On Self-tuning Networks-on-Chip for Dynamic Network-Flow Dominance Adaptation

Xiaohang Wang
Guangzhou Institute of Advanced Technology, CAS
Email: xh.wang@giat.ac.cn

Terrence Mak
The Chinese University of Hong Kong
Email: stmak@cse.cuhk.edu.hk

Mei Yang and Yingtao Jiang
University of Nevada, Las Vegas
Email: mei.yang@unlv.edu, yingtao@egr.unlv.edu

Masoud Daneshtalab
University of Turku,
Email: masdan@utu.fi

Maurizio Palesi
University of Enna, Kore, Italy
Email: maurizio.palesi@unikore.it

***Abstract*—Modern networks-on-chip (NoC) systems are required to handle complex run-time traffic patterns and unprecedented applications. Data traffics of these applications are difficult to be fully comprehended at design-time so as to optimize the network design. However, it has been discovered that the majority data flows in a network are dominated by less than 10% of the specific pathways. In this paper, we introduce a method that is capable of identifying critical pathways in a network at run-time and, then, can dynamically reconfigure the network to optimize for the network performance subjected to the identified dominated flows. An online learning and analysis scheme is employed to quickly discover the emerged dominated traffic flows and provides a statistical traffic prediction using regression analysis. The architecture of a self-tuning network is also discussed which can be reconfigured by setting up the identified point-to-point paths for the dominance data flows in large traffic volumes. The merits of this new approach are experimentally demonstrated using comprehensive NoC simulators. Compared to the conventional network architectures over a range of realistic applications, the proposed self-tuning network approach can effectively reduce the latency and power consumption by as much as 25% and 24%, respectively. We also evaluated the configuration time and additional hardware cost. This new approach demonstrates the capability of an adaptive NoC to handle more complex and dynamic applications.**

***Keywords*—networks-on-chips, self-tuning, regression, reconfigurable**

I. Introduction

A large number of on-chip many-core systems have been designed for a wide range of applications, including scientific computing, the Internet-based services, the newly emerging applications of recognition, mining, and synthesis (RMS) [1], among many others [2]. One of the key components of an on-chip many-core system is its on-chip network (OCN) or network-on-chip (NoC), which has to provide efficient communication bandwidths for the processor cores and other resources with low latency and low power.

Modern architectural optimization techniques applied to NoCs in many/multi-core systems [3]–[7] assume a general purpose packet-switching fabric where packets are transmitted through complex router pipelines in a hop-by-hop manner. Such scheme, however, incurs high communication latency and power consumption due to the contention for the shared channels, buffering, and long pipeline stages [8] [9]. Some of the existing approaches like the express virtual channel (EVC) [8] and the duo [10] try to bypass part of or full router pipeline stages so as to optimize data transmission.

Another big problem of most of the NoC architectural designs is that they fail to exploit the applications' traffic behavior, although applications running in these systems exhibit stable and predictable traffic behaviors [10]. A couple of noticeable exceptions to this kind of NoC architectural designs is the work in [11], which investigates the self-similarity of traffic and the duo approach in [10], which analyzes the spatiotemporal distribution of traffic flows.

A more subtle alternative to the conventional general purpose packet-switching fabric rests on the on-line learning of the applications' traffic behavior and dynamically change the network configuration. Thus, channel contention, excessive buffering, and complex pipeline stages could be avoided. The potential benefit of having such a self-tuning NoC is attributed to the fact that a small percentage of the flows account for a disproportionally large number of the packets transmitted; this phenomenon is therein referred as *flow dominance* [10].

As an example, let us consider a 64-core system running benchmarks through its NoC with 64 threads[1]. Here a *flow* is defined as the data traffic flowing between a source-destination pair. With the distribution function of the packets sent by the flows, Fig. 1 shows the flow dominance observed in different benchmarks. The Y-axis in Fig. 1 indicates the percentage of the packets injected by the corresponding flow divided by the sum of the total packets. We can see that flow dominance in some applications is more apparent than that in the others. For example, for barnes, the top 20 flows (only 7% of the total flows) inject about half of the total packets. Here we define the *dominant flows* as the ones with the highest data volume. Optimizing the network for the dominant flows will obviously bring in the biggest performance improvement.

The above mentioned flow dominance feature shall be explored to help build *intelligent self-tuning NoC* to optimize both latency and power. In this paper, we propose a design method to help building such application-aware intelligent

[1]The readers can refer to Section III.A for detailed configuration of the experiments.

978-1-4673-6493-5/13/$31.00 ©2013 IEEE

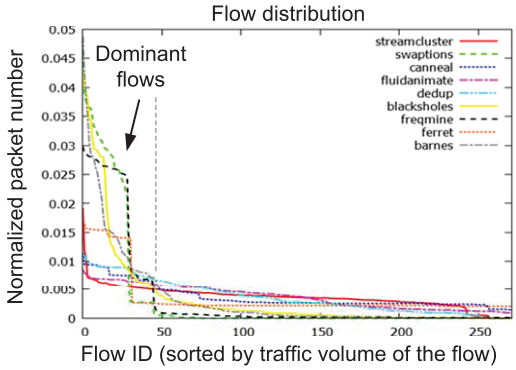

Fig. 1. Dominant flow analysis. Flows are sorted according to their traffic volume in the descending order. The X-axis represents the flow IDs (each flow is associated with a source-destination pair), while the Y-axis represents the ratio of the packet number injected by the corresponding flow over the total number of the packets in the network. This figure is the distribution function of the packets sent by the flows.

NoC (AIN) with online learning capability. AIN works in two phases.

