Prediction Router: Yet Another Low Latency On-Chip Router Architecture *

Hiroki Matsutani[1], Michihiro Koibuchi[2], Hideharu Amano[1], and Tsutomu Yoshinaga[3]

[1]Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama,
JAPAN 223-8522
{matutani,hunga}@am.ics.keio.ac.jp

[2]National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo,
JAPAN 101-8430
koibuchi@nii.ac.jp

[3]The University of Electro-Communications
1-5-1, Chofugaoka, Chofu-shi, Tokyo, JAPAN 182-8585
yosinaga@is.uec.ac.jp

Abstract

Network-on-Chips (NoCs) are quite latency sensitive, since their communication latency strongly affects the application performance on recent many-core architectures. To reduce the communication latency, we propose a low-latency router architecture that predicts an output channel being used by the next packet transfer and speculatively completes the switch arbitration. In the prediction routers, incoming packets are transferred without waiting the routing computation and switch arbitration if the prediction hits. Thus, the primary concern for reducing the communication latency is the hit rates of prediction algorithms, which vary from the network environments, such as the network topology, routing algorithm, and traffic pattern. Although typical low-latency routers that speculatively skip one or more pipeline stages use a bypass datapath for specific packet transfers (e.g., packets moving on the same dimension), our prediction router predictively forwards packets based on a prediction algorithm selected from several candidates in response to the network environments. In this paper, we analyze the prediction hit rates of six prediction algorithms on meshes, tori, and fat trees. Then we provide three case studies, each of which assumes different many-core architecture. We have implemented a prediction router for each case study by using a 65nm CMOS process, and evaluated them in terms of the prediction hit rate, zero load latency, hardware amount, and energy consumption. The results show that although the area and energy are increased by 6.4-15.9% and 8.0-9.5% respectively, up to 89.8% of the prediction hit rate is achieved in real applications, which provide favorable trade-offs between the modest hardware/energy overheads and the latency saving.

*This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with STARC, e-Shuttle, Inc., and Fujitsu Ltd.

1 Introduction

As semiconductor technology improves, the number of processing cores integrated on a single chip has continually increased. More recently, commercial or prototype chips that have 64 or more processing cores have already been produced [5, 19, 21]. To connect such many cores, Network-on-Chips (NoCs) [6, 2, 3] that introduce a packet-switched network structure have been widely employed instead of traditional bus-based on-chip interconnects.

NoCs are quite latency sensitive, since their communication latency determines the application performance on the many-core architectures, and it is becoming more dominant factor when the number of cores on a chip increases. Unfortunately, the communication latency in packet-switched networks tends to be large compared with that in bus-based systems, since packets are forwarded via many routers, each of which performs complex operations including the routing computation, switch allocation, and switch traversal for every packet or flit, as shown in Figure 1.

To reduce the communication latency on interconnection networks, various router architectures have been recently developed. A speculative router speculatively performs different pipeline stages of a packet transfer in parallel [18, 7, 11]. The look-ahead routing technique removes the control dependency between the routing computation and switch allocation in order to perform them in parallel [7, 15, 16]. In addition, aggressive low-latency router architectures that bypass one or more pipeline stages for specific packet transfers have been proposed [13, 8, 14, 17, 12].

As yet another low-latency router architecture, we propose the prediction router that predicts an output channel being used by the next packet transfer and speculatively completes the switch arbitration. In the prediction routers, incoming packets are transferred without waiting the routing computation and switch arbitration if the prediction hits. Otherwise, they are transferred through the original pipeline stages without any additional latency overheads. Thus, the

primary concern for reducing the communication latency is the hit rates of prediction algorithms, which vary from the network environments, such as the network topology, routing algorithm, and traffic pattern. For example, a predictor that exploits the path regularity of dimension-order routing on meshes or tori [8] would work well for packet transfers that span multiple hops, but it rarely hit when a given traffic pattern contains a lot of neighboring communications. Although existing low-latency routers that speculatively skip one or more pipeline stages use a bypass datapath for specific packet transfers [13, 8, 14, 17, 12], our prediction router predictively forwards packets based on a prediction algorithm selected from several candidates in response to the network environments.

Although only the concept of predictive switching was proposed in [22] and [23] for interconnection networks of massively parallel computers, they neither showed their on-chip router architecture nor evaluated its performance with various prediction algorithms, cost, and energy consumption based on the detailed router design. In this paper, first, we propose the prediction router architecture that can select a prediction algorithm from several candidates in response to the network environments. Second, we present formulas that estimate the prediction hit rates of various algorithms on meshes, tori, and fat trees. Third, we provide three case studies, each of which assumes different many-core architecture. We have implemented a prediction router for each case study by using a 65nm CMOS process, and evaluated them in terms of the prediction hit rate, zero load latency, hardware amount, and energy consumption. Our claim is that the low-latency routers should support multiple prediction algorithms (not a single algorithm!) in order to deal with various traffic patterns, and this paper would be a first guide to find which prediction algorithms should be supported for a given network topology, routing algorithm, and traffic pattern.

The rest of this paper is organized as follows. Section 2 introduces typical low-latency router architectures. Section 3 explains the prediction router architecture. Section 4 analyzes the hit rates of various prediction algorithms on meshes, tori, and fat trees. Then, Section 5 evaluates the prediction router through the three case studies. Section 6 concludes this paper.

2 Related Work

We first show a pipeline structure of a baseline router, and introduce two conventional low-latency techniques, called the speculative and look-ahead transfers, to shorten the router pipeline stages. Then we describe the other aggressive low-latency router architectures that speculatively skip one or more pipeline stages.

2.1 Conventional Router

A 4-cycle router quoted from [7] is assumed as a baseline router in this paper. In the router, a header flit is transferred through four pipeline stages that consist of the routing computation (RC) stage, virtual channel allocation (VA) stage for output channels, switch allocation (SA) stage for allocating the time-slot of the crossbar switch to the output channel, and switch traversal (ST) stage for transferring flits through the crossbar.

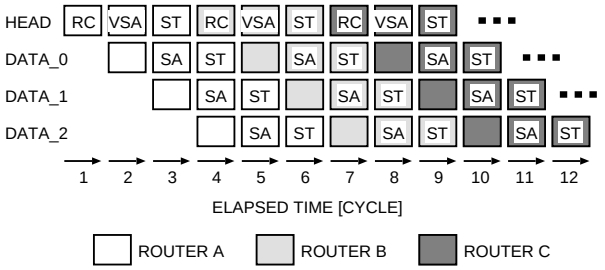


Figure 1. Router pipeline structure (3-cycle)

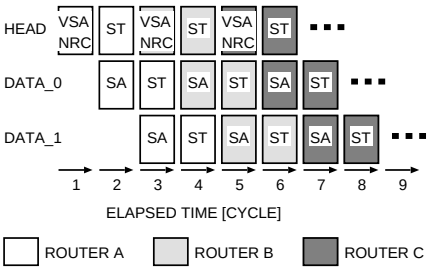


Figure 2. Router pipeline structure (2-cycle)

2.2 Speculative Router

A well-known technique to reduce the number of pipeline stages in a router is the speculative transfer that performs different pipeline stages in parallel [18, 7, 11]. Figure 1 shows an example of the speculative router that performs the VA and SA in parallel. These operations are merged into a single stage, called the virtual channel and switch allocation (VSA) stage. Notice that when the VA operation in the VSA stage cannot be completed due to the conflicts with other packets, the SA operation also fails regardless of its result and must be re-executed. For further reducing the pipeline stages, the double speculation that performs RC, VA, and SA operations in parallel is possible, but it would degrade the performance due to the frequent miss speculations and retries.

