Application-Driven End-to-End Traffic Predictions for Low Power NoC Design

Yoshi Shih-Chieh Huang, Kaven Chun-Kai Chou, and Chung-Ta King
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
Email: {yoshi, kavenc, king}@cs.nthu.edu.tw

***Abstract*—As chip multiprocessors keep increasing the number of cores on the chip, the network-on-chip (NoC) technology is becoming essential for interconnecting the cores. While NoCs result in noticeable performance boost over conventional bus systems, they consume a non-negligible fraction of the system power. One promising solution is to dynamically adjust the working frequencies/voltages of the switches as well as the links between switches in the NoC to match the traffic flows. The question is when to adjust and by how much. Most previous works take a passive approach by reacting to fluctuations in local traffic flows. Unfortunately, this approach may be too slow and too conservative in adjusting the working frequencies/voltages. Since applications often exhibit periodic behaviors, we propose a hardware mechanism to proactively adjust the frequencies/voltages of switches and/or links in NoC by predicting the application runtime traffic. The evaluations show that our design achieves 86% dynamic power savings of the links in the on-chip network, and the resulting overheads from mispredictions are tolerable.**

***Index Terms*—Network-on-chip, low-power design, many-core, end-to-end traffic prediction, power management, DVFS.**

I. Introduction

As chip multiprocessors keep increasing the number of cores on a chip, the network-on-chip (NoC) technology is becoming essential to interconnect the cores. Examples include Tilera's TILE64, MIT's Raw, and UT Austin's TRIPS [1], [2], [3]. As NoC occupies a substantial portion of the chip, the power consumption and heat induced by NoC become non-negligible. For example, in a previous study [2], the NoC of the MIT Raw consumes 36% of the total chip power. The same is true for the NoC of the Alpha 21364 processor [4], which consumes about 18.4% of the total chip power. In an extreme case [5], the power consumption of NoC may be more than 50% of the total chip power.

To solve the power consumption problem in NoC, previous works have proposed techniques such as *dynamic voltage/frequency scaling* (DVFS) to adjust the power mode of the switches and links to match the traffic flows. The challenge is to predict the traffic flowing through the switches and links in the next time interval. For example, if a switch or a link can be predicted to be idle for a while, then it can be set to a low power mode [6], [7] by lowering its voltage or frequency. Unfortunately, most previous works predict the traffic flows by observing only the changes in the local states (recent activities) of the switch or link, such as the fullness of the packet buffers [8]. It may be too slow and too conservative in adjusting the working frequencies/voltages, particularly for sudden changes in traffic patterns.

What is required is a more aggressive and intelligent prediction mechanism. The key is to trace back to the source of the traffic in the NoC, i.e., the application. It is intuitive, and confirmed by many studies, that applications often run in phases and exhibit repetitive behaviors, including communications between parallel tasks or threads in the application [9], [10], [11]. If the communication patterns can be captured and used as clues for predicting the traffic through individual switches in the NoC, then it is possible to adjust the frequencies/voltages of the switches and links accordingly to minimize waste of power.

According to this observation, in this paper we propose a novel application-driven approach for predicting traffic in NoC and performing DVFS on communication links. We consider message-passing many-core architectures, in which cores communicate with each other directly through explicit message passing. The basic idea is to capture the communication patterns between parallel tasks, i.e., the end-to-end traffic, by using a small table in the network interface (NI) of each core to record the outgoing messages from that core. The novel data structure is called the *Application-driven Traffic Pattern Table* (ATPT). With the support of ATPTs, the amount of data injected into the NOC from each core can be predicted.

Once the predictions are made, the utilization of each individual link can also be derived. The voltage/frequency (VF) level of the link can thus be adjusted proactively based on the predicted link utilization. In comparison to previous studies that make the DVFS decision based on the hardware status, our approach uses the data transmission behavior of the application as the VF scaling reference. The data transmission behavior is a better guide, because it is more predictable and the repetitive behavior exists in the execution phase.

To summarize, this paper makes the following specific contributions:

- A table-based traffic predictor, ATPT, is proposed, which can capture application traffic pattern at runtime.
- An NoC power management mechanism based on ATPT is proposed that can reduce the waste of NoC power while keeping the packet latency small.
- Three different strategies on DVFS in communication links are proposed that satisfy different optimization goals.

This paper is organized as follows: In Section II, a motivating example is introduced to show the repetitive commu-

nications behaviors in parallel applications. Section III gives a formal definition of our problem. In Section IV, our traffic prediction design is presented, followed by the mechanism for adjusting the working frequency of the communication links. Implementation considerations of our design are also discussed. In Section VII, we evaluate the accuracy of our predictor and the effects of different design parameters. The efficiency of the resultant frequency scaling is also studied. Related research works are discussed in Section VIII, followed by the conclusion.

II. A Motivating Example

To motivate our discussions, let us consider the LU decomposition from the SPLASH-2 benchmarks suite. The program involves the decomposition of a given dense matrix to the product of a lower and an upper triangular matrices. The parallel program arranges the executing cores into a two-dimensional grid, with each core handling one submatrix. Data will be passed between cores along rows and columns of the grid, and the data transmission behavior appears to have repetitive patterns.

For this study, we use Tilera's TILE64 [1], a message-passing many-core system. The LU decomposition kernel was ported to TILE64 and ran on a 4×4 tile array. The routing algorithm is X-Y dimensional routing. The experiment setup is described in detail in Section VII. In the following discussion, we use the form of $(source \rightarrow destination)$ to describe the transmission pairs. Figure 1 shows the data traffic trace of the east port of router 4. The first diagram shows all the traffic arriving at the east port of router 4. The other three diagrams show the decomposed traffic. Note that the traffic relayed by router 4 is omitted.

We can see from the first diagram that the input data traffic of the east port of router 4 is actually a mix of all traffic destined for router 4. The mixed traffic is irregular and difficult to predict. If we use the traffic predictors proposed in previous studies, which check only the hardware status, it is impossible to make sensible predictions out of such irregular traffic patterns. However, if we examine the end-to-end data traffic of the pairs: $(5 \rightarrow 4)$, $(6 \rightarrow 4)$ and $(7 \rightarrow 4)$, as shown in the remaining diagrams, they are more regular and predictable. Each end-to-end data transmission above is issued by one of the parallel threads of the running application, and it exhibits some repetitive patterns in the observed time intervals.

By utilizing the repetitive characteristic of the end-to-end data traffic, we can predict the data transmission accurately using the recorded history. Once the traffic of all the transmission pairs can be predicted, the total workload of any link in the next time interval can be predicted by summing all the predicted end-to-end data transmissions that pass through the link.

