Data-Driven IoT Platform Architecture with Overload Avoidance Capability

Shuji SANNOMIYA[1,4], **Yukikuni NISHIDA**[2], **and Hiroaki NISHIKAWA**[3,4]
[1]Faculty of Engineering, Information and Systems, University of Tsukuba, Ibaraki, Japan
[2]Graduate School of Systems and Information Engineering, University of Tsukuba, Ibaraki, Japan
[3]Headquarters for International Industry-University Collaboration, University of Tsukuba, Ibaraki, Japan
[4]DDSNA, Inc., 1-1-1 Tennodai, Tsukuba, Ibaraki, Japan

Abstract— *To keep the maintenance cost of large sensor networks acceptable, abnormalities such as failures and cyberattacks should be detected and localized autonomously. To realize such function, self-timed data-driven processor is introduced because its overhead-free real-time multiprocessing results in the proportional relation between the processing load and consumption current of the processor as long as the load is within a design target. This relation enables the detection of neighboring platforms' abnormalities via the consumption current of own platform processing data sent from the neighbors and also the isolation of the neighbors by ignoring the neighbors' data. In this paper, an overload avoidance mechanism is proposed to assure the proportional relation. The mechanism prevents the processing load from unexpected increases beyond the design target by postponing the data increase until the data decrease events occur. The effectiveness of the proposed mechanism is verified by simulation.*

Keywords: data-driven processor, self-timed pipeline, real-time multiprocessing, overload avoidance

1. Introduction

With the development of Internet of Things (IoT) technologies, the number of networked devices is becoming large, especially in sensor network system in which trillion sensors are expected to be used per a year in the future. In order to network and sustain such a large number of sensor devices with low maintenance cost, abnormalities such as failures and cyberattacks should be detected and their untoward effects should be localized internally and autonomously in networking systems because any external checking structures force additional costs for maintaining themselves.

To realize such autonomous abnormality detection and localization, we have already studied data-driven sensor networking system whose platform is realized by ultra-low-power data-driven chip multiprocessor (ULP-DDCMP) that realizes real-time multiprocessing essential for communication processing without any run-time overheads resulting in both power dissipation and throughput degradation [1], [2]. Because of this overhead-free real-time multiprocessing, the consumption current of the ULP-DDCMP have strong correlation and is in proportion to the processing load that is the number of data in processing, i.e. the power is consumed only for processing data, as long as the load is within a design target.

By virtue of this strong correlation, the consumption current in operation under a typical amount of processing data is foreseeable, and the foreseeable consumption current makes it possible to detect devices' failures and cyberattacks that change the number of data in processing, by comparing the monitored consumption current with the foreseen value at run-time. For instance, a wireless sensor device transferring unusual data changes the number of data in processing in the processor of its neighboring devices, and this change can be detected via the consumption current in the neighboring devices. Moreover, the untoward effects of the wireless sensor device with abnormality can be localized by dropping the data sent from that device in the neighboring devices.

However, the proportional relation is not established when the number of data in processing exceeds a design target [3]. Although a technique that distributes the data over plural processors is already proposed [4], it is still difficult to assure the run-time fluctuation of the number of the data even with this kind of technique. This is because the number of operations that can be executed in parallel may change depending on the timing of data input even when the input data rate is constant and exhaustive simulations covering all combinations of the variations of data arrival timing is impractical. The situation where the number of data in processing exceeds the design target is named overload.

In this paper, an overload avoidance mechanism is proposed to guarantee the proportional relation between the consumption current and the number of data in processing. The proposed mechanism postpones the increase of the number of data flowing in the processor until the data decreases, in order to enable the assurance of the number of data in processing. Its effectiveness is shown based on a simulation with a protocol handling application.

2. Data-Driven IoT Platform

In this section, the autonomous abnormality detection and localization scheme in our data-driven sensor networking system is explained and the architectural requirements to realize IoT platforms implementing this scheme are discussed.

2.1 Essential power consumption

The platform of the data-driven sensor networking system is realized by ULP-DDCMP (Ultra-Low-Power Data-Driven Chip Multiprocessor) and its peripherals such as sensors and wireless communication unit. The ULP-DDCMP is a chip multiprocessor that is realized by interconnecting an ultra-low-power version of self-timed data-driven processor, named ULP-CUE [5]. To achieve ultra-low-power consumption in which only essential power is consumed, the ULP-CUE implements data-driven principle by which operation execution is initiated on the arrival of input data as long as computational resources (i.e. pipeline stages) are available.

