Chia Proof of Space, Plotting, Accelerated by GPU

Ting-Wei SU
Texas A&M University
College Station, United States
willytwsu@tamu.edu

ABSTRACT

Recently, Blockchian has become a new popular terms and area people talk about. Not only does it has tons of innovative application in technology but also the impact it impose is also huge which will overturn the world that we live in. Decentralized is the key component that catch everyone's attention. Ranging from financial technology, semiconductor industry, entertainment industry and art collection, blockchain is providing a whole new architecture of protecting and contributing the system integrity and security at the same time.

Chia network, a new network utilizing the proof of space to verify the farmer for farming power, is a recent spotlight in blockchian forum which came live at June of 2021. The utilization of disk to contribute proof of power is not only innovative but also ecologically friendly. Compared with the most well-known cryptocurrency, bitcoin, the energy consume in chia network is 100 times less than the power required during bitcoin mining.

The motivation of my approach is to accelerate the process of chia farming, which is analogous to bitcoin mining, via a accerlation unit, a GPU computing node on super-computer.

1 INTRODUCTION

The key concept that blockchain was based on decentralized consensus is like a voting system where every node participating in the network could have equal contribution to the decision of the network with

what is proof of space? When a person owns a storage disk with 1 GB, the person can request to join the network by implementing the network protocol to claim and tell the network that he or she has a storage disk with 1 GB. How do we let the network know that you did actually hold a 1 GB storage space? In order to prove you have 1 GB space, we could simply give you a 1GB file and tell you to store the file on your disk. When we need a specific segment of information in the file, we request it from you, you should be able to pull out the exact same information. In order to store the useful information in the File that the network gave it to the farmer and create the proof of space of earn the rewards, three stages are required in the participating of Chia network: In the following chapters, we will discuss more in detail.

Plotting -> Farming -> Verifying Consensus algorithm: VDF Timelord creating the challenge Farmer extracts the proof of space

2 CHIA NETWORK

Firstly, let's start with what is Chia network? Chia is a decentralized consensus network that uses the combination of Proof of Space and Proof of time (VDF) as the fundamental consensus algorithm. Two kinds of people are included in the network. The farmer, persons who provide the storage space, and the verifier, persons who request the information stored by farmer.

3 CHIA PLOTTING PROCESS

The plotting process is when a farmer preperation work that the farmer has to completed before contributing to the network. The plotting process begins with storing many hash-like number in a file that stores on the farmer disk. Seven tables is initiate in the plot file with a specific plot constant k.

Plot constants k The plot constants k represent the number of entries each table, which is determine by the farmer. The minumum disk space of the plot with $k = 32$ is 515.4 (GB) for temperarily storage and 104.7 (GB) for final storage.

$$\text{Number of entries in each table} = 2^k, 32 \leq$$

$$\text{Temporarily plot file size} \approx 3.75 \times k \times 2^k (byte) \text{Final plot file size} \approx 0.762 \times k \times$$

Plot seed The plot seed is a 32 byte constant received from the network to initialize each plot file, so that no two plot file will be the same.

$$\text{plot\_seed} \in [2^{256}]$$

That is said, all the farmer need within the plotting process is the plot seed receiving online, everything else will be able to construct offline. The design of chia plotting process is intentionally design so that whomever gets a powerful machine, such as a ASIC or GPU won't create absolute advantage over those who don't, because the main motivation is letting the farmer purely use The plotting process will be distinguish into the following four parts.

1. Forward propogation The forward propogation is to insert data in each entries of the table, from table 1 to table 7. The dependency of the table is table i + 1 will relie on the talbe i (i = 0, 1, ... 6). Because this dependency, farmer who wish to accelerate the plotting process will not have the ability to predict what the next table content without waiting for the completion of the previous table. This will create a bottle-neck for those who wish to create a faster plotting method using a high performance machine.

2. Backward propogation Backward propogation is to reduce the size of each table by getting rid of those entry that's not in use.

3. Compression The compression stage is to sort the content in each table.

4. Checkpoint Checkpoint will create a faster index for future proof retrieval.

Adaptive DLP. Except for scheduling optimization, [1] proposes a method applying different DLP approaches in specific situations. Although this paper mainly focuses on the kernel utilization, we can treat binary translation processing elements as enormous kernels inside. Data partitioning can be done according to the properties of basic blocks. If the basic blocks are comprised of small items, machine tends to adopt kernel partitioning to fit with the instructions. In contrary, basic blocks with big items are appropriate for inter-kernel strategy.