III. Problem Formulation

To simplify the discussion, we focus on message-passing many-core architecture in this paper. The chip-multiprocessor consists of $M \times M$ tiles arranged in a 2D $M \times M$ mesh such as that in Tilera's TILE64 [1]. Each tile contains a processor core,

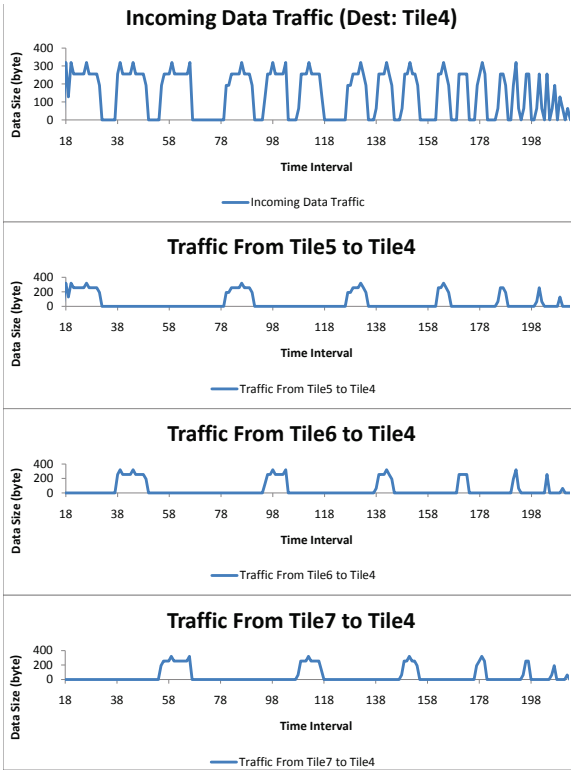

Fig. 1. The data traffic trace of the east port of router 4

a memory module, and a network router. Between each pair of adjacent tiles, there are two uni-directional communication links for sending and receiving data respectively. Each router consists of at most five input ports and five output ports for the north tile, east tile, west tile, south tile, and local processor. This results in a 2D mesh NoC. Note that our solution is not confined to 2D mesh and should be applicable to any topology. The routing algorithm is assumed to be deterministic X-Y routing, which is the most common routing algorithm in the current NoC implementations.

We assume that each tile runs one task. This assumption can be easily relaxed by duplicating the ATPT. Thus, the terms *tile* and *task* will be used interchangeably in the following discussions. Now consider a task $i \in T$ in the system, where T is the set of all tasks. Let $comm_i$ be the set of tasks with which i communicates, denoted as $comm_i = \{j | i \rightarrow j\}$. Let $p_t(j)$ be the function for predicting the traffic from task i to task j at the t-th time interval, where $j \in comm_i$. Note that this prediction function is for the end-to-end traffic between tiles i and j. If the end-to-end traffic can be predicted accurately, the workload of each link in the NoC can then be calculated by *layering* all the end-to-end communication paths together and its utilization can be predicted accordingly.

Given the utilization of a link for the next time interval, the frequency/voltage of the link can be adjusted dynamically to increase its power efficiency. A high link utilization indicates that more data are transmitted through this link in the next time interval. A higher frequency/voltage is thus needed for the link to meet the throughput requirement. On the other hand, a lower utilization means that the link remains idle most of

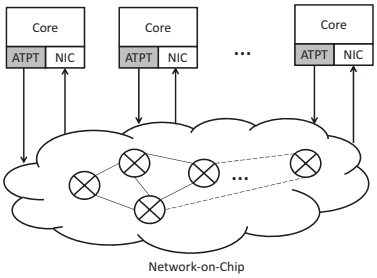

Fig. 2. The high-level design of Application-driven Traffic Pattern Tables (ATPTs), which track the amount of data transmitted in the past and predict the amount of data to be transmitted later.

the time. Consequently, the link frequency and voltage can be decreased to save power without significant performance degradation.

Let E be the set of all the links in the NoC and $e_i \in E$ denote the i-th link. The utilization of e_i at the *t*-th time interval is defined as: $Util_i(t) = \frac{D_i(t)}{\delta \times W}, 0 \leq Util_i \leq 1$, where $D_i(t)$ denotes the total amount of data transmitted by e_i in the *t*-th time interval, δ is the period of time intervals, and W is the maximum bandwidth of a communication link. Thus $\delta \times W$ denotes the maximum possible size of data transmitted through a link in one time interval.

We assume that the communication links support DVFS and can function at the maximum frequency of f_{link}. The links support N discrete frequency levels [9]. At the change of the frequency, the voltage is also correspondingly scaled to the lowest possible value. We use VF levels to describe the possible power modes for the communication links.

From the above utilization function, we can see that if the utilization of link e_i at the *t*-th time interval can be predicted correctly, then the most power-efficient frequency for that link at the *t*-th time interval is $OL_i(t) = Util_i(t) \times f_{link}$. However, since the link can only take N discrete VF levels, only the VF level that has the minimum difference with $OL_i(t)$ can be chosen. Let $CL_i(t)$ denote that VF level. The goal of our dynamic power management is to minimize the difference between $OL_i(t)$ and $CL_i(t)$ throughout the application execution.

Assume that the total execution time of the application takes H time intervals. The goal of our work is to minimize $\Delta_i(t)$, where: $\Delta_i(t) = |OL_i(t) - CL_i(t)|, \forall e_i \in E, \forall t \in H$

IV. System Design of ATPT

The ATPT is a two-level table for predicting the end-to-end traffic between the local tile with other tiles in the next time interval [11]. The design is inspired by the branch prediction. Figure 2 shows the high level design. An ATPT sits besides each network interface card (NIC) for monitoring the amount of data transmitted out of the NIC in a time interval.

A. Basic Ideas

An ATPT supports two predictors, *last value predictor* (LVP) and *pattern-oriented predictor* (POP), which are designed for different purposes. Consider a tile i in the system. Suppose that it has communications with another tile j, i.e. $i \in comm_i$. The LVP predicts the amount of data to be transmitted from tile i to tile j in the next time interval based on the amount of data transmitted from i to j in the last interval. The latter can be recorded easily with a counter $counter_{i,j}$. The LVP is usually accurate if the application has a continuous behavior. On the other hand, the POP makes predictions for the communication with another tile j based on the communication patterns between tiles i and j in the past. Different communication patterns result in different predictions. To unify the two predictors, ATPT uses a selector to decide which predictor to use at runtime.

