Efficient Trace-Driven Metaheuristics for 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

As industry moves towards many-core chips, networks-on-chip (NoCs) are emerging as a scalable communication fabric for interconnecting the cores. With increasing core counts, there is a corresponding increase in communication demands in multi-core designs to facilitate high core utilization, and a consequent critical need for high-performance NoCs. Another megatrend in advanced technologies is that power has become the most critical design constraint. In this paper, we focus on trace-driven virtual channel (VC) allocation in application-specific NoCs. We propose a new *significant VC failure* metric to capture the impact of VCs on network performance and efficiently drive NoC optimization. Our proposed metaheuristics achieve up to 38% reduction in the number of VCs under a given average packet latency constraint. In addition, compared to a recently proposed trace-driven VC allocation approach [13], we obtain up to an $O(|L|)$ speedup, where $|L|$ is total number of links in the network, with no degradation in the quality of results.

1. INTRODUCTION

Networks-on-Chip (NoCs) are emerging as the de facto interconnection fabric of choice for both general-purpose chip multiprocessors (CMPs) [16, 26, 28] and application-specific multiprocessor systems-on-chip (MPSoCs) [6, 20]. With increasing core counts, there is a corresponding increase in communication demands in multi-core designs to facilitate high core utilization and the corresponding critical need for high-performance NoCs. At the same time, power has become the most critical design constraint in advanced technologies [9, 32], with power affecting virtually all "grand challenges" for NoC design.

The design of an on-chip network can be broken into its various building blocks: topology, routing, flow control, router microarchitecture, and link architecture. Among these building blocks, router microarchitecture is of utmost importance due to its significant impact on communication latency. As a result, significant research effort has been spent to reduce router latency through modified router architectures and designs [8]. NoCs can be designed for general-purpose CMPs [16, 26, 28] or application-specific MPSoCs [6, 20]. The challenges are different in each case. Since general-purpose CMPs 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, such as the number of virtual channels, are typically *uniform* across all routers in the design. On the other hand, since application-specific MPSoCs 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 minimal loss in performance. Though the problem of NoC configuration for application-specific MPSoCs is not new, prior approaches [4, 10, 11] 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 for real applications. Figure 1 contrasts actual vs. average traffic of two real applications from the PARSEC [3] benchmark suite. The actual traffic behavior tends to be very bursty, with substantial fluctuations over time. This motivates a hypothesis that average-rate driven approaches may be misled by average traffic characteristics, resulting in poor design choices that are not well-matched to the actual traffic characteristics.

To illustrate the potential suboptimality of average-rate driven approaches, we consider the problem of application-specific VC allocation. 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 now the first-order design constraint, the decision that needs to be made is how to allocate these resources such that certain performance criteria are satisfied while the number of allocated VCs is kept at a minimum. In particular, we are interested in metrics that reflect the performance of the actual application, for example the average packet latency with respect to a given application trace. In particular, we considered the following problem.

Given:

- Application trace, A_{trace}
- Network topology, $T(C, L)$, where C is the set of processing cores and L is the set of physical links
- Deterministic XY routing algorithm, R
- Target latency constraint, 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}

978-1-4244-8194-1/10/$26.00 ©2010 IEEE

Objective:

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

Subject to:

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

To quantify the limitations of average-rate driven approaches, Kahng *et al.* [13] have recently proposed two application-specific VC allocation heuristics as just one facet of the router configuration problem. The authors of [13] have also evaluated an existing average-rate driven VC allocation method [11] on the applications in the PARSEC benchmark suite. The evaluation is based on minimizing the total number of virtual channels allocated to achieve a given average packet latency performance. The results in [13] show that the same average packet latency performance achieved using an average-rate driven optimization method can be matched by their trace-driven heuristics with 35% fewer VCs and a corresponding reduction in buffer (and, power and area) requirements.

Despite being quite successful in capturing the application specificity, simple trace-driven approaches [13] incur significant runtime cost compared with average-rate approaches. There are two ways to enhance the runtime of the proposed trace-driven optimization schemes: (1) improve the runtime of trace simulation, and (2) use different metrics to capture the average packet latency such that fewer trace simulations are required to make optimization decisions. In this paper, we focus on (2) and propose *efficient* trace-driven metaheuristics for the problem of router VC allocation which substantially improve the runtime complexity of the approach proposed in [13] with no degradation in quality of results.

The contributions of our work are listed below.

- We propose two metrics: (1) significant VC failure (SVCF), and (2) queueing delay to capture the impact of addition of a single VC on average packet latency. In addition, we show that these metrics can appropriately account for actual runtime VC resource contentions.
- We develop architecture-level optimization techniques based on a *trace-driven* paradigm that directly incorporates actual application-traffic behavior and workloads into the optimization process.
- We evaluate our proposed heuristics on a set of real applications. In particular, we evaluate our methods on the PARSEC benchmark suite, which contains multi-threaded programs that are representative of emerging workloads.
- We also compare our proposed heuristics with a recently proposed trace-driven approach [13] and observe an $O(|L|)$ speedup with no degradation in the quality of results, where $|L|$ is the total number of links in the network.

