Trace-Driven Optimization of Networks-on-Chip Configurations

Andrew B. Kahng, Bill Lin, Kambiz Samadi and Rohit Sunkam Ramanujam
ECE Department, University of California San Diego, La Jolla, CA
Email: {abk,billlin,ksamadi,rsunkamr}@ucsd.edu

ABSTRACT

Networks-on-chip (NoCs) are becoming increasingly important in general-purpose and application-specific multi-core designs. Although uniform router configurations are appropriate for general-purpose NoCs, router configurations for application-specific NoCs can be non-uniformly optimized to application-specific traffic characteristics. In this paper, we specifically consider the problem of virtual channel (VC) allocation in application-specific NoCs. Prior solutions to this problem have been average-rate driven. However, average-rate models are poor representations of real application traffic, and can lead to designs that are poorly matched to the application. We propose an alternate trace-driven paradigm in which configuration of NoCs is driven by application traces. We propose two simple greedy trace-driven VC allocation schemes. Compared to uniform allocation, we observe up to 51% reduction in the number of VCs under a given average packet latency constraint, or up to 74% reduction in average packet latency with same number of VCs. Our results suggest that average-rate driven methods cannot effectively select appropriate links for VC allocation because they fail to consider the impact of traffic bursts. As a case study, we compare our proposed approach with an existing average-rate driven method [9] and observe up to 35% reduction in the number of VCs for a given target latency.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design

General Terms

Algorithms, Design, and Performance

Keywords

Networks-on-Chip, virtual channel, and greedy heuristics

1. INTRODUCTION

Diminishing returns in the performance of uniprocessor architectures have led to multi-core chips. In these designs, networks-on-chip (NoCs) are emerging as the interconnection fabric of choice. With the increasing number of on-chip cores, the need for a scalable and high-bandwidth communication fabric becomes more evident [5, 17]. NoCs can be designed for general-purpose multi-core processors [13, 25] or application-specific multi-core systems-on-chip (SoCs) [21, 29]. The challenges are different in each case. Since general-purpose multi-core processors are designed to run a wide range of applications, the application traffic characteristics are inherently unknown *a priori*. Hence, the configurations of on-chip routers, e.g., with respect to the number of virtual channels (VCs), are typically *uniform* across all routers in the design. On the other hand, since application-specific multi-core SoCs are designed to implement specific functions efficiently, the configuration of each router in the network can be *non-uniformly* optimized to the traffic characteristics of the particular application.

With power being a first-order design constraint, network resources must be carefully allocated to minimize overall power with no or minimum loss in performance. Among all router resources, input buffers consume a significant (e.g., 28% [7]) portion of the total communication power. Hence, minimizing the number of buffers is important for reducing router power consumption. Buffers are used in routers to house incoming flits[1] that cannot be immediately forwarded to the output links because of contention. Earlier router architectures used packet-level flow control [6, 8] whereby buffer and channel resources are allocated at the level of packets. Packet-level flow control requires large input buffers to store entire packets, and also increases contention latency as a packet must wait for the previous packet to be transmitted before it can acquire a channel. This problem was alleviated with the introduction of wormhole routers [4] which operate at flit-level granularity. However, wormhole routers without virtual channels are prone to head-of-line blocking, which significantly limits router performance. Virtual channels provide alternate paths for incoming packets to bypass a blocked packet, and hence can improve router performance. However, performance is improved at the expense of extra power consumption and area overhead.

Though the problem of NoC configuration for application-specific multi-core SoCs is not new, prior approaches [3, 8, 9] have been *average-rate driven* in that the traffic characteristics have been modeled with *average data rates*. Unfortunately, average-rate models are poor representations of actual traffic characteristics of real applications. For example, consider the actual traffic behavior of two real applications from the PARSEC [2] benchmark suite shown in Figure 1. As shown in these two applications, the actual traffic behavior tends to be very bursty with substantial fluctuations over time. Therefore, average-rate driven approaches may be misled by average traffic characteristics, which may lead to poor design choices that are not well matched to the actual traffic characteristics.

In this paper, we propose an alternate paradigm based on a *trace-driven* approach that uses actual application traffic traces to drive the optimization of NoC configurations.To illustrate the benefit of a trace-driven approach, in this paper we specifically consider the problem of application-specific VC allocation. In particular, we propose two greedy trace-driven heuristics in which we use PopNet [23], a cycle-accurate flit-level on-chip network simulator, to constantly evaluate the impact of adding or deleting a VC on average packet latency of a given application. To decouple the VC allocation and the buffer space allocation problems we assume that

[1] A flit is a fixed-sized portion of a packetized message.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC'10, June 13-18, 2010, Anaheim, California, USA
Copyright 2010 ACM 978-1-4503-0002-5 /10/06...$10.00