Kernel Improvement. [2] introduces a solution efficiently applying kernel computation. It duplicates the kernels with potential following computations (multiplication). Machine operates the sequential computation in parallel that decreases the time spending on recursive computations. The additional computation cost is acceptable in exchange of performance reinforcement. [1] also includes improvements about kernels. It demonstrates how to operate kernel partitioning. Small pieces of kernels help accelerate data fetching and computations due to shorter distance between PE and targets.

SIMD Exploitation. With the support of SIMD engine, such as ARM NEON and Intel AVX, the main processor could utilize vectorized code regions to realize DLP. Along with the dynamic Assembler we could extend the DLP to a dynamically detection at the run-time execution. The approach could be achieve by vectorizing the instruction set architecture (ISA) or convert the short instruction into longer instruction.

Previously-mentioned methods are seemingly solutions to deal with assembly sequences. Machine should pay extra cycles to complete their preliminary works. This cause them not capable of universally adopted. In spite that there are approaches available to address the loss in translation, they do not have a in-depth research throughout the whole computer architecture. Current papers simulate the data stream without taking penalty and exceptions into account. They have deficiency though, we can still figure out more and more possibilities following their directions. Notable advances will be made if the algorithm of DLP is optimized and avoid exceptional hardware efforts.

4 MECHANICS WHILE EXECUTING BINARY TRANSLATION

Binary Translation Improvement. A Hybrid Multi-Target Binary Translator (HMTBT) was presented in [3] proposing the architecture of dynamically selecting the best performance before executing the Instruction-level Parallelism (ILP) engine or Data-level Parallelism (DLP) engine on the chosen target accelerator would create the adaptable solution for different application. ILP is exploited by using a CGRA engine through the process of (A) Dependency Analysis (B) Mapping (C) Configuration Build. DLP is exploited using the ARM NEON engine to operate the process of the binary translation on loop detection process when the iteration is unknown during compile time. After the ILP and DLP optimization of the code region (Mutual Regions), a dispather will receive CGRA configuration and SIMD instruction to decide which acceleration is dispatched in the Translation Cache. In addition, a History Table is used to stored the optimized code region for future use.

In [4], researcher also proposes a dynamic SIMD Assembler (DSA) generating the SIMD instruction to trigger the ARM NEON engine by detecting the vectorized code regions. The need of the DSA to detect the vectorized loop run-time is to compensated the weakness of the ARM auto-vectorization compiler count loop detection at compile time. Such condition as dynamic range loop, conditional loop and loop with a function call are three condition that the compiler couldn't possibly has optimized; thus, it will required the assistance from the run-time engine to parallelize the execution.

Combining the result in [4] and [3] both utilizing ARM NEON DLP engine to accomplish DLP, we could see that NEON couldn't have became effective without the co-existence of CGRA ILP engine. The effect of ARM NEON is only limited to certain scanerio and the performance enhancement is not as outstanding than the CGRA engine along. The CGRA engine could have operated in certain scenario with certain amount of optimization, but with the help of ARM NEON, the result could reach optimal in most of the scenario. That is to say, the combination of both of the ILP and DLP could create an effect better than both operates individually combined.

Different from [4] and [3], [5] proposed the binary translation between the AVX ISA, with 256-bit register wide, and the x86 SSE ISA, with 128-bit register wide, could become beneficial to software developer by eliminating the migration effort from older architecture to newer architecture. The paper proposed to use the static loop detection mechanism at compile time to translate the ISA to a lower-cost processor which create a future opportunity for software programmer to effortlessly migrate the old version application to the new version architecture when the technology would have dramatically improved.

An additional case has been taken into account when the loop exist the loop-carried dependency stating that such dependency will limit the parallelism factor for the program. Consider the following case

```
for (i = 0; i < 400;i++)
  a[i+4] = a[i] + 2;
```

The dependency will limit the maximum parallelism to a factor of 4 only.

Power Efficiency. Not only could we see that the the speed-up from exploiting DLP has outstanding effect but also energy saving has been reduced enormously especially with the highly data-dependent application in the [6]. Those research [5] which use static compiler analyzer to exploit DLP has not expressed their advantage over the energy consumption. Therefore, our survey is concluded that run-time dynamic binary translator could have a huge advantage on saving energy consumption.

