Accelerating the Task Activation and Data Communication for Dataflow Computing

Zheng Du
Department of Computer Science and Technology,
Tsinghua University, Beijing, China
douz18@mails.tsinghua.edu.cn

Wenjie Zhao
College Of Computer Science and Software Engineering,
Shenzhen University, Shenzhen, China
1910272042@email.szu.edu.cn

Zhiwei Wen
College Of Computer Science and Software Engineering,
Shenzhen University, Shenzhen, China
2018133007@email.szu.edu.cn

Qiuming Luo*
College Of Computer Science and Software Engineering,
Shenzhen University, Shenzhen, China
lqm@szu.edu.cn

ABSTRACT

The hybrid dataflow/von-Neumann [1] architectures may differ in implementations but all follow similar principles: they harness the parallelism and data synchronization inherent to the dataflow model, yet maintain the programmability of the von-Neumann model. In this paper, we raise a new kind of hybrid dataflow/von-Neumann architectures, which contains TAU (Task Activated Unit) and SPM [9] (scratchpad memory) components, by which we can enhance parallel efficiency. We also implement the prototype design, integrated with peripheral devices and verify the whole system on FPGA. Finally, we deploy operating system on the hardware system and profile the performance. The experimental results show that the performance is improved by 3.07%~10.32% under the random data flow graph, the performance of inter-core communication is improved by 4% and the hardware acceleration effect is achieved.

CCS CONCEPTS

• **Computer systems organization -> Architectures -> Parallel architectures -> Multicore architectures; Data flow machine.**;

KEYWORDS

Dataflow/von-Neumann architecture, parallel computation, scratchpad memory, RISC-V

ACM Reference Format:
Zheng Du, Wenjie Zhao, Zhiwei Wen, and Qiuming Luo*. 2022. Accelerating the Task Activation and Data Communication for Dataflow Computing. In *51th International Conference on Parallel Processing Workshop (ICPP Workshops '22), August 29–September 01, 2022, Bordeaux, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3547276.3548523

1 INTRODUCTION

Since the concept of hybrid dataflow/von-Neumann architectures have been raised by many computer pioneers, the explorations [3] of this kind of machines has been going on for decades. In order to enhance the efficiency of parallel computation in traditional platforms, the hybrid dataflow/von-Neumann machines insert many dataflow units, which are in the form of software or hardware, into the traditional von-Neumann system.

The idea of data flow computing is not abandoned by the industry like the pure data flow execution model, but further penetrated into the control flow execution model. The mixed heterogeneous multi-core architecture further improves the efficiency and performance of the control flow execution model. For example, CUDA Programming of GPU integrates data related instructions into an execution node, and each execution node executes asynchronously. The condition of execution is whether resources and data are ready. Therefore, the program can be disordered and executed quickly, which also reflects the idea of data flow calculation [16]. In addition, tensorflow [17], a deep learning framework, programmers need to explicitly describe the relationship between neural network layers, and construct a data flow graph from finite class operators to reduce the programming difficulty of the underlying acceleration library. Even some programming models of streaming processing also draw on the calculation idea of data flow, such as steam-kernel [14].

At the beginning of the 21st century, aiming at the advantages and disadvantages of the two execution models, the hybrid execution model of data flow and control flow has become the mainstream direction of this research topic. Researchers [5] have sought to play a balance between their strengths and complement their weaknesses, and proposed data flow architectures such as trips [4], ddm [19] and task superscalar [20]. Even, FPGA acts as an accelerator and provides interfaces through software and hardware APIs [21]; Non mainstream solutions based on data flow language description and automatic generation of multi-core processors in specific fields through software tools [22] have also been proposed from time to time. However, the concepts involved in data flow computing are huge after all. There are a lot of gaps in the underlying chip design, programming language specification, operating system and software application development [6], etc. the project research and development cycle are often very long, so the hybrid execution model is still on the way to explore.