1) The *dominant flow identification* phase, in which identifies the behavior of the flow traffic and exploit the flow dominance with predictable patterns. Data regression models will be used to predict the flows' traffic and dominant flows are also identified in this phase.
2) The *reconfiguration* phase, in which a reconfigurable NoC structure is employed to customize point-to-point paths for the identified dominant flows.

To the best of our knowledge, this work is the first to introduce a framework in building intelligent self-tuning NoC architectures. The rest of the paper is organized as follows. Section II presents overall framework, including the dominant flow identification and the description of the reconfigurable NoC structure. Section III discusses the experimental results. Section IV reviews related work. Section V concludes the paper.

II. Self-tuning NoC Framework

A. Overview of the self-tuning NoC framework

The framework of the self-tuning NoCs operates in two phases. There is a master core performing the path setup. During the dominant flow identification phase, data regression models with their parameters are identified for prediction purpose at each node. In the reconfiguration phase, the flows' traffic can be predicted according to the regression models, after which the dominant flows can be sorted out. Correspondingly, the network can be reconfigured to optimize the dominant flows.

The basic steps performed in each phase include:

- *Phase I: dominant flow identification*
  1) Each source node records the flow's traffic volume at each time interval. At the end of this phase, the regression model for each flow's traffic volume is built. The parameters of the models of each flow are sent to the master core.
  2) At the master core, the regression models of the flows are collected.
- *Phase II: reconfiguration*
  1) The master core[2] predicts the flows' traffic volume based on the parameters of the regression models. After sorting, the dominant flows are found.
  2) Point-to-point connections are set up by reconfiguring the network to optimize the dominant flows. In this phase, only the master core is involved.

Algorithm 1 shows the overall flow of framework in an algorithmic manner. The input is the flow traffic of the NoC running applications, and the output is the point-to-point paths set up for the identified dominant flows.

Algorithm 1: Self-tuning NoC

Input: F: NoC flow traffic of each node (Table I)
Output: Point-to-point paths for the predicted dominant flows.
Function: Identify dominant flows & find point-to-point paths for them.
begin
  `/* Phase I: dominant flow identification (Section II.C) */`
  for *each node* i **do**
    for *each destination* j **do** `/* flow` $f_{<i,j>}$ `*/`
      1) get the parameters of ARMA and polynomial models;
      2) select the model with min error (the best model);
      3) send the parameters of the best model to the master core;
    end
  end
  `/* Phase II: reconfiguration (Section II.D) */`
  sort and find the predict dominant flows DF in the next time interval;
  `HeuristicBasedPathSetup` (DF) ;
  `/* point-to-point path setup (Algorithm 2). */`
end

B. System architecture and definitions

The reconfigurable NoC structure is composed of two parts, (*i*). the basic NoC layer based on an existing (packet-switching) NoC system, *e.g.*, 2D mesh, 2D concentrated mesh, or even 3D NoC topologies, *etc.*, and (*ii*). the reconfigurable layer (Fig. 2 (a)) which offers low latency and low complexity. The reconfigurable layer can be realized with the lightweight configurable switches (CS). Each router in the packet switching NoC has one additional port connecting to the corresponding CS by the through-silicon vias (TSVs). This port only requires data buffers, and no crossbar switching or arbitration logic is needed. The switches in the reconfigurable layer can be configured to provide point-to-point connections/paths for the dominant flows. In a simple word, the dominant flows can

[2]Without losing generality, the master core is assumed to be the top-left core in the mesh like topologies.

TABLE I. NOTATIONS USED IN THE PAPER

$F = \{F_t\}$,where $F_t = \{f_{<1,1>,t}, ...,f_{<K,K>,t}\}$	$f_{<s,d>,t}$ represents the traffic volume of the flow from source node s to destination node d at interval t. K is the network size after concentration. F_t is the set of the flows' traffic volume at interval t. F is the set of the flows' traffic volume over the whole execution time.
DF=$\{df_i\}$ = $\{(s,d)\}$, 0<i<S	The top S flows after ranking (dominant flows). df_i corresponds to the i-th dominant flow.
MP_i	All of flow df_i's minimal paths.
$mp_{i,k} \in MP_i$	The k-th minimal path of flow df_i.
$df_{i,k}$	A binary variable indicating if the k-th minimal path of flow df_i is set up or not.
$PATH_{i,j}$	The set of reconfigurable layer links in the j-th minimal path taken by flow df_i.
K	The network size after concentration.

traverse in the reconfigurable layer as if it provides these flows with dedicated wires. As a result, the router pipelines, excessive buffering, and channel contention, are all avoided for these dominant flows.

As shown in Fig. 2(b), each CS consists of 5 4-to-1 multiplexers (MUXs) for bidirectional transmission. A global line connecting the master core and the CSs controls the configuration of each CS. The master core sends the control signals to the configurable switches via the global control line after performing the path setup algorithm in Phase II. Each control signal has the format of <CS ID, MUX ID, MUX select>. For the concentrated mesh (CMesh) network where a total of 16 configurable switches is used, the global control line thus has a width of $log_2 16 + log_2 5 + 2 = 8$ bits. The power and area can be estimated based on this circuit. The latency of the reconfigurable layer is assumed to be 1 cycle/hop.

Table I lists the notations used in this section. At each source node, the traffic volume of the flows is recorded. Assume the network size is K after concentration. At node i, $f_{<i,j>,t}$ represents the traffic volume of flow from node i to node j at time instance t, where $0 \leq j < K$. The set of the flows' traffic volume at interval t is represented as $F_t = \{f_{<1,1>,t}, ..., f_{<K,K>,t}\}$.

