Non-Minimal Routing Strategy for Application-Specific Networks-on-Chips

Hiroki Matsutani, Michihiro Koibuchi [*], Yutaka Yamada,
Akiya Jouraku, Hideharu Amano
Department of Information and Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, JAPAN 223-8522
{matutani,koibuchi,yamada,jouraku,hunga}@am.ics.keio.ac.jp

Abstract

We propose a deterministic routing strategy called flee which introduces non-minimal paths in order to distribute traffic with a high degree of communication locality in Networks-on-Chips. In the recent design methodology, target system and its application of the Systems-on-a-Chip are designed in system level description language like System-C, and simulated in the early stage of design. The task distribution is statically decided in this stage, and the amount of traffic between nodes can be analyzed. According to the analysis, a path that transfers a large amount of total data is firstly assigned with a relaxed limitation, thus it is mostly minimal. On the other hand, paths for small amount of total data, are secondly established so as not to disturb previously established paths, thus they are sometimes non-minimal. Simulation results show that the flee routing strategy improves up to 28.6% of throughput against the dimension-order routing on typical stream processing application programs.

1 Introduction

On-chip interconnect, which connects intellectual property (IP) cores, is one of the most crucial components in Systems-on-a-Chips (SoCs), in terms of performance and hardware cost. An on-chip bus, which transfers data logically on wires shared by all connected modules, has been widely used as an on-chip interconnect. Despite various techniques to improve performance [2][13], buses still create bandwidth bottlenecks when connecting larger numbers of modules, and their clock frequencies are difficult to increase because wiring delays have become relatively increased in recent process technologies.

In order to avoid both bandwidth and wiring-delay bottlenecks which bus structures create, Networks-on-Chips (NoCs)[5][10][1][9] have been studied as on-chip interconnects for the next generation. NoC architectures are similar to those used in parallel computers and System Area Networks (SANs). In these networks, source nodes [1] generate packets that include headers as well as data, then routers transfer them through connected links, and destination nodes decompose them. Since different packets can be simultaneously transferred on multiple links, bandwidth of NoCs is much larger than that of buses. Also, the wiring-delay problem is resolved, since each flit of a packet is transferred on a limited-length point-to-point link, and stored in the buffers of each router. By introducing error detection and re-transmission protocols, dynamic transmission errors caused by crosstalk, which will come up in future process technologies, can be also solved.

Since a node is often a simple special-purpose unit, the corresponding router should be small. Thus, a router in NoCs usually employs a deadlock-free deterministic routing[10][1][9]. The NoC architecture itself is desired to be flexible, scalable and applicable so as to fit for various types of SoCs. However, once an SoC architecture is fabricated, the number of nodes, achievable performance of nodes, their functionalities, and connection topologies are fixed, and never changed in most cases. Thus, a network architecture should be customized for its application rather than that for general purpose.

One of the major targets of SoCs is embedded applications, like media processing[8] mostly for consumer equipments. In the recent design methodology, target system and its application are designed in system level description language like System-C, and simulated in the early stage of design. The task distribution is statically decided in this stage, and the amount of traffic between nodes can be analyzed. Topology design policy according to the analysis of traffic patterns has been researched[7]. However, a routing algorithm has not been widely investigated, and simple minimal routing methods for parallel computers[4] have been used.

[*]Presently with National Institute of Informatics

[1]In this paper, we use the term “nodes” for IP cores, which are the connection targets in a chip.

In this paper, we propose a deterministic routing strategy called *flee* by introducing non-minimal paths for NoCs according to records of traffic analysis in the design stage of SoCs. Access traffic of streaming application often includes a high degree of communication locality, in which distance between communication pairs of nodes is too short to distribute a path by minimal routing. By introducing non-minimal paths, the *flee* routing strategy increases alternative paths to avoid congestion links. Then, it establishes a deterministic path taking account into the data sizes communicated by each pair of nodes.

The rest of this paper is organized as follows. In Section 2, characteristics of stream processing are described. In Section 3, existing deadlock-free routing methods which can be used for NoCs are surveyed, and in Section 4, the *flee* routing strategy is proposed. In Section 5, evaluation results using real application traces are shown, and Section 6 is the conclusions.