The remainder of this paper is organized as follows. In Section 2, we contrast against prior related work. Section 3 describes our proposed efficient trace-driven non-uniform VC allocation heuristics. In Section 4, we propose two efficient metaheuristics using our basic heuristics described in Section 3, and show that they can provide significant runtime improvement compared with a recently proposed trace-driven approach [13]. Section 5 shows our experimental setup and testcases and presents our results. Finally, Section 6 concludes the paper.

2. RELATED WORK

Among the various components of an input-buffered router, the configuration of input buffers has been shown to have a major impact on both the overall performance of an on-chip network as well as its energy consumption [9, 15, 18, 12]. Therefore, determining the optimal buffer size and VC allocation is of critical importance in maximizing performance and minimizing of power consumption.

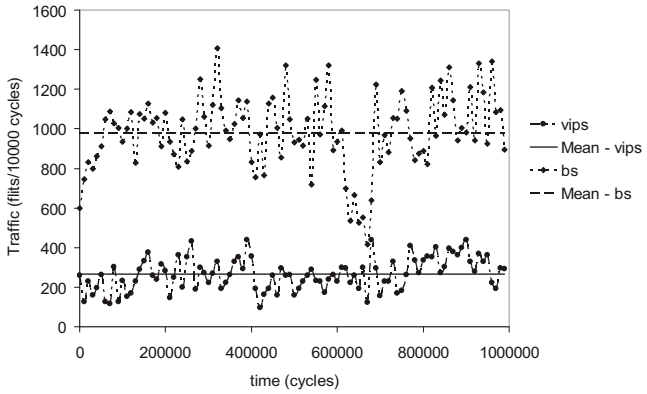

Figure 1: Actual versus average communication traffic between two arbitrary nodes in the network, for two different PARSEC benchmark applications.

For buffer sizing, a number of methods have been proposed in the literature.

- Chandra *et al.* [4] propose a sizing algorithm based on bursty transmission rates of packets. A significant drawback of this work is their use of synthetic traffic models, which can potentially impact the relevance of their solution.
- Manolache *et al.* [17] propose a traffic shaping methodology in which packets are delayed at the source to avoid contention between different flows, as to reduce the total amount of buffer usage along intermediate nodes.
- Hu *et al.* [10] propose a probabilistic approach for the sizing of input buffers along the intermediate nodes of a network. Their main goal is to minimize the average packet latency of all packet transmissions in the network while remaining under an overall target buffer area budget. However, in their work [10], their method assumes packets as atomic units of storage (i.e., store-and-forward). On the other hand, modern routers use flit-level flow control to achieve better latency performance and area.

For VC allocation, several approaches have also been proposed.

- Huang *et al.* [11] propose a queueing-based algorithm for VC allocation. Their focus is on determining the number of VCs to allocate to each link, with the assumption that the network topology, the mapping of tasks to the NoC, and the choice of deterministic routing are all given. They proposed a greedy algorithm that increases the number of VCs allocated to a given link only if the addition reduces the average packet latency.
- Al Faruque *et al.* [2] proposed a two-step VC allocation approach that aims to minimize the number of VCs required to achieve a certain quality of service (QoS) level. Their VC allocation approach is carried out during the task mapping stage. By assuming a Poisson packet arrival process, they try to estimate the size of each VC for each output port. The main shortcoming of this approach is their reliance on Markovian assumptions for multimedia applications. These assumptions have been shown to be unreliable by other works (e.g., [30]).

- In addition to static buffer and VC allocation approaches discussed above, which are decided at design time, several methods have been proposed to dynamically allocate buffers and number of VCs at runtime. Several methods have been proposed [25, 21, 7] for dynamic buffer space allocation. For example, the dynamically allocated multi-queue approach [25] uses linklists to allocate VCs to each port. However, to update the logical pointer to the free list, a 3-cycle delay is incurred at every flit arrival/departure, making this method unsuitable for high-performance applications.

- The fully-connected circular buffer approach [21] uses registers to selectively shift flits within buffers. However, this solution requires a large $P^2 \times P$ crossbar instead of a 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 overhead.

- Finally, Nicopolous *et al.* [22] propose a dynamic VC allocation approach called ViChaR in which the number of VCs and the depth of buffers per VC are dynamically adjusted based on the traffic load. However, since there can be as many VCs as there are flit buffers, control logic becomes complicated.

In summary, runtime dynamic allocation of buffers seems more desirable for general-purpose and reconfigurable design platforms that execute different workloads. However, design-time allocation of VCs appears more desirable for application-specific NoCs that are intended to implement a specific application or a limited class of applications. In this paper, we propose efficient VC allocation metaheuristics that can significantly reduce the number of VCs needed to achieve a target performance for a given application.

3. TRACE-DRIVEN VC ALLOCATION

In this section, we propose a new metric called *significant VC failure* (SVCF) to implicitly capture the impact that VC allocation has on network performance. Using this new metric, as well as an existing queueing delay metric, we propose new efficient heuristics that can significantly reduce the number of simulations that are needed in a general trace-driven optimization approach. These heuristics are described in the following three subsections.