Under this background, our team design a new kind of hybrid dataflow/von-Neumann machine, which innovatively contains TAU hardware and SPM. TAU is a hardware that can be set by special instructions, which can serve as activation links among dataflow

ICPP Workshops '22, August 29–September 01, 2022, Bordeaux, France
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9445-1/22/08.
https://doi.org/10.1145/3547276.3548523

Table 1: Comparison with other data flow processors ('/': not mentioned in the original paper)

	Task granularity	ISA	Optimize Scene	Generality
stream-dataflow [10]	Coarse granularity	Specific ISA	Repeat Pipeline Calculation	Wider
Flexflow [11]	Fine granularity	/	CNN Neural Network	narrow
SPU	Instruction Level Fine Granularity	RISC	Loop Iterative Data Flow Calculation	Wider
This Work	Thread Level Coarse Granularity	RISC-V	Multithread-based data flow	wide

tasks. SPM subsystem is used to replace traditional cache system, with which processor cores can directly communicate each other through shared memory space at the speed of cache, rather than the mix of cache and main memory. With the help of TAU and SPM, the communication between threads can be boost, which can enhance efficiency of parallel computation.

In this paper, we design and deploy the machine. The physical machine is based on RISC-V MINI architectures and the OS is derived from XV6. Our team expand the original RISC-V MINI architectures from single core into double core, and we also develop many peripheral devices to get input-output result. We perform physical experiment on FPGA hardware to test and verify this machine, during which we collect many data related to performance and efficiency. In summary, we raise a architecture in our new concepts, realize it and test its performance.

The RISC-V data flow architecture implemented in this paper is a multi-threaded programming data flow machine, with threading level tasks (thousands or thousands of instructions) as nodes, which are compared with other data flow processors in recent years as shown in Table 1 .Although RISC-V data flow architecture is not as granular as SPU's instruction level in optimizing parallel performance, it uses RISC-V instruction set which is popular in recent years, which is easier to process in compilation and facilitates scientific iteration. In the optimization scenario, RISC-V data flow architecture differs from flexflow in that it optimizes parallel computing only for CNN neural networks, but uses a common thread to describe tasks. In addition, one of the difficulties of data flow processors is that the corresponding operating systems and software interfaces are scarce in the market, and multithreaded programming-oriented data flows can be compatible with most control flow operating systems and have good scalability, that is, they have the advantage of wide usage.

The main contributions of this work compared to precious research are (1) we design a module of hybrid dataflow/von-Neumann architecture that contain TAU and SPM. (2) we implement it and test its preliminary performance.

2 SYSTEM ARCHITECTURE

The basic structure can be explained by the Figure 1. The processor core is provided by RISC-V MINI structure, which is a open source processor using RISC-V32I instruction set with three stages pipeline and is developed in chisel language by UC Berkeley.

The TAU is connected to both cores, and it can serve as a special link between threads by special operation, which can speed up the execution of dataflow task.

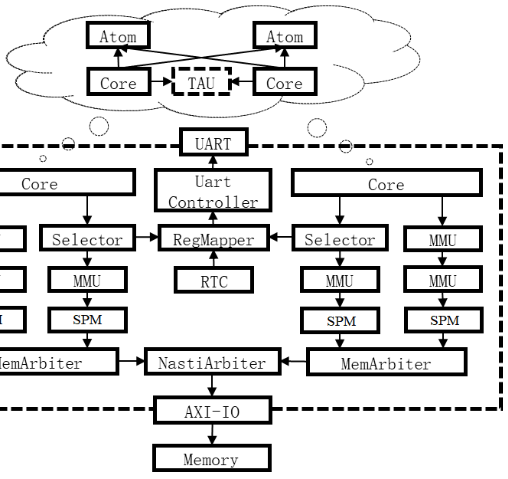

Figure 1: The system architecutre of the machine

The MMU stands for Memory Management Unit, and the Reg Mapper maps peripheral devices registers to memory address space, and the Atom stands for the atomic operation unitrelated to the mutex for operating system. What's more, necessary arbiter organizations are added, such likes the MemArbiter and NastiArbiter.

The SPM multiplexing the main storage address space, which means low address will corresponding to a set of storage unit for speed and high address will corresponding to the main storage, through address decoding unit for real time data synchronous.

As for the peripheral devices, the UART is used for interaction likes receiving command and output information. The RTC (Real-Time Clock) serves as system clock, which is a counter in essence and response to the scheduling time slice of the threads. Finally, the main storage is connected through AXI bus, which can flexibly be BRAM, DRAM and DDRAM etc.