Machine Learning Applications. Most binary translations are manually manipulated, including parameters, validation, and inspection. [7] demonstrates a novel perspective of binary translation that operating under unsupervised machine learning. Training and testing dataset is formed the same compiler. Input the dataset to recurrent neural network (RNN), which is universally used for natural language processing (NLP). After a couple training procedure, machine can determine the targeted assembly code at the accuracy of 95%. The performance is said to be enhanced if utilizing more powerful neural networks, e.g. long short-term memory (LSTM). Nevertheless, problems accompany with novel ideas. How to verify the semantical equivalence between two structures and how to compile useful dataset for training are two major concerns. We can imagine that the performance of binary translation will be significantly improved once the proper solution is proposed. It is a promising lecture that we can try to conquer.

We have presented the role of binary translator in different scenario ranging from the dynamically SIMD assembler, static loop detection mechanism and several application. The importance of binary translator in data-level parallelism is indispensable.

5 CONCLUSIONS

After evaluating the enhancements and implementation difficulties, we propose our view of possible solutions on this topic. Intra-window with partial profile provides enough information for machine to execute DLP operations. Dynamic scheduling is able to utilize the partial profile with data dependencies and balance the workload between processors. Lastly employs HMTBT to pursue significant translation performance. Integrating these three novel approaches into a whole machine requires huge efforts. We believe this concept could be a prospective direction to forward researches.

Our survey shows that DLP assists the system performance either in aspect of energy or speedup perspective. Considering computer structure limitations in overall could we find the best architecture to our targeted design. We aim to fulfill a DLP optimization with more practical constraints, e.g. memory dependencies, cold start overhead, data stream in reality with the proposed solutions. By doing so, unexpected conditions have a chance to be addressed. Probably, novel methods will be realized while we address them.

6 APPENDIX

Below is Authors and contributions

Chun-Sheng Wu: Beforehand Information Fetching, Data Partitioning Strategy, Kernel Partitioning Strategy, Scheduling Optimization, Adaptive DLP and Machine Learning Algorithm (RNN).

Ting-Wei Su: Binary Translation Optimization, Dynamic Binary Translation, SIMD Optimization.

7 REFERENCE

[1] Lili Song et al. "C-Brain: A Deep Learning Accelerator That Tames the Diversity of CNNs through Adaptive Data-Level Parallelization". In: *Proceedings of the 53rd Annual Design Automation Conference*. DAC '16. Austin, Texas: Association for Computing Machinery, 2016. ISBN: 9781450342360. DOI: 10.1145/2897937.2897995. URL: https://doi.org/10.1145/2897937.2897995.

[2] Franjo Plavec, Zvonko Vranesic, and Stephen Brown. "Exploiting Task- and Data-Level Parallelism in Streaming Applications Implemented in FPGAs". In: *ACM Trans. Reconfigurable Technol. Syst.* 6.4 (Dec. 2013). ISSN: 1936-7406. DOI: 10.1145/2535932. URL: https://doi.org/10.1145/2535932.

[3] Tiago Knorst et al. "An energy efficient multi-target binary translator for instruction and data level parallelism exploitation". In: *Design Automation for Embedded Systems* (Jan. 2022), pp. 1–28. DOI: 10.1007/s10617-021-09258-6.

[4] Ding-Yong Hong et al. "Exploiting Longer SIMD Lanes in Dynamic Binary Translation". In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. 2016, pp. 853–860. DOI: 10.1109/ICPADS.2016.0115.

[5] Nabil Hallou et al. "Dynamic re-vectorization of binary code". In: *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 2015, pp. 228–237. DOI: 10.1109/SAMOS.2015.7363680.

[6] Michael Guilherme Jordan et al. "Boosting SIMD Benefits through a Run-time and Energy Efficient DLP Detection". In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2019, pp. 722–727. DOI: 10.23919/DATE.2019.8714826.

[7] Fei Zuo et al. "Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs". In: *Proceedings 2019 Network and Distributed System Security Symposium* (2019). DOI: 10.14722/ndss.2019.23492. URL: http://dx.doi.org/10.14722/ndss.2019.23492.