2 Stream Processing

In most streams processing such as Viterbi, JPEG or MPEG coder, a series of processing is performed to a certain amount of data. A unit of such processing is called the "Task". Figure 1 shows a task diagram of JPEG2000 decoding. In the processing, each task can be mapped onto each node, and is performed in the pipelined manner. In this case, communication is limited only between neighboring two nodes. However, the framed part called EBCOT (Embedded Block Coding with Optimal Truncation) requires a high computation power and bottlenecks the whole stream flow if each processing in EBCOT is assigned into a single node. For equalizing the stream flow, the processing of EBCOT should be distributed into several nodes and executed in parallel as shown in task flow graphs of Figure 2 and 3. In parallelized one, the communication pattern between nodes includes stream fork and join.

NoCs usually employ simple network topologies, such as a two-dimensional mesh[1][9] or a folded torus[5][10]. When assigning such tasks into those topologies, access patterns between nodes include a high degree of locality as shown in Figure 3. Thus, by pre-analyzing the target application, routing strategies should cope with its high access locality, so that communication paths are well-distributed.

3 Existing Routing Methods

Existing NoCs employ deadlock-free deterministic routing in most cases[10][1][9]. Unlike adaptive routing that dynamically changes paths of packets, a path is fixed statically in deterministic routing, and it has the following advantages: 1) simple switch without selecting an output

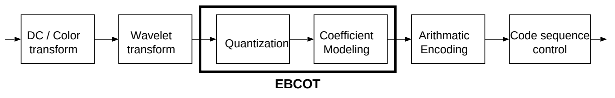


Figure 1. A task graph for JPEG2000 decoding. EBCOT requires high computation load.

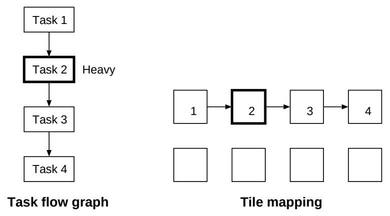


Figure 2. Sequential model. Task 2 bears a heavy load.

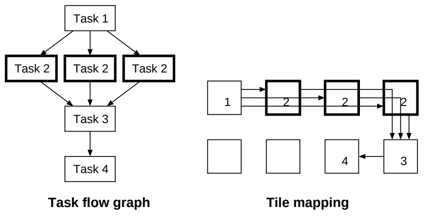


Figure 3. Parallel model. Task 2 is distributed into three nodes and executed in parallel.

channel dynamically from alternative channels can be used; 2) in-order packet delivery, which communication protocol often requires, is guaranteed.

A simple and popular deterministic routing in NoCs is dimension-order routing[4], which uses y-dimension channels after using x-dimension channels in 2-D mesh and torus. Dimension-order routing uniformly distributes minimal paths between all pairs of nodes.

On the other hand, path selection techniques used for selecting a path from alternative paths in adaptive routing will be considerable to arrange a deterministic path set, because the traffic pattern of stream application is predictable. The path selection techniques in SANs with irregular topologies[12] could be applied to this strategy. Since various kinds of parallel programs are executed in parallel computers, access patterns are difficult to be predicted in networks for SANs. Thus, such techniques are mainly intended to make shortest paths to achieve stable and high throughput under unknown general traffic in PC clusters. In addition, pairs of nodes make different amount of communication data, which would be a key factor to establish paths in NoCs. However, current path selection methods and deterministic routings don't consider this directly, and they are not optimized to make application-specific paths.

4 Flee Routing Strategy

In this section, a deterministic routing strategy called *flee*, which exploits non-minimal paths, is proposed. As demonstrated in Section 2, access traffic of streaming application often includes a high degree of access locality. The *flee* routing strategy increases alternative path sets by simply introducing non-minimal paths so as to mitigate path congestion. It establishes a deterministic path taking account into the data sizes, which each pair communicates.

This strategy is divided into two steps: 1) static analysis of communication pattern, and 2) path establishment under deadlock-free condition. In this algorithm, a path that transfers a large amount of total data is firstly assigned with a relaxed limitation, thus it is mostly minimal. On the other hand, paths for small amount of total data, are secondly established so as not to disturb previously established paths, thus they are sometimes non-minimal.

