A Case for Dynamic Frequency Tuning in On-Chip Networks

Asit K. Mishra[†] Reetuparna Das[†] Soumya Eachempati[†]
Ravi Iyer[§] N. Vijaykrishnan[†] Chita R. Das[†]
[†]Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA-16801, USA
{amishra,rdas,eachempa,vijay,das}@cse.psu.edu
[§]Integrated Platforms Lab
Intel Corporation
Hillsboro, OR 97124, USA
ravishankar.iyer@intel.com

Abstract

Performance and power are the first order design metrics for Network-on-Chips (NoCs) that have become the de-facto standard in providing scalable communication backbones for multicores/CMPs. However, NoCs can be plagued by higher power consumption and degraded throughput if the network and router are not designed properly. Towards this end, this paper proposes a novel router architecture, where we tune the frequency of a router in response to network load to manage both performance and power. We propose three dynamic frequency tuning techniques, *FreqBoost*, *FreqThrtl* and *FreqTune*, targeted at congestion and power management in NoCs. As enablers for these techniques, we exploit Dynamic Voltage and Frequency Scaling (DVFS) and the imbalance in a generic router pipeline through time stealing. Experiments using synthetic workloads on a 8x8 wormhole-switched mesh interconnect show that *FreqBoost* is a better choice for reducing average latency (maximum 40%) while, *FreqThrtl* provides the maximum benefits in terms of power saving and energy delay product (EDP). The *FreqTune* scheme is a better candidate for optimizing both performance and power, achieving on an average 36% reduction in latency, 13% savings in power (up to 24% at high load), and 40% savings (up to 70% at high load) in EDP. With application benchmarks, we observe IPC improvement up to 23% using our design. The performance and power benefits also scale for larger NoCs.

Categories and Subject Descriptors

C.1.2 [**Multiprocessors**]: Interconnection architectures.
C.1.4 [**Parallel Architectures**]: Distributed architectures.

General Terms

Design, Experimentation, Performance.

1. INTRODUCTION

On-chip interconnects or Network-on-Chip (NoC) architectures have become an important research focus in recent years for designing multicores/Chip Multi-Processors (CMPs) and System-on-Chip (SoC) architectures that can scale to hundreds of cores in the future. This is because an on-chip network plays a critical role in determining the performance and power behavior of a multicore/SoC architecture. While the performance implications of the underlying communication architecture is well understood in designing multiprocessors over the years, power consumption has become a first order design metric specifically in the nanometer regime. It is predicted that NoC power can be a significant part of the entire chip power and can account for up to 40 to 60 watts [3] with technology scaling for a mesh based network with 128 nodes. A few commercial designs also support this trend, where up to 28% of the entire chip power is devoted to the interconnect [11]. Thus, on-chip interconnects that can optimize both performance and power pose intriguing research challenges. This is evident from the large body of literature covering multiple facets of NoC design [16, 13, 25, 8, 22, 28].

Router frequency is one of the critical design parameter that directly affects both performance and power, albeit in a contradictory fashion. With a sophisticated design of a router pipeline, it is possible to increase the operating frequency [18, 23], but higher router frequency leads to higher power consumption. On the other hand, a mismatch between processor and router/network frequency can result in significant performance penalties [6]. Thus, a prudent control of the router frequency can help in optimizing both performance and power.

We motivate the fine balance that exists between power and performance in an on-chip network with a relative power-performance trade-off analysis with respect to offered network load. Figure 1 shows the relative growth of network power versus network latency for an 8x8 mesh with a synthetic traffic mixture of Uniform Random, Transpose, Nearest-Neighbor and Self Similar traffic (detail network configuration is mentioned later in Table 3(a)). The bars indicate network latency/power normalized with respect to the network latency/ power at no load (idle network). At low load, the network power consumption is less. However, the rate of growth of network power is much higher as compared to the rate of growth of network latency. For example, as shown in Figure 1, the network power grows to 30x as the injection rate varies from 1% to 40%, whereas the network latency grows only 7x. We leverage our insights from these trends to optimize the network at low load for performance and at high load for power. An activity based power-management technique, which was recently implemented in the Intel 80-core routers [11, 33], shares a similar view of optimizing the network power based on activity, albeit in a different fashion by clock-gating the idle ports.

Since performance and power are directly proportional to frequency, we dynamically modulate the router frequency in response to network load to facilitate these optimizations, and demonstrate the advantages at system level. Specifically, at low load we operate the routers at peak frequency. At high load, we dynamically determine the operating frequency of individual routers in the network. The dynamic schemes that determine the operating frequencies of the routers are designed to a) reduce power consumption and b) manage

MICRO'09, December 12–16, 2009, New York, NY, USA.

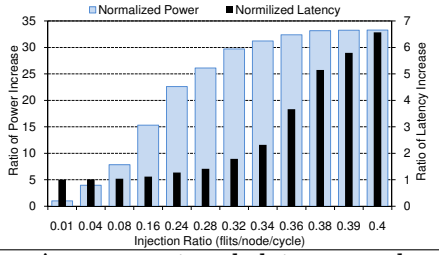


Figure 1: Average network latency and power behavior of an 8x8 mesh network.

congestion in the network, by selectively stepping up and down the frequency of a *subset of routers* in the congested regions of a network. We propose a two-prong approach to vary the baseline router frequency: *clock scaling* and *time-stealing*. We employ Dynamic Voltage and Frequency Scaling (DVFS) [29, 35] to scale up and down the router clock frequency below the nominal frequency by switching the operating voltage levels. The time stealing technique is employed to boost the baseline router frequency by exploiting the timing imbalance between router pipeline stages, such that a router can operate at the average cycle time of all the pipeline stages in contrast to the delay of the worst case pipeline stage.

We explore three techniques for dynamic frequency tuning to simultaneously address power-performance trade-offs. The first technique, called *FreqBoost*, initially employs time-stealing to operate all routers at a boosted frequency. This helps in enhancing the performance at low load conditions, while slightly increasing the power consumption. However, as the network gets congested, power consumption becomes a key challenge. Hence, it throttles the frequency/voltage of selected routers using DVFS. The second mechanism, called *FreqThrtl*, initially operates all routers at the baseline frequency and selectively employs time-stealing and DVFS to either increase or decrease the frequency at the onset of congestion. This scheme, unlike *FreqBoost*, can modulate frequency of routers bi-directionally (higher or lower) and consequently can help reduce power and manage congestion at high load more effectively. Using this technique, the frequency of a congested router is boosted at the onset of congestion and the frequency of a router adjacent to this congested router is throttled. *FreqTune* is a hybrid of the above two schemes that dynamically switches between *FreqBoost* and *FreqThrtl* as the network load varies from low to high.

We evaluate the performance and power implications of the proposed techniques using a wormhole-switched mesh interconnect with synthetic and application benchmarks and compare them with respect to a baseline router/network. To further emphasize the efficacy of our approach, we compare our results with adaptive routing and with a baseline design that employs time-stealing but no congestion management.

The primary contributions of this paper are the following:

• We propose novel frequency tuning algorithms to reduce latency and power consumption in NoC by distributed throttling and boosting of router frequencies depending upon network load. To the best of our knowledge, this is the first work to propose a distributed congestion management scheme that is based on operating individual routers at different frequency levels. Our proposal leads to 36% reduction in latency at high load, 13.5% savings in power (up to 24% at high load) and average 40.5% reduction in energy delay product (EDP) (maximum 70% at high load). With application benchmarks, we achieve IPC improvements up to 23.1% using our schemes. Moreover, the power-performance benefits increase when these techniques are applied to large networks.