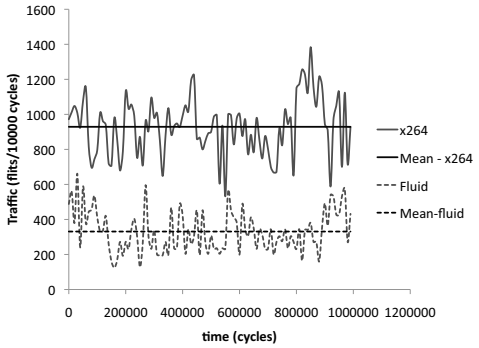

Figure 1: Communication traffic between two arbitrary nodes for two different applications.

each VC has a fixed buffer length which is larger than the average packet size of the application. Our approach is simple, but is shown to be quite powerful in achieving more efficient VC resource allocation. Although "trace-driven" suggests high runtime complexity (cf. Subsection 3.3), we believe that in practice ample computing resources are available to enhance any given application-specific NoC configuration using our techniques.

The contributions of our work are listed below.

- We propose two greedy VC allocation algorithms based on our proposed trace-driven optimization paradigm.
- We evaluate our proposed algorithms on a set of real applications. In particular, we evaluate our methods on the recently developed PARSEC benchmark suite [2], which contains multi-threaded programs that are representative of emerging workloads.
- In comparison with uniform configurations, we observe up to 51% reduction in the number of VCs needed to achieve the same performance, and up to 74% reduction in average packet latency with the same number of VCs.
- We also compare our proposed approach with an existing average-rate driven VC allocation method [9]. In this comparison, we achieve up to 35% reduction in the number of VCs, which demonstrates the benefits of a trace-driven approach over average-rate driven approaches.

The remainder of this paper is organized as follows. In Section 2 we contrast against prior related work. Section 3 describes our trace-driven non-uniform VC allocation problem formulation. In Section 4 we describe our experimental setup and testcases, and present our results. Finally, Section 5 concludes the paper.

2. RELATED WORK

Input buffers affect the overall performance of interconnection networks and consume a major portion of router power and area [7, 10, 12, 15]. Hence, determining the optimal buffer size that maximizes performance with little or no impact on power is of utmost importance. Several buffer sizing methods have been proposed in the literature. Hu *et al.* [8] propose a probabilistic approach to size the input buffers of the intermediate nodes of the network. The main objective is to minimize the average packet latency of all communications occurring in the network while keeping total buffer area under a certain budget. In their approach, the authors of [8] use packets as atomic units of storage (i.e., store-and-forward); however, packet-level flow control mechanisms are not efficient due to their negative impact on latency and area.

Chandra *et al.* [3] propose a sizing algorithm based on production and consumption rates of packets. A major disadvantage of this work is the use of synthetic traffic models which can potentially impact the relevance of solutions. In a different approach, Manolache *et al.* [14] propose a traffic shaping methodology in which packets are delayed at the source to avoid contention between different flows, and hence reduce the total amount of buffer space required at intermediate nodes.

In recent works, virtual channel (VC) routers have gained prominence. Virtual channels allow packets to pass blocked packets by multiplexing the physical channel, and thus mitigate the head-of-line blocking problem in wormhole routers. Huang *et al.* [9] propose a queueing-based algorithm for VC allocation. Given a network topology, a mapping of tasks to the NoC, and a deterministic routing algorithm, they determine the number of VCs allocated to each link. The proposed algorithm proceeds in a greedy fashion: it increases the number of VCs for a given link only if this reduces the average packet latency. Similarly, Al Faruque *et al.* [1] propose a two-step approach to minimize the number of virtual channels under certain quality of service criteria. In their approach, they first minimize the number of VCs during mapping of communication tasks; then, based on the assumption that packet arrivals follow a Poisson distribution, they estimate the size of each VC for each output port. A significant shortcoming of this approach is that it relies on Markovian assumptions for multimedia applications which are contradicted by, e.g., the experimental results in [28].

All of the above approaches are static: buffer sizes (including number of VCs) are determined at design time based on analysis of different traffic patterns. In contrast, buffer space can also be allocated dynamically, i.e., at runtime [6, 18, 24]. Dynamically allocated multi-queue [24] uses linked lists to allocate VCs to each port. However, a three-cycle delay at every flit arrival/departure, to update the logical pointer to the free list, makes this approach unsuitable for high-performance designs. Fully-connected circular buffers [18] use registers to selectively shift flits within buffers. However, this method requires a $P^2 \times P$ crossbar instead of the conventional $P \times P$ crossbar, where P is the number of ports. It also requires existing flits to be shifted when new flits arrive. Hence, this approach has significant latency and power overheads. Finally, Nicopolous *et al.* [19] propose ViChaR, a dynamic VC allocator, in which the number of VCs and the buffer depth per VC are dynamically adjusted based on the traffic load.