3 TAU HARDWARE

This paper investigates the execution of coarse-grained data flows with the thread granularity size as the task node of the data flow. In the aspect of data flow programming, it uses data flow programming based on multithreaded programming, such as DFC data flow programming language [2]. However, when describing data flow diagrams in multithreaded programming, access synchronization control through mutexes is unavoidably required. For example, in the case of Figure 2, the structure description of task node F on

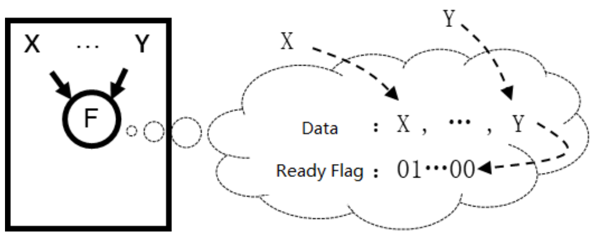

Figure 2: Example of data ready signal write conflict

the left often includes the dependent data chain and ready identification. When data X and Y are ready at the same time, they all need to modify the bits in the ready identity (assuming that the task node is ready when the bits are all modified to 1), and there is a write conflict.

To resolve access conflicts in multithreaded programming, pthread_can be used Mutex and other forms of mutex exclusion. However, the use of mutexes poses an efficiency issue, that is, it can cause user programs to crash, causing context switching, which can cause additional time overhead. This efficiency issue is one of the main starting points for designing dedicated hardware for data streaming.

This article considers storing task nodes'dependencies on data in a specific hardware structure instead of using a software-based ready identity. When both data ready signals occur at the same time, it acts as an out-of-core hardware module structure, not affecting the processor flow in progress, but using buffers to ensure that the ready signal is not lost. This structure is referred to as Task Activated Unit (TAU), in which the dependency of tasks on data determines the readiness of tasks, which can then be scheduled by the operating system

TAU (Task Activated Unit), mentioned in this paper, is independent expanded hardware unit, which stored the data dependency of task nodes among dataflow graph, with the data signal, data ready signal as input and task ready signal as output. Since it is based on muti-thread programming, task node can be represented by its PID of the thread. As shown in Figure 3, the structure of TAU is with 5 parts. The TAU is directly connected to processor cores, the DataReady Channel transports the data ready address, the Data Dependence Channel transports the PID as well as dependency data address, and the Interrupt Channel transports the ready node PID with state signal. TAU is consisted of buffer, Write Arbiter, Interrupt Arbiter, TDC (Task Dependence Counter) and TDU (Task-Data dependence Unit).

When TAU trace a dataflow task, Pid and data address will be written into TDU through the Data Dependency Channel and go by the TDC. During this process, the TDC counts the quantity of the unready dependency data of task node, and the TDU stores the dependency relation between two nodes. Since it is dual core structure, the Data Dependency Write Channel is connected to TDC through Write Arbiter, which means that when both cores about to write, the arbiter will allow only one core to perform the write operation, the other write will be temporarily stored in the buffer.

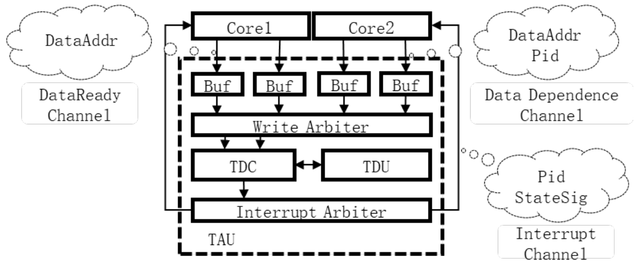

Figure 3: The basic structure of TAU

	31 25	24 20	19 15	14 12	11 7	6 0
	Saved	rs2	rs1	Funct3	rd	Opcode
TW	0000000	DataAddr	Pid	000	result	1110100
TR	0000000	DataAddr	00000	000	00000	1110101

Figure 4: The basic structure of the machine

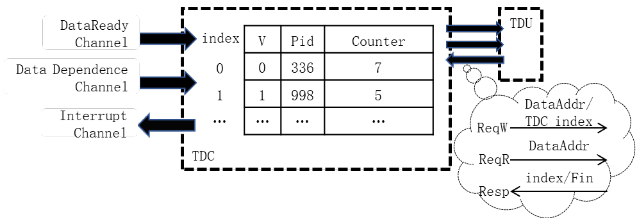

Figure 5: The basic structure of the TDC

Since it's hard to operate TAU in original RISC-V instruction set, we develop extended instructions, which can write PID and data address into TAU in one instruction to directly express the dependency between each other. The reason of why not use "store" to finish this job is that original "store" can only put one data into the address stored in register, and if using a series of "store" to finish it, the process will easily be interrupted and be hard to express the dependency between instructions.