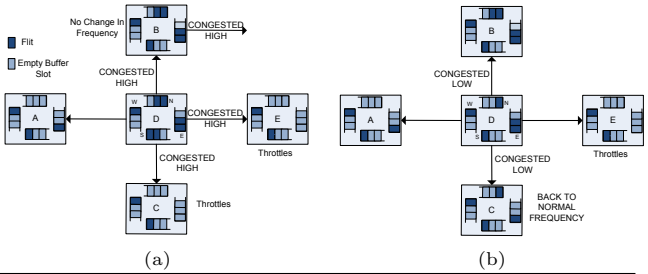


Figure 2: An example showing actions taken by a congested router.

• Our analysis corroborates the pipeline stage imbalance found in other routers proposed in [18, 6, 23]. While this imbalance can be removed using power optimizations such as variable voltage (supply/threshold) and gate sizing optimizations, we focus on time-stealing techniques to *boost* performance. Time-stealing in NoC routers can lead to 25% reduction in zero load latency. In addition, we use the well-known DVFS technique at the granularity of an individual router to tune the router frequency and power based on its load. We believe, this is the first paper to apply time-stealing and DVFS techniques for performance and power management in on-chip routers.

• We demonstrate that the proposed techniques are not only much more effective in delivering better performance and reducing power consumption compared to the monolithic, single frequency design, but also can outperform other pure performance enhancement techniques such as using adaptive routing and simply increasing the operating frequency without any congestion management. Moreover, we also show how coarse-grain frequency tuning can help in reducing the overheads involved in these techniques. All these results make a strong case for implementing variable frequency routers in on-chip interconnects.

The rest of this paper is organized as follows: the three performance and power management techniques are presented in section 2. In section 3, we elaborate on clock scaling and time stealing techniques for deploying these schemes, and the required hardware support. Section 4 discusses the experimental platform and results. Prior work is discussed in section 5, followed by the concluding remarks in section 6.

2. FREQUENCY TUNING RATIONALE

We use a congestion metric (buffer utilization) per port in a router to decide whether this port of the router is likely to get congested in the next few cycles, and if so, it signals the upstream router to throttle. The intuition behind such an approach comes from the fact that if a router is getting congested, it is due to the pressure from its neighboring routers. The congested router is unable to arbitrate and push out its flits fast enough compared to the rate of flit injection into its buffers. To handle this mismatch and reduce the contention in the congested router, we throttle the upstream router by lowering its frequency. This decrease in frequency of the upstream router leads to a lower rate of arrival of the flits into the congested router, giving the congested router some leverage to push out its flits, and hence, reduce the overall blocking latency in the network.

To better understand our proposed techniques, we present an example using Figure 2. Figure 2(a) shows the central (D) router's North (N),South (S) and East (E) port buffers are at the onset of congestion. Thus, router D signals congested port's corresponding upstream routers B,C and E to throt-

Table 1: (a) Settings for $FreqBoost$ and $FreqThrtl$ (b) $Threshold_{throttled}$ Settings for FreqBoost, FreqThrtl and FreqTune. The table shows throttling frequency that a neighboring router uses based on its buffer utilization(BU) after receiving a *congested_high* signal.

F_{base}	F_{boost}
2.20 GHz	2.75 GHz
$Threshold_{congestion}$	$Threshold_{low}$
0.60	0.40

(a)

	$BU > 0.60$	$0.50 < BU < 0.60$	$0.40 < BU < 0.50$	$BU < 0.40$
$FreqBoost$	F_{boost}	$0.9*F_{boost}$	$0.85*F_{boost}$	$0.8*F_{boost}$
$FreqThrtl$	F_{base}	$0.9*F_{base}$	$0.85F_{base}$	$0.8*F_{base}$
$FreqTune$	F_{boost}	$0.85*F_{boost}$	$0.8*F_{boost}$	F_{base}

(b)

tle. Due to similar reasons, router B's East port signals its upstream router to throttle. The upstream routers in turn, depending on their own congestion status, decide whether to throttle themselves or operate at the normal frequency. In the example shown, router B does not throttle itself since it is at the verge of congestion, while routers C and E throttle themselves. A few cycles later, when buffer availability in East and South ports of router D increases, it signals the throttled routers to boost their frequency back to the normal level. This is shown in Figure 2(b). Based on these premises, we design three techniques, *FreqBoost*, *FreqThrtl* and *FreqTune*.

2.1 *FreqBoost* Technique

In the *FreqBoost* technique, we operate the network at a higher frequency, F_{boost} (2.75 GHz), compared to the nominal operating frequency, F_{base} (2.2GHz), right from the beginning using time-stealing technique and apply DVFS to throttle a router for congestion management. This technique and all other techniques proposed later in this paper use two thresholds, $Threshold_{congestion}$ and $Threshold_{low}$ (shown in Table 1(a)), for triggering the proposed schemes. $Threshold_{congestion}$ is used to decide pro-actively whether a particular port is likely to get congested in the next few cycles. Upon detection of such an onset of congestion, this port signals a *congested_high* to the upstream router. We assume that it takes one cycle for this signal to reach the upstream router similar to the credit-flow information. Upon receipt of the *congested_high* signal, the upstream router compares its *overall* buffer utilization with the $Threshold_{throttled}$ settings, shown in Table 1(b), and depending on its overall buffer utilization, the upstream router decides its operating frequency. Table 1(b) shows the different frequency settings for each level of buffer utilization in the router. If the total buffer utilization in the router that received the *congested_high* signal is high, then it will not throttle itself by a big margin, whereas, if the buffer utilization is low, then it would throttle itself aggressively. The reason behind using overall buffer utilization as a metric in choosing the throttled frequency settings and not a port's buffer utilization, is to handle asymmetric traffic, where buffers across some ports are heavily utilized compared to other ports. This scenario arises in X-Y routing, where buffers in the X-direction are heavily utilized compared to buffers in the Y-direction. The $Threshold_{throttled}$ settings ensure that the buffer utilization of a router to be throttled is not close to the congestion threshold, and thereby, the slowed-down router can sustain throttling, without itself getting congested.

When a congested router's buffer utilization goes below $Threshold_{low}$, it signals *congested_low* to its upstream router. After receiving a *congested_low* signal, the throttled router increases its frequency back to F_{boost}.

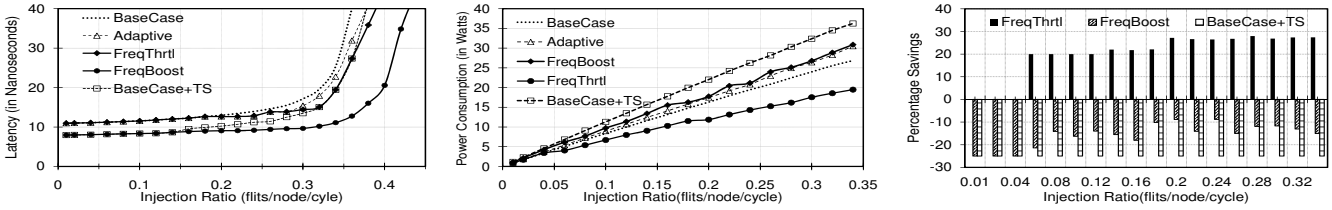
2.2 *FreqThrtl* Technique

With $FreqThrtl$, we increase the frequency of a congested router to F_{boost} only at the onset of congestion and throttle the upstream router to manage congestion, otherwise the base frequency,F_{base} (2.2GHz), is not enhanced during normal operation. Increasing the frequency of the congested router helps in servicing the flits faster through higher rate of arbitration and flit traversal. Additionally, slowing down upstream routers helps to reduce the pressure of injections and helps to ease out the traffic in the congested router.