3.1 SVCF-Driven VC Allocation

We propose to apply the concept of significant VC failure (SVCF) to implicitly capture the impact of virtual channels on average packet latency. A significant VC failure occurs when a new packet cannot acquire a virtual channel because all virtual channels that use the same output link are already held by packets that are "blocked" from proceeding further by downstream contentions. In this scenario, the output link unnecessarily remains idle until a packet that already holds a virtual channel can proceed.

To give a flavor of what we mean by a significant VC failure, consider the example shown in Figure 2. Suppose we have three packets with the following ($source$, $destination$): (A, F), (B, D), and (E, D), with all three packets 10-flits in size. Packet (E, D) is injected at time 0, and Packets (A, F) and (B, D) are injected at time 1. We assume in this example that links only have a single VC (i.e., wormhole configuration) with 10 flits per VC. Link 2 will carry two packets from (A, F) and (B, D), and Link 3 will also carry two packets from (B, D) and (E, D). We observe that Packet (A, F) is "blocked" from proceeding because Link 2 is held by Packet (B, D) which itself is "blocked" from proceeding since Packet (E, D) has already held Link 3. Note that, as long as Packet (B, D) is blocked, Packet (A, F) is also blocked even though it is heading to a different destination. Only when all 10 flits of Packet (E, D) have traversed Link 3, then Packet (B, D) can proceed; however, Packet (A, F) needs to wait until all flits of Packet (B, D) have traversed Link 2 before it can proceed to its destination, F. However, if we add one VC to Link 2 then Packet (A, F) can bypass Packet (B, D) while Packet (B, D) is being blocked by Packet (E, D). Each time that Packet (A, F) tries to acquire a VC, a *significant* VC failure occurs until Packet (B, D) leaves Link 2. On the other hand, when Packet (B, D) requests for a VC on Link 3, the VC failure is *not* considered significant because Packet (E, D) is using Link 3.

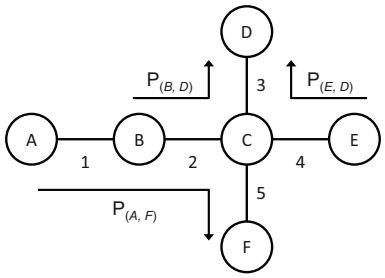

Figure 2: An example of significant VC failure.

Algorithm: SVCF-Driven
Input: Application trace, A_{trace}; Network topology, $T(C, L)$; Deterministic routing algorithm, R; Target latency constraint, D_{target}
Output: VC configuration vector, $\mathbf{n}_{VC}$, which contains the number of VCs associated with each link $l \in L$

1. **for** l = 1 to $|L|$
2. $\quad n_{VC}^{current}[l] = 1$;
3. $N_{VC} = |L|$;
4. **while** $(N_{VC} \leq budget_{VC})$ {
5. $\quad \mathbf{n}_{SVCF}^{current} = ComputeSVCF(\mathbf{n}_{VC}^{current})$;
6. $\quad$ **find** l^* that maximizes $n_{SVCF}[l^*]$;
7. $\quad n_{VC}^{current}[l^*] = n_{VC}^{current}[l^*] + 1$;
8. $\quad N_{VC}$++;
9. }

Figure 3: Significant VC failure-driven VC allocation heuristic.

Figure 3 shows the SVCF-driven VC allocation heuristic. The algorithm initializes every link, l, with one VC; this is equivalent to wormhole routing (Line 2). Hence, the total number of VCs in the network, N_{VC} is initialized to the total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration the significant failures of all the $|L|$ possible links in the current configuration, $\mathbf{n}_{VC}^{current}$, are calculated using $ComputeSVCF$ (Line 5). We use trace simulations to evaluate $ComputeSVCF$. Subsequently, one VC is added to the link with the maximum number of significant failures, $n_{SVCF}[l^*]$. The algorithm stops when the total number of allocated VCs exceeds the VC budget, $budget_{VC}$.

Figure 4 shows the average packet latency of each intermediate configuration using SVCF-driven heuristic and the approach proposed by [13] for two different traces, *ferret* and *blackscholes (bs)* from the PARSEC benchmark suite. We observe that our proposed SVCF-driven heuristic can achieve up to 20% and 23% reduction in number of allocated VCs compared with 160 VCs required for uniform-2VC and 208 VCs required for uniform-3VC configurations, respectively. In addition, we achieve average packet

latency values within 9% of those achieved by [13] while providing a speedup of $O(|L|)$. As network size increases, the method proposed by [13] requires significantly more computing resource, i.e., $O(|L|)$ simultaneous simulations, compared with only 1 simulation per iteration in our proposed SVCF-driven heuristic. We note that our proposed SVCF-driven heuristic cannot reduce the average packet latency as much as the method of [13], and we attribute this to: (1) the fact that we are not directly minimizing average packet latency, and (2) links with highest SVCF count may not have the heaviest traffic load, and thus cannot reduce APL as much.