4.1 Communication Analysis

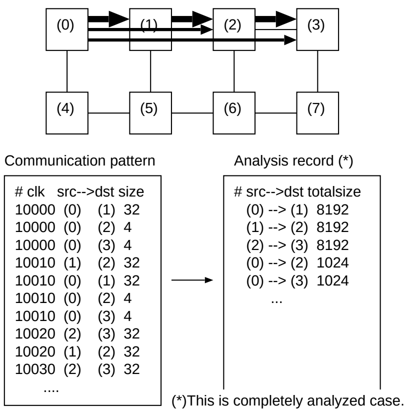
In the recent design methodology, target application is described in system level description language including System-C or Spec-C, and simulated in the early stage of design. After fixing the type of nodes and assignment of task, the communication pattern can be analyzed with System-C level simulation. Through this analysis, the total amount of communication data between each pair of nodes is ranked as follows:

1. Count the total size of data transferred between each pair of nodes in the target application.

2. Sort communication pairs by their total data amount.

As shown in Figure 4, communication pattern, which consists of clock, source, destination, and data size, is listed during the simulation for the design. Then all source-destination pairs are sorted in order of total communication amount to give priority of pairs, and recorded as an “analysis record”. That is, a source-destination pair with the large amount of total communication data has high priority to set paths in the next stage.

It is rarely possible that the communication data size cannot be known even in stream processing. In this case, only communication-pair list is used instead of the list shown in Figure 4. Although an analysis record including communication data amount is better to establish well-distributed paths, we can employ the *flee* even when the amount of communication data is unknown. We show the influence of the analysis records (completely analyzed case and incompletely analyzed case) on performance of the *flee* in Section 5.2.3.



```
# clk   src-->dst size
10000  (0)   (1)  32
10000  (0)   (2)  4
10000  (0)   (3)  4
10010  (1)   (2)  32
10010  (0)   (1)  32
10010  (0)   (2)  4
10010  (0)   (3)  4
10020  (2)   (3)  32
10020  (1)   (2)  32
10030  (2)   (3)  32
      ....
```

```
# src-->dst totalsize
  (0) --> (1)  8192
  (1) --> (2)  8192
  (2) --> (3)  8192
  (0) --> (2)  1024
  (0) --> (3)  1024
        ...
```

Figure 4. Communication pattern analysis. Communication pattern of target application is given and all source-destination pairs are sorted in order of total traffic amount.

4.2 Building Paths

Each path is established in order of the priority in the analysis record shown in Section 4.1. The deadlock-free deterministic path is made based on the total cost of channels passed through as follows.

1. Set the cost of all channels to 1 (minimum channel cost).

2. Select a source-destination pair, whose path is still not assigned, with the highest priority in the analysis record.

   (a) Establish a minimum cost path (not always minimal hop path) by means of the Dijkstra’s algorithm, which finds the shortest path in a weighted direct graph, under the condition that all assigned paths are deadlock-free.

   (b) Increase the cost of all channels passed through by its total data amount, or 1.

3. Repeat Steps 2-3 until all source-destination pairs are assigned into paths.

In Step 2 (a), deadlock-free condition is satisfied by employing the restriction of existing deadlock-free non-minimal adaptive routing, such as the Turn-Model[6].

In Step 2 (b), when communication pairs are completely analyzed, that is, the total amount of communication data is known, increase by the communication amount between the pair. In this case, total bit-length or byte-length of the communication data can be applied as the unit of communication amount. Otherwise, at incompletely analyzed case, increase by 1.

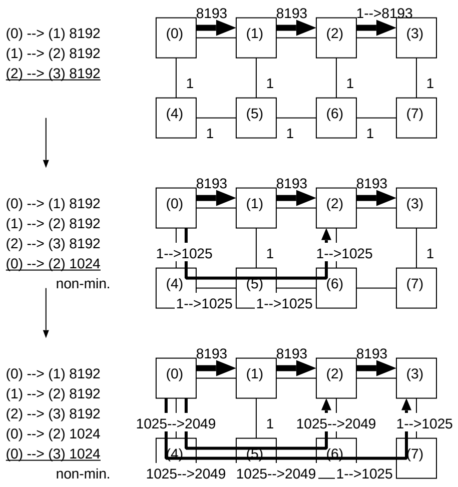


Figure 5. Paths of the flee. The communication pair (0)-(2) and (0)-(3) take a non-minimal path.

Figure 5 shows an example of path search under the West-first Turn-Model[6], when communication pairs of nodes are completely analyzed. As shown in Figure 5, high-priority paths, which transfer a large amount of data, are routed first, and almost minimal paths are assigned. As the routing progresses, the channel cost is increased from 1. That is, hot-spots channels will have high cost. Low-priority paths sometimes become non-minimal so as to avoid such hot-spot channels. The *flee* routing keeps mainstream minimal paths, while low-priority paths try to be established to make the best use of entire network resources.

The time complexity to establish all paths with *flee* routing strategy is $O(n^2k)$, where n is the number of switches and k is the number of paths to be established.