Next, we describe how the communication patterns between tiles i and j are tracked in POP. The idea is to track and record the amount of data transmitted from i to j in the past l time intervals. In other words, for each possible destination tile j, we keep a vector of l elements, each recording the amount of data transmitted to j in one of the last l time intervals. Now, the amount of data may be large and it is difficult to make a pattern out of the data sizes. Our idea is to quantize the amounts according to the number of discrete VF levels, N. For example, suppose that the maximum bandwidth of each link is *W* bits/s, we use a quantizer $G = W/5$ bits/s to discretize the communication amounts, and we track $l = 5$ time intervals. Thus, given the vector $(5, 3, 1, 2, 4)$, the value 5 in the first element means that the amount of data transmitted from tile i to j in five time intervals ago is between W and $4W/5$ bits/s. Similarly, the value 4 in the fifth element indicates that the amount of data transmitted in the last interval is between $4W/5$ and $3W/5$ bits/s. We say that $(5, 3, 1, 2, 4)$ is the communication pattern from tile *i* to *j*, denoted $history_j$ for tile i.

Given the communication pattern $history_j$, it can be used to index a table $table_j$ in tile i, in which each entry gives a prediction value of the amount of data to be transmitted in the next time interval. Actually, the value recorded in each entry is the amount of data transmitted from tile i to j when the corresponding communication pattern was encountered the last time. Then, if the same communication pattern appears again, the recorded value is used as the prediction value.

1) Design of Prediction Tables: As discussed above, ATPT supports two predictors, LVP and POP. LVP only needs a counter for each possible destination tile j to record the amount of data transmitted from the local tile to that tile in the last interval. On the other hand, POP needs to track the communication pattern from tile i to j during the last l time intervals. Combining the two, each entry in the ATPT thus has a counter for recording the total size of data transmitted in the current time interval and a set of l counters for tracking the communication pattern. The communication pattern is used to index a pattern history table. The pattern history table stores the prediction for each corresponding communication pattern.

This essentially creates a two-level hierarchical table for tracking and predicting the data transmissions. The first-level table (L1-table) tracks all data transmissions injected by the local processor, as shown in Figure 3. When the local processor sends a message, the destination and size of the message are recorded in the Destination and Data Size columns in the L1-table, respectively. At the end of the time interval, the total transmitted size is quantized according to the possible link

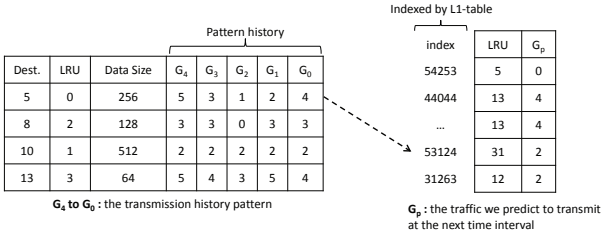

Fig. 3. An example of an ATPT-based predictor. The columns G_4 to G_0 record the quantized size of transmitted data of the last 5 time intervals. L2-table is indexed by the communication pattern $(G_4, G_3, G_2, G_1, G_0)$. The value stored in the corresponding entry G_p is the predicted amount of data to be transmitted in the next time interval.

frequency levels, and is then shifted G_0. The original value in G_0 is shifted into G_1 and so on. The data size counter is then reset. Since a tile often communicates with a limited number of tiles in one time interval, the number of rows for the L1-table can be reduced using an LRU mechanism.

The communication pattern recorded in the L1-table is next used to index the second-level table (L2-table) to obtain the predicted amount of data to be transmitted in the next time interval. As Figure 3 shows, the L2-table stores the communication pattern $(G_5 : G_0)$ and the predicted value G_p. Let Q_d be the number of levels to quantize the data size. The maximum number of entries in the L2-table is thus Q_{d^l}. Again, the size of the L2-table can be reduced by using the LRU mechanism since the transmission behavior for a tile can usually be covered by a limited number of patterns.

The two tables are updated at the end of each time interval. The recorded data sizes in the L1-table are used to check the correctness of the prediction made at the last time interval. If the prediction was wrong, the value of G_p at the L2-table for the corresponding communication pattern will be updated to the data size recorded in the L1-table. If the communication pattern cannot be found in the L2-table, the system will either create a new entry or replace the existing entry by LRU in the L2-table, and use the last value (G_0) as the predicted data size to be transmitted.

B. Design of Selector for Predictors

ATPT supports two predictors. When predicting the traffic from tile i to $j \in comm_i$, the predicting function is defined as:

$$p_t(j) = \begin{cases} c_{t-1}(j), & s(j) = 0; \\ table_j(history_j), & s(j) = 1. \end{cases}$$

where $p_t(j)$ is the prediction for the traffic from tile *i* to tile *j*, and $c_t(j)$ be the actual amount of communication from *i* to *j*. The prediction either comes from LVP or POP, determined by a selector function *s*. The selector function can be different according to the system requirements. In this paper, we use a 2-bit saturating counter.

As Figure 4 shows, the 2-bit saturating counter changes its state according to the prediction results. While the LVP suffers from a high prediction error rate, the 2-bit saturating counter gradually changes its state and finally switches to the POP.

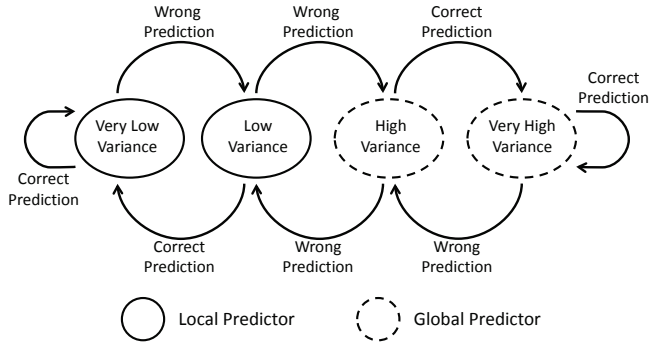

Fig. 4. State diagram of a 2-bit saturating counter

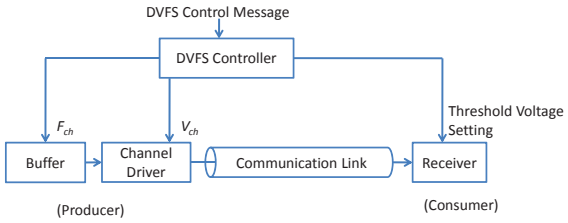

Fig. 5. A DVFS-enabled on-chip interconnection link

V. Link-DVFS by ATPT-Based Predictor

In this section, we discuss how to adjust the frequency/voltage of a link if the utilization of the link for the next time interval can be predicted. To support link-level DVFS, we use the on-chip DVFS link model proposed by Worm *et al.* [12]. As Figure 5 shows, a DVFS controller is added to the original link architecture. The DVFS controller receives the DVFS control message from the producer router. At the producer end, the DVFS controller scales the voltage used to drive the communication channel according to the DVFS control message. The frequency of the data buffer is also changed to fit the driving voltage. At the consumer end, the threshold voltage of the receiver logic is adjusted in order to correctly convert the voltage value to a digital signal.