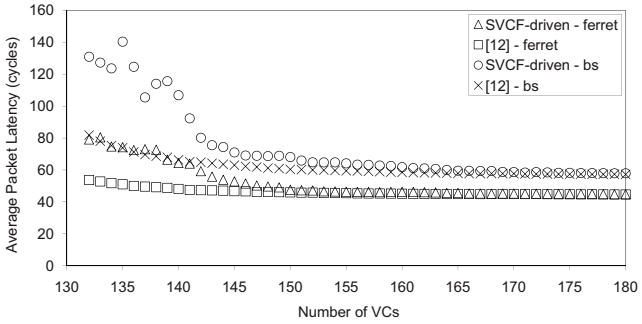

Figure 4: Performance of SVCF-driven VC allocation heuristic on $ferret$ and $blackscholes$ traces.

3.2 Queueing Delay-Driven VC allocation

In this subsection, we describe another simple VC allocation heuristic in which we use the queueing delays observed at each link to drive the VC allocation. In a conventional input-buffered router, queueing delay is the time a flit needs to wait in the buffers before it gets access to its designated output link. The queueing delay of a flit at link l of a router is measured as the difference between the time a flit enters the input buffer of the router until it departs the router through output link l. The queueing delay of a link l is the sum of the queueing delays of all flits passing through link l.

This approach is structurally similar to the SVCF-driven heuristic except that we use queueing delay as the driving metric. However, the queueing delay-driven approach can better capture bottleneck links early in the VC allocation process as shown in Figure 5. We note that our proposed queueing delay-driven heuristic fails to improve average packet latency after certain number of VCs have been allocated. This is because after a few VC allocations the bottleneck shifts from the nodes with heavy traffic to downstream nodes; however, VC failure at downstream nodes causes the queueing delay of the flits residing in the source nodes to increase. Hence, our proposed queueing delay-driven approach will allocate VCs to links in source nodes instead of links in downstream nodes. Figure 5 shows the average packet latency of each intermediate configuration using queueing delay-driven, SVCF-driven heuristic as well as the approach proposed by [13] for *canneal* trace from the PARSEC benchmark suite.

3.3 Top-k Selection Heuristic

In this subsection, we propose an efficient, and yet simple approach for improving the performance of our SVCF- and queueing delay-driven heuristics. The hypothesis is that we can significantly improve the quality of solutions obtained by SVCF- and queueing delay-driven heuristics by simultaneously evaluating more than one of their suggested solutions. In other words, we will evaluate the top-k solutions and determine the best solution based on the maximum reduction in average packet latency.

Figure 6 shows our top-k SVCF-driven VC allocation heuristic. The algorithm initializes every link, l, with one VC (Line 2). Hence, the total number of VCs in the network, N_{VC} is initialized to the

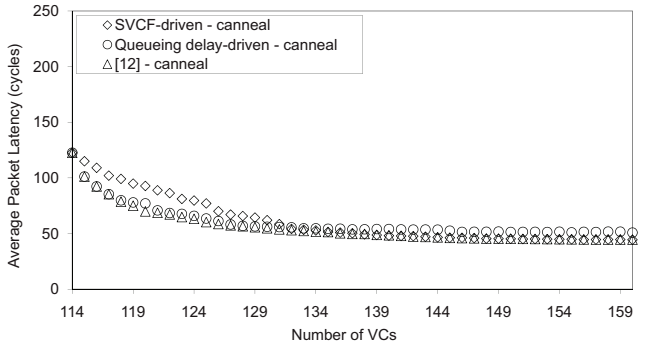

Figure 5: Comparison of SVCF-driven, queue delay-driven, and [13] VC allocation heuristics on $canneal$ trace.

total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration, significant VC failure values corresponding to each link in the current configuration, $\mathbf{n}_{VC}^{current}$, are computed using $ComputeSVCF$, where we use trace simulation to evaluate $ComputeSVCF$ (Line 5). Then, we find the top k links that have the highest number of significant VC failures (Line 6), and add one VC to each of these k configurations (Line 8). Next, we run k parallel trace simulations to evaluate the quality of each configuration, and pick the one that minimizes the average packet latency the most (Line 11). In the shown pseudocode (Figure 6), $ComputeAPL$ performs trace simulation on a given VC configuration and reports the average packet latency for that VC configuration. The algorithm stops when the total number of allocated VCs exceeds $budget_{VC}$.

Algorithm: Top-k SVCF-Driven
Input: Application trace, A_{trace}; Network topology, $T(C, L)$; Deterministic routing algorithm, R; Target latency constraint, D_{target}
Output: VC configuration vector, $\mathbf{n}_{VC}$, which contains the number of VCs associated with each link $l \in L$

1. **for** l = 1 to $|L|$
2. $\quad n_{VC}^{current}[l] = 1$;
3. $N_{VC} = |L|$;
4. **while** $(N_{VC} \leq budget_{VC})$ {
5. $\quad \mathbf{n}_{SVCF}^{current} = ComputeSVCF(\mathbf{n}_{VC}^{current})$;
6. $\quad$ **find** top-k l_i^* corresponding to top-k $n_{SVCF}[l_i^*]$;
7. $\quad$ **for** i=1 to k {
8. $\quad\quad n_{VC}^{current}[l_i^*] = n_{VC}^{current}[l_i^*] + 1$;
9. $\quad\quad APL[i]$=$ComputeAPL(n_{VC}^{current}[l_i^*])$;
10. $\quad$ }
11. $\quad$ **find** m_i^* that minimizes $APL[m_i^*]$
12. $\quad \mathbf{n}_{VC}^{current} = n_{VC}^{current}[l_m^*]$;
13. $\quad N_{VC}$++;
14. }

Figure 6: Top-k significant VC failure-driven VC allocation heuristic.