5 Performance Evaluation

In this section, we firstly evaluate the *flee* routing strategy and dimension-order routing on mesh and torus. Second, we compare the *flee* routing strategy and existing path selection algorithms[12] in collaboration with the up*/down* rule on mesh with faults.

Fault-tolerant system is sometimes required in NoCs, for link or node faults in the fabrication process. In such cases, fault redundant techniques for memory yield improvement[3] can be applied. That is, spare nodes are reserved for fault recovery, and faulty nodes are replaced after initialization. Thus, we use not only completely regular topologies but also those with faults.

Since access data of most stream applications can be pre-analyzed, we use the completely analyzed record to build paths in both first and second evaluations. However, in the last subsection, we will compare the incomplete- and complete-analyzed records on the *flee*.

5.1 Simulation Environment

5.1.1 Network Model

A flit-level simulator written in C++ was developed for analysis. As target topologies, 16-node 2-D mesh and torus are used. They are known as typical connection topologies of NoCs [5][1][9]. Every router has five ports; a port is connected to a single node, and others are used for neighboring routers. A simple model consisting of channel buffers, crossbar, link controller, and control circuits is used for the switching fabric in the router. A header flit requires at least three clock cycles to be transferred; one cycle for routing, one cycle for transferring a flit from an input channel to an output channel through a crossbar, and the remaining cycle for transferring the flit to the next router or host. Wormhole switching is used as a switching method for the router. The simulation time is set to more than 1,000,000 cycles.

We employ the West-First Turn-Model[6] to perform deadlock-free condition in the *flee* routing strategy on mesh and torus without faults. In this method and dimension-order routing, the number of virtual channels is one or two in mesh or torus, respectively. On the other hand, the up*/down* routing (rule) with one virtual channel is used for the *flee*, and path selection algorithms in mesh and torus with faults. Two faulty links are randomly selected among the links that bear the highest communication load through each application to be simulated. We employ the same breadth-first spanning tree for the up*/down* routing. Here, the *flee* routing strategy is compared with the two path selection algorithms: 1) random selection, and 2) the traffic balancing algorithm, which uses static analysis of routing paths, proposed by Sancho *et al.*[12] for SANs. In the simulations, the random and Sancho's algorithm select a path from only shortest paths, but the *flee* routing strategy uses both shortest and non-shortest paths.

5.1.2 Traffic Pattern

In the *flee* routing strategy, since traffic pattern is an important performance factor, we used practical stream processing application programs: JPEG codec, Viterbi decoder and IPsec accelerator[14]. They were originally developed for implementation on NEC electronics' dynamically reconfigurable processor DRP[11]. The target systems are consisting of 16 tiles, each of which can process a task of the target streaming processing. In this case, it is assumed to be a node. These applications are designed with C-level

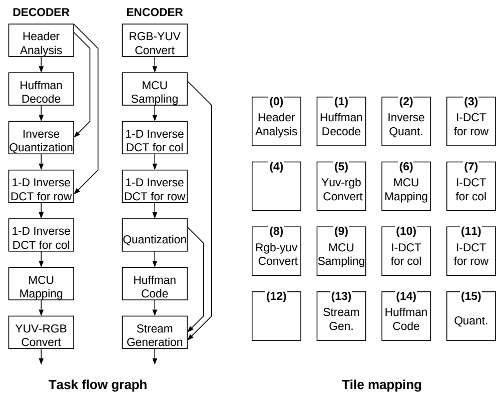


Figure 6. Task flow graph and mapping result for JPEG decoder and encoder.

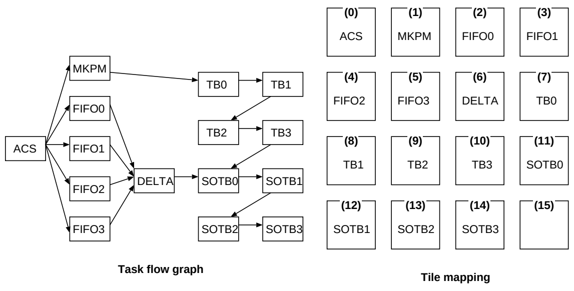


Figure 7. Task flow graph and mapping result for Viterbi decoder.

language, and the amount of communication data is analyzed in the C-level simulation. In this evaluation, total bit-length of each communication pair is used as its total amount of communication, which increases the cost of all channel passed through.