Note that more adjustable levels of voltages incur more costs, since on-chip regulators occupy a considerable area [6], [7], [13]. However, according to the power consumption formula: $p \propto v^2 f$, adjusting frequencies can still reduce substantial power consumption. Without loss of generality, in the following discussions, we assume that there are five adjustable levels of voltages for each link.

A. Superposing Link Utilization

As described in Section III, the utilization of a communication link e_i in the network at the *t*-th time interval is denoted as $Util_i(t)$. Let $PredictedUtil_i(t)$ be the predicted utilization of e_i at the *t*-th time interval. The goal of the network traffic prediction is to minimize the value of $\Delta_i(t)$ for all $e_i \in E$ and all $t \in H$, while $\Delta_i(t)$ is defined at the end of Section III.

In the NoC that implements deterministic routing algorithms, the data transmission path between any pair of source and destination routers can be determined before the data transmission starts. Thus, for any communication link e_i in the network, the total size of data that is predicted to traverse

through the link can be calculated by summing the data sizes of all the transmissions passing through e_i at the *t*-th time interval. Let $d_i(t)$ denote that sum. Therefore, the predicted utilization of e_i is: $\textit{PredictedUtil}_i(t) = \frac{d_i(t)}{\delta \times W}$

B. Strategies for Link-DVFS

We have discussed in Section IV how ATPT predicts the communication traffic injected from a tile into the NoC. By exchanging the predictions between tiles, the network traffic at the next time interval can be calculated. Consequently, the utilization of each link in the network can be predicted. Using the predicted link utilization, we propose three different link DVFS strategies, *Direct-Set* DVFS (DS-DVFS), *Latency-Aware* DVFS (LA-DVFS), and *Power-Aware* DVFS (PA-DVFS).

In DS-DVFS, at the end of each time interval, the VF level of a link is set according to the predicted size of data to be transmitted. For any link e_i in the network, the predicted traffic of all the end-to-end transmissions passing through this link are summed, which gives the parameter $d_i(t)$. The VF level is set to the minimum level that can transmit $d_i(t)$ amount of data through the link within one interval. However, if $d_i(t)$ is larger than $\delta \times W$, it means that the data needs more than one interval to transmit. Thus the VF level of this link is set to the highest level to maximize the throughput.

LA-DVFS takes into account the latency caused by link DVFS. If the predicted utilization is smaller than current one, LA-DVFS only lowers one VF level in the next time interval. For other cases, it sets the VF level directly according to the utilization prediction. This strategy avoids slowing down links too sharply. Instead, it steps down the link speed gradually for stability.

PA-DVFS emphasizes power consumption more. If the predicted utilization is higher than the current one, PA-DVFS raises only one VF level in the next interval. For other cases, it sets the VF level directly according to the utilization prediction. This strategy applies inertia to raise the VF level but allows quick drop of the link speed for aggressive power saving.

In Section VII-B4, the three DVFS strategies are evaluated for their performances.

C. Discussions

Since traffic predictions and DVFS adjustments are both based on a single time interval, the VF level of a link, e_i, should be set to complete the transmissions of all the predicted workload from all the end-to-end transmissions passing through e_i in one interval. The problem is that DVFS and traffic predictions affect each other. In the following, we examine the effects. Consider a link e_i in the NoC. Let us examine all the links that may feed data to e_i. There are two cases:

1) The utilizations of all these links are correctly predicted. The VF level of e_i can then be set accordingly, which should satisfy the required bandwidth. Thus, no congestion will be created.
2) The utilizations are predicted wrongly (either over- or under-estimated) in some of these links. The ATPT predictors will detect the errors and refine their predictions. In addition, ATPTs use hybrid predictors. When the POP predictor fails to find patterns, ATPT will switch between POP and LVP to better match the actual traffic. One may argue that the correcting process may be too slow and delayed data may be accumulated to cause congestion. This is actually a matter of how aggressively we adjust the VF level on mispredictions. In Section VII-B4, we will examine the effects of different strategies to DVFS adjustment.

VI. Implementation Considerations

In this section, we examine various considerations in implementing the proposed ATPT predictors. Due to the space limitation, parts of the implementation details are moved to our technical report [14].

A. Area Occupancy

We next analyze the area overhead of the ATPT in terms of the number of transistors in chip. Assuming that to store a bit needs 6 transistors, we apply all the space-reducing techniques discussed in [14] except the LRU replacement mechanism in the second-level table. When the number of cores is 64, the amount of transistors needed to implement ATPT is about 0.1M. When the number of cores is 36, the number reduces to about 0.06M. As a reference, the AsAP developed by UC Davis contains 55M transistors [3], and Tilera's TILE64 has 615M transistors. It follows that the ATPTs occupy 0.11% and 0.017% of space in AsAP and TILE64, respectively. The space overhead is quite small and tolerable. Furthermore, we also model ATPTs with a SRAM implementation (similar to cache implementation) and estimate the area overhead by CACTI [15] under a 32nm process technology. The estimated area is $0.133708 \times 0.056544 mm^2$, about 0.01744% of the area of TILE64. This further confirms our estimation above.

B. Energy Consumption of ATPT-Based Predictors

The energy consumption of ATPTs is important, for otherwise the benefits of link DVFS will be offset. Moreover, the energy overhead of scaling voltage levels should also be considered. We will examine these two factors in this subsection.

1) Energy consumption of ATPTs: We model ATPTs with CACTI [15] to estimate the power consumption. The parameters are the same as those in Section VI-A. The estimated energy consumption per access is 0.00394686 nJ. Assuming that every ATPT is accessed in every cycle, the ratio of energy overhead to energy saved is only 2.01407×10^{-7}. This indicates that the energy saving by ATPTs is more than compensated by its overhead.