To determine an appropriate value for k, we perform a simple sensitivity analysis in which we run the heuristic for multiple k values with reduced trace length. In our sensitivity experiments, we are interested in the number of allocated VCs to satisfy an average

packet latency constraint for a given VC configuration. We perform the sensitivity experiments for $k \in \{1, 3, 5, 8, 10, 15\}$ and assess at which point the increase in k does not improve the quality of the solutions. Figure 7 shows the sensitivity analysis results for all PARSEC traces. The x-axis shows the value of k, and the y-axis is number of VCs required to satisfy the average packet latency of a uniform configuration with 2 VCs per link. From Figure 7, we observe that for all of the traces, except for $vips$, k= 5 gives the best tradeoff between quality of results and runtime, i.e., the reduction in average packet latency is within 2% beyond k = 5. For $vips$ trace, k = 10 gives an additional 8% improvement in average packet latency compared with k = 5; however, in our experiments we assume k = 5 for all the PARSEC benchmark traces. Similarly, we have performed sensitivity analysis for our queueing delay-driven heuristic and have determined that k = 15 offers the best tradeoff between quality of the results and runtime. The results presented are for a mesh network with 16 nodes and 64 links (details of the setup are explained in Section 5.1).

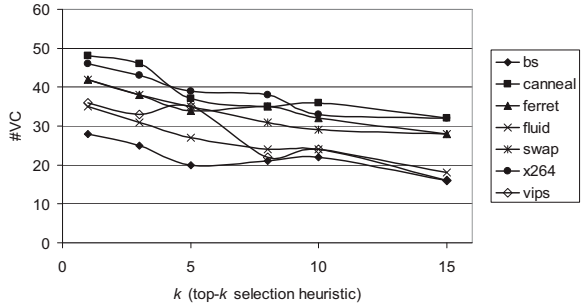

Figure 7: Sensitivity analysis of the k parameter for the PARSEC benchmark traces.

4. EFFICIENT METAHEURISTICS

In this section, we propose two efficient and more robust VC allocation metaheuristics by combining the heuristics described in Section 3.

4.1 Hybrid Metaheuristic

In our proposed hybrid metaheuristic, we combine the top-k SVCF-driven and queueing delay-driven VC allocation heuristics such that in each iteration we pick the best configuration out of these two heuristics. The key to our hybrid metaheuristic is that SVCF-driven and queueing delay-driven VC allocation heuristics each performs quite well in different VC regimes. In other words, the SVCF-driven heuristic seems to perform well once there are already a few VCs inserted in the network, whereas the queueing delay-driven heuristic performs well in the beginning, i.e., where there are only few VCs in the network.

Figure 8 shows our proposed hybrid metaheuristic. The algorithm initializes every link, l, with one VC; this is equivalent to wormhole routing (Line 2). Hence, the total number of VCs in the network, N_{VC} is initialized to the total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration, the significant VC failures associated with all of the $|L|$ possible links in the current configuration, $\mathbf{n}_{VC}^{current}$, are calculated using $ComputeSVCF$ (Line 5). In parallel, we also run $ComputeQD$ on $\mathbf{n}_{VC}^{current}$ to determine the total queueing delay associated with all of the $|L|$ possible links (Line 6). Subsequently, we find the corresponding top-k and top-k' links with the highest number of significant VC failure and queueing delay, respectively (Lines 7-8). Then, we run $k + k'$ parallel trace simulations in which we add one VC to each of the suggested $k + k'$ configurations and evaluate their impact on average packet latency (Lines 9-12 and 13-16). Finally, we pick the VC configuration with the lowest average packet latency and set that as the starting configuration for the next iteration. The algorithm stops when the total number of allocated VCs exceeds the VC budget, $budget_{VC}$.

Algorithm: Hybrid Metaheuristic
Input: Application trace, A_{trace}; Network topology, $T(C, L)$; Deterministic routing algorithm, R; Target latency constraint, D_{target}
Output: VC configuration vector, $\mathbf{n}_{VC}$, which contains the number of VCs associated with each link $l \in L$

1. **for** l = 1 to $|L|$
2. $\quad n_{VC}^{current}[l] = 1$;
4. **while** $(N_{VC} \leq budget_{VC})$ {
5. $\quad \mathbf{n}_{SVCF}^{current} = ComputeSVCF(\mathbf{n}_{VC}^{current})$;
6. $\quad \mathbf{n}_{QD}^{current} = ComputeQD(\mathbf{n}_{VC}^{current})$;
7. $\quad$ **find** top-k l_i^* corresponding to top-k $n_{SVCF}[l_i^*]$;
8. $\quad$ **find** top-k' m_i^* corresponding to top-k $n_{QD}[m_i^*]$;
9. $\quad$ **for** i=1 to k {
10. $\quad\quad n_{VC}^{current}[l_i^*] = n_{VC}^{current}[l_i^*] + 1$;
11. $\quad\quad APL[i]$=$ComputeAPL(n_{VC}^{current}[l_i^*])$;
12. $\quad$ }
13. $\quad$ **for** i=1 to k' {
14. $\quad\quad n_{VC}^{current}[m_i^*] = n_{VC}^{current}[m_i^*] + 1$;
15. $\quad\quad APL[i+k]$=$ComputeAPL(n_{VC}^{current}[m_i^*])$;
16. $\quad$ }
17. $\quad$ **find** n_i^* that minimizes $APL[n_i^*]$
18. $\quad \mathbf{n}_{VC}^{current} = n_{VC}^{current}[l_n^*]$;
19. $\quad N_{VC}$++;
20. }

