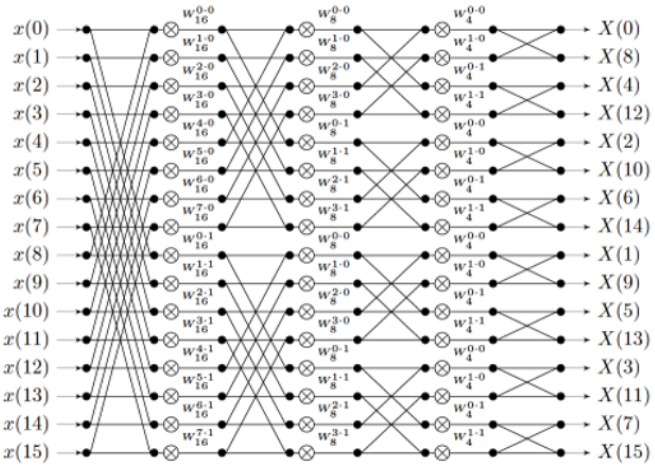

Fig. 1. 16 floating point base-two-FFT signal flow diagram.

Figure 1 shows the flow of the base-two-FFT algorithm, taking an FFT with 16 sampling points as an example. Firstly, the FFT of 16 sampling points is decomposed into the FFT of the odd-even combination of two 8-sampling points, and then the FFT of two 8-sampling points is decomposed into the FFT of four 4-sampling points in parallel. The cycle is carried out until the FFT is decomposed into two sampling points, then put into equation 1 for calculation.

2.2 Related research

Baidu's Parallel Distributed Deep Learning Platform (Paddle) [6] integrates GPUs and FPGAs into clusters to accelerate applications. IBM's Coherent Accelerator Processor Interface (CAPI) provides the POWER8 core, the system's available memory architecture, and a high-bandwidth, low-latency path between peripheral devices [7]. Microsoft developed a custom FPGA board, Catapult [8], which will be placed on the server of each cluster of 1632 nodes. Catapult has increased throughput per server by 95% and reduced tail latency by 29% under high load. Yu Ting Chen [8] and Ehsan Ghasemi [9] connected the host program of JVM and OpenCL through JNI and used the computing framework provided by OpenCL to control and manage the link between FPGA and CPU. Inspur cooperated with IFLYTEK [10] to conduct accelerated research on the DNN speech recognition algorithm based on deep learning on a heterogeneous platform. The results show that FPGA has a noticeable performance and energy consumption ratio advantage. Huang [11] realized the acceleration of MuTect2 based on FPGA. The experimental results showed that the FPGA implementation achieved a maximum acceleration effect of nearly thirty times and an average acceleration effect of about three times for each node in the load balancing test. In the above work, only the platform test results are involved, but the integration scheme of heterogeneous framework and acceleration design of accelerator are not introduced in detail.

Aiming at the slow development process of the FPGA heterogeneous platform, this work designed a distributed heterogeneous computing platform based on the latest SDSoC whole system optimization compiler, which shortened the overall development cycle. Using Vivado HLS high-level synthesis tool and accelerating strategies such as pipeline and cyclic unrolling, an efficient accelerator IP core is designed and realized for the base-2-FFT algorithm. In the simulation and test, the acceleration effect is better than that of single CPU platform.

3 Design of heterogeneous computing cluster

In this study, the cluster includes one master node and one slave node. The slave node is equipped with a Xilinx Artex 7@200MHz board card. The slave node is equipped with Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz. It includes 3 PCIe bus interfaces, including an X4 PCIe bus interface connected to the Xilinx Artix-7 FPGA development board. Artix-7 has 126800 registers for locking data; 63,400 LUTs for control logic, gate circuits, and selectors; 135 BRAMs for caching small amounts of data; 240 DSPS, which are internal computing resources for multiplication.

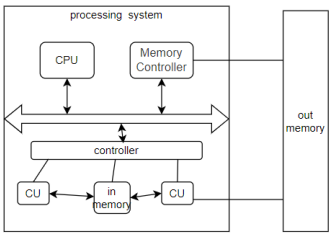

Fig 2. The top-level design framework of a heterogeneous Spark cluster.