2) Energy consumption of DVFS switching: Scaling voltage/frequency in DVFS consumes energy. Although frequent adjustment of the VF level of a link according to its utilization can better match the traffic flows, the transition energy of the

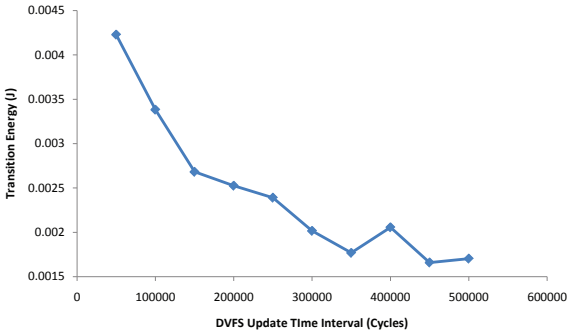

Fig. 6. The transition energy of scaling VF level with different intervals of updating DVFS

scaling can be too high. It is possible to strike a balance between these two factors by adjusting the frequency of updating the VF level. The energy overheads for changing the voltage level from V_1 to V_2 can be calculated with the following equation: $E_{overhead} = (1-\eta) \cdot C \cdot |V_2^2 - V_1^2|$, where C is the capacitance of the power supply regulator and η is the power efficiency. In the following experiments, the capacitance is assumed to be $5\mu F$ and the power efficiency is set to 90%. Details can be found in [14].

Figure 6 shows the energy overhead of scaling the VF level when running the LU decomposition in the SPLASH-2 suite [16]. The transition energy becomes lower as the DVFS update interval is longer. With a longer update interval, the number of times in voltage transition is reduced. However, this also lengthens the time for adjusting the VF level of the links, resulting in a mismatch between the speed of the link and the traffic flowing through it. This in turn wastes energy and even causes traffic congestion. It follows that the design parameters need to be carefully considered while applying the link DVFS techniques. In Section VII, we will evaluate the accuracy of ATPT-based predictor with different DVFS updating intervals. The results show the energy overhead in VF scaling can be neglected.

VII. Evaluation

In this section, we evaluate the accuracy of our proposed ATPT predictions using different settings of time intervals, and compare the predictors LVP and POP. A simple example will be used to demonstrate that the ATPT-based predictors are capable of dynamically learning the communication pattern of an application and recovering from misprediction.

A. Evaluation for ATPT-Based Predictors

We use Tilera's TILE64 as the evaluation platform [1]. TILE64 has a mesh-based NoC with five independent networks to support different functions, namely the static network (STN), the tile dynamic network (TDN), the user dynamic network (UDN), the memory dynamic network (MDN), and the I/O dynamic network (IDN). Application programs can perform tile-to-tile communications through the UDN by calling the `iLib` Library API provided by Tilera. For our study, we only track and predict UDN traffic, because it reflects the communication behavior of the application. The configuration of the TILE64 platform for our evaluation is summarized in Table I.

TABLE I
Configuration of the Evaluation Platform

TILE64 Platform Settings	
Processor family	Tilera Tile64
Frequency	866 MHZ
Number of cores (tiles)	64
Interconnection network	Multiple 8-by-8 meshes
Memory	16KB L1+64KB L2 per core 4 DDR2 memory controllers
Routing algorithm	Dimension-order

To evaluate ATPT, TILE64 is used in a hybrid way as follows. We dedicate half of the tiles in TILE64 to execute the benchmark program directly. Another half of the tiles are used to simulate the operations of the corresponding ATPTs, one tile simulating one ATPA for one tile in the execution plain. Thus, if we are going to evaluate ATPT for a parallel benchmark program running on an n-core multiprocessor, we will use $2n$ tiles of TILE64: n of them execute the program directly and another n tiles simulate the behavior of the corresponding ATPTs. We define *WORKER* as the set containing the tiles that execute the application program and *UPDATER* as the set containing the tiles that simulate the behavior of corresponding ATPTs. The relationship between a tile wt_j in *WORKER* and a tile ut_i in *UPDATER* is: $ut_i \in$ *UPDATER* will simulate the ATPT for $wt_j \in$ *WORKER* if $i = j$.

We use a modified blocked LU decomposition kernel from the SPLASH-2 suite [16] as the benchmark. The original blocked LU decomposition kernel from the SPLASH-2 suite is a data-parallel shared-memory program. We modified the program in two ways. First, in the initialization stage, we determine the allocation of data to the tiles in *WORKER* and then let them cache their own portion of the data. Second, when a *WORKER* tile wt_i needs data that are allocated to another *WORKER* tile wt_j, we use the iLib tile-to-tile message passing API to transmit the data from wt_j to wt_i. In the mean time, a notification message containing information regarding this transmission will be sent to ut_j to simulate the operations in the corresponding ATPT. Note that, since direction execution of application program in the *WORKER* tiles is much faster than simulation of ATPT in the *UPDATER* tiles, the *WORKER* tiles have to be paused until the *UPDATER* tiles can complete their simulation.

We run the modified blocked LU decomposition to solve 50 arrays. Each of them is a 512×512 64-bit floating point array generated randomly at initialization. Each array is separated into 1024 16×16 subarrays and dispatched to 16 tiles for parallel processing.

The time interval in tracking and predicting traffic is a critical factor affecting the prediction accuracy and the applicability of the prediction results. In this experiment, we try to find a *suitable* time interval for our benchmark program. A *suitable* time interval should be able to capture the communication behavior of the application and predict the network traffic with acceptable accuracy. We have tested the time interval settings ranging from 1,000,000,000 cycles to

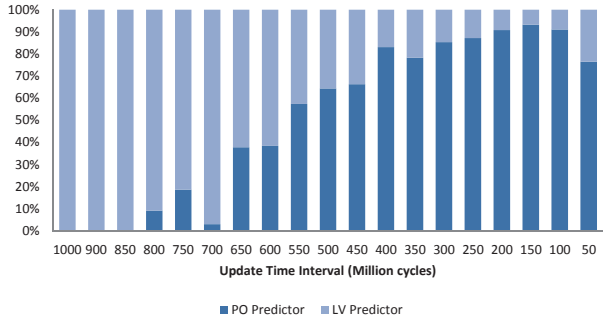

Fig. 7. The ratio of using POP and LVP under different time intervals

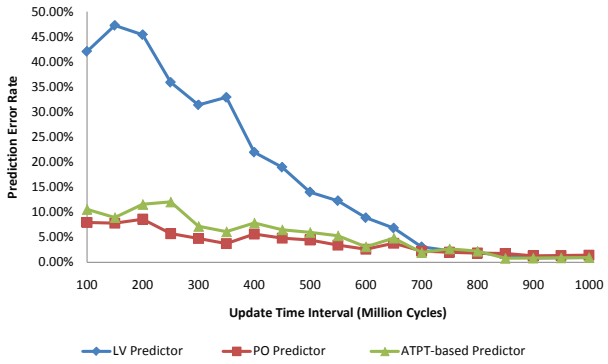

Fig. 8. Prediction error rate of three types of predictors

50,000,000 cycles, as shown in Figure 7.