Figure 8: Hybrid metaheuristic using top-k SVCF-driven and queueing delay-driven VC allocation heuristics.

4.2 Multi-Stage Metaheuristic

From previous subsections, we observe that our queueing delay-driven heuristic performs well starting from an initial configuration (i.e., wormhole configuration), and that our SVCF-driven heuristic performs well when there are already a number of VCs allocated. Knowing this, we propose a 2-stage metaheuristic in which we start with our top-k queueing delay-driven algorithm and will switch to top-k SVCF-driven algorithm once the difference in average packet latency of two consecutive configurations falls below a certain thershold, s.

Figure 9 shows our proposed two-stage metaheuristic which is similar to our hybrid metaheuristic, but uses some constant number less trace simulations with no degradation in results. The algorithm initializes every link, l, with one VC; this is equivalent to wormhole routing (Line 2). Hence, the total number of VCs in the network, N_{VC} is initialized to the total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration the algorithm picks either top-k SVCF-driven or top-k queueing delay-driven heuristic based on the average packet latency improvement threshold, s, which is defined as the difference in average packet la-

tency for two consecutive iterations. Based on our previous findings (cf. Subsection 3.2), we start with queueing delay-driven heuristic first (Lines 7-15). Once, the APL improvement threshold falls below the defined value, the algorithm chooses the top-k SVCF-driven heuristic (Lines 19-26). We use trace simulations to evaluate both $ComputeSVCF$ and $ComputeQD$. The algorithm stops when the total number of allocated VCs exceeds the VC budget, $budget_{VC}$. To find an appropriate value for s, we have performed similar sensitivity analysis as described in Subsection 3.3, and have chosen s=0.5.

Algorithm: Two-Stage Metaheuristic
Input: Application trace, A_{trace}; Network topology, $T(C, L)$; Deterministic routing algorithm, R; Target latency constraint, D_{target}
Output: VC configuration vector, $\mathbf{n}_{VC}$, which contains the number of VCs associated with each link $l \in L$

1. **for** l = 1 to $|L|$
2. $n_{VC}^{current}[l] = 1$;
4. $switch$ = false;
5. **while** $(N_{VC} \leq budget_{VC})$ {
6. **if**(!switch) {
7. $\mathbf{n}_{QD}^{current} = ComputeQD(\mathbf{n}_{VC}^{current})$;
8. **find** top-k m_i^* corresponding to top-k $n_{QD}[m_i^*]$;
9. **for** i=1 to k {
10. $n_{VC}^{current}[l_i^*] = n_{VC}^{current}[l_i^*] + 1$;
11. $APL[i]$=$ComputeAPL(n_{VC}^{current}[l_i^*])$;
12. }
13. **find** n_i^* that minimizes $APL[n_i^*]$
14. $\mathbf{n}_{VC}^{current} = n_{VC}^{current}[l_n^*]$;
15. N_{VC}++;
16. **if**$(APL_{N_{VC}-1} - APL_{N_{VC}} < s)$
17. $switch$=true;
18. }
19. **else** {
20. $\mathbf{n}_{SVCF}^{current} = ComputeSVCF(\mathbf{n}_{VC}^{current})$;
21. **find** top-k' l_i^* corresponding to top-k $n_{SVCF}[l_i^*]$;
22. **for** i=1 to k' {
23. $n_{VC}^{current}[m_i^*] = n_{VC}^{current}[m_i^*] + 1$;
24. $APL[i]$=$ComputeAPL(n_{VC}^{current}[m_i^*])$;
25. }
26. **find** n_i^* that minimizes $APL[n_i^*]$
27. $\mathbf{n}_{VC}^{current} = n_{VC}^{current}[l_n^*]$;
28. N_{VC}++;
29. }
30. }

Figure 9: Two-stage metaheuristic using top-k SVCF-driven and queueing delay-driven VC allocation heuristics.

4.3 Runtime Analysis

As we have noted earlier, simple trace-driven approaches [13] require relatively larger runtime compared with average-rate approaches. In this paper, we focus on defining new metrics (i.e., significant VC failure and queueing delay) that can more efficiently capture the impact of VC allocation on performance. The runtime complexities of our proposed heuristics are as follows.