2.3 Look-Ahead Router

The look-ahead routing technique removes the control dependency between the routing computation and switch allocation in order to perform them in parallel, by selecting the output channel of the i-th hop router in the $(i-1)$-th hop router [7]. That is, each router performs the routing computation for the next hop (denoted as NRC), as shown in Figure 2. Since the computational result at the NRC stage of the $(i-1)$-th hop router is used in the i-th hop router, the result does not affect the following VA and SA operations

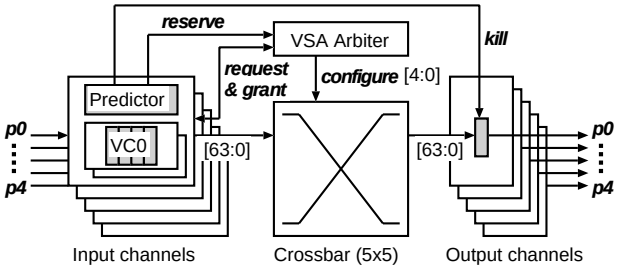


Figure 3. Prediction router architecture

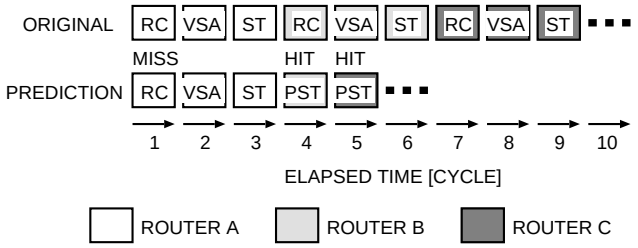


Figure 4. Prediction router pipeline

in the $(i-1)$-th hop router; therefore the NRC and VSA operations can be performed in parallel (Figure 2).

However, the NRC stage should be completed before the ST stage in the same router, because the hint bits in a packet header, which are the results of the NRC, must be updated for the NRC/VSA stage of the next router before the packet header is sent out. Thus, the control dependency between the NRC and ST stages in a router still remains difficulty for shortening to a single cycle router without harming the frequency, although some aggressive attempts using this approach have been done [7, 15, 16].

2.4 Bypassing Router

This section introduces existing aggressive low-latency router architectures that bypass one or more pipeline stages for the specific packet transfers, such as paths frequently used so far [17], packets continually moving along the same dimension [8], and paths pre-specified [14, 12].

Express virtual channels (VCs) have been proposed to reduce the communication latency by bypassing pipeline stages in intermediate routers between non-adjacent nodes [13]. This method is efficient for reducing the communication latency of long-haul packet transfers that span multiple intermediate routers; however, it does not work well for communications between neighboring routers. Dynamic fast path architecture also reduces the communication latency of frequently used paths by sending a switch arbitration request to the next node before flits in the fast path actually enter the next node [17].

Mad-postman switching exploits the path regularity of dimension-order routing on meshes for reducing the communication latency on off-chip networks that use bit-serial physical channels [8]. In Mad-postman switching, a router forwards an incoming packet to the output channel in the same dimension as soon as it receives the packet. Thus, it also has a problem with neighboring communications that do not go straight.

Preferred path employs a bypass datapath that connects input and output channels without using the crossbar switch in a router, in addition to the original datapath that goes through the crossbar [14]. The bypass datapath can be customized so as to reduce the communication latency between the specific source-destination pairs; however, it cannot adaptively select the bypass (or original) datapath for each packet in response to the recent traffic pattern. Also, Default-backup path (DBP) mechanism provides a low-latency unicursal ring network that spans all routers on a chip, though it has been originally proposed for on-chip fault-tolerance [12]. Only the packets that move along the DBP can be transferred in a single cycle.

All techniques listed above can bypass some pipeline stages only for the specific packet transfers. However, the network environments (e.g., routing algorithm and traffic pattern) will be easily changed, since multiple applications are usually running on an NoC. A technique that accelerates only a specific traffic may not work well for another one. The low-latency routers should thus support multiple prediction algorithms in order to deal with more than one application. In addition, we do need a guide to find which prediction algorithms should be supported for a given network topology, routing algorithm, and traffic pattern.

3 Prediction Router

In this section, we propose the prediction router architecture, which predictively forwards packets without waiting the RC, VA, and SA operations, based on the prediction algorithm selected from several candidates in response to the network environments, such as the network topology, routing algorithm, and traffic pattern.

The prediction router architecture and its pipeline structure are described in Section 3.1 and Section 3.2, respectively. Six prediction algorithms designed for the prediction router are introduced in Section 3.4.

3.1 Router Components

For purposes of illustration, we first introduce the baseline router architecture, and then we show the changes required for the prediction router.

As a baseline router, we assume a dimension-ordered wormhole router that has five physical channels, each of which has two virtual channels. It thus has five input channels, five output channels, a 5×5 crossbar switch, and an arbitration unit, as shown in Figure 3. When an input channel receives a packet, it computes an output channel to be used based on the destination address stored in the header flit. It asserts the *request* signal to the arbiter in order to allocate a time-slot of the crossbar for the requested output channel. The channel access is granted by the arbiter when the request wins the arbitration. Then it will be released after the corresponding tailer flit goes through the crossbar.

Algorithm 1 State transitions of an input channel (idata: an input flit, odata: an output flit)

```
1: if current_state == RC && idata is a packet header then
2:     port ← routing_computation(idata); current_state ← VSA;
3:     if predicted_port is free then
4:         odata ← idata; {predictive switch traversal (PST)}
5:         if predicted_port == port then
6:             current_state ← ST; {the prediction hits}
7:         else
8:             kill ← predicted_port; {remove the mis-routed flit
               at the output channel}
9:         end if
10:    end if
11: else if current_state == VSA then
12:    normal VSA operation;
13: else if current_state == ST then
14:    normal ST operation;
15: else if current_state == RC && no packet arrives then
16:    predicted_port ← prediction(); {an output channel likely
       to be used is selected and reserved}
17: end if
```

Figure 3 illustrates the prediction router architecture changed from the original one mentioned above. The changes required for the prediction router are as follows: 1) adding a predictor for each input channel, 2) changing the arbitration unit so that it can handle the tentative reservations from predictors, and 3) adding the *kill* signal for each output channel in order to remove mis-routed flits when the prediction fails.

The predictor in an input channel forecasts which output channel will be used by the next packet transfer while the input channel is idle. Then it asserts the *reserve* signal to the arbiter in order to tentatively reserve a time-slot of the crossbar for the predicted output channel, as shown in Figure 3. The details about the prediction algorithms are described in Section 3.4.

The arbiter handles the *request* and *reserve* signals from each input channel (Figure 3). Obviously, the former has higher priority than the latter; thus an output channel that has been reserved by an input channel will be preempted by another input channel that requests the output channel in the next cycle.

Although the predictive switching drastically reduces the packet delay at a router if the prediction hits, it generates mis-routed flits (or dead flits) if the prediction fails. To remove such dead flits inside a router, the input channel asserts the *kill* signal to the mis-predicted output channel if it predictively forwards a packet to the wrong output channel. The output channel masks all incoming data if the *kill* signal for it is asserted; thus the dead flits never propagate to the outside of the router.