Note that, we do not change the routing algorithm during congestion, thus, the algorithm does not have to deal with any deadlock situations. Also, monitoring the buffer utilization per port and throttling the corresponding upstream router helps us to manage congestion across traffic flows, i.e. at the onset of congestion in a router, the scheme selectively throttles only those neighboring routers which can lead to congestion in the current router. Additionally, our schemes work on top of the credit-based flow control. With the credit-based flow control, a flit's traversal into the downstream router would pause when there are no available credits. However, with our approach, we pro-actively detect whether the downstream router port is likely to get congested (before all buffer slots get filled up) and slow down the rate of injection into this downstream router, making it different from the credit-based approach, where flit traversal is paused till credit availability. This approach leads to reduction in blocking and queuing latency per flit since each flit now sees reduced contention upon arrival at each input port.

Figure 3 shows the load-latency and power consumption/ savings in an 8x8 network for Uniform Random (UR) traffic. $BaseCase$, in the figures depicts a network, where no time-stealing or DVFS techniques are applied. For comparison purposes, we have also plotted latency curves with a minimally-adaptive routing algorithm (shown as *Adaptive* in the figure) and a case where time-stealing alone (no DVFS) is employed on top of the base case to boost the performance of the network (shown as $BaseCase + TS$). As can be seen, $FreqBoost$ and $FreqThrtl$ increase the throughput of the network at higher injection rates when $BaseCase$ starts saturating. $FreqBoost$ always gives the best performance compared to all of the schemes. With $FreqBoost$, we operate the network at a 25% higher frequency due to time-stealing, and hence, consumes 25% more power at low-injection rates. However, at low injection rates, absolute power consumption in the network is low (less than 7W till 12% injection rate) and $FreqBoost$ does not, therefore, increase the absolute power envelope significantly. At higher injection rates, $FreqBoost$ starts throttling routers to manage congestion and thus, power consumption in the network decreases. The power curves for *FreqBoost* lie in between $BaseCase{+}TS$ and $BaseCase$. Comparison of performance and power curves for $BaseCase{+}TS$ and $FreqBoost$ shows that simply increasing the frequency of a router does not lead to significant performance benefits. Since $FreqBoost$ employs intelligent throttling of routers along with frequency boosting, the performance difference between $FreqBoost$ and $BaseCase{+}TS$ can solely be attributed to our congestion management scheme, where we tune the frequency of individual routers depending on load conditions.

With $FreqThrtl$, we always save power (Figure 3(b)), which



(a) Load-Latency Graph (b) Power Consumption (c) Savings in Power w.r.t. BaseCase

Figure 3: Performance and network power analysis of $FreqBoost$ and $FreqThrtl$ with UR traffic.

comes mostly when few routers under high network load operate at lower than nominal voltage and frequency. However, when compared with $FreqBoost$, $FreqThrtl$ saturates earlier. Therefore, while $FreqBoost$ helps in performance enhancement and consumes more power, $FreqThrtl$ helps in reducing power consumption while providing smaller performance enhancement margins. Since $FreqThrtl$ at low load behaves similar to $BaseCase$ until congestion thresholds are reached, initial power savings with $FreqThrtl$ are negligible until the load increases to about 4% injection rate. On an average, we find about 4.5-5.5 nanoseconds (10-12 cycles) reduction in zero and light load latency and about 40% increase in the throughput of the network with $FreqBoost$. With $FreqThrtl$, there is about 12% increase in network throughput. Average power saving with $FreqThrtl$ is 23%, while $FreqBoost$ consumes on an average 14% more power compared with $BaseCase$. Figure 3(c) depicts that simply boosting the network frequency by 25% ($BaseCase{+}TS$) leads to a consistent 25% more power consumption. The figures also show that both $FreqBoost$ and $FreqThrtl$ outperform adaptive routing in average latency and $FreqThrtl$ provides much better power savings due to congestion management in the network. Our simulations show a similar power and performance trend with other non-uniform traffic patterns as well. Hence, in subsequent sections, we do not present any evaluation results comparing adaptive routing to our techniques to preserve clarity.

2.3 *FreqTune* Technique

Leveraging our insights from the above results, we propose an adaptive technique that utilizes $FreqBoost$ and $FreqThrtl$. It uses $FreqBoost$ at low load for performance enhancement and switches to $FreqThrtl$ at high load to save power and manage congestion. The EDP results shown in later sections reinforce this choice. We call this hybrid technique $FreqTune$, since using this technique, the network dynamically *tunes* to the traffic conditions. With FreqTune, each router selects its operating mode using information from its neighbors. This leads to distributed throttling of routers in the network with a mix of schemes, wherein some regions operate in the nominal frequency mode, some in FreqBoost mode and others operate in FreqThrtl mode. Thus, with $FreqTune$, a router transitions from using high-frequency, F_{boost}, at low load to using nominal frequency, F_{base}, at medium load and again to high frequency at high load if it gets congested. This distributed dynamic frequency transition, based on network load, manages congestion as well as saves power in the network.

3. ROUTER AND NETWORK ARCHITECTURE

In this section, we discuss the enablers for our proposed techniques: on-chip DVFS in routers and time-stealing. We then describe the architectural modifications to the router design for supporting frequency scaling and time stealing techniques, their hardware implementation and the corresponding overheads.

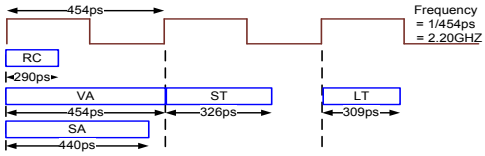
3.1 Frequency Scaling

Our approach extends the concept of per-core on-chip DVFS [17] to per-router DVFS in NoCs for congestion and power management. In order to minimize the overhead of supporting multiple power domains in the network, we also experiment with one regulator being shared among a group of routers. We adopt the two-step voltage regulator configuration as proposed by Kim et al. [17]. An off-chip regulator performs the initial step down from the Li-ion battery (3.7V) to 1.8V followed by multiple on-chip voltage regulators, where each of them steps down voltage from 1.8V to 1V. The 2-level approach amortizes the degradation in conversion efficiency of employing only off-chip regulators. A multi-phase buck converter that can provide three voltage and frequency levels and a programmable voltage controlled ring oscillator [10] are used. The programmable ring oscillator is required for different frequencies at the same voltage to support the time-stealing technique described next. The on-chip regulators operate at 125MHz switching frequency and provide voltage transitions between 1V to 0.8V. The routers can operate at the lower frequency during frequency step-down by speedily ramping down the frequency before the voltage steps down. For stepping-up the frequency using DVFS, we first step-up the voltage before ramping up the router frequency. Thus, the overhead in every transition is primarily the voltage settling time which is 13ns for every 100mV change [17]. Hence, due to higher than required voltage during step-down (and lower frequency during step-up), the power consumed by the router during a transition lies between the values before and after the scaling. All our evaluations take this overhead into account.