Fig. 1 shows the ULP-CUE's circular pipeline that is indispensable to naturally realize the iteration of operation execution in which operation result is transferred to the input of the succeeding operation. In the ULP-CUE, every data is packed with tags such as generation and node number and the packed data is called token. The programs of the ULP-CUE are expressed by data-flow graph, and the generation is used to identify the data and the node number identifies the operation in the data-flow graph. The circular pipeline consists of matching memory (MM), program storage (PS), functional processing unit (FP), memory access (MA), merge (M), and branch (B)

In the MM, two tokens which are the operands of binary operations are found out based on their generation and node number, and they are packed into a single token and output. The first comer of the two tokens is stored to an internal memory until the other is input to the MM. The PS stores the operation codes and data dependencies among the operations of the data-flow graph. Among those stored information, the corresponding operation code and its dependency data are read out by using the node number of the token output from the MM. The read operation data and the token are output to the FP. According to the operation data, the FP conducts token manipulations such as arithmetic and logic operations, the copy and absorption (erasing) of token, alteration of generation and node number. These manipulations can be selected by using conditional expressions. The MA realizes the read and write of data memory, and the token whose data is written into the data memory may be absorbed (erased) from the pipeline. The M stages accept data from two preceding stages in order of arrival and transfers the accepted data to a succeeding stage, and the B stages transfer each data to one of two succeeding stages selectively. The M and B stages are used to make the circular data path.

With this structure, the concurrent operations of target programs are naturally exploited over the circular pipeline as long as the pipeline stages are available, and they are processed independently from each other without run-time overheads such as context switching resulting in both power

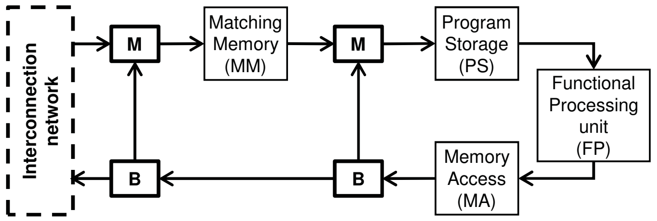


Fig. 1: Ultra-low-power data-driven networking processor: ULP-CUE.

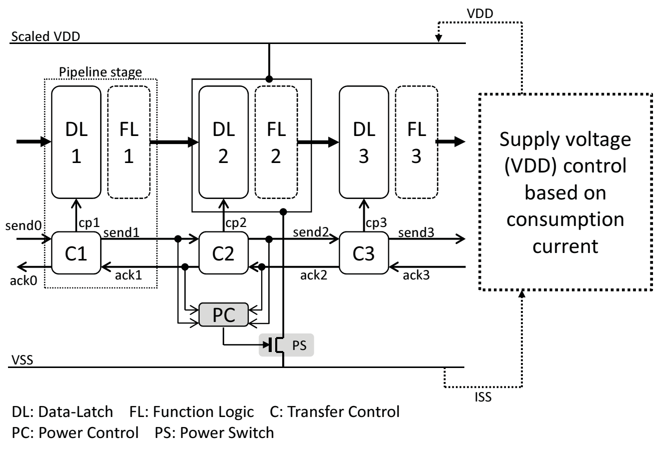

VDD
Scaled VDD
Pipeline stage
DL 1
FL 1
DL 2
FL 2
DL 3
FL 3
Supply voltage (VDD) control based on consumption current
send0
cp1
send1
cp2
send2
cp3
send3
C1
C2
C3
ack0
ack1
ack2
ack3
PC
PS
VSS
ISS
DL: Data-Latch FL: Function Logic C: Transfer Control
PC: Power Control PS: Power Switch

Fig. 2: Self-timed pipeline with power control mechanism.

dissipation and throughput degradation. To enhance the power-performance efficiency of protocol handling in which unary operations occupy the 80% of operations, the circular pipeline is optimized to bypass the MM that is required essentially only for binary operations when unary operations are executed [5].

Fig. 2 illustrates the basic structure of the self-timed pipeline [6] whose pipeline stage consists of a data-latch (DL), functional logic (FL), and a transfer control unit (C). The self-timed pipeline is a type of asynchronous bundled data pipeline that employs a four-phased handshake [7]. Based on the handshake, the valid data is transferred between adjacent stages according to the following procedure.