3.2 Pipeline Structure

Figure 4 shows a timing diagram of a packet transfer from router (a) to router (c) in the cases of the 3-cycle speculative router (denoted as original) and the prediction router. In the prediction router, the prediction hits in router (b) and router (c), while it fails in router (a).

Since the prediction router completes the RC, VA, and SA operations prior to packet arrivals, it performs the predictive switch traversal (PST) as soon as a new packet arrives; thus the header flit transfer is completed in a single cycle without waiting the RC and VSA stages if the prediction hits. Otherwise, although dead flit(s) are forwarded to the wrong output channel, their propagation is stopped inside a router and their copy is forwarded to the correct output channel through the original pipeline stages (i.e., RC, VSA, and ST stages) without any additional latency overheads. Such state transitions of an input channel in the prediction router are summarized in Algorithm 1.

Notice that the predictive switching can be applied to various router architectures, such as the look-ahead 2-cycle router mentioned in the previous section. We will apply this concept to three case studies in Section 5.

3.3 Predictor Selection

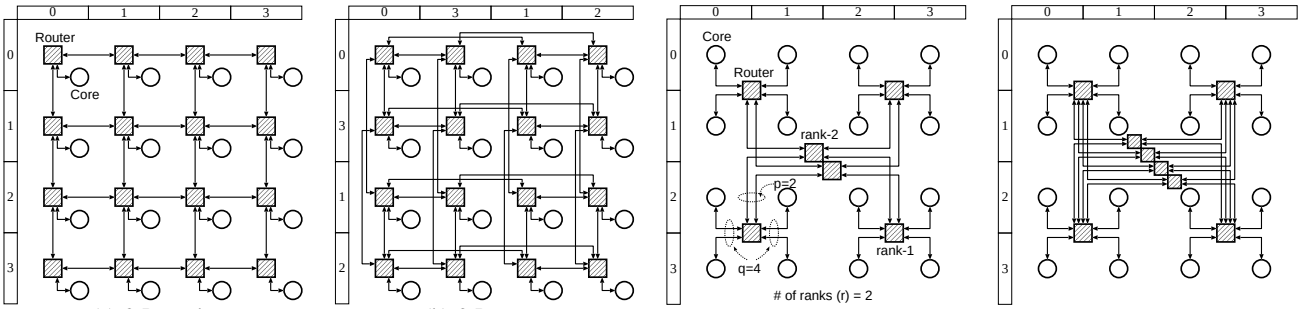
Each input channel can select a single prediction algorithm from all algorithms supported in the channel. The reconfiguration of the predictor is classified into two schemes: *static* and *adaptive*. A single algorithm is statically selected at the beginning of each application in the *static* scheme, while it is dynamically switched in the *adaptive* scheme. In this paper, we assume the *static* scheme, since it is efficient for many on-chip applications whose traffic pattern can be pre-analyzed at the design time by system-level simulations.

For unknown applications, here we introduce the *adaptive* scheme. Assume that an input channel supports n algorithms. All n algorithms predict their preferred output channel when the input channel receives a packet. Each algorithm increments its own counter when its prediction hits. For every m packet transfers, a prediction algorithm with the largest counter value is selected as the next algorithm. The value m determines the frequency of the reconfigurations. The *adaptive* scheme consumes more energy, since every algorithm predicts for each packet, though it can adaptively adjust the predictor even with unknown traffics.

3.4 Prediction Algorithms

Since some pipeline stages are skipped only when the prediction hits, the primary concern for reducing the communication latency is the prediction algorithm to be used. Fortunately, sophisticated value predictors have been developed for the speculative execution in microprocessors [4] and the universal information theory [9]. The prediction router can use some of them. Here we briefly describe the ones applicable to our lightweight prediction routers.

- **Random:** The simplest prediction algorithm is *Random*, which randomly predicts an output channel that will be used by the next incoming packet. In the case of dimension-order routing on 3-D torus, for example, a packet in an input channel on the y-dimension is transferred to an output channel on the y- or z-dimension, or local core. Thus, it randomly selects one of them.



(a) 2-D mesh (b) 2-D torus (c) Fat tree (2,4,2) (d) Fat tree (4,4,2)

Figure 5. Target network topologies

Since it exploits neither the topological regularity nor the traffic locality, its prediction hit rate tends to be quite low.

- **Static Straight (SS):** A simple and practical prediction algorithm optimized for dimension-order routing on meshes or tori is *SS*, which assumes that all incoming packets are continuing along the same dimension, as in Mad-postman switching [8]. In the case of dimension-order routing on 2-D mesh, for example, the *SS* predictor fails at most two times per a flight, since a packet may turn from the x-dimension to y-dimension in addition to the destination core. Therefore, packets that travel a long distance increase the prediction hit rate, whereas the communication locality negatively affects the *SS* predictor. It does not require any history tables (i.e., it is stateless); hence its implementation cost is low.
- **Custom:** The *Custom* strategy allows users to define which output channel each input channel should predict, as in Preferred path [14]. In the case of a router with five physical channels, for example, a 3-bit register to store the preferred output channel is required for each input channel.
- **Latest Port Matching (LP):** The *LP* strategy predicts that the next incoming packet will be forwarded to the same output channel as that of the previous packet. The *LP* predictor requires only a single history record in each input channel. The killer applications of *LP* are the traffic patterns that have strong access regularity, such as the straight-after-straight communications, and the repeated communications between two neighboring nodes.
- **Finite Context Method (FCM):** The nth-order FCM strategy predicts the most frequently used value after the last n-context sequence [4]. As n increases, the two-dimensional history table consisting of the n-context sequence and next value frequency gets bigger, while the prediction accuracy improves. For the sake of simplicity, in this paper, we have implemented and evaluated the 0th-order FCM predictor, which just selects the most frequently used output channel.
- **Sampled Pattern Matching (SPM):** The *SPM* algorithm was originally proposed as a universal predictor [9]. It selects a value which has the highest probability after a suffix sequence, called a marker, in a given data. The predicted value is calculated by applying a majority rule to all values appearing at positions just after the markers in the data. We can use it to predict an output-channel number by finding a value of the most frequently occurring number after the (longest) suffix sequence of communication history.

A preferred output channel is statically fixed in *SS* and *Custom*, while the other algorithms dynamically predict it according to the traffic pattern or their history table. Each algorithm requires different predictor circuit and provides different hit rate for a given traffic pattern. The following sections illustrate these trade-offs. In the next section, we analyze the prediction hit rates of these algorithms on meshes, tori, and fat trees. Their area and energy overheads are evaluated in Section 5 through three case studies.

4 Hit Rate Analysis

The reliable analytical model is essential to expect the prediction hit rate and communication latency of various prediction algorithms on a given network environment (e.g., topology, routing, and traffic). In this section, we present formulas that estimate the prediction hit rates of the proposed prediction algorithms on tori, meshes, and fat trees, respectively (see Figure 5). Using the analytical model, we can select or dynamically update the prediction algorithm in response to the environment.

The accuracy of the model will be confirmed through the simulations of three case studies in Section 5. Such confirmation is also important to guarantee the value and reliability of the analytical model.

4.1 Torus (k-ary n-cube)

To analyze the prediction hit rate on uniform traffic, we first count the number of paths passing through each channel on a given k-ary n-cube, assuming that the dimension-order routing is used on it. Then we derive the formulas that calculate the prediction hit rates of *SS*, *FCM*, *LP*, *SPM*, and *Random* algorithms.