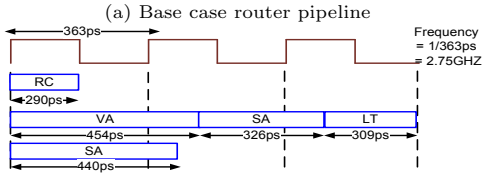
The power consumption (at activity factor of 0.5) for our regulator with conversion efficiencies similar to that in [17] is given in Table 2(a). The area overhead in an 8x8 mesh for 64 on-chip regulators is $4mm^2$ which is about 25% of the area. The average power overhead is around 8% (at 1V) of all routers in the network. This is based on our router area of 0.245 mm^2 and power of 0.35W at 65nm node based on our synthesized design (see Section 3.3.2). This area and power overhead reduces by 4x when we use per-column regulators (described later in Section 4.3).

3.2 Time Stealing in Router Pipeline

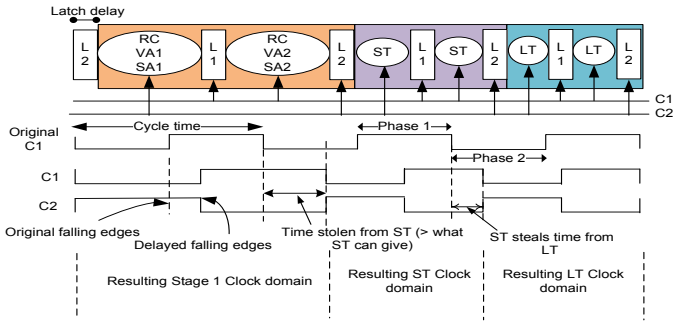
A generic on-chip router pipeline stage delays are quite imbalanced unlike the processor pipeline [6, 26, 24]. In order to boost the router frequency from the nominal frequency, we apply time stealing techniques, where a slower stage in the router gains evaluation time by *stealing* it from successive or previous router pipeline stages. In order to steal slack from a previous pipeline stage, an early clock signal needs to be available and this may be limited by the earliest time a clock signal can be obtained from the clocking system. A cycle may steal time from subsequent stages by delaying the triggering



(a) Base case router pipeline



(b) Time stealing in router pipeline



(c) Time stealing with a 2-phase clocked pipeline

Figure 4: Time stealing in a generic router pipeline with a 2-phase clock.

edge of the clock to all the subsequent latches. We use the later method for our router design.

Our baseline router is a 2-stage speculative router in which the first stage performs the routing computation (RC), the virtual channel allocation (VA) and the switch allocation (SA) in parallel and the second stage is the switch transfer (ST) as shown in the Figure 4(a). The link traversal (LT) stage takes an additional cycle. The router stage delays (shown in Figure 4(a)) are obtained after synthesis using the Synopsys Design Compiler, and the LT stage delay is obtained using wire models from PTM [1]. A non-overlapping symmetric two-phase clock is used for boosting the router frequency (shown as C1 and C2 in 4(c)). In our 2-stage router, since the VA stage (454ps) is the bottleneck, it steals time from the ST stage. Since the time required by the VA stage is greater than the slack available in the ST stage, ST stage will need to steal time from LT as shown in Figure 4(b). Consequently, we delay the active clock edge of C1 and C2 for both first and second stages (shown in Figure 4(c)). Let the new enhanced clock time after time stealing be T_c ($=\frac{T_{VA}+T_{ST}+T_{LT}}{3}$). The active clock edge for the VA stage is delayed by $S_1 = T_{VA} - T_c$. This is the extra time that is required by VA (stolen from ST and LT put together). The slack time ST can provide is $S_2 = T_c - T_{ST}$. The remaining time of $S_1 - S_2$ is stolen from the LT stage by the ST stage. Thus, the clock edge to the ST stage is delayed by $T_{VA} + T_{ST} - 2 * T_c$.

The extra time required to delay the falling edge is introduced using tunable delay buffers comprising of inverter chains [32]. We assume a hierarchical network consisting of an H-tree at the top level and local mesh grids at the lower level for distributing the clock on the chip similar to the Itanium [31] clock-distribution design. A local grid typically constitutes about 1250 flip-flops [9], which is about the size of a single stage of our router. Thus, the clock signal to the router stages are supplied by the distinct local clock grids. Hence, accommodating for time stealing technique will involve introducing the extra delay buffer for the clock grid supplying a router stage. Assuming a two-stage inverter chain for the buffer, the size of the second inverter required for introducing the clock skew in the clock distribution network is shown in Table 2(b). For a delay of 27ps, a single inverter sized to 2.4 times the minimum sized inverter is sufficient. These values were obtained from HSPICE simulations at 65nm, and we find that the power and the area overheads for introducing these extra buffers are minimal.

To study the effectiveness of the time-stealing approach across more diverse VA stage designs and implementation choices, the frequency improvement was evaluated by varying the VA stage delay by ±25% from our design. Further,

Table 2: (a) On chip regulator power consumption and (b) Inverter sizes for introducing clock skew.

Output voltage	Power overhead
1V	52.3mW
0.9V	41.5mW
0.85V	37.9mW
0.8V	34.1mW

(a)

Skew	Inverter Size	Skew Stage @ Freq
91ps	4.75	VA @ 2.75GHz
54ps	2.41	ST @ 2.75GHz
46ps	1.9	VA @ 2.47GHz
27ps	2.41	VA @ 2.34GHz

(b)

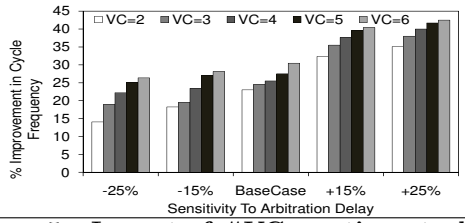


Figure 5: Impact of #VCs on time-stealing.

the number of virtual channels (VCs) is also varied to analyze the percentage benefits with time-stealing due to variation in both VA stage latency as well the number of VCs. Such delay variations are done to mimic changes due to different arbiter designs or circuit optimizations. Figure 5 shows the percentage improvement in cycle frequency using time-stealing technique as the number of VCs vary and as the cycle-time of the VA stage varies from the BaseCase design. There is a 25% frequency improvement using time stealing over the base router frequency (2.2 GHz at 1V and 4VCs). If the delay of the VA stage is reduced by 25%, it no longer becomes the bottleneck stage in the router and the ST stage becomes the delay dominant stage. Under such scenarios, the ST stage steals time from the LT stage. On an average, there is 32% improvement in cycle-time across ±25% variations in VA stage time as the number of VCs change.

3.3 Architectural Support for Frequency Adaptation and Hardware Implementation

3.3.1 Asynchronous Communication

In order for the network routers to operate at different frequencies, they should be able to communicate asynchronously with each other. The router control logic, switching logic and arbitration logic remain unaffected. This architecture essentially *mimics* a Globally Asynchronous Locally Synchronous (GALS) design within the network. To support the communication between routers operating at different frequencies, we utilize the dual clock I/O buffer design from [12] for our router buffers. In this design, the buffers use independent read and write clock signals along with control to prevent synchronization problems when read and write pointers approach each other. In our design, the write clock of a buffer is controlled by the clock of the feeding upstream router and the

read clock is from the current router. When two neighboring routers operate at the same frequency and the read and the write pointers are in separate locations in the buffer, the read and write operations are independent. Two operational situations that could cause problems are when the two address pointers approach each other in the buffer. These situations correspond to buffer emptying and buffer full. Prevention of synchronization problems is handled separately for each of these cases without introducing any delay penalty using additional circuitry as proposed in [12]. Note that, with our proposed distributed frequency tuning schemes, the entire network does not become asynchronous, only neighboring routers that are operating at different frequencies communicate asynchronously. The feasibility of such asynchronous clocking styles has been demonstrated recently. For example, the 80-core prototype chip from Intel [33] uses mesochronous clocking, where there is no control over the clock phase reaching a particular module and data signal is transmitted along with a strobe signal, which is translated by the receiver as an incoming clock. Our proposed distributed frequency tuning techniques would benefit with these kinds of clocking styles for on-chip networks.