The JPEG encoder and decoder derived from [1] have been mapped onto 16-node 2-D mesh and torus. Their task flow graph and mapping results are shown in Figure 6. Tasks for the JPEG decoder are mapped onto nodes (0)-(7), while tasks for the encoder are mapped onto (8)-(15). A 16×16 pixel data is sequentially processed by each task in the pipelined manner.

The Soft-Input Soft-Output (SISO) Viterbi decoder has been implemented for 16-node NoCs as shown in Figure 7. The input data is transferred to the Add-Compare-Select (ACS) logic. ACS logic generates the Branch Metric and stores it into an FIFO. ACS also generates the Path Metric. Then, the most probable state is selected in Make Path Metric (MKPM), and the most probable Branch Metric is selected in DELTA. Finally, an error corrected code is generated in Trace Back (TB) and Soft Output Trace Back (SOTB).

Figure 8 shows task flow graph and mapping results for

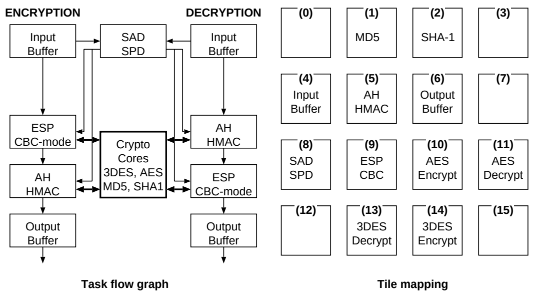


Figure 8. Task flow graph and mapping result for IPsec accelerator.

IPsec accelerator. The cryptographic parameters such as algorithm and key to be used are stored in Security Association Database (SAD), and security policies are managed in Security Policy Database (SPD). For Encapsulated Security Payload (ESP), input packets are encrypted/decrypted with Triple-DES-CBC or AES-CBC. For Authentication Header (AH), the hashed value is calculated with HMAC-MD5 or HMAC-SHA-1.

For comparison, we also evaluated the case when *uniform* traffic is used. It is a synthetic traffic in which all nodes sends packets to every other node randomly. A host injects a packet independently of each other, and we set packet length for 259 flits including two header flits.

The main stream of the JPEG trace is sequentially processed, and the Viterbi trace includes fork and join patterns. As mentioned at Section 2, most of stream applications have more than one stream in which a series of task is sequentially processed. Thus, we consider that simulation results of the three traces, which are used in this evaluation, can be applicable for other stream processings.

5.2 Simulation Results

5.2.1 Two-dimensional Mesh and Torus

The *flee* routing strategy under the Turn-Model is compared with dimension-order routing (DOR) on regular topologies with the three stream applications and uniform traffic.

Figure 9 and 10 show the accepted traffic versus latency with the traffic from Viterbi on mesh and torus, respectively. In these figures, “Flee” shows the latency when the *flee* is used, while “DOR” is for the dimension-order routing. The average hops of both methods are also shown in the parenthesis. On the 2-D mesh topology (Figure 9), the average hops of packets are 2.52 for the *flee*, and 1.84 for the dimension-order routing. Thus, the *flee* actually takes a number of non-minimal paths in the stream application. By employing non-minimal paths, the *flee* routing strategy distributes the paths, and improves the accepted traffic

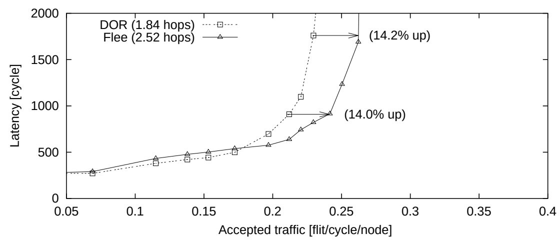


Figure 9. Viterbi trace on 4x4 mesh.

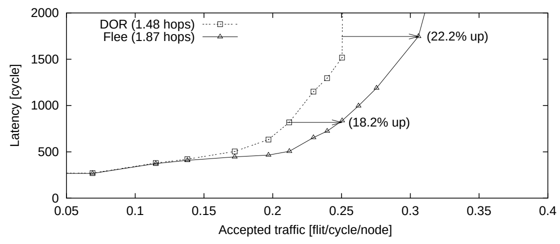


Figure 10. Viterbi trace on 4x4 torus.