In summary, VC allocation is an important issue in optimization of NoCs configurations as it directly affects the overall performance of the network and consumes a major portion of router power and area. There are two existing VC allocation approaches: (1) dynamic and (2) static. Dynamic allocation of VCs at runtime seems more desirable for general-purpose and reconfigurable design platforms executing different workloads. However, allocating VCs at design time is more desirable for application-specific NoCs running a specific application or a limited class of applications. Previous VC allocation approaches are all average-rate driven whereas in this paper, we propose two simple yet effective trace-driven VC allocation heuristics that significantly reduce the number of VCs while matching the latency attributes of a uniform VC allocation scheme. As far as the authors know, there is no existing trace-driven VC allocation approach; however, we note that similar trace-driven approaches have been used to optimize traditional bus-based communication architectures [20, 22].

3. PROBLEM FORMULATION

In a typical design of virtual channel routers, a fixed amount of hardware resources (i.e., queues) is set aside to implement the VC buffers. With power being a first-order design constraint, these resources must be allocated efficiently such that certain performance criteria are satisfied while keeping the number of VCs to a minimum. When the application is known, as discussed above, the concept of a trace-specific resource allocation becomes relevant. We formulate the trace-driven VC allocation problem as follows.

Given:

- Application communication trace, C_{trace}
- Network topology, $T(P, L)$, where P is the set of processing elements and L is the set of physical links
- Deterministic routing algorithm, R
- Target latency, D_{target}

Determine:

- A mapping $\mathbf{n}_{VC}$ from the set of links L to a set of positive integers, i.e., $\mathbf{n}_{VC} : L \rightarrow Z^+$, where for any $l \in L$, $n_{VC}(l)$ gives the number of VCs associated with link l, such that $(\sum_{l \in L} n_{VC}(l))$ is minimized while average packet latency

using routing algorithm R, $D(\mathbf{n}_{VC}, R)$, is within a target latency constraint D_{target}

Objective:

- Minimize $\sum_{l \in L} n_{VC}(l)$

Subject to:

- $D(\mathbf{n}_{VC}, R) \leq D_{target}$

In the next two subsections, we propose two greedy heuristics, whose performance will be discussed in Section 4.

3.1 Greedy Addition VC Allocation

Our first VC allocation heuristic is shown in Figure 2. The algorithm initializes every network link, l, with one VC (Lines 1-3); this is equivalent to wormhole routing. Hence, the total number of VCs in the network, N_{VC}, is initialized to the total number of links, N_L (Line 5). Then, the algorithm proceeds in a greedy fashion: in each iteration, the performance of all the N_L possible perturbations of the current VC configuration, $\mathbf{n}_{VC}^{current}$, are evaluated simultaneously. Each perturbation consists of adding exactly one VC to one link (Line 9). The average packet latencies of all perturbed VC configurations are examined, and the configuration with the smallest average packet latency, $\mathbf{n}_{VC}^{best}$, is chosen (Line 12). Subsequently, $\mathbf{n}_{VC}^{best}$ is set as the starting configuration of the next iteration. The algorithm stops if either the total number of allocated VCs exceeds the VC budget, $budget_{VC}$, in which case the VC budget needs to be increased to achieve the target latency, or a configuration with better performance than the target average packet latency, D_{target}, is found. Runtime analysis of our proposed heuristics is explained in Subsection 3.3.

Algorithm: Greedy Addition
Input: Application communication trace, C_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency, D_{target}
Output: Vector $\mathbf{n}_{VC}$, which contains the number of VCs associated with each link $l \in L$

1. **for** i = 1 to N_L
2. $n_{VC}^{current}(l) = 1$;
3. **end for**
4. $\mathbf{n}_{VC}^{best} = \mathbf{n}_{VC}^{current}$;
5. $N_{VC} = N_L$;
6. **while** $(N_{VC} \leq budget_{VC})$
7. **for** l = 1 to N_L
8. $\mathbf{n}_{VC}^{new} = \mathbf{n}_{VC}^{current}$;
9. $n_{VC}^{new}(l) = n_{VC}^{current}(l) + 1$;
10. **run** trace simulation on $\mathbf{n}_{VC}^{new}$ and **record** $D(\mathbf{n}_{VC}^{new}, R)$;
11. **end for**
12. **find** $\mathbf{n}_{VC}^{best}$;
13. $\mathbf{n}_{VC}^{current} = \mathbf{n}_{VC}^{best}$;
14. **if** $(D(\mathbf{n}_{VC}^{best}, R) \leq D_{target})$
15. break;
16. **end if**
17. N_{VC}++;
18. **end while**

Figure 2: Greedy addition heuristic.