From the figure we can see that when using larger time interval settings (over 800,000,000 cycles), the prediction error is relatively low, but the POP is rarely selected. Although we can get higher prediction accuracy by using larger time interval settings, the low POP usage means that the ATPT-based predictor does not observe large variations in the communication behavior. This indicates that it misses the dynamism of the NoC traffic and thus the prediction is not applicable. Similar results can be observed if applying relatively small time interval settings (under 50,000,000 cycles).

For our benchmark, with the time interval settings ranging from 100,000,000 cycles to 400,000,000 cycles, the POP is frequently selected. This indicates that ATPT-based predictor observes a varying communication behavior, and the prediction can tell us whether the NoC is busy or not. In summary, the time interval settings ranging from 100,000,000 cycles to 400,000,000 cycles allow ATPT to capture the communication behavior of our benchmark program.

Next, we study the effects of hybrid predictors in ATPT versus pure LVP or POP. Note that ATPT uses both predictors and dynamically selects between the two. The prediction error rates of the three different predictors with different update interval settings are shown in Figure 8. The diamond dotted line shows the performance of using only the local predictor (LVP), which suffers from high error rates under time intervals ranging from 150,000,000 to 650,000,000 cycles. This is because the communication traffic with these time interval settings varies widely as monitored by ATPT. While using larger time intervals, the local predictor has good prediction accuracy since the communication patterns are almost the same, again as shown in each ATPT update.

On the other hand, the POP (diamond dotted line) performs well in all the evaluated time intervals. The POP can work as an LVP when the two entries of **00000000** and **11111111** have been filled, which stand for the communication patterns of *no transmission* and *transmit at all times*. The result is highly accurate predictions in low-variance communication patterns. The POP is also capable by its nature of predicting accurately in high-variance communication patterns.

However, the POP uses more memory space than the LVP. The accuracy of the POP is related to how many patterns can be tracked in the 2nd-level table. Currently, we have not limited the size of the 2nd-level table, and therefore all the patterns can be tracked and used to make predictions. But in a practical implementation, considering the transistor count overhead, the 2nd-level table may be not large enough to hold all the patterns throughout application execution, so only a subset of the patterns could be recorded. Consequently, the accuracy will be dramatically affected by the replacement policy. When a longer pattern history is needed, the number of total tracked patterns (the difference between the POP and the ATPT-based predictor) will be broadened as Figure 9 shows.

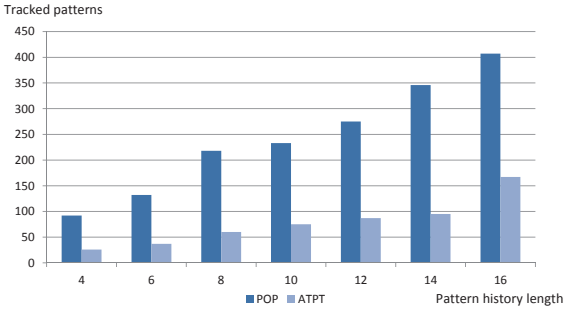

Fig. 9. Total patterns tracked by pure POP and ATPT-based predictor. Running a modified SPLASH-2 blocked LU decomposition benchmark with 16 tiles and 600 million cycles as the update time interval.

To summarize, the ATPT-based predictor (triangle dotted line) adaptively selects an LVP in low-variance traffic, and a POP for high-variance traffic. Moreover, the experiments show that the ATPT uses fewer entries than the POP and is suitable for memory-constrained design.

1) Pattern learning and misprediction recovery: We give a simple example to demonstrate how the POP (as a part of the ATPT-based prediction) learns the communication pattern and recovers from misprediction at runtime. Figure 10 shows two different time periods of the benchmark program execution, where the POP encountered the same communication patterns. The first time, the POP recognized the pattern of **10111101**, and it made an inaccurate prediction (**1** stands for there is a transmission in that time interval, and **0** stands for no transmission). The POP will change the prediction for that communication pattern in the 2nd-level table when a misprediction is detected. Afterwards, when the same pattern appears, the POP can make a correct prediction.

To further validate the accuracy and the efficiency of LRU design, four benchmarks (`Bcast`, `Scatter`, `Gather`,

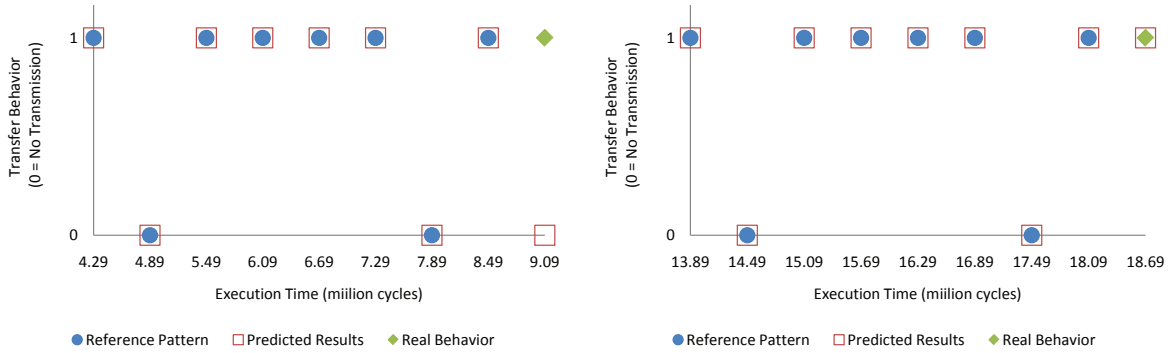

Fig. 10. The communication pattern trace from Tile2 to Tile3 of different time intervals in the modified SPLASH-2 LU decomposition kernel execution. The POP suffers from misprediction for a certain communication pattern and recovered in the next encounter with the same pattern.

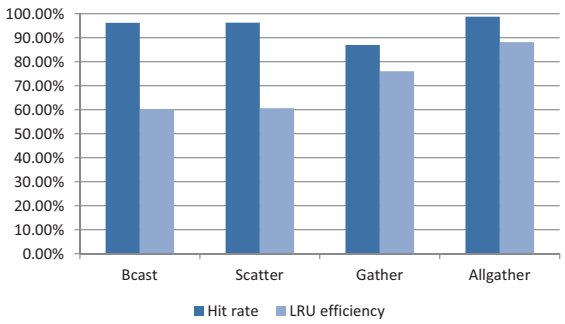

Fig. 11. Prediction results of Intel MPI benchmarks.

TABLE II
DEFAULT SETTING OF THE SYSTEM PARAMETERS FOR THE EXPERIMENTS.