In this case, we develop two instructions, TW and TR, following the principle of RISC-V Standard User Instruction Manual. The TW is for writing data to TAU, and the TR is for transporting data ready signal to TAU. As shown in Figure 4, the operate code "1110100" and "1110101" are unused in RISC-V32I, which means it is safe to extend instruction in that operate code.

The main function of TDC is counting the unready dependency data as well as sending interrupt signal and state signal to processor core. TDC is not only connected to TAU signal channel, but also connected to TDU through three signal channels. As shown in Figure 5, the TDC stores the valid bit, pid and dependency counter,

The main task of TDU (Task-Data Dependency Unit) is to store the counting values of unready dependency data of the tasks, and to find out the corresponding task node according to the data address provided by the TDC request, and to feed back the TDC-Index to the storage row where the task node is stored.

TDU only interacts with TDC, requesting no redundant description of the signal, while the Resp feedback signal channel contains

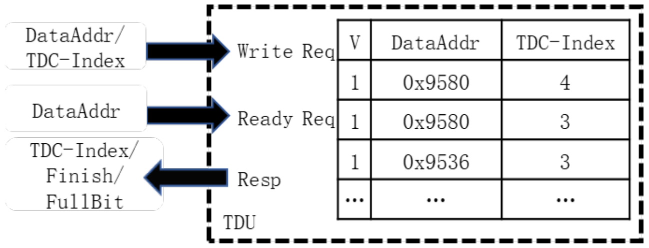

Figure 6: The basic structure of the TDU

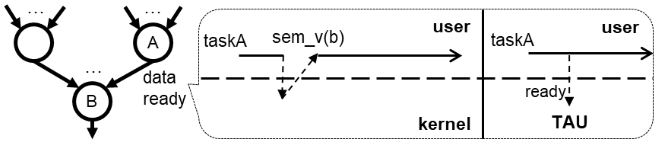

Figure 7: Trigger mode of data ready signal (no tau structure on the left; tau structure on the right)

three main signals: TDC-Index, Finish, FullBit. The Finish signal is used to notify the TDC to request the end of the query. The FullBit signal is used to determine whether the response is full when a TDC request is written.

The structure of TDU is shown in Figure 6, there are mainly valid bits V stored internally. TDC stored row ordinal TDC-Index (task node identity Pid unique correspondence), and DataAddr dependent data address. The relationship between TDU and TDC is many-to-many, because there is a case where data is dependent by multiple task nodes in the task description of data flow, and there is also a case where task nodes depend on multiple data, so there may be the same DataAddr value or TDC-Index value between the stored rows.

Describes coarse-grained data flow diagrams based on multi-threaded programming, as shown in Figure 7 on the left, where nodes perform tasks for threads. An edge pointing to a node represents the node's dependent data and is signaled ready by the start node of the edge. For example, when data ready information is sent out after task node A calculation is completed, the data ready identity of task node B needs to be modified - this data ready identity can be considered a semaphore for a P/V operation. Here task node B enters a sleep wait state when described in the data flow diagram, waiting for the ready identification, that is, the semaphore B wakes up. The semaphores for P/V operations are implemented by the kernel itself and are provided to user threads as system call interfaces. This results in the cost of context switching performed by the processor (involving register saving and stack space switching, etc.). With data flow hardware TAU, the data ready signal of task node A can be modified to the ready identity of task node B in TAU only by user instruction TR, without affecting the current execution stream, thus bringing about acceleration effect.

4 SPM SYSTEM

In the traditional cache system, the transfer of shared data and variables is delayed. Because of hits and misses in the cache, public data in memory cannot be synchronized to each processor core in a timely manner, so synchronization mechanisms are needed to ensure communication between different kernel threads, which greatly increases the overhead for data flow tasks.

In the SPM system, since there is no hit or miss, any processor kernel has the same number of memory cycles for reading and writing to the main memory. The first storage cycle is written to the critical zone of memory by a thread, and the second can be read by threads from other cores to achieve real-time communication between different processor cores. The only thing a user needs to do is to design a program mechanism to manage the space manually

The SPM (scratchpad memory) subsystem is consisted of storage array, decoding logic unit and IO unit, which is commonly used for embedded systems with low power consumption, small area and high real-time performance.