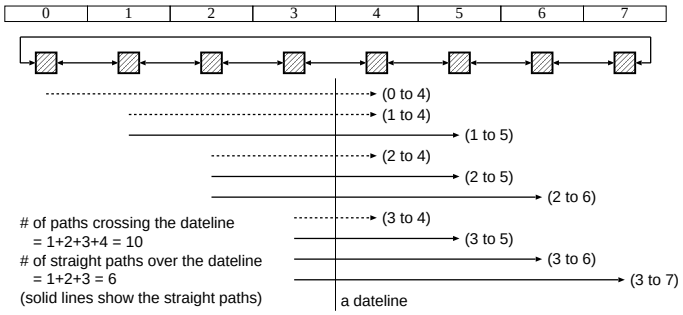


Figure 6. Path distribution on a k-ary 1-cube

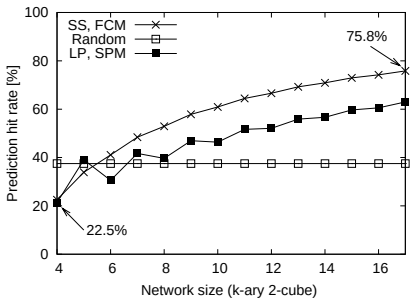


Figure 7. Prediction hit rates on k-ary 2-cube

4.1.1 Number of Paths

Here we assume that each node consists of a processing core and a router. In a k-ary 1-cube, the number of paths that go through a channel between neighboring two nodes, T_{1d}, and the number of paths that go straight at the next node, $T_{1d_{ss}}$, are calculated as follows (see Figure 6).

$$T_{1d} = 1 + 2 + \ldots + (\frac{k}{2} - \frac{1}{2}) = \sum_{i=1}^{\frac{k}{2}-\frac{1}{2}} i \quad (1)$$

$$T_{1d_{ss}} = 0+1+\ldots+(\frac{k}{2}-\frac{3}{2}) = \sum_{i=1}^{\frac{k}{2}-\frac{1}{2}} (i-1) = \sum_{i=1}^{\frac{k}{2}-\frac{3}{2}} i \quad (2)$$

We extend them to k-ary n-cubes, since most NoCs have employed two-dimensional topologies, and more recently, three-dimensional NoCs are also studied [10]. Assuming that k is an odd number, the number of paths on a channel, T, and the number of paths that go straight at the next node, T_{ss}, are calculated as follows.

$$T = k^{n-1} \sum_{i=1}^{\frac{k}{2}-\frac{1}{2}} i \quad (3)$$

$$T_{ss} = k^{n-1} \sum_{i=1}^{\frac{k}{2}-\frac{3}{2}} i \quad (4)$$

If k is an even number, on the other hand, there are two ways to reach a destination $\frac{k}{2}$ hops away in a dimension: one uses the wrap-around channel and the other does not. Assuming that both paths are equally used, the number of paths that go through a channel can be estimated as follows.

$$T = k^{n-1} \sum_{i=1}^{\frac{k}{2}} (i - \frac{1}{2}) = k^{n-1} (\sum_{i=1}^{\frac{k}{2}} i - \frac{k}{4}) \quad (5)$$

$$T_{ss} = k^{n-1} \sum_{i=1}^{\frac{k}{2}-1} (i - \frac{1}{2}) = k^{n-1} (\sum_{i=1}^{\frac{k}{2}-1} i - \frac{k}{4} + \frac{1}{2}) \quad (6)$$

In the case of dimension-order routing, a packet at an i-dimensional input channel is transferred to a j-dimensional output channel when it completes its movements on $1, 2, \ldots, (j-1)$ dimensional channels, where $i < j$. The number of paths from an i-dimensional input channel to a j-dimensional output channel, $T(j)$, and the number of paths from an i-dimensional input channel to a local processing core, $T_{PE}(i)$, are calculated as follows.

$$T(j) = (\frac{k}{2} - \frac{1}{2})^2 k^{n+i-j-1} \quad (7)$$

$$T_{PE}(i) = (\frac{k}{2} - \frac{1}{2}) k^{i-1} \quad (8)$$

4.1.2 Prediction Hit Rate

The prediction hit rate of *SS* algorithm is calculated as a percentage of the number of paths from the input channel to the predicted output channel, in the total number of paths that go through the input channel. Using the T and T_{ss}, the prediction hit rate of *SS* algorithm at an i-dimensional input channel is calculated as follows.

$$P_{ss} = \frac{T_{ss}}{T} \quad (9)$$

The *SS* predictor is equivalent to the 0th-order *FCM* predictor in the case of uniform traffic, because, for each input channel, the straight output channel in the same dimension is the most frequently used one; thus $P_{ss} = P_{fcm}$.

The *LP* predictor succeeds when two consecutive packets are transferred between the same pair of input and output channels in a router. Thus, the prediction hit rate of *LP* algorithm at an i-dimensional input channel is calculated as follows.

$$P_{lp} = (\frac{T_{ss}}{T})^2 + (\frac{T_{PE}(i)}{T})^2 + 2 \sum_{j=i+1}^{n} (\frac{T(j)}{T})^2 \quad (10)$$

The *SPM* predictor uses a history table that records the output channels used by packets. In the case of uniform traffic, since each core independently injects packets to a random destination, the hit rate of the *SPM* predictor is the same as that of the *LP* predictor; thus $P_{lp} = P_{spm}$.

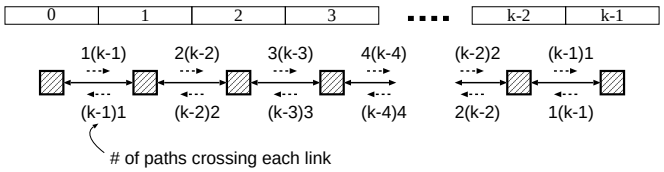


Figure 8. Path distribution on a k-ary 1-mesh

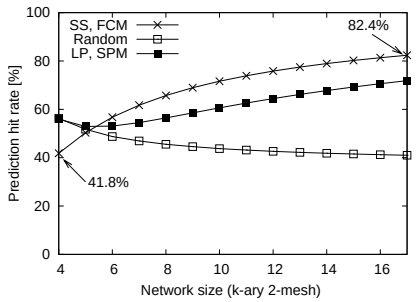


Figure 9. Prediction hit rates on k-ary 2-mesh

For the purpose of comparison, we calculate the hit rate of the *Random* predictor. In this case, a packet at an i-dimensional channel is transferred to one of j-dimensional channels or a local processing core, where $i \leq j$. Thus, the hit rate of the *Random* predictor at an i-dimensional input channel is calculated as follows.

$$P_{random} = \frac{1}{n} \sum_{j=i}^{n} \frac{1}{2(n-j+1)} \tag{11}$$

Finally, the prediction hit rate of each algorithm at a local input channel is calculated as follows.

$$P_{ss_{PE}} = P_{fcm_{PE}} = \frac{T_{PE}(n)}{k^n - 1} \tag{12}$$

$$P_{lp_{PE}} = P_{spm_{PE}} = \frac{2\sum_{j=1}^{n} T_{PE}(j)^2}{(k^n-1)^2} \tag{13}$$

Based on the formulas listed above, we have estimated the prediction hit rates of the *SS*, *FCM*, *LP*, *SPM*, and *Random* predictors on k-ary 2-cubes, where $4 \leq k \leq 17$. As shown in Figure 7, the hit rates of *SS*, *FCM*, *LP*, and *SPM* increase as the number of nodes (i.e., k) increases, while that of *Random* is constant. We have also confirmed these results by using a cycle-accurate network simulator.