Despite the effectiveness of the addition approach as shown in Section 4.2, it has an inherent disadvantage, namely, it bases its decision at each iteration on the impact (on average packet latency) of adding only a single VC. To demonstrate the drawback of this approach, consider the example of Figure 3, where two traffic flows F_1 and F_2 share links $A \rightarrow B$ and $B \rightarrow C$, both of which initially have only one VC (A, B, and C are network nodes). F_1 turns west and F_2 turns east at node C. In this case, adding a VC to either link $A \rightarrow B$ or link $B \rightarrow C$ may not have a significant impact on average packet latency of flows F_1 and F_2 because the other link with just one VC becomes the bottleneck. On the other hand, if VCs are added to both links $A \rightarrow B$ and $B \rightarrow C$, the average packet latency of the flows may be significantly reduced. This is because if one of the two flows (F_1 or F_2) is blocked at node C due to congestion further downstream, the other flow can still use the shared links $A \rightarrow B$ and $B \rightarrow C$. The greedy VC addition approach may fail to realize the benefits of these combined additions and not pick either of the links as candidates for VC allocation at a given iteration. To overcome this drawback, we propose another greedy VC allocation heuristic based on *deletion*, instead of addition, of VCs.

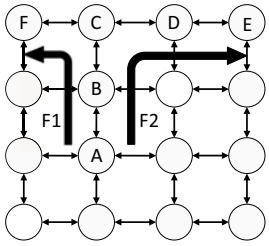

Figure 3: An example illustrating the drawback of greedy addition heuristic.

3.2 Greedy Deletion VC Allocation

Our greedy VC deletion algorithm is shown in Figure 4. The approach is structurally similar to greedy addition, except for the fact that we delete a VC, instead of adding one, at the end of each iteration. The algorithm starts with a given initial VC configuration for the network[2], with number of VCs equal to $\mathbf{n}_{VC}^{initial}$ (Line 1). At the beginning of each iteration, the current VC configuration vector, $\mathbf{n}_{VC}^{current}$, is saved into a new vector, $\mathbf{n}_{VC}^{new}$ (Line 6). Next, single VC-deletion perturbations of the current VC configuration are generated by removing a VC from each link in the network that has more than one VC (Lines 7-8). The impact of each VC removal on the average packet latency is assessed and the one resulting in best latency is selected for the next iteration (Lines 12-13).

The stopping condition for the algorithm is qualitatively different from that of the addition approach. We *generally* expect the average packet latency to increase at every iteration as VCs are removed. However, on rare occasions during the algorithm execution we may encounter a configuration with fewer VCs that has comparable or slightly lower packet latency than one with a higher number of VCs. This may be because the link for which a VC was removed was not the bottleneck link responsible for increased average packet latency. Hence, instead of stopping the algorithm as soon as the average packet latency exceeds D_{target}, it is better to continue exploring configurations even after exceeding the target latency value, and then return the minimum VC configuration that satisfies the latency constraint. The algorithm automatically stops once a wormhole configuration is reached and there are no more VCs left to be deleted.

The greedy deletion approach can overcome the drawback of greedy addition depicted in Figure 3. In that example, if the initial configuration had two VCs for links $A \rightarrow B$ and $B \rightarrow C$, deletion of a VC on either of these links would expose the link as a bottleneck. At a given iteration, a VC is deleted from one of these links only if the impact of the deletion on packet latency is less than the impact of deleting VCs from any other link in the network.

Figure 5 shows the average packet latency of each intermediate configuration using the greedy VC addition and deletion approaches for two different traces, *fluidanimate* and *vips* from the PARSEC benchmark suite [2]. The results presented are for a mesh

[2]In this paper, without the loss of generality, we assume that the algorithm starts with a given uniform VC configuration.

network with 16 nodes and 64 links (details of the setup are explained in Section 4). In general, the average packet latency decreases as VCs are increased in the addition algorithm, and decreases as VCs are removed in the deletion algorithm. However, the change in packet latency is much smoother in the case of deletion of VCs, compared to addition. This is because adding a VC at a single link may not have a significant impact on average packet latency unless the added VC relieves congestion along the entire path of a traffic flow, as illustrated in the Figure 3 example. This explains the stepwise nature of the curve for VC addition: there are periods of little improvement followed by steep descents. The intermediate solutions in the deletion approach are of a slightly higher quality because the deletion heuristic is better at detecting bottleneck links.