C. Phase I: Dominant flow identification

In this phase, two data regression models, the autoregressive moving averaging (ARMA) model [12] and the polynomial model [13], are considered in predicting the flows' traffic. These two models are popular for their simple structures and high precision in prediction. Algorithm 1 summarizes the major tasks performed by each node and the master core during the two phases as described in the previous section.

1. The autoregressive moving averaging (ARMA) model

Variable $f_{<s,d>,t}$ is used to denote the traffic volume from node s to d at time instance t. In the ARMA model, the objective is to predict $f_{<s,d>,t}$ from a linear combination of its past values (*e.g.*, $f_{<s,d>,t-1}$, $f_{<s,d>,t-2}$, ...).

An ARMA(p,q) model can be written as

$$\hat{f}_{<s,d>,t} = \varphi_1 f_{<s,d>,t-1} + ... + \varphi_p f_{<s,d>,t-p} + \omega_t + \theta_1 \omega_{t-1} + ... + \theta_q \omega_{t-q} \quad (1)$$

where $\{\varphi_1, ..., \varphi_p\}$ and $\{\theta_1, ..., \theta_q\}$ are regression parameters and $\{\omega_t, ..., \omega_{t-q}\}$ is a Gaussian white noise. The predicted value of $f_{<s,d>,t}$ is denoted as $\hat{f}_{<s,d>,t}$. To estimate the parameters $\{\varphi_n, \theta_m | n = 1, ..., p, m = 1, ..., q\}$ with a given (p,q) pair, methods like the Yule-Walker estimator or maximum likelihood estimator can be used, which are detailed in [12].

2. The polynomial regression model

In the polynomial regression, the traffic volume of a flow $f_{<s,d>,t}$ should be modeled by a polynomial function of time t given below [13]

$$\hat{f}_{<s,d>,t}(\mathbf{w}) = \omega_0 + \omega_1 t + \omega_2 t^2 + ... + \omega_M t^M = \sum_{j=1}^{M} \omega_j t^j \quad (2)$$

where M is the order of the polynomial and the coefficients $\omega_0, ..., \omega_M$ are collectively denoted as a vector $\mathbf{w}$. The value of the coefficients can be determined by fitting the polynomial to the training data. This can be done by minimizing an error function that measures the misfit between $\hat{f}_{<s,d>,t}(\mathbf{w})$ and the training set data points. Methods like maximum likelihood could be used to estimate the coefficient vector $\mathbf{w}$ for a given order of M [13]. The regression coefficients (*e.g.*, $\{\varphi_n, \theta_m | n = 1, ..., p, m = 1, ..., q\}$ or $\{\omega_0, ..., \omega_M\}$) are obtained such that they could be used to predict the flows' traffic volume.

D. Phase II: Reconfiguration

After discovering the dominant flows in Phase I, point-to-point paths could be created in the reconfigurable layer to expedite the dominant flows. As this layer is made of simple switches only, they don't have any flow control and routing computation capabilities. Thus, the key issue in path setup in this layer is to avoid path overlap. To facilitate this request, the reconfigurable layer itself could be expanded to increase the path diversity (i.e., to increase the number of minimal paths from the same source to the same destination) with additionally added switches. Expansion of the reconfigurable layer is regulated by a parameter, E. For example, if initially the reconfigurable layer has a size of $N \times N$, then the expanded layer can eventually be as large as $(2N-1) \times (2N-1)$ with $E = 2$.

The path setup problem could be described as follows:

Set up the paths for the dominant flows such that, the paths have the minimum overlap and the total distance of all the involved paths is minimized.

This problem can be formally formulated as follows. Suppose the dominant flows are sorted in DF (see Table I). For a given dominant flow df_i, all of its minimal paths are enclosed in set MP_i with $|MP_i|$ representing the total number of minimal paths of flow df_i. $mp_{i,k} \in MP_i$ represents the k-th minimal path of flow df_i. A binary variable $df_{i,k}$ is used to indicate if the k-th minimal path of flow df_i is set up or not. Let $PATH_{i,j}$ be the set of links on the j-th minimal path taken by flow df_i in the reconfigurable layer. Let V be the maximum number of flows sharing one link. Each configurable switch p has a set of binary variables, $\{C^p_{XY}\}$. C^p_{XY} represents the connection from the input port X to the output port Y, where X and Y correspond to elements in E, W, S, N, and L. In each configurable switch, one port can only be connected to one of