As shown in Figure 8, compared with traditional cache, SPM subsystem has a more concise structure, no tag storage and comparison logic, and main memory address fetch data directly through address decoding logic unit without considering complex issues such as hit ratio, swap in and out like cache. As for the spatial locality, users can manually manage the space of SPM to speed up the access of dataflow data.

As shown in the Figure 9, the left stands for inter-core communication through traditional cache system, delay of which is the sum of the delay of write back and refill. The right shows the inter-core communication through SPM subsystem, by which the CPU can directly access the SPM through address decoding logic and the delay is only two CPU cycles. So we can conclude that the efficiency of the SPM system is indeed higher than that of the traditional cache system in terms of inter-core communication alone,thus bringing about acceleration effect

5 SYSTEM PERFORMANCE

As we implement our design into FPGA and make operating system run on it, we develop and deploy many exams to test its performance.

The Arty A7-100T development board is the hardware used in this paper and is the product of Xlinx company. The company provides quite complete technical support such as the development tools and documentation of the field programmer, so its field programmer is often used in business, teaching and scientific research. The ATY A7-100T development board used in this paper is small in size, has high transmission signal bandwidth, contains 225KB block RAM, 256MB DDR3 storage, USB-UART bridge, single 100 MHz crystal oscillator and other hardware devices [15]. Block RAM can be initialized when writing a field bus, and the input frequency of 100 MHz can generate different clock frequencies via MMCMs and PLLs.

Vivado is the EDA tool used in this paper. We use chisel language [7, 13] to complete the overall hardware design, compile and generate Verilog [12] hardware description code, and store the test program and software in external memory

We loaded the riscv-xv6[18] operating system into a block ram on the development board of the field programmer that has burned the RISC-V data stream processor and run it.

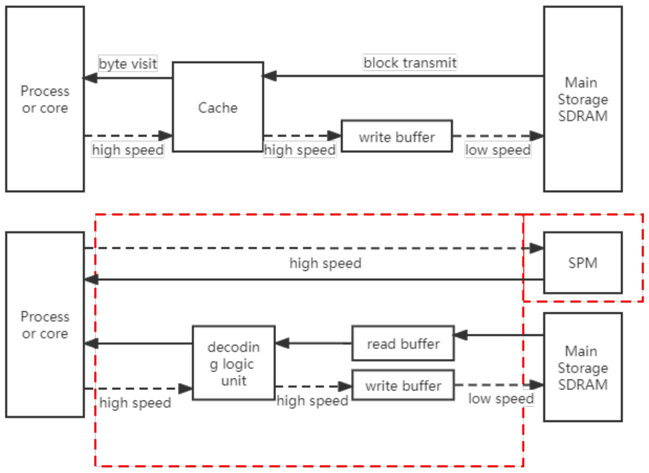

Figure 8: The comparison between cache and SPM system [8]

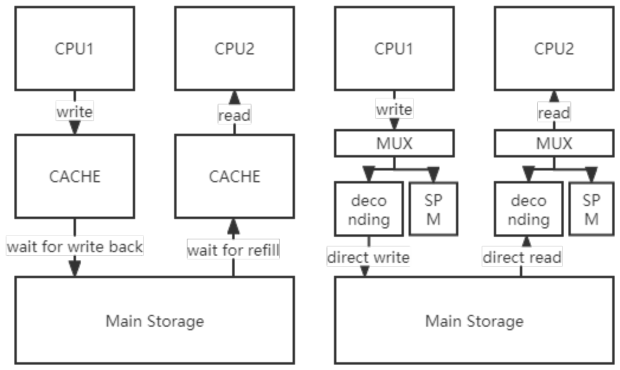

Figure 9: The difference of data flow direction between cache and SPM system

Table 2: The important configuration parameter of TGFF

Node Amount	Initial Node Amount	Maximum in-degree	Maximum out-degree	deadline for task execution
N	0.1±0.05N	αN	0.2IN	0.5N

We use TGFF [9] tool, an open-source random DAG generator builded in C++, to assess the TAU by optimistic instructions amount. The configuration of TGFF tool is shown on Table 2, among which the N stand for total nodes amount and the αN stands for maximum input degree as the variable, and the other parameter are in portion of N and αN. .

