World Journal of Innovation and Modern Technology, Vol. 7, Issue 6, (Dec) 2024
ISSN 2682-5910
DOI: 10.53469/wjimt.2024.07(06).11

A Task Parallel Accelerator with Dynamic Pipeline Balancing

Tianyuan Xu, Yihan Wang, You Zhang, Yuang Lu

University of Michigan, Ann Arbor

Abstract: ***Coarse-grained reconfigurable array(CGRA) based ac- celerator is a promising architecture to accelerate data processing workloads. CGRA-based accelerator features more flexibility in adopting various workloads while remaining powerful in comparison to the traditional application specified accelerator. However, the strength of CGRAs is limited by irregular data access and dependence patterns. The previously proposed task- based execution model enabled work-aware dynamic scheduling, but the effectiveness is still limited by the regular execution resources. To address these issues, we proposed a heterogeneous architecture to improve parallelism for pipeline-enabled task streaming. We enabled dynamic pipeline balancing on this architecture with minor modifications to the task stream annotation. We compare the execution result on various heterogeneous structures with the regular configuration. Overall, we find that our architecture can improve performance with the same overall resources.***

Keywords: Parallel Accelerator, Dynamic Pipeline Balancing, Coarse-grained reconfigurable array(CGRA).

1. INTRODUCTION

With the slowing improvements in silicon process and general purpose processors, reconfigurable accelerators are becoming a more popular option as a midpoint in the trade-off of generality and data throughput. While dataflow accelerators [4], [5], [7], [10] or CGRAs [2], [6], [8], [9], [11] have become increasingly successful in emerging applications, they are still fairly limited to regular computation [3].

With the demand for accelerating irregular workload, the TaskStream execution model has been proposed recently to exploit irregular parallelism, also known as task parallelism [1]. By supporting task parallelism, wider applicability is enabled in the case of parallel inherent data dependencies. Moreover, plenty of resource demands usually can only be determined during runtime, which means they need dynamic workload balancing. Furthermore, due to the difference in workload types, task parallelism can bring better utilization of the computation, memory, and network resources.

Task parallelism is normally not supported in common coarse- grained reconfigurable accelerators. Since such designs are aimed to exploit the structural parallelism across time or units, traditional reconfigurable accelerators are having a hard time harvesting the task parallelism: the structural parallelism cannot be applied and the benefits of accelerators are lost.

In this study, a reconfigurable accelerator with the support of task parallelism is implemented. Using abstracted primitives for dynamic task management and structured access, the accelerator can dynamically schedule resources with load balancing. The introduction of the task stream also enabled pipeline data reuse, decreasing the need for data memory access and synchronization.

While implementing the accelerator, we discover that using a heterogeneous design could potentially further exploit task parallelism. Due to the unbalanced workload and data dependencies, the hardware utilization is limited by the slowest task on-chain, limiting the overall system throughput. By deploying different sizes of computation units, memory access units, and bandwidth, we can further balance the pipeline, thus achieving a higher hardware utilization and system throughput, as shown in Figure 1.

The remainder of the paper will delve into the specifics of the TaskStream model, optimization strategies, implementation details, evaluation results, and possible applications, providing a comprehensive overview of the contributions and findings of our study.

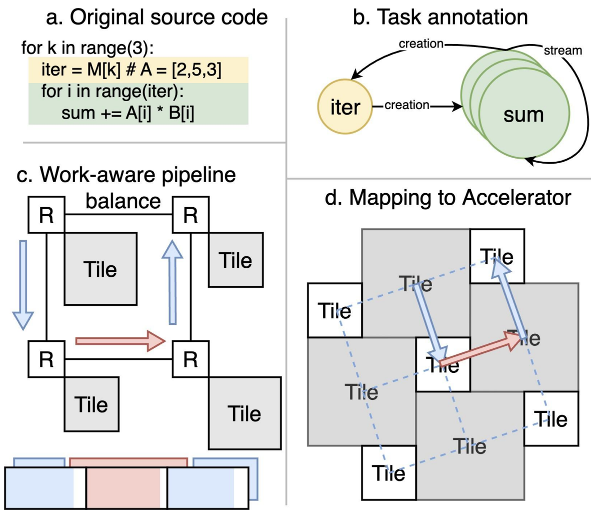


Figure 1: Overview of Task Stream and Pipeline Balancing.