4.2 Mesh (k-ary n-mesh)

We derive the formulas that estimate the prediction hit rate of mesh by using the similar approach to the torus. The hit rate of each predictor at an i-dimensional input channel on a k-ary n-mesh is calculated as follows.

$$T = k^{n-1} \sum_{j=1}^{k-1} j(k-j) \tag{14}$$

$$P_{ss} = P_{fcm} = \frac{k^{n-1} \sum_{j=1}^{k-1} j(k-j-1)}{T} \tag{15}$$

$$P_{lp} = P_{spm} = P_{ss}^2 + \frac{k^{2(n-1)} \sum_{j=1}^{k} j}{T^2} + \sum_{d=i}^{n} \sum_{j=1}^{k} \frac{k^{2(n+i-d-1)}((k-j)^2 + (j-1)^2)}{T^2} \tag{16}$$

We here omit the formulas that estimate the prediction hit rate at a local input channel on mesh, since they are quite similar to those on torus.

Figure 9 shows the prediction hit rates of the prediction algorithms on k-ary 2-meshes, where $4 \leq k \leq 17$. Since edge or corner nodes on mesh have fewer links than the others, the prediction hit rate of a mesh tends to be higher than that of the same-sized torus. We have confirmed these results in Section 5.1.3.

4.3 Fat Tree (p, q, r)

A fat tree is expressed with a triple (p, q, r), where p is the number of upward links, q is the number of downward links, and r is the number of ranks in the fat tree (see Figure 5). A router located in i-hop distance from a core is called a rank-i router, and a core is regarded as a rank-0 router, for the sake of convenience.

The adaptive up*/down* routing algorithm is employed in fat trees. That is, a fat tree provides multiple upward paths from a source core to the least common ancestor (LCA) of the source and destination cores, while it provides only a single deterministic path from the LCA to the destination core. In the upper transfers, the prediction succeeds if one of output channels towards the root has been selected as a predicted output channel; thus the prediction frequently hits in lower input channels, while it is difficult to predict a correct output channel in upper input channels, especially for uniform random traffic.

As in the cases of the torus and mesh, the prediction hit rate of each algorithm on fat tree is calculated based on the path distribution. Figure 10 illustrates the number of paths from a single source to several destinations. The number of paths at an upper output channel in a rank-i router, $T_{up}(i)$, is calculated as follows.

$$T_{up}(i) = (\frac{q}{p})^i (q^r - q^i) = \frac{q^{i+r} - q^{2i}}{p^i} \tag{17}$$

In a rank-i router, the number of paths from a lower input channel to a lower output channel, $T_{lo}(i)$, is calculated by using the regularity of fat tree.

$$T_{lo}(i) = \frac{q^{i-1}}{p^{i-1}} \tag{18}$$

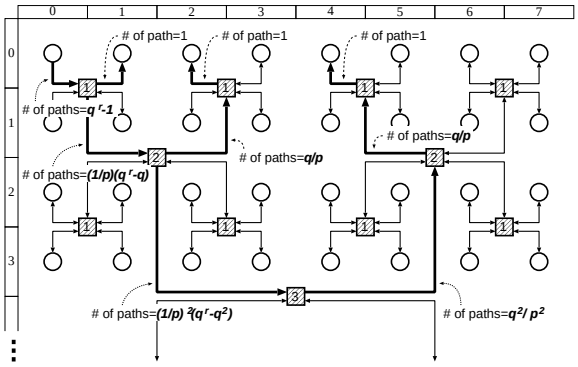


Figure 10. Path distribution on fat tree (1,4,3)

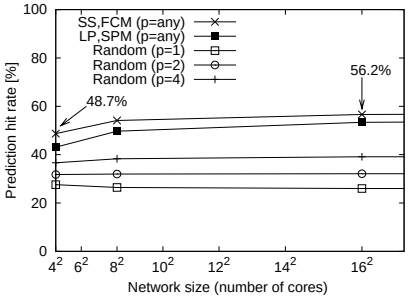


Figure 11. Prediction hit rates on fat tree (p,4,r)

The prediction hit rate of a lower input channel in a rank-i router is calculated as follows.

$$P_{ss} = P_{fcm} = \frac{pT_{up}(i)}{T_{up}(i-1)} \tag{19}$$

$$P_{lp} = P_{spm} = P_{ss}^2 + (q-1)(\frac{T_{lo}(i)}{T_{up}(i-1)})^2 \tag{20}$$

Again, in the upper transfers, the prediction succeeds if one of output channels towards the root has been selected as a predicted output channel. Notice that the *SS* predictor is equivalent to the 0th-order *FCM* predictor in uniform traffic, because the lower input channels frequently forward incoming packets to the upper output channels.

The prediction hit rate of an upper input channel is $\frac{1}{q}$, and that of a lower input channel at a rank-r router is $\frac{1}{q-1}$, since the traffic is uniformly distributed to lower channels.

Figure 11 shows the prediction hit rates of *SS*, *FCM*, *LP*, *SPM*, and *Random(p)* on fat trees $(p, 4, r)$, where $2 \leq r \leq 4$. One of case studies in the next section uses a fat tree to confirm these results (see Section 5.3).

5 Evaluations

In this section, we apply the prediction router architecture discussed in the previous sections to three case studies listed in Table 1. For each case study, we design a prediction router and compare it with an original router in terms of the prediction hit rate, zero load latency, hardware amount, and energy consumption.

5.1 Case Study 1: 256-Core Fine-Grained Operand Network

The first case study assumes a fine-grained operand mesh network that connects 256 ALUs with simple routers.

5.1.1 Router Architecture

Original Router The dimension-order routing and wormhole switching with no virtual channel are employed as the low-cost routing and switching techniques for the fine-grained operand mesh network. The router has a 4-flit input buffer for each channel. It has three pipeline stages that consist of the RC, SA, and ST stages, as shown in Figure 4. The distance between neighboring cores is typically short in fine-grained ALU networks; thus the link traversal (LT) is lumped into the ST stage in this case study.

Prediction Router We selected *SS* as a simple and practical predictor for the dimension-ordered routers on mesh. All channels except a local input channel in the router employ the *SS* predictor, while the local channel uses the *LP* predictor since *SS* does not work well in the injection channel. The prediction router transfers a header flit in a single cycle when the prediction hits. Otherwise, it takes at least three cycles depending on the packet conflicts. Dead flits are removed safely inside the router by the kill mechanism.

5.1.2 Simulated Throughput

First, we show the performance impact of the router delay in order to illustrate the advantages of low-latency routers. We used a cycle-accurate flit-level network simulator to measure the throughputs of five networks, each of which employs the 4-cycle, 3-cycle, 2-cycle, 1-cycle routers, and the *SS*-based prediction router (Pred(SS)), respectively. The other simulation parameters are listed in Table 1.

Figure 12(a) shows the simulation results. The Pred(SS) router network achieves 30.4% higher throughput compared with the 4-cycle network. The prediction router is regarded as a 1.4 cycle router when the prediction hit rate is 80%. In this case, its throughput is between the 1-cycle and 2-cycle.

5.1.3 Prediction Hit Rate

We estimated the prediction hit rate of the *SS*-based prediction router by using the cycle-accurate network simulator in order to confirm the accuracy of our formulas derived in Section 4. Figure 12(b) shows the simulation results. The curve of SS(sim) in the graph is very similar to that in the analysis results (Figure 9).