This paper uses Xilinx SDSoC 2017.4 as the primary design tool, using Vivado HLS and Vivado to compile C/C++ code into the format of Verilog (code of FPGA). The system structure is shown in Figure 2, the base-2-FFT algorithm is realized inside FPGA, the input data set is preloaded and sliced on CPU processor, and the initialization stage of FPGA is controlled by CPU. The hardware logic part mainly consists of a computing unit, on-chip memory, DMA and finite state machine controller. DMA is mainly used to preload control

parameters into the on-chip memory during the operation phase and write the final result data set back to the off-chip memory. In-chip memory is mainly used to store the result data after computing unit operation and save the data read by DMA. It corresponds to BRAM, FIFO, RAM and other buffer storage resources, respectively. The computing unit is implemented based on the LUT inside the FPGA, and different configurable logic blocks and I/O units are called inside each compute unit.

4 FPGA implementation and simulation of base-two-FFT algorithm based on heterogeneous platform

4.1 FPGA implementation of base-two -FFT algorithm

High-speed FFT operation requires multipliers, a large amount of memory and registers. This is the reason why the FFT algorithm is suitable for FPGA implementation [12]. The I/O architecture of base-2-FFT is based on the design of the butterfly processing engine, as shown in Figure 4. It is mainly comprises of RAM (FPGA internal resources), butterfly operation module, rotating factor ROM.

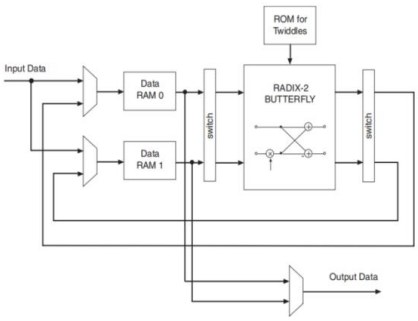

Fig 4. Diagram of radix-2 FFT processing engine.

When executed sequentially, O(NlogN) operations in FFT require O(NlogN) time steps. A common way to parallelize FFT is to organize the computation into $logN$ stages. The actions of each phase depend on the actions of the previous phase, naturally leading to pipelining across tasks. This architecture allows simultaneous calculations of $logN$ FFTS with task intervals determined by the architecture of each phase.

Each stage in FFT also contains significant parallelism because each butterfly calculation is independent of other butterfly calculations in the same stage. Each clock cycle performs $N/2$ butterfly calculations with task intervals of one. The FFT code we implemented is a nested three-tier "for- loop" structure.

The external "for-loop", marked $stage\_loop$, implements a phase of FFT during each iteration. There are $logN$ stages, where N is the size of input samples. In this experiment, the 16-point FFT has eight butterfly operations.

The second "for-loop", marked $butterfly\_loop$, performs all of the butterfly operations for the current phase. $butterfly\_loop$ has another nested "for-loop", marked dft_loop. Each iteration of dft_loop performs a butterfly operation. The first line in dft_loop determines the offset of the butterfly operation. The "width" of butterfly operations varies depending on the stage. Phase one performs a butterfly operation on adjacent elements, phase two performs a butterfly operation on elements whose indexes differ by two, and phase three performs a butterfly operation on elements whose indexes differ by four. The difference is calculated and stored in the i_lower variable. We can see the variable $numBF$ is different at

each stage. The remaining operations in dft_loop perform multiplication by rotation factor and addition or subtraction operation. The variables $temp\_R$ and $temp\_I$ retain the real and imaginary parts of the data after multiplying by the rotation factor *W*. The variables c and S are the real and imaginary parts of *W*. We choose to store the real and imaginary parts of the complex number in two separate arrays. *X_R* holds real input values, and *X_I* holds virtual values. *X_R[i]* and *X_I[i]* preserve the complex number indexed *i* in separate real and imaginary parts. Finally, the elements of *X_R []* and *X_I []* arrays are updated using the results of the butterfly calculation.

dft_loop and $butterfly\_loop$ are executed at different times depending on the phase. However, the total number of times dft_loop is executed in a phase is constant. $butterfly\_loop$ has only one iteration in phase one, two iterations in phase two, and four iterations in phase three. Similarly, the number of iterations of dft_loop changes. It iterates over the whole algorithm eight times in stage one, four times in stage two, and only two times in stage three In each stage, the dft_loop body performs the same total number of times, and a total of eight butterfly operations are performed in each stage for a 16-point FFT.