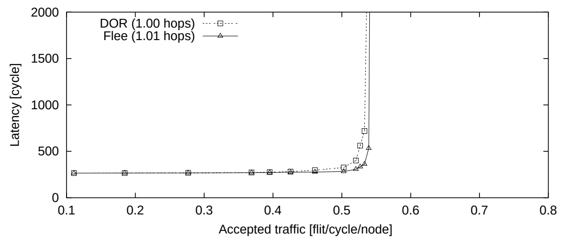


Figure 11. JPEG trace on 4x4 mesh.

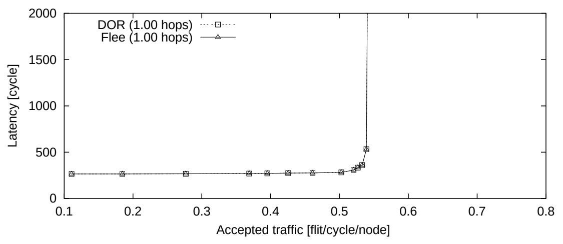


Figure 12. JPEG trace on 4x4 torus.

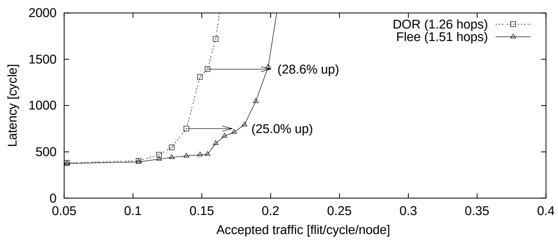


Figure 13. IPsec trace on 4x4 mesh.

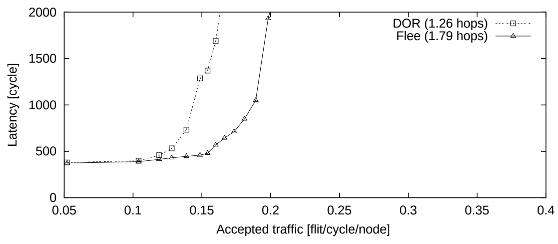


Figure 14. IPsec trace on 4x4 torus.

(throughput) up to 14.2% of dimension-order routing. In addition, on the 2-D torus (Figure 10), the *flee* improves the throughput up to 22.2% of the dimension-order routing.

Figure 11 and 12 show the accepted traffic versus latency under the JPEG traffic on the two topologies. In this application, the main-stream data is almost sequentially processed in each task as depicted in Figure 6, and each task is manually mapped onto nodes to minimize the average hops. In this case, most communications are between neighboring nodes, and the number of effective alternative non-minimal paths is limited. Thus, this comparison shows no significant difference between the *flee* and dimension-order routing.

Figure 13 and 14 show the network performance under the IPsec trace on the two topologies. The former graph also shows the up to 28.6% of throughput improvement against dimension-order routing. As shown in Figure 8, since all data streams are encrypted, decrypted, or hashed in cryptographic cores, each cryptographic core tends to be bottlenecks. Like Viterbi trace, non-minimal paths are effective for distributing traffic and reducing the congestion in this application.

For comparison, we also evaluated using uniform traffic. Although the *flee* routing strategy is designed to cope with a high degree of communication locality, we must know how it works in the worst condition. Figure 15 shows the accepted traffic versus latency on 4×4 2-D mesh topology, and Figure 16 shows that on 4×4 2-D torus. In both cases, any further path distributions are difficult to find even though the non-minimal paths are introduced. This is because the dimension-order routing already achieves the well-distributed paths. Thus, non-minimal paths only consume network resources but hardly decrease the hotspots, and these figures show the inefficiency of the *flee* on the uniform traffic. Notice that, the *flee* routing strategy tends to make non-minimal paths even in uniform traffic, because each path is assigned in order of priority in the analysis record and it tries so as not to disturb the previous paths.

Consequently, the *flee* routing strategy improves the network performance, when the network has a high degree of communication locality, typical streaming processing generates.

5.2.2 Two-dimensional Mesh With Link Faults

In this subsection, we evaluate the *flee* routing strategy on mesh with link-faults, that is the fewer available link cases.

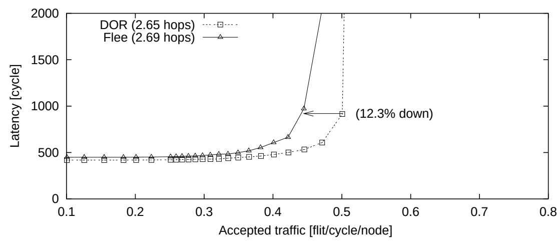


Figure 15. Uniform traffic on 4x4 mesh.