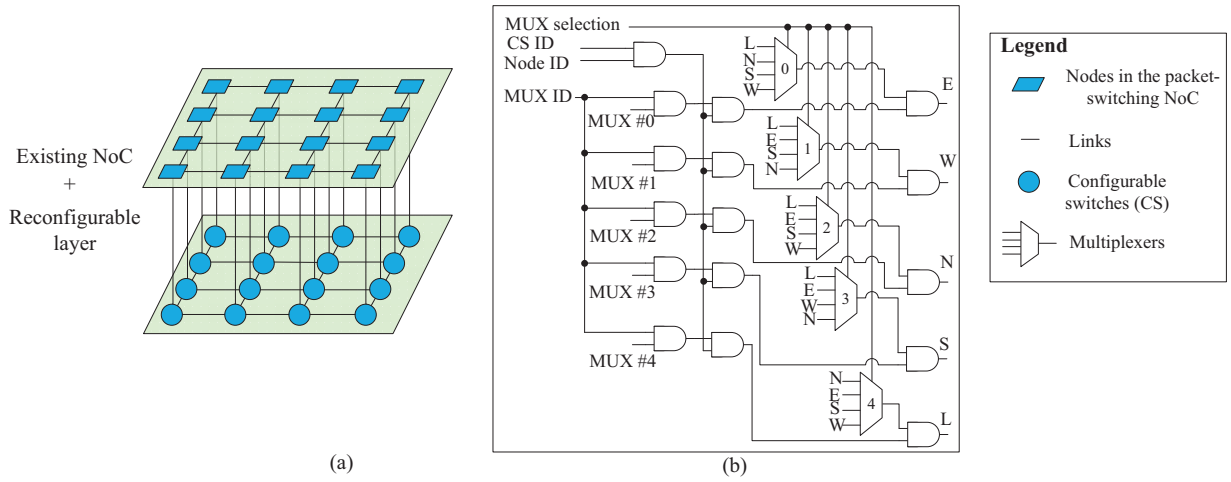

Fig. 2. The reconfigurable layer. (a) The general architecture, including an existing packet-switching NoC and the reconfigurable layer proposed in this paper. (c) The design of a configurable switch.

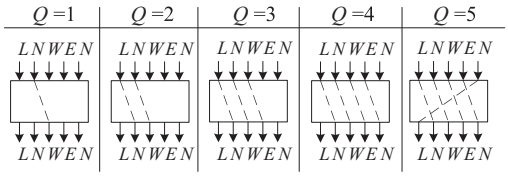

Fig. 3. Possible interconnection schemes of input and output ports in a configurable switch. Two connected ports are called a pair of ports. Q denotes the number of pairs of connected ports. The maximum value of Q is 5.

the other ports. These two connected ports are called a pair of ports. Let Q be the number of pairs of ports that are connected within one configurable switch. Fig. 3 shows examples of connected pairs of input/output ports with different values of Q. The maximum value of Q is 5 as in this figure.

The problem could be formulated as follows.

$Min(V + Q)$, Subject to:

$$\sum_{k \in |MP_i|} df_{i,k} = 1 \tag{3}$$

$$\sum_{i} \sum_{l \in PATH_{i,j}, k \in |MP_i|} df_{i,k} \leq V, \forall \text{ link } l \tag{4}$$

$$\sum_{Y \neq X} C_{XY}^{p} \leq Q, \forall p, \forall\ X, Y \in \{E, W, S, N, \text{and } L\} \tag{5}$$

$$C_{XY}^{p} \geq df_{i,j}, \forall p, \text{if } mp_{i,j} \text{ makes an } X \text{ to } Y \text{ turn in } p \tag{6}$$

Eqn. (3) ensures exactly one path is established for each dominant flow. Eqn. (4) tries to minimize the overlap among the links in the reconfigurable layer. Eqn. (5) tries to minimize the overlap inside each configurable switch, *e.g.*, an input should not be connected to more than one output. Eqn. (6) ensures that if a flow's path makes a turn from direction X to Y inside a configurable switch p, C_{XY}^{p} will be set.

Different from the work in [14] which only sets the constraint of the link bandwidth capacity, the key idea of phase II is to minimize both V and Q, corresponding to the overlap of links and switches. When solving the problem, V is typically larger than 1, i.e., more than one flows share a single link. This will cause the paths in the reconfigurable layer to overlap, which should be avoided as only simple point-to-point connections are allowed in this layer.

To deal with the overlapping problem, a simple solution is to increase the number of channels/links between two configurable switches; that is, if a link is shared by V flows, it is extended to include V links between the two configurable switches. However, this approach demands a lot hardware resource which varies from application to application.

As an alternative, if we reformulate the problem by setting V less than 2, we shall be completely free from the overlapping concerns. Under this condition, the problem objective becomes

$Min(Q)$,

The following two additional constraints are included besides the ones given in Eqn. (3)∼(6).

$$V < 2 \tag{7}$$

$$Q < 6 \tag{8}$$

Eqn. (7) ensures non-overlapping paths among the flows. Eqn. (8) set limit on the maximum value of Q as in Fig. 3. This new problem formulation implies lower hardware resource required. However, in most cases, we fail to find such solutions. The reason is that, due to minimal route overlapping, some flows may end up using the same links/configuration switches no matter which minimal paths are actually selected. For example, consider two flows $f_{<0,1>}$ and $f_{<0,2>}$ in a 4×4 reconfigurable layer, i.e. flows from node 0 to 1 and from 0 to 2 (Fig. 4). The paths of the two flows will always overlap if minimal routing is assumed. To overcome this problem, a heuristic search algorithm is used to find out the solutions with relaxed constraints. In the example, only one path actually will be selected for one flow (*e.g.*, $f_{<0,1>}$) in the reconfigurable layer, leaving the other flow (*e.g.*, $f_{<0,2>}$) to be routed in the packet-switching NoC.

Heuristic search based path setup (Algorithm 2)

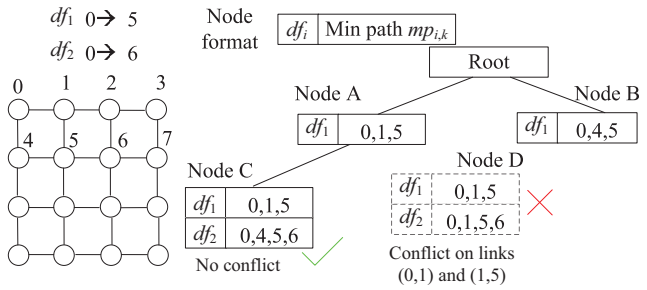

Fig. 4. The search tree example in the heuristic based path setup optimization.