For example, with the parameter configuration of N=40, α=0.2, the randomly generated DAG graph of TGFF is shown in Figure 10. There is an execution-first dependency among the nodes as a whole,

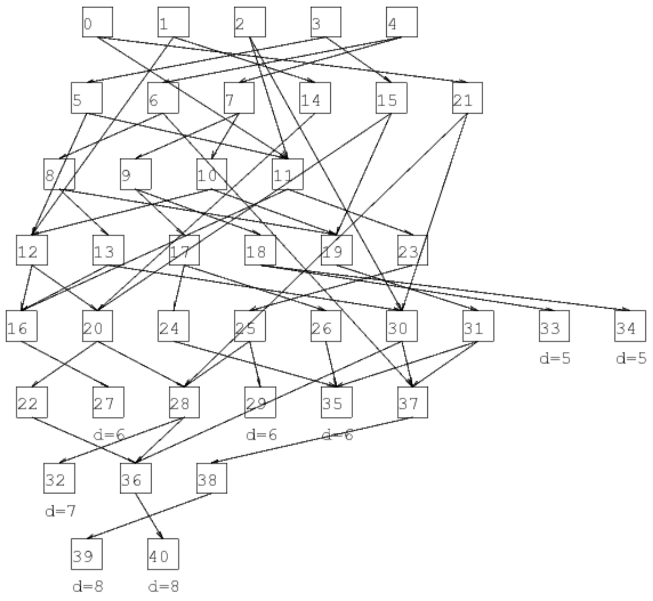

Figure 10: Random DAG generated by TGFF

which corresponds to the data dependency of the general task nodes in the data flow diagram. This paper adjusts the number and value of N. Each set of parameter adjustments generates random data flow. Then, the number of optimized V operations for each data flow graph is counted and the average within the group is calculated. It should be noted here that N is only the maximum possible value for a single node, but it is randomly chosen between [0, N] in the actual generation process, and the random algorithm is determined internally by the TGFF. Finally, we get the relationship between the number of optimization instructions and data flow diagram nodes, entry parameters Figure 11 and Figure 12.

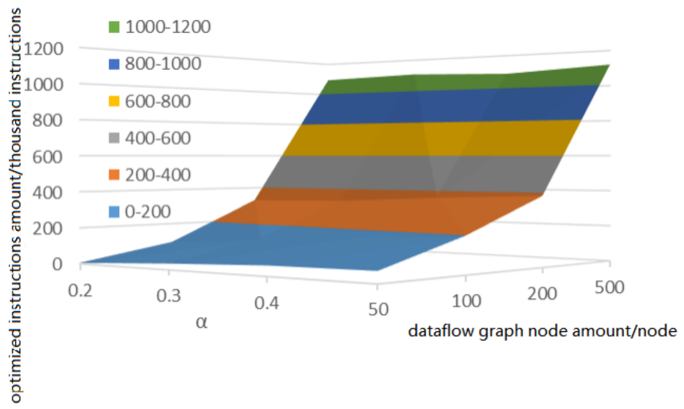

Figure 11: The relationship between the number of optimized machine instructions and data flow graph nodes and in-degree parameters

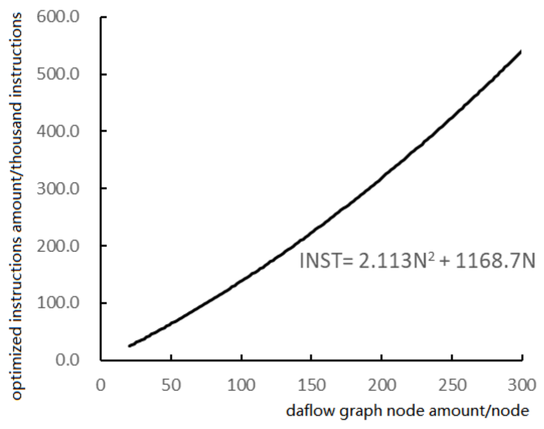

Figure 12: The relationship between the number of optimized machine instructions and data flow graph nodes and in-degree parameters

Table 3: The relation between TAU acceleration and task granularity (200 tasks)

Task granularity	500	2000	5000	10000
TAU acceleration	10.67%	10.32%	9.51%	8.42%

As Table 3 shows the acceleration ratio obtained by changing the task granularity size in the resulting data flow diagram when the number of fixed task nodes is 200. The data dependency of the data flow graph is also fixed. As the granularity of the task increases, the time to execute the task itself increases, and the time spent on synchronization decreases, affecting the TAU's acceleration ratio. From task granularity of 500 machine instructions to task granularity of 1000 machine instructions, TAU acceleration decreased from 10.67% to 8.42%. From Figure 13 You can see that the attenuation is