3.3.2 Hardware Implementation

All of our proposed techniques require only buffer utilization (BU) as the input. This information is already gathered in conventional on-chip networks for the credit-based flow control. Buffer utilization is a good indicator of network congestion since it is sensitive to network traffic and adapts well to the changes in traffic pattern. Also, to filter out short-term fluctuations in the network traffic and adapt our techniques to handle long-term traffic characteristics, we use exponential weighted average utilization of buffers by combining the current buffer utilization with past buffer utilization to decide the operating points.

The threshold values for triggering the techniques are stored in registers and the comparison operations can be implemented using simple combinational logic. For signaling congestion status to neighbors, we use a 1-bit line that does not add any significant overhead to the already existing backwiring for credit-flow information. All the overheads for voltage and frequency management discussed in Section 3.1 are included in our evaluations. In addition, the overheads for implementing the buffer utilization tracking and control signal generation are obtained through synthesis using the Synopsys Design Compiler. This shows an additional 550 logic gates per router port with a negligible 6 mW power consumption increase in the router. None of these logic gates lie in the critical path of the router pipeline, and hence, the router frequency is not affected.

4. PERFORMANCE EVALUATION

4.1 Experimental Platform

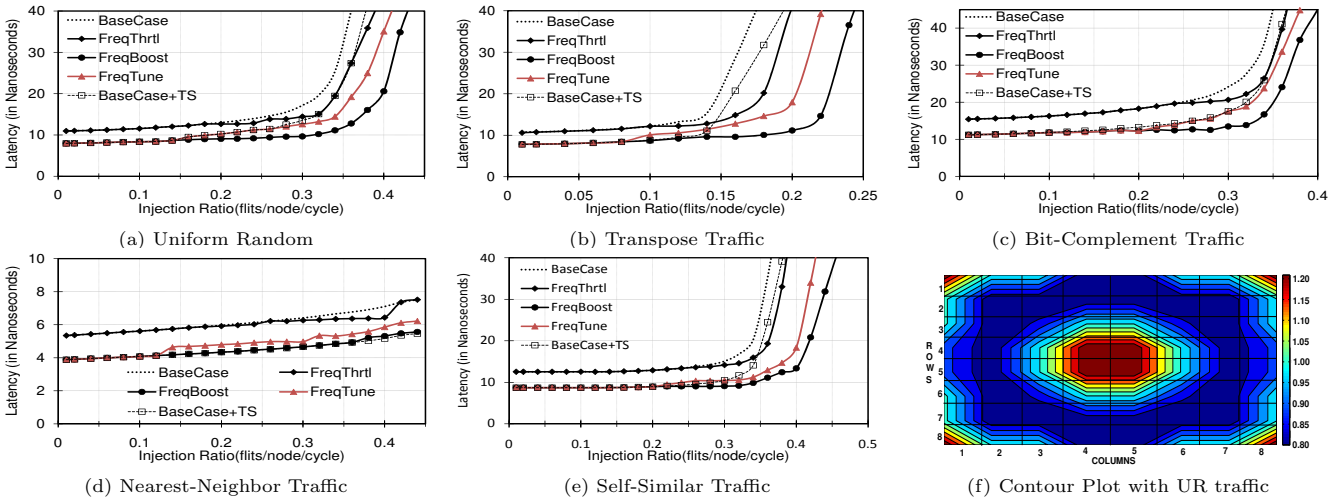
We use a 64-node network as our experimental platform with the network laid out as a 8x8 2D-mesh. We also vary the network size from 8x8 to 16x32 for conducting scalability analysis. We use a cycle-accurate NoC simulator for our simulations and model a state-of-the art two stage router pipeline based on [26]. The base case router has 5 physical channels (PCs) including the local PE-to-router port and 4 virtual channels (VCs) multiplexed on to each PC. A message (packet) consists of six 128-bit flits and we use a buffer depth of 4 flits per VC. We simulate a wormhole-switched network with the deterministic X-Y routing and credit-based flow control. The router along with the proposed modifications, described in Section 3, was implemented in structural RTL Verilog and synthesized using Synopsys Design Compiler using TSMC 90 nm cell library and then scaled down the parameters to 65 nm based on rules given in [4]. The resulting design operates at a clock voltage of 1V and 2.20 GHz (2.75 GHz using time-stealing). The dynamic and leakage power numbers were extracted using Orion [34] and incorporated in our simulator for detailed power analysis of the network.

The network is initially warmed up with 1000 packets and the statistics are collected for 100,000 packets. We measure average flit latency, average power consumption and energy delay product (EDP) across various injection rates for synthetic traffic. The injection rates are with respect to the base frequency and are unbiased to frequency scaling. For synthetic workload, we use Uniform Random (UR), Transpose (TP), Bit-Compliment (BC), Nearest Neighbor (NN) and Self-similar (SS) traffic patterns. For application workloads, we use four commercial workloads and six benchmarks from the PARSEC suite [2] and measure reduction in latency, power savings and IPC improvements when our techniques are applied.

4.2 Results with Synthetic Workload

We compare the latency and power consumption characteristics of the three techniques with the five synthetic traffic patterns. *BaseCase* in our discussion corresponds to the standard design without any frequency tuning and congestion management. *Base Case+TS* represents the case, where time-stealing is used on top of *BaseCase* to boost the frequency of the network, but no congestion management techniques are used. Figures 6(a)-(e) show the load-latency curves with different traffic patterns. We measure the saturation bandwidth when the average latency per flit is three times the zero-load latency [8]. All three of our proposed techniques outperform the *BaseCase* as well as the *BaseCase+TS* network, and as expected, *FreqTune*'s performance envelope lies between *FreqThrtl* and *FreqBoost*. At low load, *FreqTune*'s performance is similar to that of *FreqBoost*, and at high load, *FreqTune*'s performance gets close to *FreqThrtl*. On an average, we find 24% (up to 31%) increase in throughput using *FreqTune* when compared to the *BaseCase*, and up to 21% increase in throughput when compared to the *BaseCase+TS*. For NN traffic in Figure 6(d), the reduction in latency is mostly due to the frequency boosting technique. The congestion management techniques do not play a significant part in latency reduction because of the very nature of the NN traffic, where a particular node sends packets only to its neighboring nodes. Therefore, throttling a neighbor node actually hurts latency since the throttled node will eject flits to the local PE at a reduced rate.