- (0) Reset: After the assertion of the reset signal, C negates both its send signal (representing a transfer request) and its ack signal (representing acknowledge).
- (1) C asserts its ack signal after its send signal is asserted.
- (2) After the assertion of the ack signal, the preceding C negates its send signal.
- (3) After negation of the send signal, C asserts both its gate open signal (cp) and its send signal. Concurrently, it negates its ack signal if the ack signal from the succeeding C is negated. As a result, the data is latched in the stage to which the succeeding C belongs.
- The succeeding C repeats the above steps in the same manner as the current C.

The handshake also lowers power consumption by directing dynamic consumption current into pipeline stages with valid data. On the other hand, empty pipeline stages can be powered off by the power control and power switch to reduce the leakage current through the empty stages [8]. Moreover, the signal propagation delay of DL, FL, and C are changed at an equal rate according to the supply voltage. Thus, the supply voltage of the self-timed pipeline can be scaled at run time while the rate of change of the voltage is moderate enough to guarantee transistor switching.

2.2 Autonomous abnormality detection and localization function and its requirements

Fig. 3 shows the throughput and current consumption of the ULP-CUE, and they are measured by using a prototype LSI of the ULP-DDCMP. As a proof of the ultra-low-power consumption, both the throughput and consumption current increase when the occupancy rate which is the ratio of the number of valid data to the number of pipeline stages increases. This strong correlation between the consumption current and the number of data in processing makes it possible to foresee the range of the consumption current in operation against a given amount of the processing load by using a system level simulator [9].

At run-time, the consumption current may be out of the foreseen range due to some abnormality. For instance, a sensor may generate unexpectedly large amount of data due to failures. In such a case, the abnormality can be detected by monitoring the consumption current of the ULP-CUE processing the data generated by devices with abnormality, and the untoward effects of the abnormality can be localized by ignoring those data. This autonomous abnormality detection and localization can be achieved also at the network level. When a platform detects some abnormality in itself, the platform sends alerts to its neighbors, and then the neighbors ignore the data sent from the platform. Even when the platform is unable to send the alerts due to some abnormality, the neighbors can notice that some abnormality occurs in the platform if the amount of processing data sent from the platform is out of the foreseen range, and in this case, the neighbors ignore the data sent from the platform with abnormality in order to localize the untoward effects of the abnormality.

Fig. 3 also shows that the operation execution time is the shortest amount as long as the number of tokens in processing is less than the design target and thus the throughput becomes in proportion to the number of tokens in processing. This design target for keeping the shortest operation execution time has already been analyzed [3] and its maximum value is given by equation (1), where the number of processing tokens, the sum of the processing times of every pipeline stage, and the longest processing time are denoted by P_{total}, $\sum T_f$, and T_{max}, respectively. The T_f denotes the send signal propagation time which is

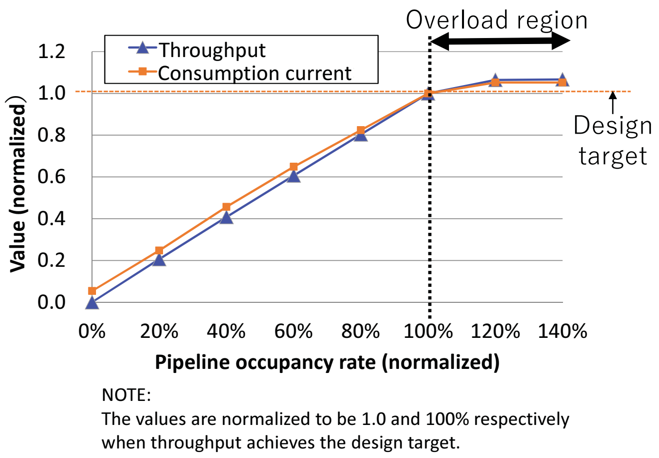


Fig. 3: Essential power consumption.

adjusted to the critical path of the corresponding FL. The T_{max} is the longest value in the values of the $(T_f + T_r)$ of every pipeline stage, where the T_r denotes the ack signal propagation time which is set to the set-up hold time of the corresponding DL.