4.2 FPGA simulation of base-2-FFT algorithm

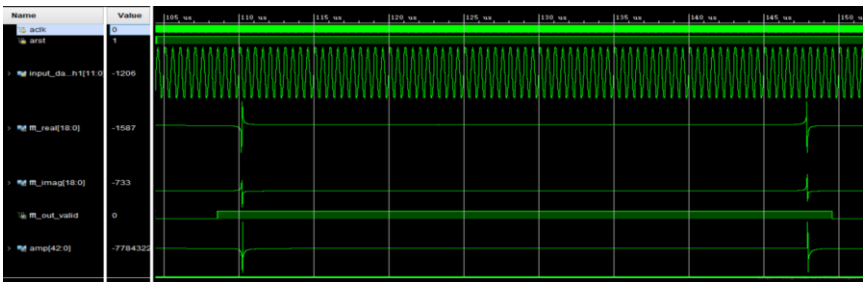

Fig. 5. Radix-2 FFT simulation of 2MHz signal.

We use MATLAB to generate 2MHz (shown in Fig 5) and 15MHz (shown in Fig 6) sine wave signals and output them to a text file. The text files of the two signals are read separately to do FFT analysis of the two single frequency signals by running the excitation program. Firstly, the FFT analysis is done for the 2MHz signal. According to the estimation in Latency above, the calculation time takes 145 μ s, so the simulation needs to run to about 150 μ s. The results are shown in Figure 5. The duration of the whole spectrum is precisely the time that the *out_valid* signal stays high.

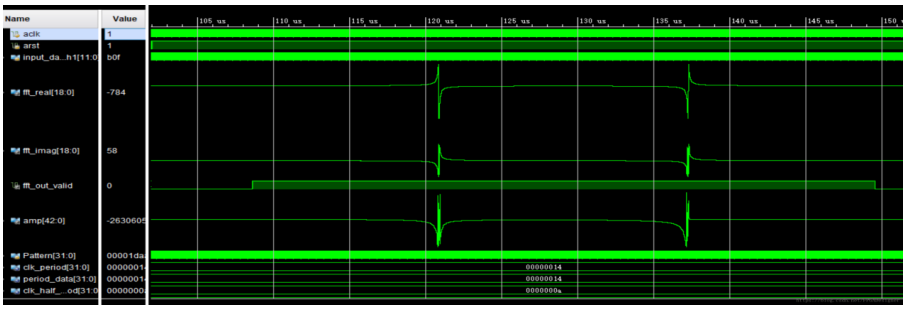

Fig. 6. Radix-2 FFT simulation of 15MHz signal.

Then the FFT simulation of the 15MHz signal is simulated, and the results are shown in Fig 6. It is observed that the peak position of the spectrum of the 15Mhz signal is significantly more centred than the 2MHz frequency, which follows the fact.

The test data set used in this experiment is the vibration signals from six sets of signal collectors with a sampling frequency of 512 and a sampling point count of 512. The data set is stored in a text file, and each data is divided by a comma. The data set will be computed using the algorithm library of this paper and MATLAB 's built-in FFT function, respectively, and the calculation results are stored in the text file. Comparing the calculation results shows that the calculation results are the same as the built-in function in MATLAB, and the correctness of the algorithm is verified.

5 Conclusion

In this section, the performance of the base-two-FFT algorithm on the heterogeneous acceleration platform is tested and analysed. First, we analyse the speedup effect by testing the overall computation time of the algorithm. Then the overall system computation time of the heterogeneous platform's and CPU's executing the same scale of the base-two-FFT algorithm is compared. Finally, the FPGA development board resource consumption is statistically analysed.