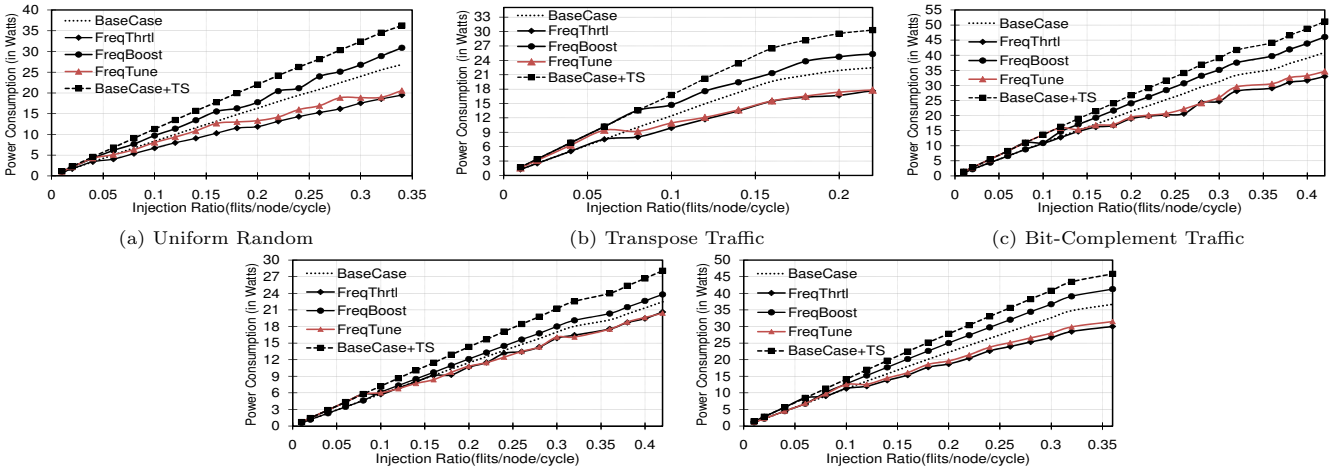
As the network load increases leading to an onset of congestion, *FreqTune* switches to the *FreqThrtl* mode, where the frequency of a congested router is boosted and the neighboring routers are throttled. The congested router can thus service its flits at a faster rate compared to the rate at which it is receiving flits from neighbors. At high injection rates, while some of the routers are congested, some routers are still un-congested. Thus, using *FreqTune*, the congested regions use *FreqBoost* and the un-congested regions use *FreqThrtl*. Figure 6(f) shows a contour plot of a snapshot with UR traffic depicting relative operating frequencies of the routers compared to *BaseCase* at high load (0.36 injection rate). The



(a) Uniform Random (b) Transpose Traffic (c) Bit-Complement Traffic

(d) Nearest-Neighbor Traffic (e) Self-Similar Traffic (f) Contour Plot with UR traffic

Figure 6: Performance with synthetic traffic and a snapshot of relative frequencies of routers at high load.



(a) Uniform Random (b) Transpose Traffic (c) Bit-Complement Traffic

(d) Nearest-Neighbor Traffic (e) Self-Similar Traffic

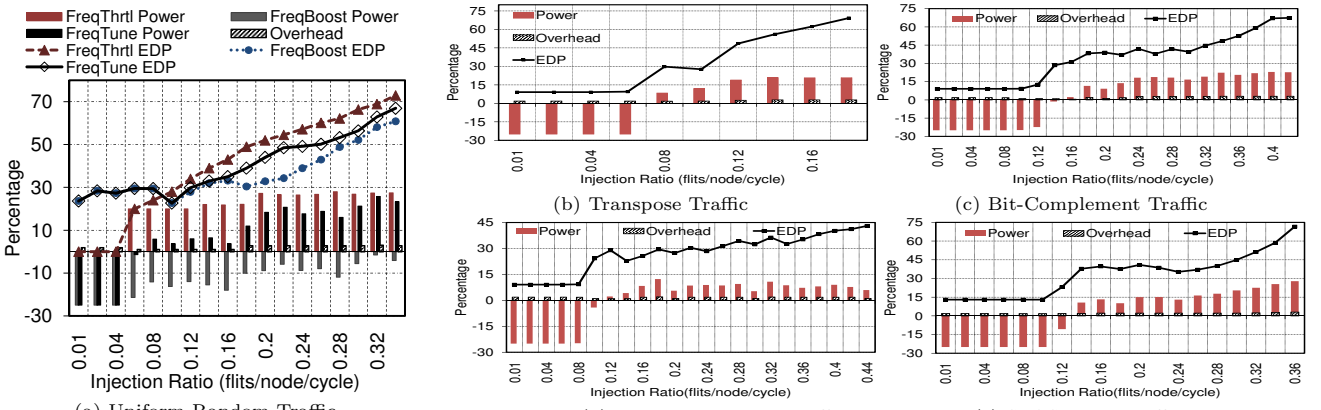
Figure 7: Network power consumption with synthetic traffic patterns.

frequencies range from the lowest possible value ($0.8*F_{base}$), shown as deep-blue color in the plot, to the highest possible value (F_{boost}), shown as dark-red color. Since the routers in the center of a mesh are more congested with X-Y traffic, they operate mostly at F_{boost} ($FreqThrtl$), while the routers around the center operate on an average at lower frequencies. The routers at the periphery are less congested, and hence, operate at F_{boost} ($FreqBoost$). This contour plot demonstrates the flexibility and the distributed frequency modulation capability of $FreqTune$ in adapting to network congestion. Comparison of all the three proposed techniques with $BaseCase+TS$ shows the benefits of a prudent congestion management scheme through frequency tuning when compared to a simple increase in frequency of the network.

Figures 7(a)-(e) show the absolute power consumption plots for different traffic patterns. *FreqBoost* does not give any power benefit since it starts at a 25% higher frequency and gradually reduces the frequency based on the congestion scenarios. At low injection rates, *FreqTune* exhibits identical power behavior as that of *FreqBoost*, and as the load increases, power saving becomes positive with an average reduction of 14.5%. The power and latency overheads of the controllers are included in all of our power-reduction plots and as is evident from the plots, the benefits due the congestion management using frequency tuning are significant compared to the overheads incurred.

Figures 8(a)-(e) show the percentage reduction in power and EDP. Figure 8(a) shows the savings in power and EDP with the three techniques for UR traffic. In addition, the overhead (in terms of percentage network power w.r.t. BaseCase) due to voltage-frequency transitions of the additional controllers in the network for congestion management is shown. In terms of the EDP metric, *FreqThrtl* provides no gain at light load but at higher load it provides the maximum EDP benefit. Both *FreqBoost* and *FreqTune* start with more than 20% saving in light traffic conditions primarily due to latency reduction. At higher network traffic, when buffer utilization increases beyond the specified thresholds, $FreqTune$ starts distributed throttling of routers using the $FreqThrtl$ technique, leading to reduction in EDP. By throttling the upstream router, we lower peak power around hot-spots as the crossbar traversal in the upstream router, traversal in the link to the downstream router and buffer write in the downstream router are all throttled. At medium load, overall buffer utilization lies in the intervals 0.55 to 0.35 and $FreqTune$ dynamically transitions between $FreqThrtl$ and $FreqBoost$. This effect is seen as fluctuations in EDP reduction during medium load in the network.