When the equation is not satisfied due to the increase of P_{total}, the operation execution time increases according to equation(2), where the difference between the P_{total} and the largest value of P_{total} satisfying the equation(1) is denoted by P_{over}. That is, the overload situation is when the equation(1) is not satisfied.

$$0 \leq P_{total} \leq \frac{\sum T_f}{T_{max}} \tag{1}$$

$$\sum T_f + T_{max} \times P_{over} \tag{2}$$

Consequently, to assure the autonomous abnormality detection and localization, the P_{total} should be keep within the value satisfying the equation (1).

3. Run-time Overload Avoidance

The number of tokens in processing (P_{total}) fluctuates depending on the arrival timing of the input data even when the amount of input data at a unit time is constant. In this section, we propose a token flow-rate peak assurance mechanism that enables the assurance of the P_{total} by postponing the increase of the P_{total} until the decrease of the P_{total} except for the input of tokens.

3.1 Run-time token flow rate fluctuation

The P_{total} increases because of (1) input to the ULP-CUE from the interconnection network and (2) copy in the FP.

The event (1) occurs in two cases. One is the case when datagrams which are protocol handling units are divided into tokens and the tokens are input to the processor, and the other is the case when tokens are transferred to a ULP-CUE from the other ULP-CUE's for load distribution or function distribution. The number of datagrams to be processed at a unit time is defined as a target throughput in system design phase, and it determines input token rate which is the number of tokens input at a unit time. Because the real-time multiprocessing can keep the operation execution time in the shortest amount while the equation(1) is satisfied, the fluctuation on the P_{total} for the determined input token rate can be estimated by using simulators such as platform simulator [9] that can simulate multiple ULP-CUE's and the data transfer among them. Consequently, against the even (1), the P_{total} can be kept within the equation(1) by adjusting the $\sum T_f$ and T_{max} according to the estimated fluctuation on the P_{total}, in the system design phase.

On the other hand, the event (2) is occurred during the execution of operations with copy, and it doubles the P_{total} at maximum and thus it may result in the overload situation. The timing of the event (2) depends on the arrival timing of the operand tokens of the operations with copy. However, it is difficult to evaluate or simulate all possible combinations of the operand tokens' arriving timings because such exhaustive simulation takes a huge amount of time.

The P_{total} increase occurred by the copy can be suppressed statically. The program of the ULP-CUE is expressed by data-flow graph in which operations and data dependencies among the operations are denoted by nodes and arcs respectively. Fig. 4(a) illustrates an example of the graph. In this figure, two arcs rooted in a node denotes that the output token of the node is copied and fed to two succeeding nodes, and rank shows the order that each operation is placed on in the critical path which is the longest path from the input to the output of the data-flow graph. The operations on the same rank are executed concurrently, and thus structuring the data-flow graph by limiting the number of the operations to one for each rank leads to the suppression of the P_{total} increase occurred by copy.

Fig. 4(b) shows one method that introduces a special binary operation S which outputs its left-hand side input token after the arrival of its operand tokens. The S is an abbreviation for synchronization. With this method, one of the copied tokens is stored into the MM during the execution of the operation S, and the execution of one of the two operations whose operand is the copied token is postponed until the other operation finishes its execution. In the Fig. 4(b), the execution of the operation C is postponed until the completion of the execution of the operation B. However, this method results in the degradation of throughput because the operation S has essentially no contribution to the target process and increase the execution time of the target process.

To avoid the throughput degradation, unary operations can be extended to accept another input token as a trigger to start their execution, as shown in Fig. 4(c) where the extended unary operation is labeled with "C'" and its left-hand side input is for the trigger. The operation C' is executed as a