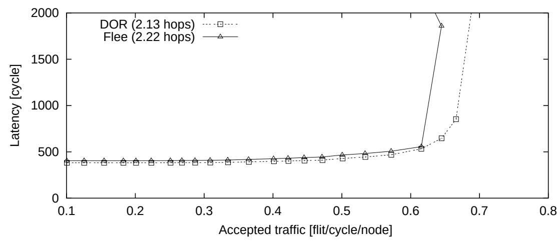


Figure 16. Uniform traffic on 4x4 torus.

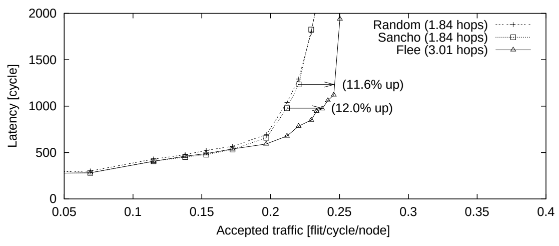


Figure 17. Viterbi trace on 4x4 mesh with no link-fault.

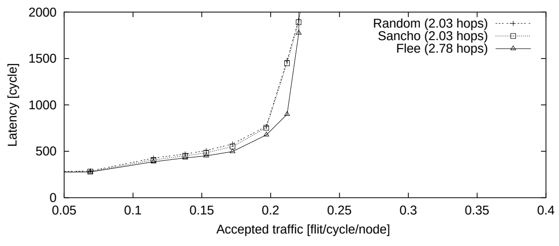


Figure 18. Viterbi trace on 4x4 mesh with two link-faults.

Accepted traffic versus latency for the Viterbi trace is shown in Figure 17 for 2-D mesh without any faulty link, and Figure 18 shows the case with two faulty links. As shown in the fault free case (Figure 17), the average hops are 3.01 for the *flee*, and 1.84 for Sancho's algorithm, then the *flee* increases the throughput up to 12.0% compared with other path selection algorithms which use only minimal paths. Contrastively, in the case of two faulty links, the *flee* still outperforms other path selection algorithms, however its advantage is degraded as shown in Figure 18. The non-minimal path originally consumes, more or less, additional network resources by just increased hops. When some of alternative paths are disabled by link-faults, the network resources of employing non-minimal paths become not negligible in the case with two faulty links. This is the reason why the *flee* degrades its advantages in the case with two faulty links.

Same evaluation is also conducted with JPEG trace and IPsec trace. For JPEG trace, as the same reason mentioned at Section 5.2.1, no significant difference is shown between the *flee* and other path selection algorithms. On the other hand, the evaluation result for IPsec trace is similar to that for Viterbi trace. In the fault free case, the *flee* reduces the latency compared with the path selection algorithms, which employ only minimal paths. In the case with two faulty links, the *flee* still outperforms others, but the difference between the *flee* and other path selection algorithms becomes small because of the fewer links.

Consequently, the *flee* successfully distributes the congestion especially at traffic patterns with high degree of access locality, and it improves throughput and latency of most of typical stream application under various conditions.

5.2.3 Effect of Analysis Records

To demonstrate the effect of analysis records in the *flee*, we shift to compare the complete analysis record, and incomplete analysis record including no entries of communication data amount. Figure 19, 20, and 21 show evaluation results under various access patterns. In these figures, "Flee.in" and "Flee" are the *flee* with incomplete- and completely-analysis records. When using incomplete analysis record, the performance of the *flee* is not stable, and sometimes, it is close to that of the dimension-order routing as shown in Figure 19. This is because that paths transfer various sizes of communication data, and communication data sizes are a key factor to improve routing paths. At incomplete analysis case, the *flee* still establishes paths based on the order of heavy-communication pairs, but channel cost function to increment by 1 is not enough to balance traffic in some cases. As described in Section 4, both a flow and size of data are completely pre-analyzed in most of stream processing applications. However, at incomplete analysis case (it is rare), the *flee* is sometimes infirm compared with that with complete analysis record.

6 Conclusions

In this paper, we proposed a deadlock-free deterministic routing strategy called *flee* for NoCs. The access pattern is often predictable in NoCs, and it includes a high degree of access locality, in which distance between communica-

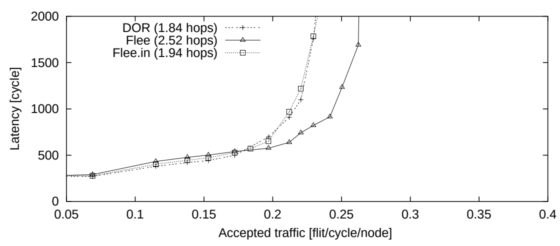


Figure 19. Incomplete analysis with Viterbi trace on 4x4 mesh under Turn-Model.