Let m be the number of VCs added to an initial VC configuration. The runtime of our SVCF- and queueing delay-driven heuristics, $T_{SVCF/QD}$, for any given input trace is expressed as follows:

$$T_{SVCF/QD} = m \times T(tr) \quad (1)$$

where $T(tr)$ is the average runtime of (cycle-accurate, flit-level) trace simulation simulator over all VC configurations explored by the algorithm. The above two heuristics do not require $O(|L|)$ simultaneous simulations as in [13], which results in an $O(|L|)$ speedup.

The runtime for our proposed top-k SVCF- and queueing delay-driven heuristics to insert m VCs is

$$T_{\text{top-}k\ SVCF/\text{top-}k\ QC} = m \times k \times T(tr) \quad (2)$$

which also results in an $O(|L|)$ speedup (since k is a small constant) when compared with the method of [13]. Finally, the runtime of our proposed hybrid and two-stage metaheuristics to insert m VCs are

$$T_{hybrid} = m \times (k + k') \times T(tr) \quad (3)$$

$$T_{two-stage} = r \times k \times T(tr) + (m - r) \times k' \times T(tr) \quad (4)$$

where k and k' are derived using sensitivity analysis as explained earlier (cf. Subsection 4.1), and r denotes the number of iterations that the two-stage metaheuristic chooses the queueing delay-driven heuristic.

5. EVALUATION AND DISCUSSIONS

In this section, we describe our experimental setup, and then we evaluate the performance of our proposed metaheuristics with respect to (1) uniform VC configurations, and (2) simple trace-driven approach recently proposed by [13]. We show that our proposed approach can achieve the same quality of results with significant runtime improvement (i.e., $O(|L|)$).

5.1 Experimental Setup

To determine the average packet latency of a given application trace, we use PopNet [24], a flit-level, cycle accurate on-chip network simulator. PopNet models a typical input-buffered VC router with four pipeline stages. Route computation and VC allocation are performed in the first pipeline stage, followed by switch arbitration, switch traversal and link traversal. The head flit of a packet proceeds through all four stages while the body flits bypass the first pipeline stage and inherit the output port and output VC reserved by the head flit. Non-uniform VC configurations are implemented by individually configuring the number of VCs at each router port. The routers are connected in a two-dimensional mesh topology and dimension-ordered routing is employed where packets are first routed in the X dimension followed by 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.

To evaluate our VC allocation heuristics, we use seven applications from the PARSEC benchmark suite, namely, *canneal*, *blackscholes*, *fluidanimate*, *ferret*, *swaptions*, *vips* and *x264* [3]. These benchmarks are multithreaded programs encompassing diverse application domains and representative of emerging future workloads. The network traffic traces were generated by running these programs on *Virtutech Simics* [33], a full system simulator,

Table 1: Processor configuration for generation of PARSEC benchmark traces.

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

and capturing the program's memory trace. The GEMS toolset [19] runs on top of *Simics* and performs 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. For the purpose of trace collection, the GARNET interconnect model [1] with 8 VCs per link is used.

In all seven traces, every node in the mesh both sends and receives packets. A 4×4 mesh has 48 links and 16 injection ports at each node. The number of VCs for each of these links can be individually configured. To decouple the VC allocation and buffer space allocation problems, we assume that each VC has a fixed buffer length of 10 flits/VC, which is larger than the maximum packet size of the application. We statically allocate 4 VCs to each of the 16 injection ports in our heuristics to ensure that there is no head-of-line blocking and that the performance bottleneck is shifted to the regular links and not the injection ports. Hence, we start with a wormhole configuration with 112 VCs (1 VC/link + 4 VCs/injection port) and set our VC budget to 256 VCs, equivalent to an uniform configuration with 4 VCs at every link. Every iteration of the addition and deletion heuristics creates at most 48 perturbations which are run in parallel on a computational grid.

5.2 Significance Assessment

In this subsection, we present the results of our proposed VC allocation metaheuristics on average packet latency reduction and reduction in number of VCs under a target performance. We compare our proposed approaches with uniform VC allocation, and the simple trace-driven heuristics recently proposed by [13] using seven traces from PARSEC [3] benchmark suites. Figures 10 and 11 show that our proposed 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 with the recently proposed trace-driven approach [13].

Figure 10 shows the number of allocated VCs using our hybrid and two-stage metaheuristics to achieve the same average packet latency as that of a uniform configuration with 2VCs per port (uniform-2VC). We observe that both of our proposed metaheuristics can achieve up to 24.4% reduction in number of allocated VCs. On average, our hybrid and two-stage metaheuristics reduce the number of VCs by around 13.5% and 14.5% across all benchmarks with respect to uniform-2VC and uniform-3VC configurations, respectively. Figure 11 shows the number of VCs required using our algorithms to achieve the same average packet latency as an uniform configuration with 3VCs per port (uniform-3VC). We see high reductions of up to 38% for both hybrid and two-stage metaheuristics, compared to the 208 VCs required by the uniform configuration.

Figures 10 and 11 also show our proposed metaheuristics match the solution quality of the approach recently proposed by [13] while reducing the runtime complexity by $O(|L|)$. Figure 12 shows the number of simulations required by our proposed metaheuristics, versus the approach of [13], to achieve the same average packet latency as uniform-2VC and uniform-3VC. In our experiments, we observe up to 90% reduction in the total number of simulations compared to the approach proposed in [13].