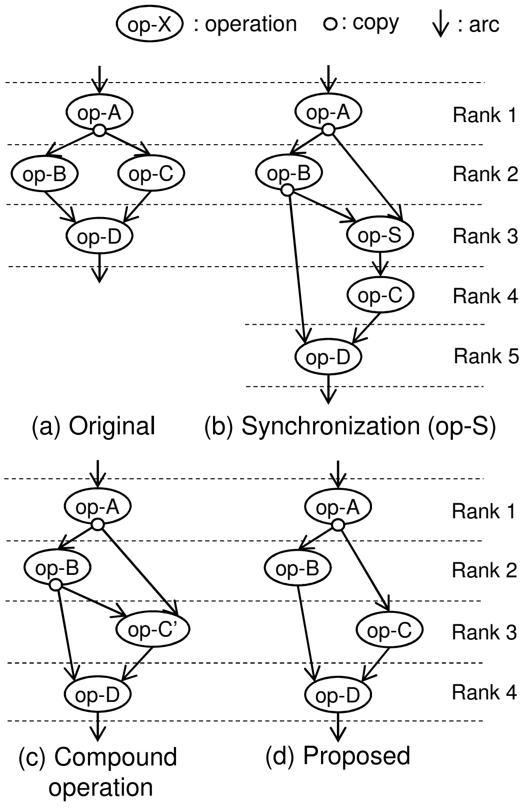


Fig. 4: Adjustment.

binary operation, and one of the two input tokens waits for the other in the MM. Once the two input tokens arrive, the operation is executed with ignoring the trigger token, i.e. the trigger token is used to postpone the execution of the operation C' until the completion of the operation B. In fact, the number of operations executed concurrently can be reduced [10]. Although the number of operations at the same rank can be 1 with this method, the P_{total} becomes 2 temporarily while the copied token input to the operation C' moves to the MM from the FP in the execution of the operation A. Hence, the P_{total} may become double temporarily depending on the input token arrival timing and the problem still remains even with this method.

During the data-flow graph execution, the P_{total} decreases due to (3) output to the interconnection network from the ULP-CUE, (4) temporal storing in the MM, (5) absorption in the FP, and (6) absorption in the MA. By utilizing this fact, the overload avoidance can be realized by dynamically postponing the increase of the tokens in processing until the occurrence of the events (4), (5), and (6).

3.2 Token flow-rate peak assurance mechanism

The token copy is realized by transferring an input token to the succeeding pipeline stage consecutively and twice, in the last pipeline stage of the FP [5]. To avoid the increase of the token transfer time resulting in throughput degradation, the postponing of the token copy should be realized without stopping the handshake between the last pipeline stage and its precedent stage. On the other hand, only one valid token can be held in a pipeline stage, and thus one of the copied tokens should be evacuated from the pipeline temporarily. Under these constraints, the token evacuation and return function is implemented into the ULP-CUE.

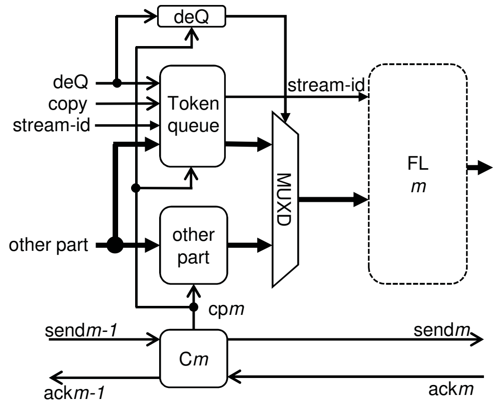
For evacuating the copied tokens, the internal memory of the MM and the data memory of the MA can be used; however, in this case, the copied tokens are transferred in the pipeline until they reach one of the memories and thus they increase the P_{total} temporarily. To eliminate the temporal increase, a queue holding the copied tokens temporary is introduced at the last stage of the FP and the copied token is queued at the same time when it is copied. This queue is named token queue in this paper.

Fig. 5(a) shows the FP's last stage (m-th stage) with the token queue. In this stage, the input token is fed to both the DL and the token queue, and the token queue stores the fed token into one of FIFO buffers if copy flag is set to 1. The copy flag is a part of the operation code in the token's tag and "1" indicates the occurrence of the copy while "0" do not. The two copied tokens are directed to different operations, and they are labeled left-hand side or right-hand side according to their destination. The selection which side of token to be queued has no effects on the result and throughput of the data-flow graph execution, but it governs the number of tokens in the token queue, and thus it is determined so as to reduce the number of queued tokens according to some static analysis such as preprocessing of the data-flow graph.