In this algorithm, a search tree is built where each tree node keeps a record of some of the flows passing through it. Each flow record has two fields: *<flow ID, the minimal path taken by the flow>*, as shown in Fig. 4. Each flow in the sorted dominant flow list DF is scanned and checked to find all its minimal paths. A new node is inserted if one of the minimal paths does not overlap any of the flows recorded in the parent node. Note that, if all of the minimal paths of a new flow are found to conflict with at least one of the flows in the parent node, the new flow is skipped and the next flow will be checked. This process continues until all of the flows in DF are checked. For example, in Fig. 4, only two dominant flows are considered, $f_{<0,5>}$ and $f_{<0,6>}$. Two tree nodes A and B are created with two minimal paths (i.e. paths (0, 1, 5) and (0, 4, 5)) for the first flow $f_{<0,5>}$. These two tree nodes are attached to the root. For the second flow $f_{<0,6>}$, the first tree node C in the last level with the path (0, 4, 5, 6) does not conflict with the path of $f_{<0,5>}$ in node A. However, the second tree node D contains the path (0, 1, 5, 6) which conflicts with $f_{<0,5>}$'s path (0, 1, 5) in node A, in that they both use the links (0, 1) and (1, 5). Thus, the second tree node D of $f_{<0,6>}$ will not be attached to node A.

Algorithm 2: HeuristicBasedPathSetup

Input: Dominant flow list $DF = \{df_1, ..., df_S\}$, where S is the number of dominant flows (Table I)
Output: Non-overlapping point-to-point paths for the dominant flows.
Function: Find point-to-point paths for dominant flows
begin
  1) sort the dominant flows in descending order according to their traffic volumes;
  2) set the root of tree as empty set;
  for *each flow* df_i *in the sorted list* **do**
    find all the minimal paths MP_i between the source and destination;
    for *each minimal path* $mp_{i,k}$ *of this flow* **do**
      if *no conflict with parent tree nodes* **then**
        add the $< df_i, mp_{i,k} >$ as a new node to the tree;
    end
  end
end

TABLE II. PARAMETERS USED IN THE SIMULATION

Number of processors	64 (MIPS ISA 32 compatible)
Fetch/Decode/Commit size	4 / 4 / 4
ROB size	64
L1 D cache (private)	16KB, 2-way, 32B line, 2 cycles, 2 ports, dual tags
L1 I cache (private)	32KB, 2-way, 64B line, 2 cycle
L2 cache (shared) MESI protocol	64KB slice/node, 64B line, 15 cycles, 2 ports
Main memory size	2GB
On-chip network parameters	
NoC flit size	72 bits
Data packet size	5 flits
Meta packet size	1 flit
NoC latency	latency: router 2 cycles, link 1 cycle
NoC VC number	4
NoC buffer	5 × 12 flits
Benchmarks	
PARSEC	streamcluster, swaptions, ferret, fluidanimate, blackscholes, freqmine, dedup, canneal
SPLASH-2	barnes, raytrace

III. EXPERIMENTAL EVALUATIONS

A. Experimental setup

We use an event-driven many-core simulator to model the NoC architecture that is designed following the self-tuning framework described in previous section. Table II lists the NoC parameters that were plugged into the many-core simulator. The ORION 2.0 power library [15] is integrated with our simulator. Evaluation is performed over a suite of benchmarks adopted from SPLASH-2 [16] and PARSEC [17]. The benchmarks are cross-compiled into MIPS-compatible binaries. In our experiment, 8 benchmarks in PARSEC were cross-complied. All the benchmarks in SPLASH-2 could be compiled, and all of them were included into our experiments. Of these compiled benchmarks, we deliberately picked two benchmarks from SPLASH-2, (i) barnes, whose data traffic shows a clear flow dominance feature, and (ii) raytrace with less flow dominance, for reporting and analysis.

In the experiments, we select the 2D concentrated mesh (denoted as CMesh) as the basic topology which can be augmented following the proposed AIN. In this paper, we assume that each node or tile has a processor, an L1 cache, an L2 bank with directory (or just a memory controller), and an NI/router. In the concentrated mesh, four such nodes are considered to be a "meta-node". In the experiments, the sampling interval is set to be 5M cycles, which balance between the computation time and precision of the regression models.

We compare the performance improvement of architectures augmented with AIN, EVC and duo [10]. The baseline router is assumed to have two pipeline stages. We also compare the result with that obtained from an ideal router with only one pipeline stage.

B. Expansion factor of the reconfigurable layer

In Section II, we see that the reconfigurable layer can be expanded by inserting more switches to help increase path diversity that is needed for accommodating more flows in the reconfigurable layer. It is interesting to see how the size of the reconfigurable layer affects the number of non-overlapping paths built by Algorithm 2. Fig. 5 shows the sensitivity of the

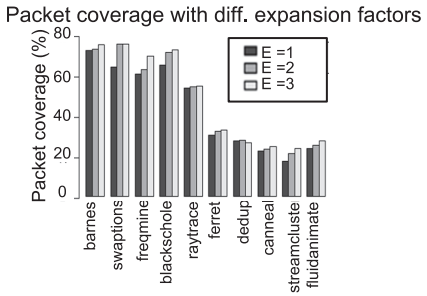

Fig. 5. The packet coverage with different expansion factors. E is the expansion factor (Section II.D). The packet coverage is defined as the packet number of the flows whose paths could be setup by Algorithm 2 over the total packet number.