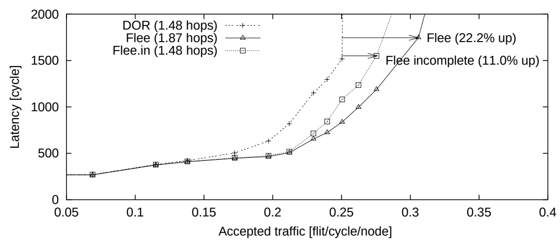


Figure 20. Incomplete analysis with Viterbi trace on 4x4 torus under Turn-Model.

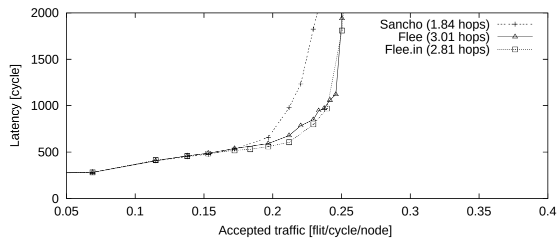


Figure 21. Incomplete analysis with Viterbi trace on 4x4 mesh under up*/down* rule.

tion pair is too short to distribute a path by minimal routing. Thus, the *flee* routing strategy introduces non-minimal paths so as to avoid path congestion. In the *flee*, the amount of communication which is estimated in the early stage of SoC design is taken into account to establish paths. According to the amount of communication, minimal paths are mostly assigned for communication pairs with a large amount of total communication data. Whereas paths that communicate a small data sometimes becomes non-minimal so as not to disturb the former paths. Simulation results show that the *flee* routing strategy successfully distributes the congestion over the NoCs using non-minimal paths, and improves up to 28.6% of throughput against existing minimal routing strategies on typical stream application programs. The *flee* routing strategy is able to be applied both when application traffic is completely analyzed, and when it is not. Simulation results also show that, by completely analyzing a total amount of data, the *flee* routing strategy improves performance compared with that using incompletely analysis record under part of applications.

References

[1] K. Anjo, Y. Yamada, M. Koibuchi, A. Jouraku, and H. Amano. BLACK-BUS: A New Data-Transfer Technique using Local Address on Networks-on-Chips. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, page 10a, Apr. 2004.

[2] ARM Ltd. Multi-Layer AHB Overview, 2001. available at http://www.arm.com/miscPDFs/1745.pdf.

[3] K. Chakraborty and P. Mazumder. *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*. Prentice Hall PTR, 2002.

[4] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transaction on Computers*, 36(5):547–553, May 1987.

[5] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the 38th Design Automation Conference*, pages 684–689, June 2001.

[6] C. J. Glass and L. M. Ni. The Turn Model for Adaptive Routing. *Proceedings of International Symposium on Computer Architecture*, pages 278–287, 1992.

[7] W. H. Ho and T. M. Pinkston. A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns. In *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture*, pages 377–388, Feb. 2003.

[8] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens. Programmable Stream Processors. In *IEEE Computer*, pages 54–62, Aug. 2003.

[9] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier. An Architecture and Compiler for Scalable On-Chip Communication. *IEEE Transactions on Very Large Scale Integration Systems*, 12(7):711–726, July 2004.

[10] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs. In *Proceedings of the Field-Programmable Logic and Applications (FPL)*, pages 795–805, Sept. 2002.

[11] M. Motomura. A Dynamically Reconfigurable Processor Architecture, Oct. 2002. presented in Microprocessor Forum.

[12] J. C. Sancho and A. Robles. Improving the Up*/Down* Routing Scheme for Networks of Workstations. In *Proceedings of the European Conference on Parallel Computing*, pages 882–889, Aug. 2000.

[13] Sonics Inc. SONICS Network Technical Overview, Jan. 2002. available at http://www.sonicsinc.com/.

[14] M. Suzuki, Y. Hasegawa, Y. Yamada, N. Kaneko, K. Deguchi, H. Amano, K. Anjo, M. Motomura, K. Wakabayashi, T. Toi, and T. Awashima. Stream Applications on the Dynamically Reconfigurable Processor. In *Proceedings of International Conference on Field Programmable Technology (FPT)*, pages 137–144, Dec. 2004.