2. BACKGROUNDS

TaskStream is a task parallel execution model that detects and exploits structure recovery using user labeled program. In theory, TaskStream may be used to extend a number of designs, but its simplicity and load balancing technique make it particularly well suited to reconfigurable accelerators.

2.1 TaskStream Model

A TaskStream program is made up of nodes (one for each job type) and edges (for inter-task dependencies). There are three types of edges: creation, streaming and batching. There are three states for each task: 1. Created: a task instance's parameters are built on the originating core; 2. Scheduled: the task is tied to execution resources. 3. Executing: the task is currently being computed.

Task creation: When a task creation edge connects two task nodes, it signifies that part of the source node task's outputs are utilized to activate the edge and will be input to the destination node task. The task creation edge may be marked with a sizeHint, which is a task parameter that defines the relative amount of labor for the job. This label enables the system to select the least loaded core to be the target core that processes the task.

Task Streaming: The output at the source node task activates a streaming communication with the assigned children when two nodes are connected by a streaming edge. The programmer can define a dependency distance for a streaming edge, which makes the connected tasks separated by a predetermined number of tasks.

Task Batching: A task batching edge is used to facilitate the multicasting of shared reads. When this edge is turned on, it requires three parameters: The DataID identifies if the reads are to the same data, the TaskID identifies the dependent dynamic task, and the bytes identify the read length. This enables the task scheduler to track which jobs are waiting for the same reading and reorganize them to be scheduled together. In this way, data may be broadcast to all co-scheduled processes, which is a benefit.

2.2 TaskStream Programming

One of the main benefits of programming in TaskStream is that it can handle efficient task scheduling with only a little input from the programmer.

If a code section performs different calculations, the same computations at a different pace, or has different locality behavior, it should be allocated to a new task type when porting a program. Two calculations, for example, might be integrated into a single job task to increase task granularity, or divided if the data associated with the two computations is not anticipated to behave similarly in terms of data locality.

The programmer must first define the instruction level dataflow graphs for each job type. Using the coreMask, the programmer may assign certain cores to that job type, as well as define a task parameter as a sizeHint.

The programmer will then use algorithmic expertise to locate edges between task nodes. These include assessing if any task computation is triggering another task calculation, whether the tasks are pipelined, and whether shared read-data exists across tasks. Another crucial step is to determine a type's task parameters and then establish an edge interface connecting producer ports at the source task to consumer ports at the destination task.

Finally, in Matrix-tile tasks, the number of recursive tasks formed by the self loop must be limited. This is accomplished by determining the maximum dependence chain length de- pending on hardware resource constraints.

2.3 Dataflow Accelerators

There are several models in market that utilizes similiar con- cepts, which are commenly known as dataflow accelerators.

Google Tensor Processing Unit: TPUs use a systolic array architecture to perform matrix multiplications efficiently. This architecture allows for high data reuse and minimizes data movement. It also leverages temporal dataflow paradigms, where data flows through the array in a synchronized manner, maximizing throughput and minimizing latency. TPUs are highly optimized for matrix operations and deep learning tasks but are less flexible for other types of computations.

NVIDIA's Data Processing Unit (DPU), particularly the Blue- Field series, is designed to offload and accelerate data center workloads by combining high-performance networking, stor- age, and security capabilities into a single chip. While DPUs excel in specific tasks like network processing and security, they have limitations compared to the TaskStream execution model. DPUs are highly specialized and may not adapt well to diverse and dynamic workloads, whereas TaskStream offers broader flexibility and dynamic resource allocation. Addition- ally, DPUs can introduce overhead and complexity in reconfig- uration, whereas TaskStream simplifies resource management and scheduling, making it more suitable for handling irregular and rapidly changing workloads.

Microsoft's Catapult project uses FPGAs to boost data center performance for tasks like Bing search and AI applications. However, compared to TaskStream, Catapult faces limitations such as reconfiguration overhead and complexity, making it less adaptable to dynamic workloads. Additionally, the fixed hardware resources of FPGAs can limit scalability and efficient resource utilization. TaskStream, on the other hand, offers dy- namic resource allocation and efficient load balancing, making it better suited for diverse and irregular workloads.