For reference, we estimated the simulated hit rates of the *FCM* and *LP* predictors. We also confirmed that their values are similar to those in Figure 9, though the FCM(sim) gives higher hit rates in small networks.

Table 1. Network specification of three case studies

	Case study 1	Case study 2	Case study 3
Topology	2-D mesh	2-D mesh	Fat tree (4,4,4)
# of cores	16×16 cores	8×8 cores	256 cores
Core distance	0.75mm	1.50mm	0.75mm
Traffic	Uniform	7 NPB programs + 4 synthetic patterns	Uniform
Routing	Dimension-order (deterministic)	Dimension-order (deterministic)	up*/down* (adaptive)
Switching	Wormhole; no VC	Wormhole; 2 VCs	Wormhole; no VC
Piplie structure	[RC][SA][ST]	[RC][VSA][ST][LT]	[RC][SA][ST]
Packet size	4-flit (1-flit=64-bit)	4-flit (1-flit=64-bit)	4-flit (1-flit=64-bit)
Input buffer	4-flit FIFO	4-flit FIFO	4-flit FIFO
Predictor(s)	SS	SS + LP + FCM	SS(LRU) + LP

5.1.4 Zero Load Latency

The zero load latency is the flight time of a single packet from source to destination with no packet conflicts. Based on the prediction hit rates obtained previously, we compare the original router and the prediction router in terms of their zero load latency.

Assuming a packet that consists of L flits including a single header flit goes through h wormhole routers, its zero load latency is calculated as follows.

$$T_0^{orig} = T_{lt}(h-1) + (T_{rc} + T_{vsa} + T_{st})h + L/\mathrm{BW}, \quad (21)$$

where T_{rc}, T_{vsa}, T_{st}, and T_{lt} are the latencies for the RC, VSA, ST, and LT stages, respectively.

The prediction router can skip the RC and VSA stages only when the prediction hits; thus its zero load latency is calculated as follows.

$$\begin{aligned} T_0^{pred} &= T_{lt}(h-1) + (T_{pst})hP_{hit} + \\ &\quad (T_{rc} + T_{vsa} + T_{st})h(1 - P_{hit}) + L/\mathrm{BW}, \end{aligned} \quad (22)$$

where T_{pst} is the latency for the predictive switch traversal (PST), and P_{hit} is the prediction hit rate.

Figure 12(c) shows the zero load latencies of the original router (Orig), the *SS*-based prediction router (Pred(SS)), and that with perfect (or oracle) prediction (Ideal). The prediction router reduces 48.2% zero load latency compared with the original one, in the case of the 256-core mesh.

5.1.5 Hardware Amount

To estimate the gate counts of the original and the *SS*-based prediction routers, we synthesized their RTL designs with a 65nm standard cell library by using the Synopsys Design Compiler version Y-2006.06-SP2. The behavior of the synthesized NoC designs was confirmed through gate-level simulations at an operating frequency of 500MHz.

Figure 12(d) shows the gate counts of the original router (Orig) and the prediction router (Pred(SS)). Although the prediction router requires the predictors, kill signals, and a modified arbiter, its area overhead is only 10.1% compared with the original one that requires at least three cycles to forward a header flit.

5.1.6 Router Critical Path

Figure 12(e) shows the maximum delay of each pipeline stage of the original and prediction routers. As shown, the critical paths of both routers are on their VSA stage, and their differences are quite small (i.e., 6.2%). Note that the prediction router forwards a flit from its RC stage if the prediction hits (see line 4 of Algorithm 1). This is the reason why the delay of RC stage in the prediction router is much larger than that in the original one.

5.1.7 Energy Consumption

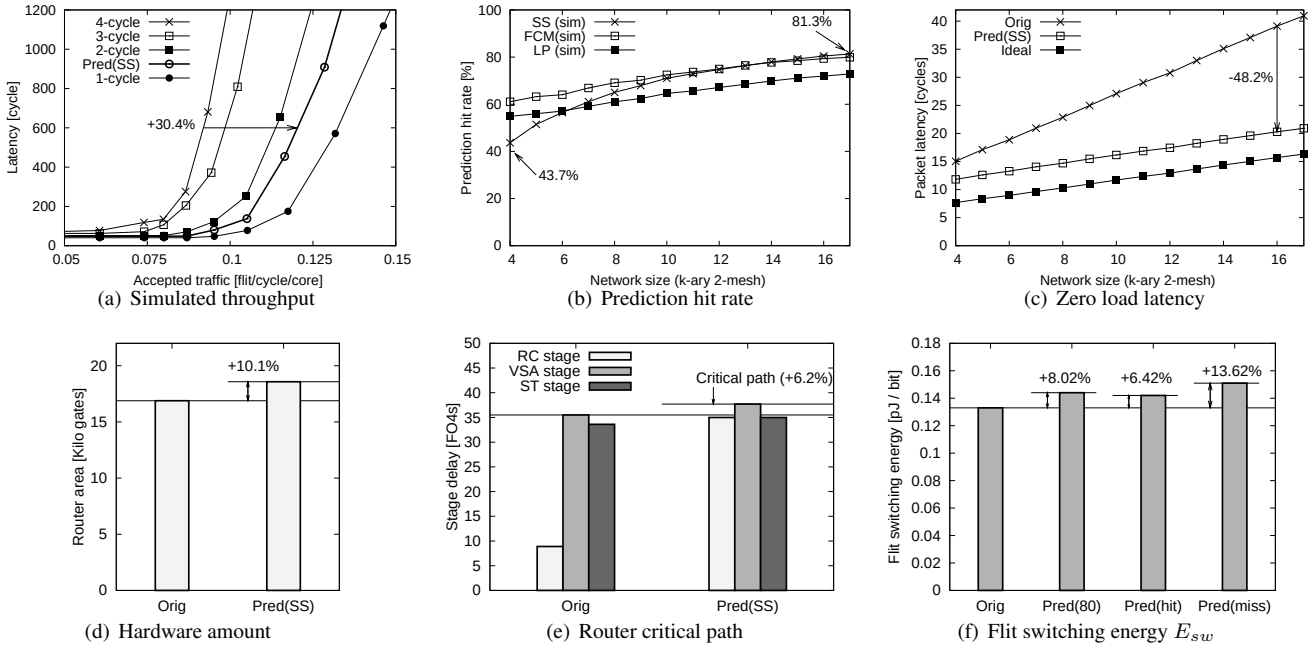
The average energy consumption to transmit a single flit from source to destination can be estimated as [20]

$$E_{flit} = wH_{ave}(E_{sw} + E_{link}), \quad (23)$$

where w is the flit-width, H_{ave} is the average hop count, E_{sw} is the average energy to switch the 1-bit data inside a router, and E_{link} is the 1-bit energy consumed in a link. Here we compare the prediction router with the original one in respect to E_{sw}, since their E_{link} values are the same.

We used the Synopsys Power Compiler in order to extract the E_{sw} of these routers. They were synthesized, placed, and routed with the 65nm standard cell library. The switching activities of the running routers were captured through the gate-level simulations operating at 500MHz with a 1.2V core voltage.