Parameters of Target NoC	
Topology	4×4 2D Mesh
Bandwidth	1 flit/cycle
Flit size	8 bytes
Routing algorithm	Dimension-order
Physical per-link length	2603 μm
Simulation cycle	2×10^9 cycles
Number of entries in L1-table	8
Number of entries in L2-table	128
Traffic predictor update time interval	2500000 cycles
Number of DVFS levels per link	5 levels

`Allgather`) from Intel MPI Benchmarks (IMB) suites [17] are also evaluated, the results also show that ATPT-based predictor has good high performance, and the efficiency of LRU is ranging from 60% to 88%, which shows the existences of repetitive behaviors.

In summary, according to the experiments above, we claim that ATPT integrates the advantages of the LVP and the POP with small chip area overhead.

B. A Case Study for DVFS on Links

In this section, we present a case study to evaluate the efficacy of the application-driven link DVFS method.

1) Simulation Setup: The Tilera's TILE64 platform is used to run the benchmark programs and collect data transmission traces. The data transmission traces are then used as the input of a NoC power simulator to obtain the power consumption of the benchmark. TILE64 is a many-core platform that consists of 8×8 tiles connected by mesh NoC architecture. No hardware virtual channels are implemented. The routing algorithm is X-Y routing. The NoC power simulator originates from the *PoPNet* network simulator. We extend *PoPNet* to make it support dynamically adjusting the frequency and voltage of individual communication links at runtime.

2) Methodology: On the TILE64 platform, we use 4×4 tiles to run the benchmark program and another 4×4 tiles to simulate the ATPT-based predictor of each router, respectively. We refer to the tiles that run the benchmark program as *WORKER* tiles and the ones that simulate the ATPT-based predictor as *UPDATER* tiles.

When one of the *WORKER* tiles issues an outgoing data traffic, the information of this data transmission is recorded by its corresponding *UPDATER* tile. The *UPDATER* tiles periodically simulate the update process of the ATPT-based predictor, and make the data transmission prediction for the next time interval.

After making the data transmission prediction, one of the *UPDATER* tiles is responsible for aggregating all the predicted value and make the DVFS decision for each link. A DVFS message is then inserted into the data transmission trace. The extended *PoPNet* has the ability to read the DVFS messages, and change the frequency levels of links in the network during the simulation. Finally, the extended *PoPNet* returns the power consumption of the benchmark program running on a NoC architecture with application-driven link DVFS.

We have ported the blocked LU decomposition to the TILE64 hardware platform as a case study. The input of the LU decomposition is a 512×512 64-bit floating point array randomly generated at initialization. The array is divided into 1024 16×16 subarrays and distributed evenly to 16 tiles on the TILE64 platform for parallel processing. Default system parameters are provided in Table II.

3) The accuracy of link voltage and frequency level adjustment: We compare the VF level of the communication links set by the proposed method with the best-fit VF value. Figure 12 shows the link VF setting a comparison between the best-fit VF value and the result of DS-DVFS. The black line shows the best-fit VF setting for this communication link at each time interval. The best-fit VF setting is calculated derived from the formulation proposed in Section III using

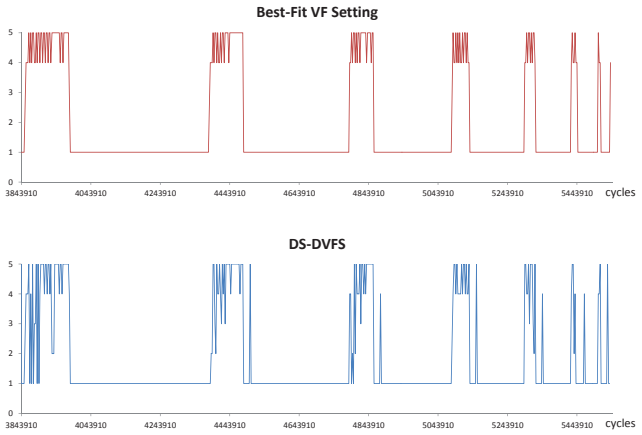

Fig. 12. Best-fit link VF setting and the result of DS-DVFS of the communication link from $tile_4$ to $tile_0$

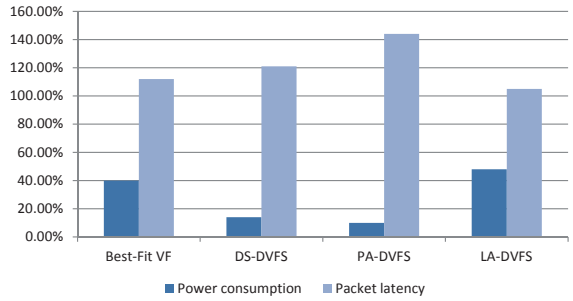

Fig. 13. Performance of power consumption and corresponding packet latency, including our three DVFS strategies and the best-fit VF. (Normalized to full-speed link execution).

the collected data traffic trace after the application execution. The grey line shows the VF level at each time interval set by the DS-DVFS method at application runtime. The DS-DVFS chooses the same VF level with the best-fit VF value most of the time. The average VF level difference between best-fit VF value and DS-DVFS is only 0.28 levels.

4) Communication links power reduction results: In Figure 13, the power consumption and the average packet latency of the best-fit VF setting and the three DVFS methods proposed in Section V-B are provided. The result of best-fit VF setting is generated by a 2-pass simulation. In the first pass, we collect the data traffic trace of the target application, and calculate the VF setting using the formulation described in Section III. In the second pass, the target NoC settings are shown in Table II. We use the calculated VF setting to simulate the power consumption of the target application. Namely, the calculated VF settings is *best-fit* for the next interval.

By using the best-fit VF setting, 60% power is reduced on communication links while causing low latency overheads (12%). On the other hand, DS-DVFS provides greater power reduction but also brings about huge packet latency increases. DS-DVFS switch the VF level of links directly according to the traffic prediction, and thus the performance is closely related to the prediction accuracy. Since the application we use performs a small amount of data transmissions during the execution, our ATPT-based traffic predictor tends to indicate the data traffic size as smaller. Consequently, DS-DVFS usually sets a lower VF level than the best-fit VF value.

Based on DS-DVFS, we provide two different DVFS approaches to make the power reduction mechanism more flexible. PA-DVFS aims to reduce power consumption while ignoring the impact of increasing packet latency. Using PA-DVFS, we receive a 90% power reduction and cause 44% latency overheads. LA-DVFS tries to keep the packet latency low while saving the link power. LA-DVFS gives a 52% power reduction and 5% latency overheads. In general, LA-DVFS has the all-round performance considering power consumption and incurred packet latency.

VIII. Related Works

Traffic Prediction in NoC. In the embedded system area, application graphs are widely used to depict the communication relationship among tasks [18], [19]. The knowledge of the applications can be extracted by off-line analysis and profiling data. However, the actual workloads and the communications among tasks often change over time, oncoming tasks are unknown, so it is not enough to confirm the traffic workload at runtime.