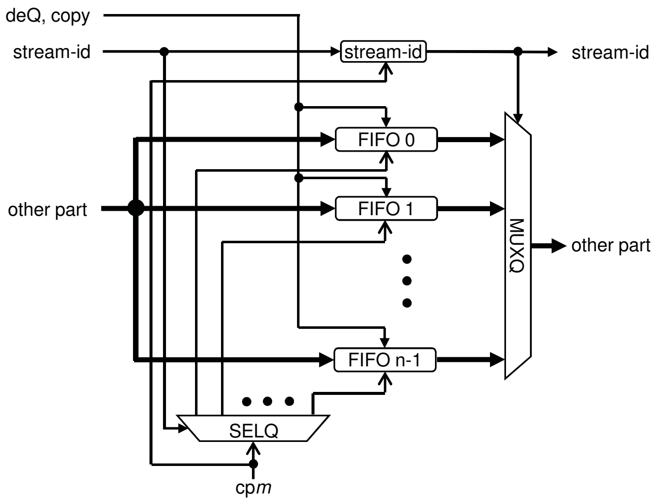
To assure that the queued tokens are returned to the pipeline after the decrease of the P_{total}, the tokens to be output or absorbed are used to draw (dequeue) the queued tokens. The token for dequeuing is named dequeue token in this paper. The dequeue token is transferred to the FP, and the FP works as follows when the dequeue token is input.

- If the token queue stores more than one token, the first stored token is drawn out and output to the succeeding pipeline stage for resuming the execution of an operation to which the drawn token is directed.
- At the same time, the dequeue token is absorbed.

The dequeue token is generated when the P_{total} decrease events occur. As for the event (3), the B outputs the dequeue token to the M at the same time when its input token is output toward the interconnection network. The dequeue token is generated by adding a flag indicating the dequeue (deQ) to the input token, and the B conducts the handshake with the M and B at the same time in order to eliminate a copy for generating the dequeue token. In addition, at the same time when the input token of the MM is stored to the internal memory in the event (4), the input token is attached the deQ flag and output to the next pipeline stage instead of absorbing the input token. Furthermore, when a token is



(a) Pipeline stage with queuing function



(b) Token queue

Fig. 5: Circuit implementation of copied token queuing function.

absorbed in the FP due to the event (5), the FP conducts the dequeuing from its token queue. Meanwhile, the MA outputs the dequeue token as similar to the MM when the event (6) occurs.

To share the token queue among plural datagram handling processes, its internal FIFO buffers are discerned by using a part of the tag's generation, that is used to specify each datagram and denoted by stream-id.

Although the addition of the deQ flag may increase the width of the pipeline data path, such circuit size can be suppressed by circuit optimization techniques such as encoding the deQ flag to the operation code in the token's tag.

The token queue is placed in parallel with the DL of the last pipeline stage of the FP, as shown in Fig.5(a). Although the latency of the token queue is partially concealed to that of the DL, the rest of the latency increases the processing time of the last pipeline stage. On the other hand, the maximum throughput of pipeline is determined by the pipeline stage whose processing time is the longest in the pipeline, and it is given by the inverse of the longest processing time, i.e. it is $1/T_{max}$ [token/sec.]. According to the previous LSI implementation of the ULP-CUE, the RAM (Random Access Memory) access of the PS and MA is atomic and its latency governs the T_{max} [5]. Because the FIFO buffers stores not the entire program or long-storage data but only active tokens temporarily, its size is rather small in comparison to the RAM, and thus the T_{max} is expected to be still governed by the PS and MA. That is, the throughput of the ULP-CUE with the token queue can be equal to that of the original ULP-CUE.

With this mechanism, the copy of a token is postponed until the absorbing or output of the other token, and the P_{total} can be assured.

3.3 Example of behavior

With a data-flow graph shown in the Fig. 4(d), the behavior of the proposed mechanism is explained. The execution of the data-flow graph is composed of 5 steps, as shown in Fig. 6.

- Step 0 shows the situation when an input token to operation A arrives. The token queue is empty at that time.
- After the completion of the operation A, a token holding the result of the operation A is copied. One of the copied tokens is output to operation B and the other is queued at the same time, as shown in Step 1.
- Step 2-1 shows the situation when a token holding the result of the operation B is output and it is input to a binary operation D.
- The input token to the operation D moves to the MM and it is stored the MM's internal memory. At the same time, a dequeue token is generated, and it moves to the FP. After that, the queued token is drawn out from the token queue in the FP, as shown in Step 2-2.
- The dequeued token goes round the pipeline. As a result, the operation C is executed and a token output from the operation C is input to the operation D, as shown in Step 3.
- After the execution of the operation D, the result token is output from the processor, as shown in Step 4. At the same time, a dequeue token is generated in the B, and it moves to the FP. In the FP, the token queue is empty, and thus the dequeue token is absorbed.