2.4 Limitations And Solutions

TaskStream model assumes that all computational tiles are of equal capacity. However, when the user annotated programs share a wide range of computational loads, this design could be sub-optimal. In this study, a heterogeneous design is proposed so that tiles are equipped with different computing resources and memory bandwidth. The TaskStream model is modified accordingly to enable fast task processing and efficient load balancing.

3. OPTIMIZATION STRATEGIES

The accelerator model is an innovative approach that aims to exploit task parallelism. This type of parallelism occurs when a program's tasks are created and scheduled dynamically based on runtime computations, rather than being predefined. This dynamic nature allows for more efficient handling of workloads that have inherent data dependencies and varying resource demands. From the high level, our design supports the following features.

3.1 Dynamic Task Management

The accelerator allows for tasks to be created dynamically during runtime, unlike traditional models where tasks are predefined. This means that as the program executes, new tasks can be generated based on the current state and data, allowing the system to adapt to changing workloads. The model includes sophisticated algorithms to schedule these dynamically created tasks efficiently. The scheduler must consider factors like task dependencies, resource availability, and workload balance to ensure optimal performance.

Our accelerator dynamically distributes tasks across available resources to ensure even load distribution. This prevents some resources from being overburdened while others remain idle, maximizing overall system efficiency. The system continuously monitors the workload and makes real-time adjustments to the task distribution. This dynamic balancing helps in maintaining high performance even as the workload changes.

3.2 Inter-Task Dependencies

Our accelerator supports the co-scheduling of dependent tasks, allowing for efficient data transfer between them. This reduces the need for intermediate storage and minimizes the overhead associated with data movement. By reusing data within the pipeline, our model reduces the need for repeated memory access. This not only speeds up processing but also lowers energy consumption.

The model aims to minimize the synchronization overhead between tasks. Excessive synchronization can introduce delays and reduce the benefits of parallelism, so our accelerator employs techniques to keep this overhead low. our model ensures that tasks are executed in the correct order and that data dependencies are respected. This requires careful coordination to avoid conflicts and ensure smooth execution.

3.3 Resource Allocation

Our model dynamically allocates resources such as computation units, memory, and bandwidth based on the current workload. This real-time management ensures that resources are used efficiently and that the system can adapt to changing demands. The model supports flexible utilization of resources, allowing for different types and sizes of computation units to be deployed as needed. This flexibility helps in handling a wide range of workloads efficiently.

Our accelerator can manage a heterogeneous set of resources, including different sizes of computation units and varying memory access units. This diversity allows the system to better match resources to the specific needs of each task.By deploying a mix of resource types, our model can optimize performance for different types of tasks. This heterogeneous approach ensures that the system can handle both regular and irregular workloads effectively.

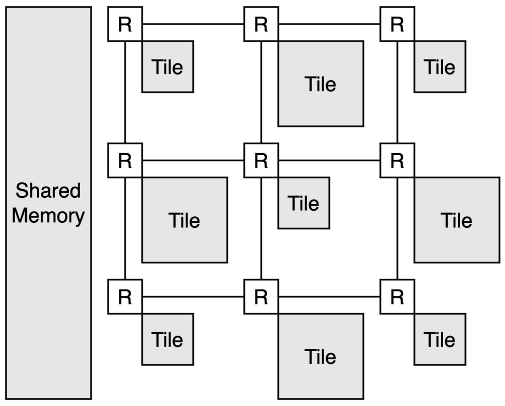


Figure 2: Topology of the interconnection network.

3.4 Pipeline Balancing

Our accelerator continuously adjusts the pipeline to ensure that no single task becomes a bottleneck. This dynamic adjustment helps in maintaining a smooth flow of tasks through the pipeline, improving overall system throughput. By balancing the workload across the pipeline, our design ensures that tasks are executed efficiently. This reduces delays and maximizes the utilization of available resources.

The model enables the reuse of data within the pipeline, decreasing the need for repeated memory access. This not only speeds up processing but also reduces the energy consumption of the system. Efficient data reuse also improves synchronization between tasks, as less time is spent waiting for data to be fetched from memory. This leads to faster and more efficient task execution.

4. IMPLEMENTATION DETAILS

Figure 2 shows the top-level architecture of the task parallel accelerator. The accelerator used a 3x3 mesh topology interconnection network. Each tile in the accelerator has a dedicated scheduler, a CGRA array, and a shared memory interface, as shown in Figure 3. The difference between tiles is the available arithmetic and memory units inside CGRAs and the memory bandwidth along with each tile. The most important modules, the scheduler and CGRA, will be detailed in the following sections.