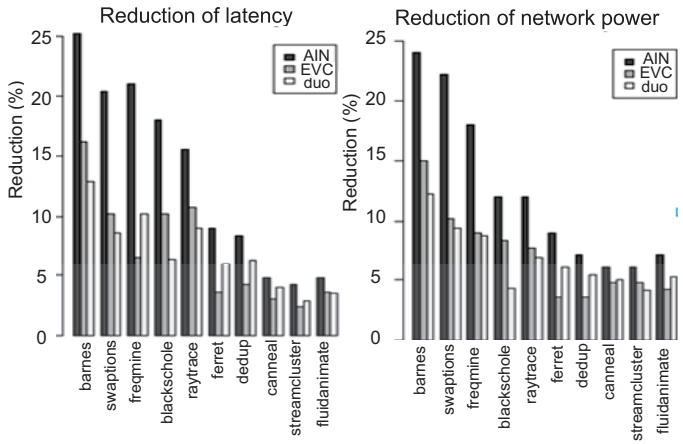

Fig. 6. The (a) latency improvement and (b) network power saving of CMesh+AIN, CMesh+EVC, and CMesh+duo against the original CMesh with 2-stage pipeline routers.

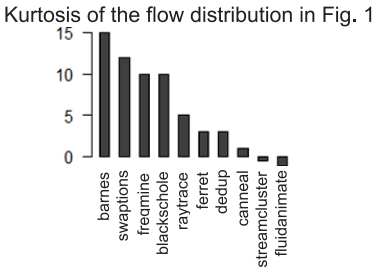

Fig. 7. The kurtosis of the flow distribution of the benchmarks as the metric of flow dominance.

expansion factor with respect to the size of the reconfigurable layer for benchmarks. The flow packet coverage is defined as the packet number of the flows whose paths could be setup by Algorithm 2 over the total packet number.

For streamcluster, we can setup non-overlapping point-to-point paths for 16 flows (accounting for 18% of total packets) without exercising network expansion (*i.e.*, expansion rate $= 1$). The path setup algorithm runs very fast without expansion, with cycles much less than 10^5 cycles. Consider the sampling interval of 5M cycles, the run time without expansion is negligible. Together 22 flows can find their point-to-point paths (accounting for 23% of total packets) with an expansion rate of 2 within one sampling interval. We can setup 24 flows (accounting for 24% of total packets) with expansion $= 3$, but this comes with longer run time. We can see that, expansion rate of 1 is sufficient for most of the applications with run time neglectably smaller than the sampling interval.

C. Performance comparison of AIN against the original NoC, EVC and duo

1. Comparison of AIN against the original CMesh

Fig. 6 shows the reductions of latency and network power with (1) AIN proposed in this paper, (2) EVC and (3) duo over the original CMesh. If CMesh is augmented with AIN, on average, the reductions in latency and network power are 14% and 13 %, respectively. For benchmarks like barnes, the reductions could be as much as 25% and 24%.

Among the benchmarks, barnes, swaptions, freqmine, and blackscholes, latency/network power are reduced by more than 10% with AIN, *e.g.*, the latency of barnes is reduced by 25% compared against the original CMesh. To investigate the difference in the latency/network power reduction among the benchmarks, a recall to Fig. 1 is necessary. From Fig. 1, we can find that the above applications have "narrow" peaks, which means the flow dominance is more obvious, *i.e.*, a small percentage of the flows account for a very large number of the packets. For the remaining applications, their curves in Fig. 1 are more "flat", indicating that flow dominance is not so obvious, *i.e.*, the traffic is more evenly distributed, instead of centered on a few flows.

Thus, we define the kurtosis of the flow distribution curve (see Fig. 1) as a metric of the flow dominance, which is the fourth moment about the mean divided by the standard deviation. Fig. 7 plots the kurtosis of the benchmarks in the descending order. As we can see that, the kurtosis of swaptions, blackscholes, freqmine, and barnes are larger than those of the remaining ones. Correspondingly, the reductions in terms of the latency/network power are more obvious for these applications.

Fig. 8 further compares the reductions in terms of latency and network power of AIN against the CMesh assuming the routers have only one pipeline stage, *i.e.*, the ideal routers. We can see from Fig. 8 that, the average reductions in terms of latency and power of AIN over the ideal CMesh are 8% and 7%, respectively. For the benchmarks with larger kurtosis values, the reduction in each metric is greater, *e.g.*, for barnes, swaptions, and freqmine, the reductions in term of latency and network power of AIN against the ideal CMesh are over 10%, respectively.

2. Comparison of AIN against EVC

Fig. 6 also shows the performance of EVC in terms of latency and network power. In all the cases, AIN achieves lower latency/network power than the EVC schemes. The performance difference between AIN and EVC is also obvious in the benchmarks with larger kurtosis values, *i.e.* whose flow dominance is more obvious. We can see that AIN reduces more than 10% latency/network power against EVC in the cases of barnes, swaptions, and freqmine, which have large kurtosis values. On average, AIN reduces 8% latency and 7% network power against EVC. As EVC uses a heuristic way to setup

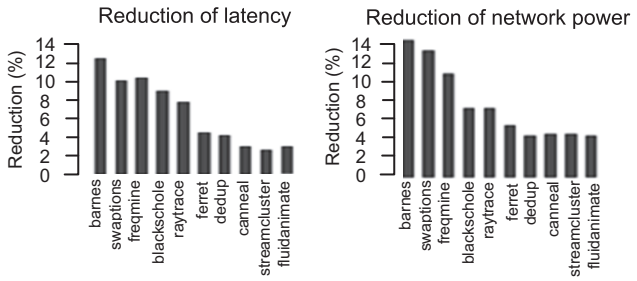

Fig. 8. The latency improvement and network power saving of AIN against the ideal CMesh with only one pipeline stage.