With the proposed mechanism, the data-flow graph execution time is increased because of the in-line execution of the operations that are originally executed concurrently. However, the throughput can be retained by exploiting the parallelism among the processing units, e.g. munificent parallelism among the datagrams can be utilized especially in the protocol handling. Moreover, the proposed mechanism

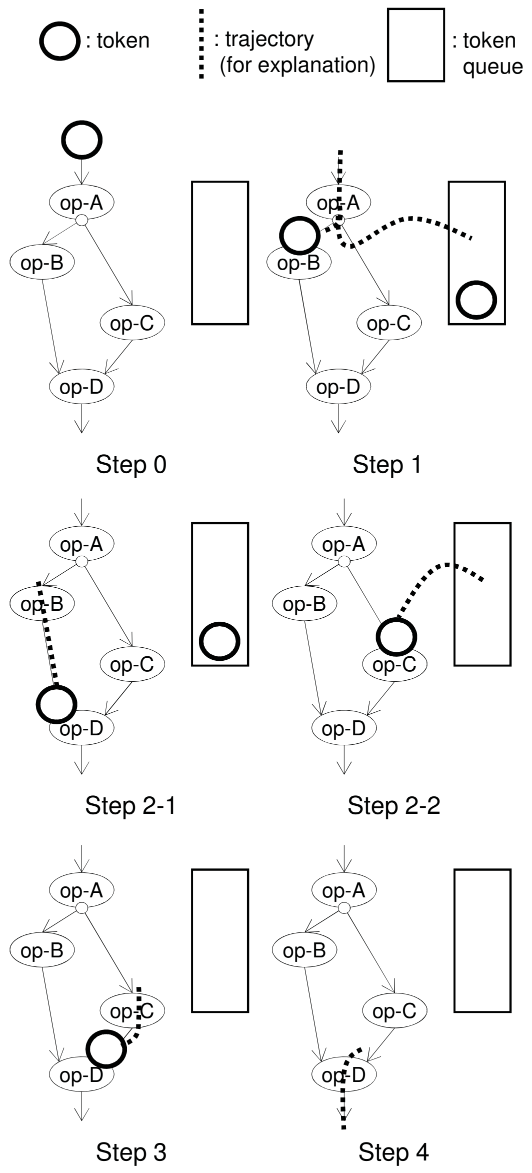


Fig. 6: Behavior of token queue.

has no effects on each pipeline stage, and thus the real-time multiprocessing is preserved.

3.4 Verification of overload avoidance

To verify the overload avoidance capability of the ULP-CUE with the proposed mechanism, the ULP-CUE is simulated by using a platform level simulator that conducts pipeline stage level simulation and outputs the P_{total} with timestamp. As an application, a UDP datagram receiving program that is a part of the UDP/IP handling is used. The UDP/IP handling is the target application of the ULP-CUE [5]. The input of the used program is tokens constituting UDP datagrams. In the execution of the program, the checksum of the datagrams is calculated and the tokens are stored to the data memory in the order in which the tokens are processed, meanwhile the stored tokens are serially read out from the memory and output in the order of the position in the datagram.

The size of the datagram is set to 512 Byte and each datagram is divided into 128 tokens whose data length 4 Byte (32 bit). The input token rate is set to 1.69 M [token/sec.] (=54M bps / 32 bit) based on the maximum throughput of IEEE802.11g, 54 M bps.

The processing time of each datagram increases when the ULP-CUE becomes overload, on the other hand, the processing time remains if the ULP-CUE with the proposed mechanism avoids the overload situation. According to this fact, the processing time of each datagram is measured. The measured results are shown in Fig. 7 in which the horizontal axis shows the number of UDP datagram processed concurrently and the vertical axis means the measured processing time.

The results in the case of the original ULP-CUE are shown with a label "Original". In the original ULP-CUE, the P_{total} may increase and become greater than the number of input tokens to be processed concurrently, and thus the original ULP-CUE may fall into the overload situation with less multiplicity in comparison with the ULP-CUE with the proposed mechanism. When the overload situation happens, the P_{total} increases due to the input tokens, and finally all of the pipeline stages are filled with the tokens and the handshake stops. In fact, the original ULP-CUE becomes inoperative due to the overload when the multiplicity exceeds 3, as shown in the Fig. 7.