Algorithm: Greedy Deletion
Input: Application communication trace, C_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency, D_{target}; Initial VC configuration, $\mathbf{n}_{VC}^{initial}$
Output: Vector $\mathbf{n}_{VC}$, which contains the number of VCs associated with each link $l \in L$

1. $\mathbf{n}_{VC}^{current} = \mathbf{n}_{VC}^{initial}$;
2. $\mathbf{n}_{VC}^{best} = \mathbf{n}_{VC}^{current}$;
3. $N_{VC} = \sum_{l \in L} \mathbf{n}_{VC}^{current}(l)$;
4. **while** $(N_{VC} \geq budget_{VC})$
5. **for** l = 1 to N_L
6. $\mathbf{n}_{VC}^{new} = \mathbf{n}_{VC}^{current}$;
7. **if** $(n_{VC}^{current} > 1)$
8. $n_{VC}^{new}(l) = n_{VC}^{current}(l) - 1$;
9. **run** trace simulation on $\mathbf{n}_{VC}^{new}$ and **record** $D(\mathbf{n}_{VC}^{new}, R)$;
10. **end if**
11. **end for**
12. **find** $\mathbf{n}_{VC}^{best}$;
13. $\mathbf{n}_{VC}^{current} = \mathbf{n}_{VC}^{best}$;
14. **if** $(D(\mathbf{n}_{VC}^{best}, R) \leq D_{target})$
15. break;
16. **end if**
17. $N_{VC}--$;
18. **end while**

Figure 4: Greedy deletion heuristic.

3.3 Runtime Analysis

Let m be the number of VCs added to (by greedy addition) or deleted from (by greedy deletion) an initial VC configuration. The runtime of the two proposed heuristics, $T_{heuristic}$, for any given input trace is

$$T_{heuristic} = m * N_L * T(\textit{trace simulation}) \quad (1)$$

where $T(\textit{trace simulation})$ is the average time to run trace simulation on all VC configurations explored by the algorithm. The above expression for runtime holds if the performance of each perturbation of the current VC configuration is evaluated sequentially. However, the code for the two heuristics can be easily parallelized by evaluating all the perturbations in parallel. This results in an improved runtime of

$$T_{heuristic} = m * T(\textit{trace simulation})_{max} \quad (2)$$

Here, $T(\textit{trace simulation})_{max}$ represents the average of the maximum runtimes of trace simulation at each iteration, where the maximum is taken over the runtimes of all perturbations in the iteration.

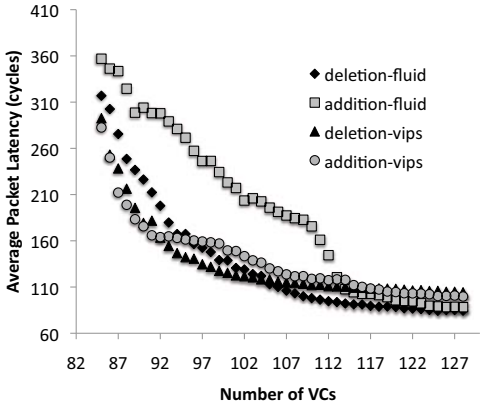

Figure 5: Performance of addition and deletion VC allocation heuristics on *fluidanimate* and *vips* traces.

4. EVALUATION

In this section, we first describe our experimental setup and then present the results of our proposed VC allocation heuristics. Finally, we assess the impact of our approach on network power and area.

4.1 Experimental Setup

We use PopNet [23], a flit-level, cycle-accurate on-chip network simulator, to determine the average packet latency (APL) of a non-uniform VC configuration for a given traffic trace. PopNet models a typical four-stage on-chip router pipeline comprising of route selection and VC allocation in the first pipeline stage followed by switch arbitration, switch traversal and link traversal in the remaining three stages. The head flit of a packet traverses all four pipeline stages while the body flits bypass the first stage and inherit the output port and output VC reserved by the head flit. The number of virtual channels at the input ports can be individually configured to implement any non-uniform VC configuration at a router. The routing algorithm is always fixed to be dimension-ordered routing where packets are first routed along the X dimension, and then along the Y dimension. The latency of a packet is measured as the delay between the time the header flit is injected into the network and the time the tail flit is consumed at the destination. The APL value reported by PopNet is the average latency over all packets in the input traffic trace.

Table 1: Processor configuration for trace generation

Cores	16
Private L1 cache	32KB
Shared L2	1MB distributed over 16 banks
Memory latency	170 cycles
Network	4×4 mesh
Packet sizes	8B data packets, 72B control packets

To evaluate our VC allocation heuristics, we use seven different benchmarks from the PARSEC benchmark suite, namely, *blackscholes*, *canneal*, *ferret*, *fluidanimate*, *swaptions*, *vips* and *x264* [2]. These benchmarks are multithreaded programs, representative of emerging future workloads and encompassing diverse application domains. The network traffic traces were generated by running these programs on *Virtutech Simics* [30], a full system simulator, and capturing the program's memory trace. The GEMS toolset [16] running on top of *Simics* was used to perform accurate timing simulation. We simulate a 16-core tiled CMP architecture arranged in a 4×4 mesh, with parameters shown in Table 1. Each tile consists of an in-order SPARC core with private L1 and shared L2 cache. DRAM is attached to the chip using four memory controllers that are at the corners of the mesh. In all seven traces, every node in the mesh both sends and receives packets. Also, all 64 links of the 4×4 mesh are used. Hence, for the addition heuristic we start with a wormhole configuration with exactly 64 VCs (1 VC/link) and set

our VC budget to 256 VCs, equivalent to an uniform configuration with 4 VCs/link. In the case of deletion, we start with a uniform configuration with 4 VCs/link and delete one VC at every iteration until we reach a wormhole configuration. Each iteration of the addition and deletion heuristics creates at most 64 perturbations which are run in parallel on a computational grid.