Figure 12(f) shows the flit switching energy E_{sw} [pJ/bit] extracted from the switching activities. Pred(hit) and Pred(miss) show the energy of the prediction router when the prediction hits and misses, respectively. Regardless of hit or miss, the prediction router requires slightly more energy due to the prediction. In addition, more energy is consumed when the prediction misses, since the dead flits are removed and a copy of them is transferred to the correct output channel as well as the original router. Pred(hit) and Pred(miss) respectively require 6.4% and 13.6% more energy compared with the original value. Although the energy overhead of Pred(miss) is costly, approximately 80% of the predictions hit in a 16×16 mesh, as mentioned in Section 5.1.3. Thus, the expected E_{sw} assuming that the 80% of the predictions hit (denoted as Pred(80)) is close to Pred(hit),



(a) Simulated throughput (b) Prediction hit rate (c) Zero load latency

(d) Hardware amount (e) Router critical path (f) Flit switching energy E_{sw}

Figure 12. Evaluation results of case study 1

and its overhead is only 8.0%. Note since both routers consume the same E_{link} for each flit, the total energy overhead for transmitting a flit is less than 8.0%, depending on the link length.

Throughout this case study, the evaluation results of the prediction router showed that although the area and energy are increased by 10.1% and 8.0% respectively, the communication latency is reduced by 48.2%; thus the prediction router presents favorable trade-offs between these modest overheads and the latency saving.

5.2 Case Study 2: 64-Core Chip Multiprocessor Network

The second case study assumes a chip multiprocessor network that connects 64 processors with virtual channel routers. Multiple prediction algorithms are evaluated on the network with seven parallel applications taken from the NAS parallel benchmark (NPB) [1] programs, in addition to typical synthetic traffic patterns, in order to clearly show their strong and weak points.

5.2.1 Router Architecture

Original Router Dimension-order routing and wormhole switching with two virtual channels are employed as a routing and a switching technique, respectively. The router has a 4-flit input buffer for each virtual channel. Since we use one cycle for the link traversal (LT), a header flit is transferred to the next router or core in at least four cycles.

Prediction Router We need multiple prediction algorithms, since various parallel applications are running on the network. Here we selected *SS*, *LP*, and *FCM* as simple and practical predictors. We implemented them on two prediction routers. Pred(SS+LP) supports *SS* and *LP*, while Pred(SS+LP+FCM) supports *FCM* in addition to *SS* and *LP*. They can dynamically change their prediction algorithm in a single cycle in response to a given traffic pattern. They forward a header flit in two cycles when the prediction hits. Otherwise, they use at least four cycles.

5.2.2 Prediction Hit Rate

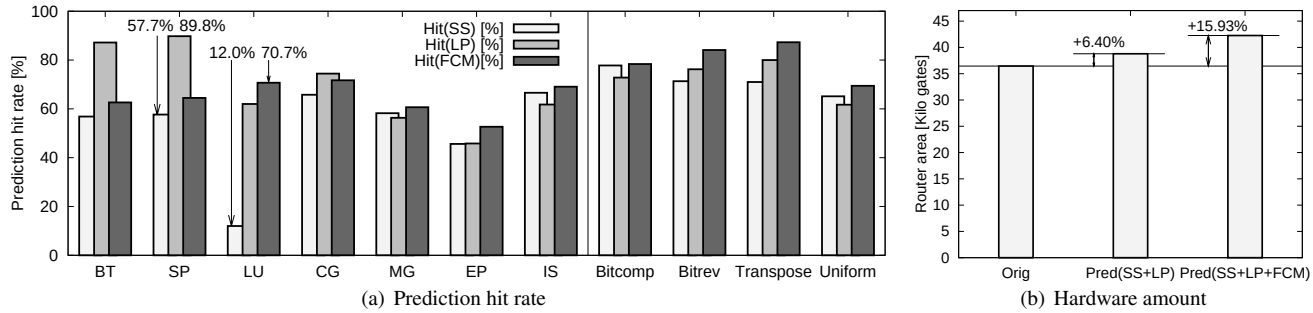
Figure 13(a) shows the prediction hit rates of *SS*, *LP*, and *FCM* algorithms on seven NPB programs (BT, SP, LU, CG, MG, EP, and IS) and four synthetic patterns (bitcomp, bitrev, transpose, and uniform [7]). The prediction hit rates for the synthetic traffics are better than those for the application traffics, since the destination distributions of synthetic ones are simpler than those of real applications.

Although the *SS* predictor achieves more than 80% of hit rates in large networks (e.g., the 256-core mesh in case study 1), its prediction hit rates are not so high in this 64-core mesh. Especially, its hit rate is extremely low (12.0%) in LU traffic that contains a lot of 1-hop communications the *SS* never hits. In most applications, the *LP* and *FCM* predictors provide higher hit rates than the *SS* does. Particularly, *LP* achieves 89.8% of hit rate in SP traffic that contains a lot of repeated short-distance communications.

Each prediction algorithm has strengths and weaknesses. Our prediction routers that support multiple algorithms would be able to accelerate wider range of applications.

5.2.3 Hardware Amount

The original router (Orig), the prediction router with *SS* and *LP* (Pred(SS+LP)), and the prediction router with *SS*, *LP*, and *FCM* (Pred(SS+LP+FCM)) were synthesized with the 65nm CMOS standard cell library in the same way as in



(a) Prediction hit rate

(b) Hardware amount

Figure 13. Evaluation results of case study 2

case study 1. As shown in Figure 13(b), the area overheads of Pred(SS+LP) and Pred(SS+LP+FCM) are 6.4% and 15.9%, respectively. The latter one needs more gate counts, since it supports the *FCM* predictor, which requires a history table (e.g., 4-bit binary counter) in order to record the number of references to each output channel. Though, it gives high and stable hit rates in most traffic patterns, as shown in Figure 13(a).

5.2.4 Energy Consumption

Here we estimate the flit switching energy E_{sw} [pJ/bit] of Orig and Pred(SS+LP+FCM) routers. As shown in Section 5.1.7, the energy overhead of prediction routers is depending on their prediction hit rate. In this case study, the maximum hit rate on each application is higher than 70% except MG and EP traffics (Figure 13(a)). Assuming that the 70% of the predictions hit, the expected energy overhead of the prediction router is estimated as 9.5%.

Throughout this case study, we showed that the hit rates of the three prediction algorithms range from 12.0% to 89.8% on the seven NPB programs. This means that the prediction routers that support multiple algorithms can accelerate wider range of applications, though their area overheads range from 6.4% to 15.9%, depending on the prediction algorithms supported. We have illustrated these tradeoffs through this case study.

5.3 Case Study 3: 256-Core Adaptive Tree Network

The third case study assumes a fine-grained operand network that connects 256 ALUs as in case study 1, but its connection topology is a fat tree, which uses the adaptive up*/down* routing.

5.3.1 Router Architecture

Original Router To connect 256 ALUs, we use fat tree (4,4,4), in which each router except rank-4 routers has four physical channels for upper and lower connections, respectively. For packets moving toward the root of the tree (i.e., upper transfers), each router adaptively forwards an incoming packet to one of four upper channels, while it selects a single lower channel in the case of lower packet transfers. The router has a 4-flit input buffer for each input channel, and its pipeline structure is the same as in case study 1.

Prediction Router Here we use the modified version of *SS* algorithm called *LRU* for lower channels. The *LRU*-based predictor speculatively forwards an incoming packet to the least recently used upper channel, in order to distribute the congestion over multiple upper channels. In the case of upper channels, on the other hand, it is difficult to predict a single correct output channel among four candidates. We implemented two prediction routers. Pred(LRU) employs *LRU* for lower channels but no predictor for upper ones, while Pred(LRU+LP) employs *LRU* and *LP* for lower and upper channels respectively. Pred(LRU+LP) can reduce more latency by the aggressive prediction, although it incurs more energy overhead due to more dead flits.