4.1 Scheduler

Schedulers receive messages from the router, control core, and CGRA. It serves as an intermediary agent, forwarding data, acknowledgment, and scheduling tasks based on messages.

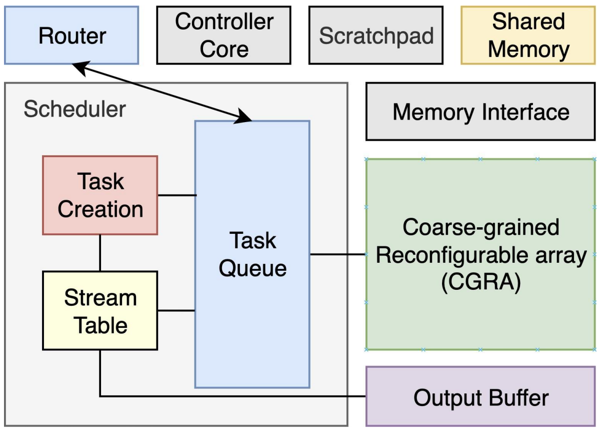


Figure 3: Architecture of a single tile.

The main function of schedulers is to schedule variable-size tasks and streaming tasks to CGRAs.

For variable-size tasks, the scheduler keeps track of each core's current bearing workload. When scheduling new tasks, it uses these values as a reference to decide which core to assign new tasks so that each core will process almost the same workload.

For steaming tasks, Schedulers divide a single task into different parts and sent it to different cores. Therefore, many streaming tasks of the same type can be efficiently pipelined. For best throughput and avoid tasks being stuck in a certain core, the total time for each core to compute their part of a task and forward the result to the next core should be almost the same. Therefore, two different scheduling algorithms, a distance-aware algorithm, and a workload-aware are created to determine which cores are best to participate in streaming tasks.

The distance-aware algorithm is used in two scenarios, either the workload of different parts is evenly distributed, or the accelerator is homogeneous, e.g., different cores of the accelerator have the same compute capabilities. In either case, the time cost of forwarding results to the next core plays a more important role in the latency of different stages. Thus, it is better to schedule different tasks in adjacent cores. Our Distance-ware algorithm finds cores that can form the pipeline with the lowest communication cost. It applies a greedy algorithm to find the nearest core that has the minimum compute capabilities to the previous core.

The workload-ware algorithm is suitable for streaming tasks with a biased workload on heterogeneous accelerators. In this case, the scheduler will find cores with the highest compute capabilities and assign different stages to them accordingly.

4.2 CGRA

CGRA or coarse-grained reconfigurable array is the unit that performs task execution. CGRA has two types of resources, arithmetic and memory resources. Arithmetic resources repre- senting the available computation units are crucial in computation workloads. Memory resources representing load-store units associated with memory bandwidth are important in memory-bound workloads.

To exploit data pipeline balancing, different strengths of CGRA are selected in terms of arithmetic and memory resources. When CGRA receives a task activity from the scheduler, it will first check its available resources for executing the incoming task. If its available resources could fulfill the task needs. By doing so, different tasks or the same tasks with different taskids could be executed concurrently to achieve higher hardware utilization.

To alleviate programming difficulties, the number of resources in a strong core will always be a factor times greater than in a weaker core. Thus, it ensures programs written for the weaker cores could easily be adopted and accelerated by the stronger cores.