4.2 Experimental Results and Discussions

In this subsection, we present the results of our proposed VC allocation heuristics with respect to average packet latency reduction and reduction in the number of VCs. We compare our greedy VC addition and deletion approaches with uniform VC allocation scheme and our reimplementation of the average-rate driven VC allocation algorithm of [9] on the seven benchmarks from the PARSEC suite. Figures 6 and 7 show that our algorithms can reduce the number of VCs required to achieve the same target latency as uniform configurations with two VCs per port and three VCs per port, respectively. We also show that our algorithms can achieve higher reductions in the number of VCs compared to existing average-rate driven approaches.

Figure 6 shows the number of allocated VCs using our greedy addition and deletion algorithms to achieve the same average packet latency as that of a uniform configuration with two VCs per port (uniform-2VC). We observe that our addition and deletion heuristics can achieve up to 36% and 34% reduction in number of allocated VCs, respectively. On an average, both our techniques reduce the number of VCs by around 21% across all benchmarks. Figure 7 shows the number of VCs required using our algorithms to achieve the same average packet latency as an uniform configuration with three VCs per port (uniform-3VC). Uniform-3VC achieves lower average packet latency than uniform-2VC and so more VCs need to be allocated even using our algorithms to achieve the same target latency. However, we see high reductions of up to 51% for greedy deletion and 48% for greedy addition, compared to the 192 VCs required by the uniform configuration. Even on an average, high reductions of 41% and 31% were observed for the greedy deletion and addition techniques, respectively. These reductions over uniform VC allocation are mainly seen because our VC allocation schemes are able to fully exploit the extra degree of freedom offered by non-uniform VC allocation, using complete knowledge of the application trace. Greedy deletion reduces the number of VCs over addition by around 25% on average when the latency target is the average packet latency of uniform-3VC. It performs comparably to greedy addition when the target latency is the packet latency of uniform-2VC. Thus, of the two heuristics, greedy deletion yields slightly better solutions for the reasons discussed in Section 3.2 above.

Figures 6 and 7 also compare our algorithms to the solutions obtained using an average-rate driven technique [9] for VC allocation. Our addition and deletion algorithms outperform the average-rate driven approach by 19% and 20%, respectively, over all traces when the target latency is set to be the latency of uniform-2VC and by 25% and 35%, respectively, when the target latency is the latency for uniform-3VC. The average-rate driven approach reduces the number of VCs over the uniform configurations for only two of the seven benchmarks. We believe that this is due to the fact that the average-rate characteristics of an application trace are not a very accurate representation of the actual traffic, which is bursty in nature for most traces with wide variations about the average rate (recall Figure 1). Our trace-driven techniques accurately comprehend the effects of these traffic bursts and can lead to significantly better solutions.

Next, in Figures 8 and 9 we highlight the potential benefits of one of our VC allocation algorithms (greedy deletion) on two of the traffic benchmarks, namely, *fluidanimate* and *vips*. In our experiments, we set a VC budget of 256 VCs or 4 VCs per link. We observe that an uniform configuration with 4 VCs per port has the best average packet latency among all uniform configurations possible within the VC budget. With our greedy deletion approach, we are able to achieve the same latency as that of an uniform configuration with 4 VCs per port, with 50% and 42% reduction in the number of VCs for the *fluidanimate* and *vips* benchmarks, respectively.

Measuring the reduction in the number of VCs to achieve a given performance criteria shows the potential power and area benefits of trace-driven VC allocation. Another outlook would be to quantify the performance benefits while keeping the number of VCs fixed. With respect to this metric, we see in Figures 8 and 9 that with a constraint of 128 VCs, the average packet latency of greedy deletion is better than that of an uniform-2VC configuration by 32% for the *fluidanimate* benchmark and by 74% for the *vips* benchmark. So, trace-driven non-uniform VC allocation can potentially be used to either reduce power within a given performance constraint or to improve performance within a given power constraint.