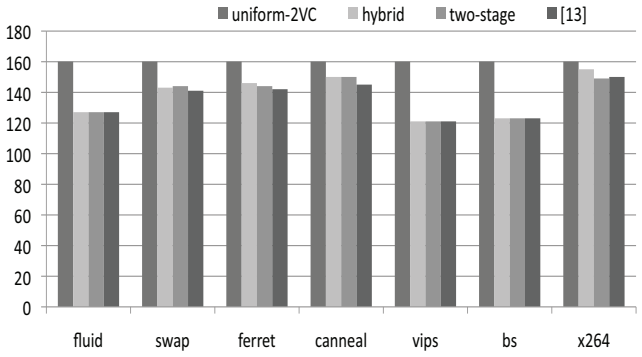

Figure 10: Comparison of hybrid and two-stage VC allocation metaheuristics versus the method of [13], and uniform-2VC configuration.

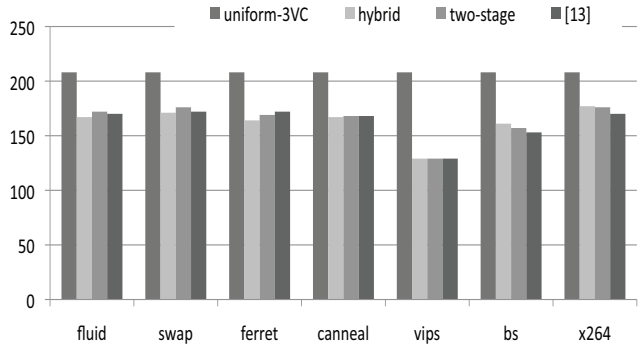

Figure 11: Comparison of hybrid and two-stage VC allocation metaheuristics versus the method of [13], and uniform-3VC configuration.

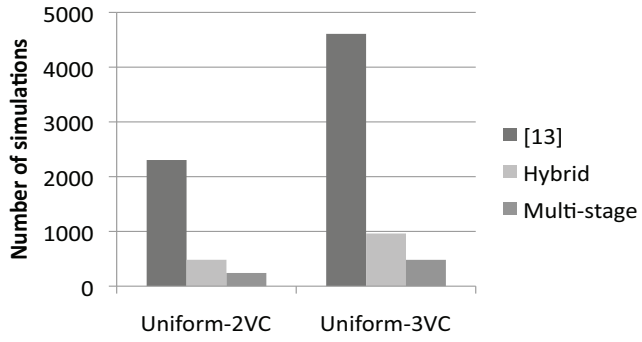

Figure 12: Comparison of number of simulations required for our proposed metaheuristics versus the method of [13].

6. CONCLUSIONS

In this paper, we propose efficient trace-driven metaheuristics for optimization of NoC configurations in today's application-specific MPSoCs. Application traces can accurately capture the temporal variability in network traffic and contention between traffic flows, which is not feasible with previously proposed solutions based on average data rates. In comparison with uniform VC allocation, our heuristics achieve up to 38% reduction in number of VCs. 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. We have also compared our proposed heuristics against a recently proposed trace-driven approach [13] and have achieved similar reductions in total number of VCs while reducing the runtime complexity by $O(|L|)$. Our results clearly show the benefits of an *efficient* trace-driven approach in the optimization of NoC configurations.

7. ACKNOWLEDGMENTS

Support from the MARCO/DARPA Gigascale Systems Research Center and NSF CCF-0811866 is gratefully acknowledged.

8. REFERENCES

[1] N. Agarwal, L.-S. Peh and N. K. Jha, "GARNET: A Detailed Interconnection Network Model Inside a Full-System Simulation Framework", *Technical report CE-P08-001*, Dept. of Electrical Engineering, Princeton University, 2008.

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

[3] 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.

[4] 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.

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

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

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

[8] N. Enright-Jerger and L.-S. Peh, *On-Chip Networks*, Synthesis Lecture, Morgan-Claypool Publishers, 2009.

[9] 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.

[10] 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.

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

[12] 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.

[13] A. B. Kahng, B. Lin, K. Samadi and R. Sunkam Ramanujam, "Trace-Driven Optimization of Networks-on-Chip Configurations", *Proc. DAC*, 2010, pp. 437–442.

[14] 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.

[15] 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.

[16] P. Kongetira et al., "Niagara: A 32-Way Multithreaded SPARC Processor", *IEEE MICRO*, 25(2), 2005, pp.21–29.

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

[18] 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.

[19] 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.

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

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

[22] 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.

[23] D. Pham et al., "The Design and Implementation of a First-Generation Cell Processor", *Proc. ISSCC*, 2005, pp. 184–185.

[24] 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.

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

[26] M. B. Taylor et al., "Evaluation of the Raw Microprocessor: An Exposed- Wire-Delay Architecture for ILP and Streams", *Proc. ISCA*, 2004, pp. 2–13.

[27] 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.

[28] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processsor in 65nm CMOS," *IEEE Journal of Solid State Circuits*, 2008, pp. 29–41.

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

[30] 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.

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

[32] *International Technology Roadmap for Semiconductors*, http://www.itrs.net.

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