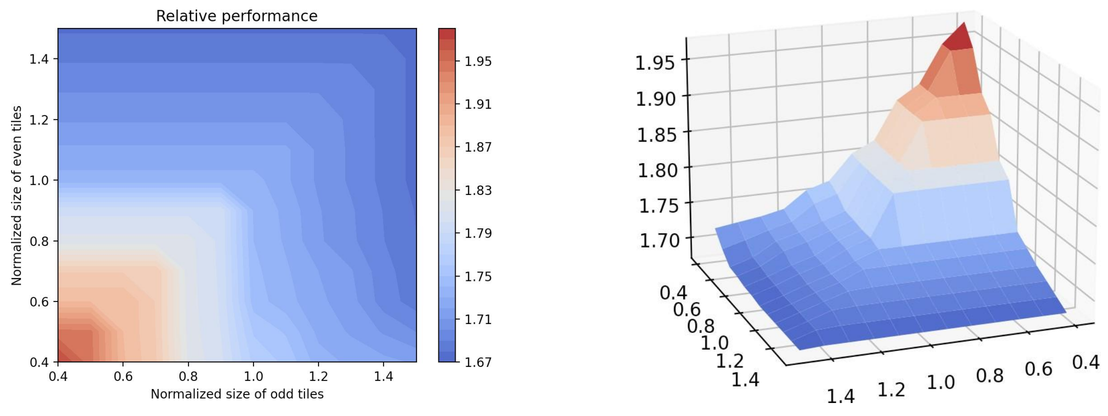
5. PERFORMANCE RESULTS

For flexibility and customizability, we constructed our eval- uation infrastructure in hardware fashion via Python. The evaluation platform is synchronized using hardware cycle counts. A delay of cycles is also added for simulating real world hardware behavior. The goal of our evaluation is to find whether our proposed data pipeline balancing can increase throughput and achieve higher hardware utilization rates with the same overall resources. Particularly, we examined the Cholesky decomposition in which a mix of task creation and different sized data pipeline streaming is presented. For the topology of heterogeneous, we focused on the square tessellation topology shown in Figure 1 as it can easily adapt to widely used mesh topology with varied sizing abilities.

5.1 Case study: Cholesky decomposition

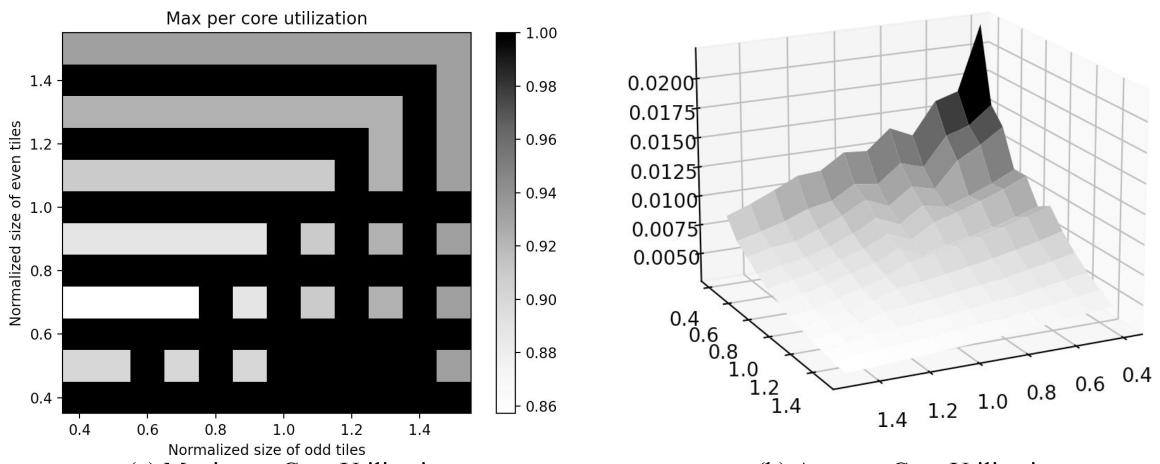
The Cholesky decomposition is a decomposition of a Her- mitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose, which is useful for efficient numerical solutions. Cholesky is challenging because it involves a variable size triangular iteration. Three types of tasks are defined for Cholesky: the Point task performs an inverse and a square root on one element in the diagonal of the matrix; the Vector task performs multiplication on an array of the matrix; the Matrix task performs a triangular iteration and subtractions. Since there is back propagation in Matrix subtraction, a pipeline data reuse edge is defined and the else of edges are defined as creation edges. A sizeHint variable is computed in the Point task and used by the scheduler to perform dynamic data pipeline balancing.

We perform 512 times three-by-three matrix Cholesky decom- position, recording its overall cycles used along with hardware utilization rates for evaluation. Two different sized tiles in terms of CGRA arithmetic units and memory bandwidth are predefined. We simulated all possible sizing in both odd or even cores for clearer demonstration. The strength of the core is normalized and the result cycles is on log-based scaling. The performance result are shown in Figure 4.



(a)Relative Performance in 2D (b) Relative Performance in 3D

Figure 4: Performance Comparison with different core strength.