Similar trends in power and EDP savings are also observed with other traffic patterns as shown in Figures 8 (b) -(e). We have not included the power and EDP results for *FreqBoost* and *FreqThrtl* in these graphs for clarity. Across all traffic



(a) Uniform Random Traffic

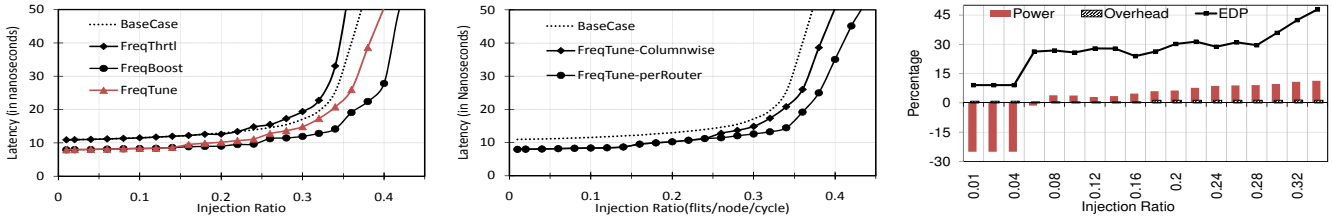
(b) Transpose Traffic

(c) Bit-Complement Traffic

(d) Nearest-Neighbor Traffic

(e) Self-Similar Traffic

Figure 8: Power and EDP reduction with synthetic traffic for *FreqTune* and controller power overheads.



(a)UR Performance with Column-wise *Freqtune*

(b) Comparison of per-router DVFS vs Column-wise DVFS

(c) UR Power with Column-wise *FreqTune*

Figure 9: UR traffic performance using Column-wise controllers.

patterns, we find the average power savings in the network to be 13.5% (up to 24% at high load) with *FreqTune* compared to the *BaseCase*. Average reduction in EDP using FreqTune is 40.5% (up to 70% at high load). Overall, *FreqTune* is able to satisfy the two requirements we discussed earlier - reduce latency at low load and conserve power at high load. However, depending on the specific requirements of a system, a designer can choose to implement any of the other two techniques if only performance enhancement or power conservation is a concern.

4.3 Results with Column-wise Controllers

To further reduce the area and power overheads in having a per-router DVFS controller, we investigate the performance and power behavior by reducing number of DVFS controllers. We chose a column-wise scheme since our evaluations show that such a scheme is quite agnostic toward many routing algorithms (e.g. adaptive, X-Y,Y-X routing). A network designer can choose many other schemes, e.g. having one controller for the routers in the center of a mesh network that have similar utilization with X-Y traffic (as is evident from the contour plot in Figure 6(f)) or have one controller for routers around a hot-spot region (e.g. around a memory controller), etc. In this work, we reduce the number of controllers from one per router to one controller per half-column, i.e. we divide a column in the network into two halves and assign a controller to each half. A controller in a particular half now modulates the frequency of all the routers in half of the column. This reduces the performance gains compared to the fine-grained frequency modulation. We experiment with the 8x8 mesh with 16 controllers (2 controller per column), reducing the area and power overheads by 4x compared to the per-router schemes.

Figure 9(a) shows that *FreqThrtl* performs worse than the *BaseCase* with column-wise controllers. This is because of the coarser granularity of frequency adjustments and adaptivity. However, we find that *FreqBoost*'s performance is not affected as much because of using high-frequency routers and thus, *FreqTune*'s latency curve is still better than the *BaseCase*. Figure 9(b) shows how latency of *FreqTune* is affected when using column-wise controllers compared to *FreqTune* with per-router controller. Figure 9(c) shows the percentage reduction in power and EDP with *FreqTune* using column-wise controllers over the *BaseCase*. With column-wise controllers, there is on an average 12% reduction in latency, 13% savings in network power and 27% reduction in EDP with *FreqTune*. Hence, although congestion management with column-wise controllers now happens at a much coarser granularity, we still save in power and EDP without sacrificing performance.

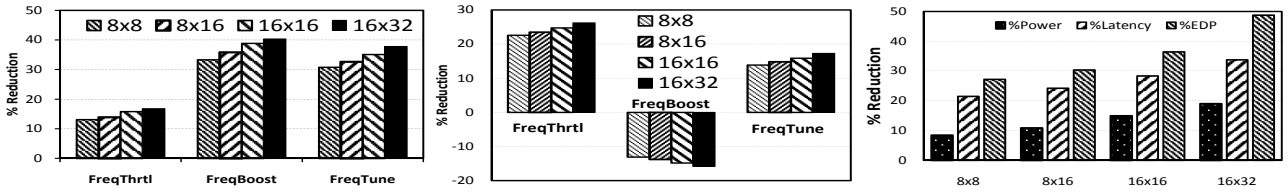
4.4 Sensitivity Analysis

4.4.1 Network Scaling Analysis

Figures 10(a)-(c) show the behavior of our techniques as the network size scales from 64 (8x8) to 512 (16x32) nodes. As the network size increases, all the three techniques get greater flexibility in terms of the number of routers that can be modulated when adapting to load in the network. This leads to even higher percentage reduction in latency, power and EDP with network size. For *FreqTune*, the latency savings increase from 30% to 38% and power savings increase from 13.5% to 18.2% as the network scales to 512 nodes. Even with the column-wise DVFS scheme, the percentage reductions in three metrics increases as network size scales.

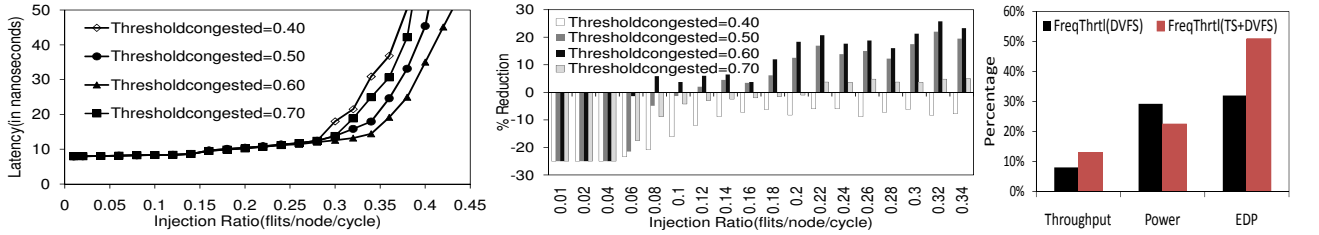
4.4.2 Sensitivity to Threshold

Next, the sensitivity of our techniques to specified thresholds is analyzed. Sensitivity results are shown only for the *FreqTune* scheme with UR traffic for brevity. The congestion thresholds depend on buffer utilization, and hence, are sensitive to the number of virtual channels and buffer depth. Figure 11(a) shows the load-latency curves and 11(b) shows the percentages in power savings compared to the it BaseCase for the it FreqTune scheme, where the threshold to trigger the technique varies across the plots. With a



(a) Latency (per router controller) (b) Power (per router controller) (c) Column-wise

Figure 10: Network scaling results for *FreqTune* with UR traffic.



(a) Load-Latency graph with UR traffic (b) Savings in power with UR traffic (c) FreqThrtl using DVFS only

Figure 11: Sensitivity of *FreqTune* to thresholds and results with DVFS only.

Table 3: CPU, Cache, Network and Workloads Configuration.