Finally, to assess the impact of our proposed approach on network power and area, we use ORION 2.0 [10]. The ORION 2.0 router model assumes the same number of VCs at every port in the router. However, we need to compute the router power for non-uniform VC configurations. Hence, we first estimate the power overhead of adding a single VC to all router ports. This is done by computing the router power for uniform configurations with one, two, and three VCs per port and averaging the power difference between the uniform-1VC (uniform-2VC) and uniform-2VC (uniform-3VC) configurations. This gives an estimate of the power overhead of adding one VC to all router ports. This value is divided by the number of router ports to determine the overhead of adding a single VC to just one port. We use a similar approach to find the area overhead of adding a single VC to one router port. The number of VCs saved by our trace-driven heuristics compared to uniform configurations is then multiplied by the power (area) overhead of adding a single VC to a router port to estimate our power (area) savings. From our experiments, we observe that our approach can reduce the network power by up to 7% and 14% compared to the uniform-2VC and uniform-3VC configurations, respectively, while achieving the same performance. Similarly, we achieve up to 9% and 16% area reduction compared to the uniform-2VC and uniform-3VC configurations, respectively.

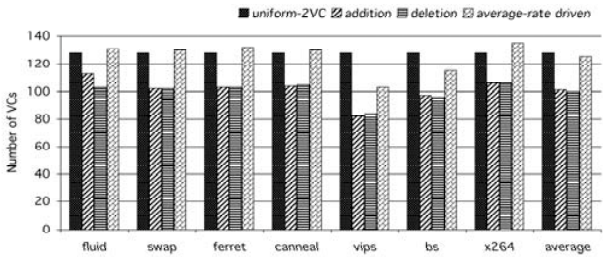

Figure 6: Performance of addition and deletion VC allocation methods versus the uniform-2VC configuration.

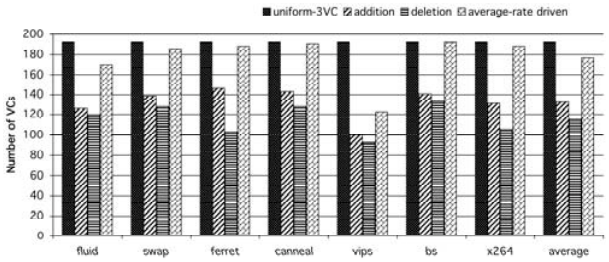

Figure 7: Performance of addition and deletion VC allocation methods versus the uniform-3VC configuration.

5. CONCLUSIONS

In this paper, we have proposed a trace-driven approach to the optimization of NoC configurations. We have specifically considered the problem of application-specific VC allocation, which seeks

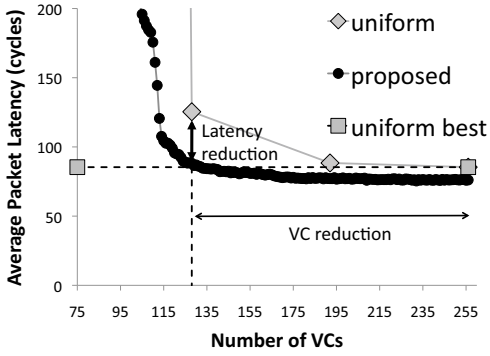

Figure 8: Average packet latency and VC reductions for the *fluidanimate* application.

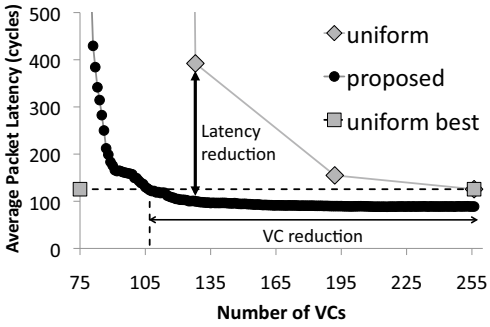

Figure 9: Average packet latency and VC reductions for the *vips* application.

to minimize the average packet latency of a given application while minimizing the VC resources required for the NoC implementation. Previous approaches to this problem have been based on the use of average rate-driven models to drive design choices, but fail to accurately capture the actual application traffic characteristics, which can lead to designs that are poorly matched to the application. In contrast, our proposed algorithms are driven by actual application traffic traces in which the selection of non-uniform VC allocations is based on the impact on actual performance. In comparison with uniform VC allocation, our methods achieve up to 51% and 74% reduction in number of VCs and average packet latency, respectively. We have also compared our proposed approach against an existing average-rate driven method [9] and have observed up to 35% reduction in number of VCs. Our results clearly show the benefits of a trace-driven approach in optimization of NoC configurations.

Acknowledgments

The authors would like to thank Niket Agarwal of Princeton University for his help in generating network traces for the PARSEC benchmark suite. The authors also acknowledge the support of the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

6. REFERENCES

[1] M. Al Faruque and J. Henkel, "Minimizing Virtual Channel Buffer for Routers in On-Chip Communication Architectures", *Proc. DATE*, 2008, pp. 1238–1243.