(a) Maximum Core Utilization (b) Average Core Utilization

Figure 5: Utilization Comparison with different core strength.

As shown in Figure 4(a), the time to finish the simulation is decreased with the increase of cores size. Along with the line where the sum of odd core size and even core size are the same, the lower spot between uniform size and huge unequal size shows the benefit of heterogeneous dynamic data pipeline balancing. In a uniformed system, performance is limited by the stronger core, but in a largely unequaled system, performance is instead limited by the weaker core. In the given example system, a ratio of 2.3x gets the highest performance.

To explain the strip in the performance result, we examined the peak utilization rates as well as the avg utilization rates. As shown in Figure 2, as the size of the core increases, the performance can't increase until it reaches a point where paralleling another task is allowed. The asymmetry of the plot also shows the benefit of dynamic pipeline balancing on utilization.

However, given the limitation of a single case study, we can't report the overall benefit of heterogeneous in various task cases with different access and compute patterns. A more convincing model for result evaluation as well as plenty of case studies are needed to fully explore the benefit of heterogeneous dynamic pipeline balancing. Further studies are also needed in combination with topology and reconfigurability.

6. STRENGTHS AND ADVANTAGES

6.1 Broader Applicability

TaskStream is designed to handle irregular workloads that do not fit neatly into predefined structures. This flexibility makes it suitable for a wide range of applications, from scientific computing to real-time data processing. The model can dynamically adapt to changing workloads, ensuring that it can efficiently manage tasks with varying sizes and dependencies. This adaptability is crucial for applications where the workload is unpredictable and varies over time.

6.2 Improved Resource Utilization

TaskStream dynamically allocates resources based on the cur- rent workload, ensuring that computation units, memory, and network resources are used efficiently. This prevents resource underutilization and maximizes overall system performance. By dynamically balancing the workload across available re- sources, TaskStream ensures that no single resource becomes a bottleneck. This even distribution of tasks leads to higher efficiency and better performance.

The model enables the reuse of data within the pipeline, reducing the need for repeated memory access. This not only speeds up processing but also lowers energy consumption. TaskStream minimizes the overhead associated with synchronization between tasks. This efficient coordination ensures that tasks are executed smoothly, without unnecessary delays.

6.3 Enhanced Performance

By dynamically managing tasks and resources, TaskStream can optimize the execution of tasks, leading to faster process- ing times. This is particularly beneficial for applications that require real-time or near-real-time performance. The model continuously adjusts the pipeline to ensure that tasks are executed efficiently. This dynamic balancing helps in maintaining high throughput and reducing processing delays.

TaskStream's sophisticated scheduling algorithms ensure that tasks are executed in an optimal order, maximizing system throughput. This leads to better performance, especially in applications with high computational demands. By deploying a mix of resource types, TaskStream can handle a wide range of tasks more effectively. This heterogeneous approach ensures that the system can achieve higher throughput and better performance.

6.4 Energy Efficiency

By reusing data within the pipeline, TaskStream reduces the need for frequent memory access, which is a major source of energy consumption. This leads to more energy-efficient processing. The model ensures that resources are used efficiently, preventing wastage and reducing overall energy consumption. This is particularly important for applications where energy efficiency is a critical concern.

6.5 Scalability

TaskStream is designed to scale efficiently, allowing it to handle larger workloads without a significant drop in performance. This scalability is crucial for applications that need to process large volumes of data or perform complex computations. The model includes efficient communication mechanisms to handle the increased communication overhead that comes with scaling. This ensures that the system can maintain high performance even as the workload grows.

7. APPLICATIONS

7.1 Graph Processing

Analyzing large social networks involves processing graphs with millions of nodes and edges. TaskStream can efficiently handle the irregular nature of these graphs, where the degree of each node (number of connections) varies significantly. This allows for faster computations in tasks like community detection, influence maximization, and recommendation systems. In bioinformatics, analyzing protein-protein interaction networks or gene regulatory networks requires handling complex, irregular graphs. TaskStream can dynamically allocate resources to different parts of the network, improving the speed and accuracy of these analyses.

7.2 Machine Learning and AI

GNNs are used for tasks like node classification, link predic- tion, and graph classification. These networks often involve irregular computations due to varying node degrees and graph structures. TaskStream can optimize the execution of GNNs by dynamically balancing the workload and reusing data within the pipeline. Many machine learning algorithms involve sparse matrix operations, where most of the elements are zero. TaskStream can efficiently manage these operations by dynamically scheduling tasks based on the non-zero elements, leading to faster and more efficient computations [1].