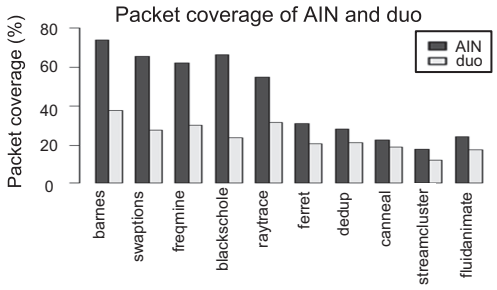

Fig. 9. Packet coverage of AIN and duo.

non-overlapping point-to-point paths for the flits, AIN learns from the global information before setting up point-to-point paths. Thus, the more flow dominance, the better AIN can optimize the flows.

3. Comparison of AIN against duo

Finally, the performance of AIN and duo is compared, which in also shown in Fig. 6. On average, AIN reduces 7% latency and 7% network power against duo. For benchmarks with larger kurtosis values, *e.g.*, barnes, swaptions, freqmine, AIN reduces more than 10% latency, network power against duo. Fig. 9 plots the packet coverage of AIN and duo. This figure shows the percentage of the total packets being optimized by customized circuits, *e.g.* the reconfigurable layer in this paper and the S-channel in the duo case. We can see that the packet coverage of AIN is higher than that of duo which means more packet transmissions are optimized in the reconfigurable layer. Thus, AIN achieves lower latency and network power than duo.

D. Cost of AIN

The cost of AIN includes three parts: memory to store model parameters, run time of the dominant flow identification /reconfigurable phases, and hardware cost of the reconfigurable layer. For the first part, an ARMA (p, q) in a K-sized network after concentration requires $2K^2 \times (p + q)$ parameters in the master core and $2p + 2q$ parameters in each of the other cores. These include the parameters$\{\varphi_n, \theta_m | n = 1, ..., p, m = 1, ..., q\}$ and the p past values of the flow variable. For a 64×64 concentrated mesh, assume an ARMA(11, 0), the master core has to store 5280 parameters, corresponding to about 10KB memory. For a polynomial model with order M, M parameters need to be stored.

The total run time for regression model calculation, path setup is in the order of 10^5 cycles. Compared with the sampling interval, which is 5M cycles, this time penalty is fairly small because the learning ends after the training phase, *i.e.*, *no* regression model building in Phase II, Phase I only takes once. In Phase II, only path setup occurs whose run time is less than 10^5 cycles. Thus the path setup takes negligible time compared to the time interval. During the path setup process, only the packet switching NoC will operate. This process takes 10^5 cycles, which happens in every 5M cycles (reconfiguration happens every 5M cycles).

The hardware cost of the reconfigurable layer is related to the design of configurable switches and wires. Assume the working frequency is 1GHz and the size of each tile in the packet-switching NoC is $1 \times 1\mathrm{mm}^2$. The length of a link (assumed to be the same width as the links in the packet-switching NoC shown in Table I) in the reconfigurable layer is 2mm as four tiles are connected to be a "meta-node" in the concentrated mesh. Each of the configurable switches (CS) has an area of 1075 um^2 and consumes 6.25uW dynamic power (switching activity is 0.5) using Synopsys Design Compiler under 45nm TSMC library. The area and dynamic power of a link are about 86606 um^2 and 0.09 W (switching activity is 0.5) respectively with the 45nm technology available from the ORION 2.0 simulator [15].

From the above analyses, we can draw some insights.

- For applications whose flow dominance is more obvious, more performance benefit can be achieved from the intelligent framework.
- The major cost of this learning/analysis framework comes from the memory to record the parameters of the models.

IV. Related Work

Studies of traffic modeling and prediction have been focused on the Internet. For example, neural network [18] is used to predict traffic flows. However, the behavior of Internet traffic is substantially different from the on-chip case. In [11], the Hurst parameter is estimated to exploit the self-similarity in NoC traffic running MPEG-2 video applications. However, this work cannot be used to model and predict online traffic dynamically. In [10], the spatiotemporal distribution of NoC flows is analyzed, but this work does not include prediction models for the flow. Machine learning methods, like reinforcement learning, are used in NoC routing algorithm to choose less congested channels. For example in [6], an adaptive routing algorithm augmented by the reinforcement learning is proposed to selected output channels based on global congestion information. This approach could be complementary to our approach proposed in this paper which optimizes the end-to-end flows.

In the literature, several NoC topologies are proposed including concentrated mesh, flattened butterfly, and multi-drop express cube structure [19], where express channels can be used to optimize communication. However, these topologies add substantial complexity to the router architecture. In addition, these are fixed topologies without the capability of being reconfigured online. An application-aware topology reconfiguration [14] is proposed which sets up paths for the communications depending on the applications traffic. As demonstrated in their experiments, this approach is only suitable for multiprocessor system-on-chip (MPSoC) applications

which have less frequent traffic. For communication-intensive CMP applications, not all the communications can find a path using the approach in [14]. Besides, [14] only has constraints on the link bandwidth capacity, while our approach tries to minimize the overlap of links as stated in Section II.D.

Work has also been proposed to make the NoC more flexible, which enables dynamic configuration. For example, express virtual channels (EVC) [8] are used to bypass router pipeline stages utilizing communication locality as in [8] [20]. Another scheme, virtual point to point connection in [9] sets up virtual point-to-point connection between senders and receivers. Both of the two methods optimize communication by bypassing router pipeline stages. However, neither of the two schemes includes a learning process which could adjust the NoC structure with global traffic info, *i.e.*, only heuristics are used without the knowledge of application traffic. In the experiments, our framework reduces 8% latency and 7% network power against EVC, on average. For applications with more obvious flow dominance (larger kurtosis values) the reductions against EVC are more than 10%.