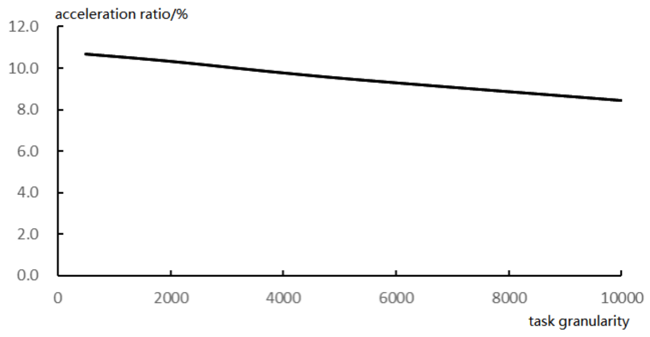

Figure 13: Curve of TAU acceleration and task granularity

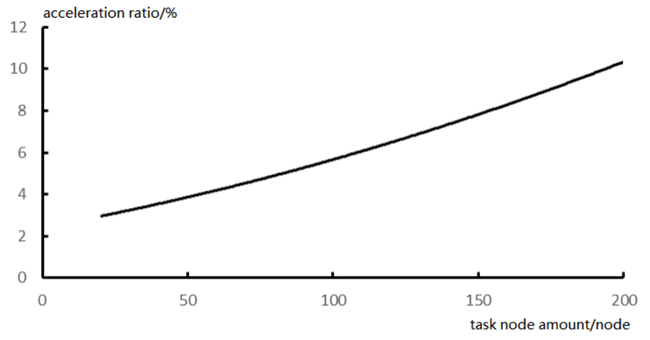

Figure 14: Curve of TAU acceleration and task amount

almost linear, and in theoretical extremes the numerical attenuation will be close to zero.

As shown in Table 4, with a fixed task granularity size of 2000 instructions, data flow diagrams with different number of nodes are generated, and their data dependencies change. The data flow graph used in this paper meets at least the condition that the dependency relationship of the data flow graph with more task nodes is more complex (that is, more nodes in and out). As the complexity of data dependency increases, the cost of data synchronization will gradually increase. From 20 task nodes to 200 task nodes, the TAU acceleration increased from 3.07% to 10.32%. As shown in Figure 14, the acceleration effect of TAU shows a slower upward trend of quadratic curve. The theory of quadratic increase in the number of optimization instructions. In extreme and ideal cases, when the number of edges in the data flow graph reaches N(N-1)/2, TAU data flow execution reaches the ideal maximum value of 50.4% (200 task nodes, 2000 instruction task granularity). In practice, however, the general data flow graph is not too dense (the randomly generated description of TGFF) and is limited by the TAU's own storage, and TAU acceleration is far less than the theoretical upper limit.

6 CONCLUSIONS

The work of this paper design and implement a new kind of hybrid dataflow/von-Neumann machine.

1) Design special hardware for data flow, namely task activation unit TAU. It implements the functions of writing the task

Table 4: The relation between TAU acceleration and task amount (2000 task granularity)

Task Amount	20	50	100	200
TAU acceleration	3.07%	3.63%	5.80%	10.32%

node dependent data relationship, triggering the dependent data ready signal, and triggering the task node ready signal. At the same time, the write instruction and ready instruction are extended to access the unit, and the programming verification is carried out in the way of assembly embedded.

2) Integrate SPM system into hybrid machine to enhance the efficiency of inter-core communication. It implements the functions of directly write or read to the main storage and storing special data in SPM unit.
3) System performance analysis of data flow execution environment. On the one hand, the random data flow graph generated by TGFF tool is used to help describe the accelerated optimization of tau, and the mathematical relationship between the number of nodes in the data flow graph and the number of optimized instructions is established. On the other hand, the overall execution environment of data flow is simulated, the acceleration ratio of running time is given, and the acceleration performance is discussed.

ACKNOWLEDGMENTS

This work was supported by teachers and students of SKT group in NHPCC (National High Performance Compute Center). This research was funded by project granted from Shenzhen Science Technology Foundation (20200811201536001).