5.3.2 Prediction Hit Rate / Zero Load Latency

By using the cycle-accurate network simulator, we estimated the prediction hit rates of the two prediction routers on 16-core, 64-core, and 256-core fat trees with uniform random traffic. Then the same tendency can be seen in both the simulation results and the analysis results (Figure 11).

Figure 14(a) shows the zero load latencies of the original router (Orig), Pred(LRU), Pred(LRU+LP), and that with perfect prediction (Ideal). The results show that Pred(LRU+LP) reduces 30.7% zero load latency compared with the original one, in the case of the 256-core fat tree.

5.3.3 Hardware Amount / Energy Consumption

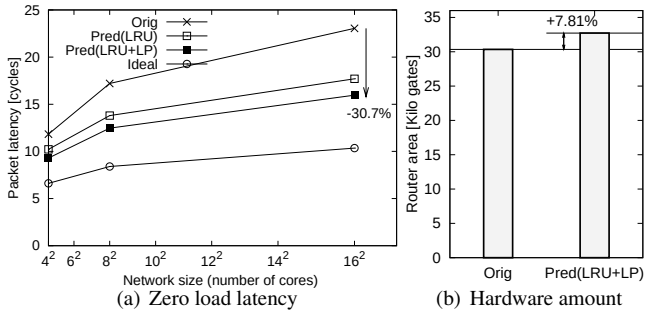
The original router (Orig) and Pred(LRU+LP) were synthesized with the 65nm standard cell library in the same way as in case study 1. Figure 14(b) shows that the area overhead of the prediction router is only 7.8%.

As for the energy consumption, we estimated the flit switching energy E_{sw} of Pred(LRU+LP) router, as in Section 5.1.7. Assuming that the 55% of the predictions hit, the expected energy overhead of the prediction router is 9.0%.

Throughout this case study, we confirmed that our prediction router architecture can be applied to the adaptive up*/down* routing on fat trees, in addition to the deterministic routing on meshes and tori.

6 Conclusions

We proposed the prediction router architecture, which predictively forwards packets without waiting the RC, VA,



(a) Zero load latency (b) Hardware amount

Figure 14. Evaluation results of case study 3

and SA operations, based on the prediction algorithm selected from several candidates in response to the network environments, such as the network topology, routing algorithm, and traffic pattern. Then, we presented the formulas that estimate the hit rates of five prediction algorithms on meshes, tori, and fat trees.

In the three case studies, we implemented prediction routers by using a 65nm CMOS process, and evaluated them in terms of the prediction hit rate, zero load latency, hardware amount, and energy consumption. Case study 1 assumed a 256-core fine-grained mesh network. We confirmed the accuracy of the formulas by comparing the prediction hit rates derived from the formulas with the simulation results. The evaluation results of the *SS*-based prediction router showed that although the area and energy are increased by 10.1% and 8.0% respectively, the communication latency is reduced by 48.2% thanks to its high hit rate. Thus, the prediction router presents favorable trade-offs between these modest overheads and the latency saving. Case study 2 assumed a 64-core chip multiprocessor network. Multiple prediction algorithms were evaluated on the network with seven NPB programs. Their hit rates range from 12.0% to 89.8% depending on the traffic pattern. Thus, the prediction routers that support multiple algorithms can accelerate wider range of applications. Though, their area overheads range from 6.4% to 15.9%, depending on the prediction algorithms each router supports. In case study 3, we also applied the prediction router to fat tree networks and showed its feasibility. Throughout these case studies, we showed that the prediction router architecture is versatile and applicable to various network environments.

Possible improvement of this study is to investigate the adaptive predictor-selection mechanism that dynamically optimizes the predictor in response to the current traffic pattern. We are designing some adaptive selection policies and will evaluate them as future work.

References

[1] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. *NAS Technical Reports NAS-95-020*, Dec. 1995.

[2] L. Benini and G. D. Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 35(1):70–78, Jan. 2002.

[3] L. Benini and G. D. Micheli. *Networks on Chips: Technology And Tools*. Morgan Kaufmann, 2006.

[4] M. Burtscher and B. G. Zorn. Hybrid Load-Value Predictors. *IEEE Transactions on Computers*, 51(7):759–774, 2002.

[5] ClearSpeed Technology. CSX700 Processor. `http://www.clearspeed.com/products/csx700/`.

[6] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. *Proceedings of the Design Automation Conference (DAC'01)*, pages 684–689, June 2001.

[7] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[8] C. Izu, R. Beivide, and C. Jesshope. Mad-Postman: A Look-Ahead Message Propagation Method for Static Bidimensional Meshes. *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing (PDP'94)*, pages 117–124, Jan. 1994.

[9] P. Jacquet, W. Szpankowski, and I. Apostol. A Universal Predictor Based on Pattern Matching. *IEEE Transactions on Information Theory*, 48(6):1462–1472, 2002.

[10] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, N. Vijaykrishnan, M. Yousif, and C. Das. A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures. *Proceedings of the International Symposium on Computer Architecture (ISCA'07)*, pages 138–149, 2007.

[11] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. S. Yousif, and C. R. Das. A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks. *Proceedings of the International Symposium on Computer Architecture (ISCA'06)*, pages 14–15, June 2006.

[12] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston. A Lightweight Fault-tolerant Mechanism for Network-on-Chip. *Proceedings of the International Symposium on Networks-on-Chip (NOCS'08)* , pages 13–22, Apr. 2008.

[13] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha. Express Virtual Channels: Towards the Ideal Interconnection Fabric. *Proceedings of the International Symposium on Computer Architecture (ISCA'07)*, pages 150–161, June 2007.

[14] G. Michelogiannakis, D. N. Pnevmatikatos, and M. Katevenis. Approaching Ideal NoC Latency with Pre-Configured Routes. *Proceedings of the International Symposium on Networks-on-Chip (NOCS'07)*, pages 153–162, May 2007.

[15] R. Mullins, A. West, and S. Moore. Low-Latency Virtual-Channel Routers for On-Chip Networks. *Proceedings of the International Symposium on Computer Architecture (ISCA'04)*, pages 188–197, June 2004.

[16] R. Mullins, A. West, and S. Moore. The Design and Implementation of a Low-Latency On-Chip Network. *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'06)*, pages 164–169, Jan. 2006.

[17] D. Park, R. Das, C. Nicopoulos, J. Kim, N. Vijaykrishnan, R. Iyer, and C. Das. Design of a Dynamic Priority-Based Fast Path Architecture for On-Chip Interconnects. *Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI'07)*, pages 15–20, Aug. 2007.

[18] L.-S. Peh and W. J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'01)*, pages 255–266, Jan. 2001.

[19] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. *Proceedings of the International Solid-State Circuits Conference (ISSCC'07)* , Feb. 2007.

[20] H. Wang, L.-S. Peh, and S. Malik. A Technology-Aware and Energy-Oriented Topology Exploration for On-Chip Networks. *Proceedings of the Design, Automation and Test in Europe Conference (DATE'05)*, pages 1238–1243, Mar. 2005.

[21] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, John F. Brown III, and A. Agarwal. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro*, 27(5):15–31, Sept. 2007.

[22] T. Yoshinaga, S. Kamakura, and M. Koibuchi. Predictive Switching in 2D Torus Routers. *Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'06)*, pages 65–72, 2006.

[23] T. Yoshinaga, H. Murakami, and M. Koibuchi. Impact of Predictive Switching in 2-D Torus Networks. *Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'07)*, pages 11–19, Dec. 2007.