7.3 Scientific Computing

Simulating the interactions between molecules involves complex computations with varying workloads. TaskStream can dynamically allocate resources to different parts of the simulation, improving the overall performance and accuracy. Simulating the behavior of celestial bodies involves handling large, irregular datasets. TaskStream can optimize these simulations by dynamically balancing the workload and reusing data, leading to faster and more accurate results.

7.4 Data Analytics

In applications like financial analytics, real-time data processing is crucial. TaskStream can handle the irregular nature of incoming data streams, dynamically allocating resources to process data as it arrives. This ensures timely and accurate analysis. Analyzing large datasets often involves irregular workloads due to varying data distributions. TaskStream can dynamically balance the workload, improving the efficiency and speed of big data analytics.

7.5 Network Security

Detecting network intrusions involves analyzing large volumes of network traffic, which can be highly irregular. TaskStream can dynamically allocate resources to different parts of the network, improving the speed and accuracy of intrusion detection. Cryptographic algorithms often involve irregular computations. TaskStream can optimize these computations by dynamically scheduling tasks and reusing data, leading to faster and more secure cryptographic operations.

8. CONCLUSION

In summery, the proposed reconfigurable accelerator with task parallelism suuport offers a promising solution for handling irregular workloads. By dynamically managing tasks and balancing resources, this approach can significantly improve system performance and utilzation. In our study, the proposed accelerator can be efficient in Cholesky decomposition with the same overall resources limitation. Further studies is need to fully examine the benefit of heterogeneous dynamic pipeline balancing with different topology and physical reconfigurabil- ity.

REFERENCES

[1] V. Dadu and T. Nowatzki, “TaskStream: accelerating task-parallel workloads by recovering program structure,” in Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022, pp. 1–13. doi: 10.1145/3503222.3507706.

[2] J. Cong, H. Huang, C. Ma, B. Xiao and P. Zhou, “A Fully Pipelined and Dynamically Composable Architecture of CGRA,” 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, 2014, pp. 9-16, doi: 10.1109/FCCM.2014.12.

[3] M. Wijtvliet, L. Waeijen and H. Corporaal, “Coarse grained reconfig- urable architectures in the past 25 years: Overview and classification,” 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016, pp. 235-244, doi: 10.1109/SAMOS.2016.7818353.

[4] T. Nowatzki, V. Gangadhar, N. Ardalani and K. Sankaralingam, “Stream- dataflow acceleration,” 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 416-429, doi: 10.1145/3079856.3080255.

[5] R. Prabhakar et al., “Plasticine: A reconfigurable architecture for parallel patterns,” 2017 ACM/IEEE 44th Annual International Sym- posium on Computer Architecture (ISCA), 2017, pp. 389-402, doi: 10.1145/3079856.3080256.

[6] S. A. Chin et al., “CGRA-ME: A unified framework for CGRA modelling and exploration,” 2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2017, pp. 184-189, doi: 10.1109/ASAP.2017.7995277.

[7] M. Vilim, A. Rucker and K. Olukotun, “Aurochs: An Architecture for Dataflow Threads,” 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), 2021, pp. 402-415, doi: 10.1109/ISCA52012.2021.00039.

[8] C. Torng, P. Pan, Y. Ou, C. Tan and C. Batten, “Ultra-Elastic CGRAs for Irregular Loop Specialization,” 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 412-425, doi: 10.1109/HPCA51647.2021.00042.

[9] D. Liu et al., “Data-Flow Graph Mapping Optimization for CGRA With Deep Reinforcement Learning,” in IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems, vol. 38, no. 12, pp. 2271-2283, Dec. 2019, doi: 10.1109/TCAD.2018.2878183.

[10] W. Lu, G. Yan, J. Li, S. Gong, Y. Han and X. Li, “FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks,” 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 553-564, doi: 10.1109/HPCA.2017.29.

[11] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram and M. Shafique, “X-CGRA: An Energy-Efficient Approximate Coarse-Grained Reconfig- urable Architecture,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 2558-2571, Oct. 2020, doi: 10.1109/TCAD.2019.2937738.