REFERENCES

[1] Fahimeh Yazdanpanah, Carlos Alvarez-Martinez, Daniel Jimenez-Gonzalez, and Yoav Etsion. 2014. Hybrid Dataflow/von-Neumann Architectures. IEEE Trans. Parallel Distrib. Syst.25, 6 (June 2014), 1489–1509. https://doi.org/10.1109/TPDS.2013.125

[2] Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. 2004. Advances in dataflow programming languages. ACM Comput. Surv. 36, 1 (March 2004), 1–34. https://doi.org/10.1145/1013208.1013209

[3] Jack B. Dennis and David P. Misunas. 1998. A preliminary architecture for a basic data-flow processor. In 25 years of the international symposia on Computer architecture (selected papers) (ISCA '98). Association for Computing Machinery, New York, NY, USA, 125–131. https://doi.org/10.1145/285930.286058

[4] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler, and Charles R. Moore. 2003. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In Proceedings of the 30th annual international symposium on Computer architecture (ISCA '03). Association for Computing Machinery, New York, NY, USA, 422–433. https://doi.org/10.1145/859618.859667

[5] Steven Swanson, Andrew Schwerin, Martha Mercaldi, Andrew Petersen, Andrew Putnam, Ken Michelson, Mark Oskin, and Susan J. Eggers. 2007. The WaveScalar architecture. ACM Trans. Comput. Syst. 25, 2, Article 4 (May 2007), 54 pages. https://doi.org/10.1145/1233307.1233308

[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[7] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a Scala embedded language. In Proceedings of the 49th Annual Design Automation Conference (DAC '12). Association for Computing Machinery, New York, NY, USA, 1216–1225. https://doi.org/10.1145/2228360.2228584

[8] Vivy Suhendra, Chandrashekar Raghavan, and Tulika Mitra. 2006. Integrated scratchpad memory optimization and task scheduling for MPSoC architectures. In Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (CASES '06). Association for Computing Machinery, New York, NY, USA, 401–410. https://doi.org/10.1145/1176760.1176809

[9] Robert P. Dick, David L. Rhodes, and Wayne Wolf. 1998. TGFF: task graphs for free. In Proceedings of the 6th international workshop on Hardware/software codesign (CODES/CASHE '98). IEEE Computer Society, USA, 97–101.

[10] Nowatzki T , Gangadhar V , Ardalani N , *et al.* Stream-Dataflow Acceleration[J]. Acm Sigarch Computer Architecture News, 2017, 45(2):416-429.

[11] Lu W , Yan G , Li J , *et al.* FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks[C]. 2017 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2017.

[12] F. Vahid. Digital design with RTL design, VHDL, and Verilog.[M] John Wiley & Sons, 2010.

[13] S. Martin. Digital Design in Chisel[M]. 2020.

[14] Dou Yong, Wang Jialun, Su Huayou, Xu Chen, Gong Xiaoli, Yang Wangdong, Weng Chuliang, Li Zhanhuai, Li Kenli, Yu Ge, Zhou Aoying. From the development of computer architecture, we can see the idea of data flow calculation [J]. Chinese Science: Information Science, 2020,50 (11): 1697-1713

[15] Arty A7 Reference Manual[EB/OL]. https://digilent.com/reference/programmable-logic/arty-a7/reference-manual .

[16] Kavi K M , Hurson A R . Design of cache memories for dataflow architecture[J]. Journal of Systems Architecture, 1998, 44(9-10):657-674.

[17] A. Martin, *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems[J]. 2016.

[18] MIT contributors,"XV6"[EB/OL]. https://pdos.csail.mit.edu .2022.

[19] Kyriacou C , Evripidou P , Trancoso P . Data-Driven Multithreading Using Conventional Microprocessors[J]. IEEE Transactions on Parallel and Distributed Systems, 2006, 17(10):1176-1188.

[20] Etsion Y , Ramirez A , Badia R M , *et al.* Task superscalar: using processors as functional units[C]. Hot Topics Parallelism, 2010, p:16.

[21] C. Penha J , B. Silva L , M. Silva J , *et al.* ADD: Accelerator Design and Deploy - A tool for FPGA high-performance dataflow computing[J]. Concurrency and Computation: Practice and Experience, 2019.

[22] Ssa C , Ua A , Tn B . A framework to generate domain-specific manycore architectures from dataflow programs[J]. Microprocessors and Microsystems, p72.