[2] C. Bienia, S. Kumar, J. P. Singh and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications", *Technical Report TR-811-08*, Princeton University, 2008.

[3] V. Chandra, A. Xu, H. Schmit and L. Pileggi, "An Interconnect Channel Design Methodology for High-Performance Integrated Circuits", *Proc. DATE*, 2004, pp. 1138–1143.

[4] W. J. Dally and C. L. Seitz, "The Torus Routing Chip", *Journal of Distributed Computing*, 1(3), 1986, pp. 187–196.

[5] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", *Proc. DAC*, 2001, pp. 684–689.

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

[7] Y. Hoskote, S. Vangal, A. Singh, N. Borkar and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor", *IEEE MICRO*, 2007, pp. 51–61.

[8] J. Hu, U. Y. Ogras and R. Marculescu, "System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design", *IEEE Trans. on CAD*, 25(12), 2006, pp. 2919–2933.

[9] T.-C. Huang, U. Y. Ogras and R. Marculescu, "Virtual Channel Planning for Networks-on-Chip", *Proc. ISQED*, 2007, pp. 879–884.

[10] 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", *Proc. DATE*, 2009, pp. 423–428.

[11] A. K. Kodi, A. Sarathy and A. Louri, "Design of Adaptive Communication Channel Buffers for Low-Power Area-Efficient Network-on-Chip Architectures", *Proc. ANCS*, 2007, pp. 47–56.

[12] A. K. Kodi, A. Sarathy and A. Louri, "Adaptive Channel Buffers in On-Chip Interconnection Networks–A Power and Performance Analysis", *IEEE Trans. on Computers*, 57(9), 2008, pp. 1169–1181.

[13] P. Kongetira, K. Aingaran and K. Olukotun, "Niagara: A 32-Way Multithreaded SPARC Processor", *IEEE MICRO*, 25(2), 2005, pp. 21–29.

[14] S. Manolache, P. Eles and Z. Peng, "Buffer Space Optimization with Communication Synthesis and Traffic Shaping for NoCs", *Proc. DATE*, 2006, pp. 95–98.

[15] R. Marculescu and P. Bogdan, "The Chip is Network: Toward a Science of Network-on-Chip Design", *Foundations and Trends in EDA* 2(4), 2007, pp. 371–461.

[16] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill and D. A. Wood, "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset", *SIGARCH Computer Architecture News*, 33(4), 2005, pp. 92–99.

[17] G. De Micheli and L. Benini, "Networks On Chip: A New Paradigm for Systems on Chip Design", *Proc. DATE*, 2002, pp. 2–6.

[18] N. Ni, M. Pirvu and L. Bhuyan, "Circular Buffered Switch Design with Wormhole Routing and Virtual Channels", *Proc. ICCD*, 1998, pp. 466–473.

[19] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yusif and C. R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers", *Proc. MICRO*, 2006, pp. 333–346.

[20] S. Pasricha, N. Dutt and M. Ben-Romdhane, "Using TLM for Exploring Bus-based SoC Communication Architectures", *Proc. ASAP*, 2005, pp. 79–85.

[21] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki and K. Yazawa, "The Design and Implementation of a First-Generation Cell Processor", *Proc. ISSCC*, 2005, pp. 184–185.

[22] K. Sekar, K. Lahiri, A. Raghunathan and S. Dey, "FLEXBUS: A High Performance System-on-Chip Communication Architecture with a Dynamically Configurable Topology", *Proc. DAC*, 2005, pp. 571–574.

[23] L. Shang, L.-S. Peh and N. K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks", *Proc. HPCA*, 2003, pp. 91–102.

[24] Y. Tamir and G. L. Frazier, "High-Performance Multiqueue Buffers for VLSI Communication Switches", *Proc. ISCA*, 1988, pp. 343–354.

[25] M. B. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe and A. Agarwal, "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams", *Proc. ISCA*, 2004, pp. 2–13.

[26] L. P. Tedesco, N. Calzans and F. Moraes, "Buffer Sizing for Multimedia Flows in Packet-Switching NoCs", *Journal ICS*, 3(1), 2008, pp. 46–56.

[27] G. Varatkar and R. Marculescu, "Traffic Analysis for On-Chip Networks Design of Multimedia Application", *Proc. DAC*, 2002, pp. 795–800.

[28] G. Varatkar and R. Marculescu, "On-Chip Traffic Modeling and Synthesis for MPEG-2 Video Applications", *IEEE Trans. on VLSI*, 12(1), 2004, pp. 108–119.

[29] *ARM* Integrated Multiprocessor Core, 2006 http://www.arm.com/ .

[30] *Virtutech AB. Simics* full system simulator http://www.virtutech.com/.