The duo approach in [10] analyzes the characteristics of traffic flows and use heuristics to configure the multi-drop channels to set up express channels. Our framework differs from that approach in the following three aspects. (1) The work in [10] focuses on traffic distribution analysis, while our work emphasizes a framework combing regression model and non-overlapping point-to-point path setup. Our framework is more general and targeted at supporting online reconfiguration of NoC. (2) In our approach, regression models are used to predict the traffic flow, such that, the point-to-point paths can be set up *in advance* in each time interval. In contrast, in [10] express paths are set up *after* collecting traffic flow values in current interval, whereas the traffic distribution could be different from that during the express path setup. Thus, the express paths might not be optimal for the real traffic. (3) In our approach, dominant flows are preferred to be optimized while in [10] flows with longer communication distance are biased. Thus the packet coverage of the two approaches differs as in Section III.C. More packets are optimized in our approach than that in [10]. Overall, our approach reduces 7% latency and 7% network power against the work in [10] on average.

V. Conclusion

In this paper, we proposed a self-tuning NoC framework to help build intelligent NoC that can deliver higher performance at the presence of changing data traffic. This framework identifies the flow dominance of the applications' flow traffic and optimizes the dominant flows in two phases. First, flows are recorded and modeled. Next, the master core identifies the dominate flows which are optimized by reconfiguring the network layer made of configurable switches. An efficient algorithm sets up non-overlapping point-to-point paths for the dominant flows on the reconfigurable layer such that these flows can traverse through the path with latency as low as the wire delay. The reconfigurable layer can be augmented to any existing NoC topologies. Our experiments showed that, existing topologies like concentrated mesh augmented with our framework can reduce as much as 25% latency and 24% network power compared against the original NoC system for both PARSEC and SPLASH-2 benchmarks. We expect that such type of self-tuning NoC systems can be used to improve the performance for a wide variety of applications.

References

[1] Y. K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nguyen, and M. Smelyanskiy. Convergence of recognition, mining, and synthesis workloads and its implications. *Proceedings of the IEEE*, 96(5):790–807, 2008.

[2] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar. A 2 Tb/s 6$, times, $4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS. *IEEE J. Solid-State Circuits*, 46(4):757–766, 2011.

[3] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das. A novel dimensionally-decomposed router for on-chip communication in 3D architectures. In *Proc. Int'l Symp. Computer Architecture*, pages 138–149. ACM, 2007.

[4] H. Matsutani, M. Koibuchi, and H. Amano. Tightly-coupled multi-layer topologies for 3-D NoCs. In *Proc. Int'l Conf. Parallel Processing*, pages 75–75. IEEE, 2007.

[5] A. Biberman and K. Bergman. Optical interconnection networks for high-performance computing systems. *Reports on Progress in Physics*, 75(4), 2012.

[6] M. Ebrahimi, M. Daneshtalab, F. Farahnakian, J. Plosila, P. Liljeberg, M. Palesi, and H. Tenhunen. HARAQ: congestion-aware learning model for highly adaptive routing algorithm in on-chip networks. In *Proc. ACM/IEEE Int'l Symp. Networks-on-Chip*, pages 19–26. IEEE, 2012.

[7] X. Wang, M. Yang, Y. Jiang, and P. Liu. On an efficient NoC multicasting scheme in support of multiple applications running on irregular sub-networks. *Microprocessors and Microsystems*, 35(2):119–129, 2011.

[8] A. Kumar, L. S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proc. Int'l Symp. Computer Architecture*, pages 150–161, 2007.

[9] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad. Virtual point-to-point connections for NoCs. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 29(6):855–868, 2010.

[10] Y. Jin, E. Kim, and T. Pinkston. Communication-aware globally-coordinated on-chip networks. *IEEE Trans. Parallel and Distributed Systems*, 23(2):242–254, 2012.

[11] G. V. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Trans. Very Large Scale Integration Systems*, 12(1):108–119, 2004.

[12] R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications*. Springer Verlag, 2010.

[13] C. M. Bishop. *Pattern recognition and machine learning*. Springer New York, 2006.

[14] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad. Application-aware topology reconfiguration for on-chip networks. *IEEE Trans. Very Large Scale Integration Systems*, 19(11):2010–2022, 2011.

[15] A. B. Kahng, B. Li, L. S. Peh, and K. Samadi. ORION 2.0: a power-area simulator for interconnection networks. *IEEE Trans. Very Large Scale Integration Systems*, 20(1):191 – 196, 2012.

[16] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *Proc Int'l Symp. Computer Architecture*, pages 24–36. ACM, 1995.

[17] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proc Int'l Conf. Parallel Architectures and Compilation Techniques*, pages 72–81. ACM, 2008.

[18] P. Cortez, M. Rio, P. Sousa, and M. Rocha. Topology aware Internet traffic forecasting using neural networks. *Artificial Neural Networks*, 4669:445–454, 2007.

[19] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Express cube topologies for on-chip interconnects. In *Proc. IEEE Int'l Symp. High Performance Computer Architecture*, pages 163–174. IEEE, 2009.

[20] M. Ahn and E. J. Kim. Pseudo-circuit: accelerating communication for on-chip interconnection networks. In *Proc. IEEE/ACM Int'l Symp. Microarchitecture*, pages 399–408. IEEE, 2010.