5.1 Accelerate platform testing and analysis

In this paper, we compare the performance of FPGA (Artix 7) and CPU (Xeon E3-1505M) implementing different scales of the base-two-FFT algorithm, and the results are shown in Figure 7. The system computation time for both schemes increases as the operation scale increases. At a scale smaller than 128 x 128 floating point numbers for the base-two-FFT (a single kernel in FPGA implementation), the performance of FPGA decreases compared to CPU. This is caused by a large number of RDD generation, computation and collection in Spark Cluster. However, at scales larger than 128 x 128 floating point numbers, FPGAs outperform CPUs, and as seen in Figure 8. The heterogeneous acceleration platform is 1.79 times faster than the CPU implementation. We will scale up the cluster size and configure each slave node with an FPGA accelerator in future work. In addition, we will optimize other time-consuming algorithms and implement a more optimized FGPA acceleration strategy.

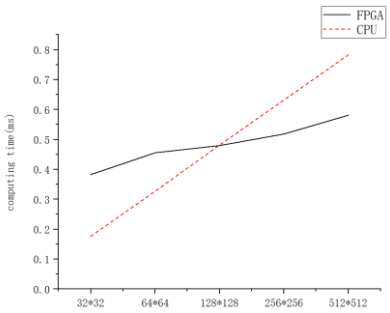

Fig. 7. Comparison of FGPA and CPU computing time of radix-2 decimation-in-time FFT.

5.2 Power consumption testing and analysis

In this subsection, we test and analyse the power consumption of the designed accelerated system to calculate the base-two-FFT algorithm. Vivado provides us with the occupancy of various resources inside the development board to analyse of the power consumption of the FPGA.

It can be seen from Table 1 that BRAM consumes fewer resources, because the system uses register groups to cache temporary calculation results of RDD between computing modules. Register resources are frequently used because the cache Register group occupies a large number of Register resources. A large number of dual-end read and write operations must be controlled by the control unit to avoid read and write conflicts. Therefore, LUT resources responsible for the control logic and data selection are often used.

Table 1. Artix resource consumption.

	Used	Total	Utilization Ratio (%)
Register	34422	126800	27.1.
LUTs	24682	63400	38.9
BRAM	11	135	24
DSP	234	240	97.5

References

1. Rui Zhao, Zhaopeng Meng, Yan Zheng, Qiangguo Jin, Anbang Ruan, and Hanglun Xie. Somr: Towards a security-oriented mapreduce infrastructure. In Trustcom/BigDataSE/ICESS, 2017 IEEE, pages 530–537. IEEE, (2017)
2. Guiming Wu. Parallel algorithms and structures for FPGA matrix computing [D]. National University of Defense Technology, (2011)
3. Charles Dierbach. Python as a first programming language. Journal of Computing Sciences in Colleges, **29(6)**:153–154, (2014)
4. Berkin Akin, Peter A Milder, Franz Franchetti, and James C Hoe. Memory bandwidth efficient two-dimensional fast fourier transform algorithm and implementation for large problem sizes.In Field-Programmable Custom Computing Machines, 2012 IEEE 20th Annual International Symposium on, pages 188–191. IEEE, (2012)
5. Gokhan Polat, Sitki Ozturk, and Mehmet Yakut. Design and implementation of 256-point radix-4 100 gbit/s fft algorithm into fpga for high-speed applications. ETRI Journal, 37(4):667–676, (2015)
6. Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). arXiv preprint arXiv:1412.6632, (2014)
7. B Brech, J Rubio, and M Hollinger. Ibm data engine for nosql- power systems edition. IBM Systems Group, Tech. Rep, (2015)
8. Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. When apache spark meets fpgas: a case study for next- generation dna sequencing acceleration. In The 8th USENIX Workshop on Hot Topics in Cloud Computing,(2016)
9. Ehsan Ghasemi,Paul Chow. Accelerating Apache Spark with FPGAs[J]. Concurrency and Computation: Practice and Experience,(2019),**31**(2)
10. Hu Leijun, Chen Naigang, Li Jian, Han Feng, Zhao Yaqian. FPGA heterogeneous computing platform and its applications [J]. Power Information and Communication Technology,(2016),**14**(07):6-11
11. Huang Ruge. Design and implementation of a parallel acceleration scheme for GATK gene analysis software [D]. Huazhong University of Science and Technology,(2019)
12. Liu Baojun, Wang Zhongxun, Zhong Qiang, Zhang Min, Lou Yang. Design and implementation of FPGA-based FFT algorithm [J]. Optical Technology Applications,(2016),**31**(03):46-49