In contrast, the P_{total} is increased only by the input tokens in the ULP-CUE with the proposed mechanism, and thus the overload situation can be avoided. As a result of the overload avoidance, the pipeline processing capability can be exhaustively exploited. "Proposed" shows the processing time in the case of the ULP-CUE with the proposed mechanism. This result proves that the multiplicity can be increased as the result of the overload avoidance.

On the other hand, the processing time increases slightly as the multiplicity increases in both the "Original" and "Proposed". This is because the data transfer between the M and B instantaneously stops when an input token flows in the M from the interconnection network.

Based on these results, it is shown that the ULP-CUE with the proposed mechanism can realize the overload avoidance and the exhaustive exploitation of the pipeline processing capability.

4. Conclusion

In this paper, a token flow-rate peak assurance mechanism is proposed to assure the proportional relation between the ULP-CUE's consumption current and processing load, that is essential to realize autonomous abnormality detection and

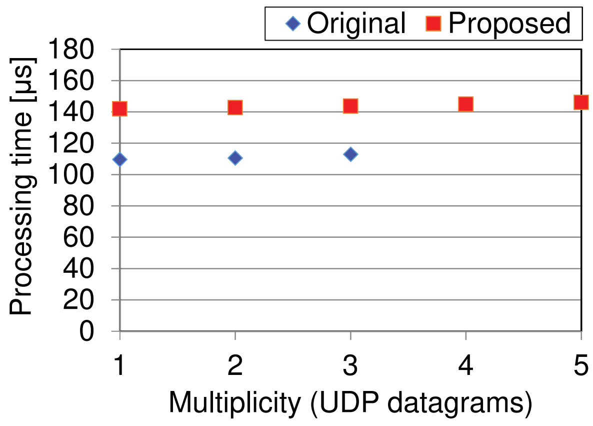


Fig. 7: Measured UDP datagram processing time

localization in the networking systems. The proposed mechanism eliminates unforeseeable token increase by postponing the token copy until the token is absorbed or output. The simulation results of the ULP-CUE with the proposed mechanism shows that the overload avoidance can be achieved and the pipeline processing capability can be exhaustively exploited.

Although the proposed mechanism assures the token flow rate of the ULP-CUE against a given token input rate, the assurance of the token input rate for every networking platform still remains an important challenge to ease the congestion of the networking system.

References

[1] S. Sannomiya and H. Nishikawa, "Highly-dependable and long-lifetime data-driven networking processor with energy assurance capability," in Proc. of PDPTA, pp.557-563, July 2015.

[2] H. Nishikawa, "Design philosophy of a networking-oriented data-driven processor: CUE," IEICE Transactions on Electronics, Vol.E89-C No.3, pp.221-229, Mar. 2006.

[3] S. Sannomiya, Y. Omori, and M. Iwata, "A macroscopic behavior model for self-timed pipeline systems," in Proceedings of Seventeenth Workshop on Parallel and Distributed Simulation (PADS2003), pp.133–140, June 2003.

[4] S. Sannomiya, H. Nishikawa, "Energy efficient data-driven networking processor with autonomous load distribution capability," in Proc. of PDPTA, pp.514-520, July 2014.

[5] S. Sannomiya, K. Aoki, M. Iwata, and H. Nishikawa, "Power-performance verification of ultra-low-power data-driven networking processor: ULP-CUE," in Proc. of PDPTA, pp.465-471, July 2012.

[6] H. Terada, S. Miyata, and M. Iwata, "DDMP's: self-timed super-pipelined data-driven processors," Proceedings of the IEEE, Vol. 87, No. 2, pp. 282–296, Feb. 1999.

[7] C. J. Myers, "Asynchronous circuit design," Univ. of Utah John Wiley & Sons, Inc., 2001.

[8] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, "Low-powered self-timed pipeline with variable-grain power gating and suspend-free voltage scaling," in Proc. of PDPTA, pp.618-624, July 2013.

[9] K. Aoki, S. Sannomiya, H. Nishikawa, "Data-driven sensor networking system simulator," in Proc. of PDPTA, pp.564-570, July 2015.

[10] S. Sannomiya, Y. Nishida, M. Iwata, and H. Nishikawa, "An overload-free data-driven ultra-low-power networking platform architecture," in Proc. of PDPTA, pp.604-610, July 2013.