Due to the simplicity and the low area overhead of the simple counters, current research is interested in exploring their usage and provides system adjustments based on clues gleaned from the counters. In [10], counters and tables are used to identify whether running tasks are computation- or memory-bound.

In [8], switch-to-switch exchange information is used to predict the pressure of the traffic and make flow control decisions. They predict traffic from the switch perspective and do not consider the behaviors of the applications. In contrast, our predictions are made from the perspectives of applications. Moreover, in ATPT-based predictor, since the data transmission is tracked before the data is injected into the router buffers, the data size can be recorded and predicted even if it is larger than the router buffer size. In this way, the bursty data transmission can be predicted more accurately than the hardware-based approach.

Table-based predictions are often used in branch prediction and cache prefetching [20]. The studies on cache prefetching concentrate on prefetching data according to temporal and spatial locality while accessing the cache. Our solution concentrates on historical patterns, and records the patterns periodically.

Power Management for NoC. In [4], the power consumption of the interconnection network has been addressed. Servers are not only energy drains; research on the chip-level multiprocessor found that the network-on-chip consumes around 36% of the whole energy of the system [2]. In [5], the architectural power model for an on-chip network has been proposed for estimating and evaluating the design of the network-on-chip. Also, the power model has been ported into a full-system simulator [21]. In [6], [7], frequency tuning for on-chip networks is well addressed.

In comparison to the previous work on this topic, our study is the first one to take advantage of the real application behaviors [22], not only to monitor and predict the NoC, but also to control the link VF by the application behaviors [11].

IX. Conclusions

In this paper, we propose a two-level table design called ATPT for predicting end-to-end traffic on the NoC. The design is introduced and the implementation details are discussed. Based on the traffic behaviors of the running applications, ATPT-based predictors can predict the workload of each communication link in the network, and the suitable working frequency and the corresponding voltages of the links can be set in advance. Experiments show that DS-DVFS achieves an 86% link power consumption reduction and incurs 21% packet latency overheads. Moreover, PA- and LA-DVFS approaches make the power reduction mechanism more flexible.

Acknowledgment

This paper is supported by NSC grants 98-2221-E-007-063- and 100-2219-E-009-022-. We also would like to thank the sponsorship by Industrial Technology Research Institute. Dr. Shau-Yin Tseng provides TILE64 as the platform for testing the correctness of our design.

References

[1] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "Tile64 - processor: A 64-core soc with mesh interconnect," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, Feb. 2008, pp. 88–598.

[2] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: a computational fabric for software circuits and general-purpose programs," vol. 22, no. 2, pp. 25–35, Mar. 2002.

[3] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, D. Truong, T. Mohsenin, and B. Baas, "Asap: An asynchronous array of simple processors," *IEEE Journal of Solid State Circuits*, vol. 43, no. 3, p. 695, 2008.

[4] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The alpha 21364 network architecture," *IEEE micro*, pp. 26–35, 2002.

[5] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Proc. DATE '09. Design, Automation. Test in Europe Conference. Exhibition*, Apr. 20–24, 2009, pp. 423–428.

[6] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "A case for dynamic frequency tuning in on-chip networks," in *Proceedings of the 42nd International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 292–303.

[7] A. K. Mishra, A. Yanamandra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "Raft: A router architecture with frequency tuning for on-chip networks," *J. Parallel Distrib. Comput.*, vol. 71, pp. 625–640, May 2011.

[8] U. Y. Ogras and R. Marculescu, "Prediction-based flow control for network-on-chip traffic," in *Proc. 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 839–844.

[9] G. Chen, F. Li, M. Kandemir, and M. Irwin, "Reducing noc energy consumption through compiler-directed channel voltage scaling," in *Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*. ACM New York, NY, USA, 2006, pp. 193–203.

[10] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 359–370.

[11] Y. S.-C. Huang, K. C.-K. Chou, C.-T. King, and S.-Y. Tseng, "Ntpt: On the end-to-end traffic prediction in the on-chip networks," in *Proceedings of the 47th Design Automation Conference (DAC)*, 2010.

[12] F. Worm, P. Ienne, P. Thiran, and G. De Micheli, "An adaptive low-power transmission scheme for on-chip networks," in *Proceedings of the 15th international symposium on System Synthesis*. ACM, 2002, p. 100.

[13] W. Kim, M. Gupta, G. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*. Ieee, 2008, pp. 123–134.

[14] Y. S.-C. Huang, K. C.-K. Chou, and C.-T. King, "Implementation details of atpt-based predictions for low power noc design," Tech. Rep., 2012.

[15] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2007, pp. 3–14.

[16] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: characterization and methodological considerations," in *ISCA '95: Proceedings of the 22nd annual international symposium on Computer architecture*. New York, NY, USA: ACM, 1995, pp. 24–36.

[17] "Intel mpi benchmarks."

[18] J. Ju and R. Marculescu, "Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures [award]," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, Feb. 16–20, 2004, p. xxix.

[19] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, Feb. 16–20, 2004, pp. 896–901.

[20] K. J. Nesbit and J. E. Smith, "Data cache prefetching using a global history buffer," in *Proc. 10th International Symposium on HPCA-10 High Performance Computer Architecture*, Feb. 14–18, 2004, p. 96.

[21] N. Agarwal, T. Krishna, L. Peh, and N. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *Proceedings of International Symposium on Performance Analysis of Systems and Software*, 2009.

[22] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *International Conference on Parallel Architectures and Compilation Techniques*, 2001, pp. 3–14.

Yoshi Shih-Chieh Huang received the BS from the Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taiwan, ROC, and the MS from the Department of Computer Science, National Tsing Hua University, Taiwan, ROC, where he is pursuing his PhD. His current research interests include system simulation, network-on-chip and design automation.

Kaven Chun-Kai Chou received the BS and MS from the Department of Computer Science, National Tsing Hua University, Taiwan, ROC. He is currently with MediaTek Inc., Hsinchu, Taiwan, ROC. His research interests include multi-core architecture, parallel processing, low-power design and network-on-chip.

Chung-Ta King received the BS in Electrical Engineering from National Taiwan University, Taiwan, ROC, in 1980, and the MS and PhD in Computer Science from Michigan State University, East Lansing, Michigan, in 1985 and 1988, respectively. From 1988 to 1990, he was an Assistant Professor of Computer and Information Science at New Jersey Institute of Technology, New Jersey. In 1990, he joined the Faculty of the Department of Computer Science, National Tsing Hua University, where he is currently a Professor and the director of the department. His research interests include parallel and distributed processing and networked embedded systems. He is a member of the IEEE.