CPU and network configuration
Processor Pipeline: 64 x86 based 2.2 GHz (nominal) processors, two-way out of order, 64-entry instruction window
L1 Caches: 16 KB per-core(private), 4-way set associative, 128B block size, 2-cycle latency, split I/D caches
L2 Caches: 1MB banks (per-core),shared, 16-way set associative, 128B block size, 6-cycles latency, 32 MSHRs
Main Memory: 4GB DRAM,up to 16 outstanding requests for each processor, 400 cycle access, 4 memory controllers
Network and Router: 8x8 mesh network(each tile consists of 64 CPUs and 64 L2 banks), 2-stage wormhole switched router, X-Y routing, 4 VCs per port, buffer depth=5, 1024 maximum packet size(8 flits/packet), 2.2 GHz base frequency, 2.75 GHz boosted frequency

(a)

Commercial workloads
App. Benchmark(sap): SAP stands for Standard Application Benchmark and represents a sales and distribution workload. **ServerWorkload(specjbb)**: SPECjbb2000 is a Java based benchmark that models a 3-tier system. We simulated 64 warehouses on 64 processors and start measurements 30 seconds after ramp-up time. **Transaction Processing(tpcc)**: TPC-C is an OLTP benchmark.TPC-C simulates a complete computing environment where a population of users executes transactions against a database. **SPEC Java App. Server(sjas)**: SJAS is a multi-tier benchmark for measuring the performance of J2EE technology-based application servers. The traces for TPC-C, SAP and SJAS benchmark were collected from a CMP server configurations at Intel Corporation and we use 64 threads from each benchmark.

(b)

PARSEC
PARSEC suite includes emerging RMS applications as well as large scale multi-threaded programs for CMPs. From this suite we choose three application benchmarks, **ferret (frrt), facesim (fsim) and vips**, and three kernel benchmarks, **canneal (canl), dedup (ddup) and streamcluster (sclst)**, and ran them with simlarge input sets. Traces were collected on a CMP platform(see config. in (a)) using Simics for 64 threads of each benchmark for 20 million L2 references and then simulated in our cycle-accurate processor-cache-network simulator.

(c)

more aggressive threshold of $Threshold_{congested} = 0.4$ and $Threshold_{low} = 0.25$, performance takes a hit. Correspondingly, percentage savings in power is also reduced. This is because, with a lower threshold, the technique is triggered earlier and starts throttling the routers even before there is a likelihood of congestion. Hence, latency of the flits is increased and flits spend more time in the network leading to increased power consumption. With a more relaxed threshold of $Threshold_{congested} = 0.7$ and $Threshold_{low} = 0.4$, performance again takes a hit, since in this case, the triggering of the congestion management schemes is delayed and this leads to network saturation. The optimal threshold values for our experiments are shown in Table 1.

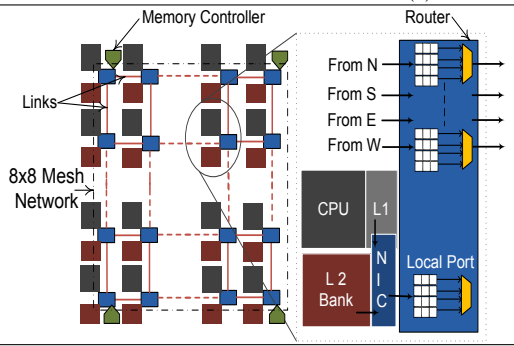
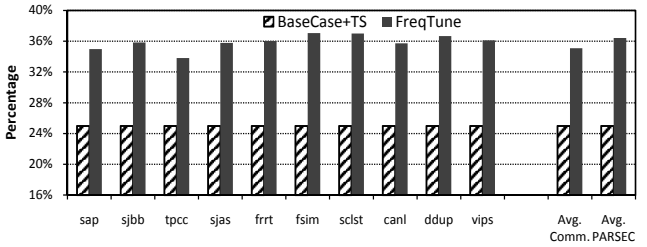


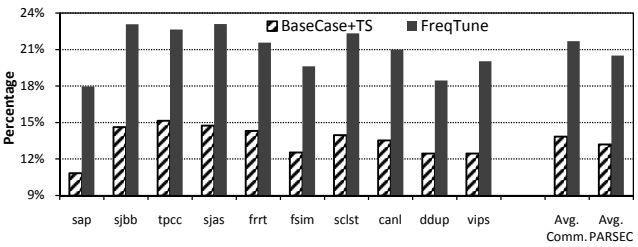
Figure 12: CMP layout.

4.4.3 *FreqThrtl* Using DVFS Only

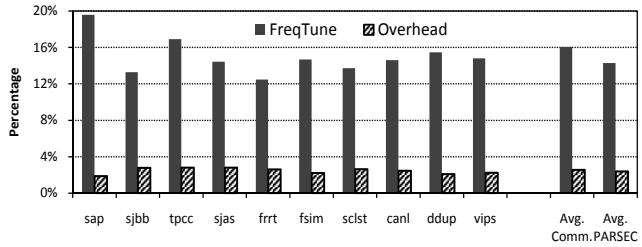
We also analyze the case when a router cannot support time-stealing to boost its frequency (to 2.75GHz). Such a scenario might arise if the pipeline stages in a router are balanced using circuit optimizations, and hence, there is no further scope for time-stealing. In this case, we can still use *FreqThrtl* and can use DVFS knobs to scale the voltage and thereby the frequency of the upstream routers. Figure 11(c) shows the improvement in throughput, reduction in power and EDP when using *FreqThrtl* with DVFS only compared to the *BaseCase* (1V, 2.2GHz) for UR traffic averaged over injection rates. We also show the results for the original *FreqThrtl* scheme that uses both DVFS and time-stealing. For *FreqThrtl* with DVFS only, we use the highest voltage/frequency knob (1V, 2.2GHz) for nominal operation and during congestion, throttle the neighboring routers to the second highest voltage/ frequency setting (1V, 1.98GHz). Even with these constraints, on an average, there is 8% increase in throughput, 28% savings in power and 33.2% reduction in EDP. All these benefits are purely due to congestion management in the network, which reinforces the advantages of frequency modulation.



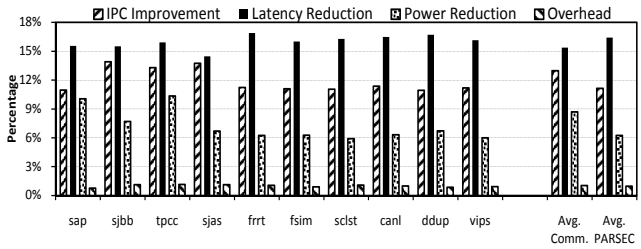
(a)Percentage reduction in latency



(b) Improvement in IPC



(c) Controller overheads and percentage reduction in power



(d)Application performance for Column-wise scheme with *FreqTune*

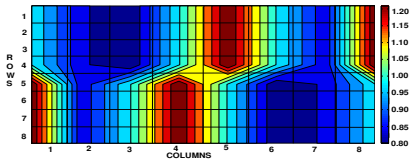
Figure 13: Results with commercial and PARSEC benchmarks.

4.5 Results with Application Benchmarks

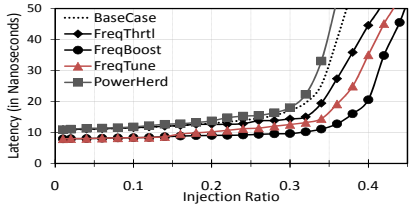
We examine the impact of our frequency tuning schemes on application performance for a 64-tile CMP. Each tile in the CMP consists of a core with a private write-back L1 cache and a L2 cache bank. The memory hierarchy uses a two-level directory-based MESI cache coherence protocol and the network connects the cores, L2 cache banks, and memory controllers. All requests and responses are faithfully modeled by the network. The CPU, network configurations and workload details are given in Table 3 along with the CMP layout in Figure 12. We collected traces using Simics [21] and then used our cycle accurate network simulator with a x86-based processor model as described in Table 3(a) for system level analysis. For our experiments, we chose four commercial benchmarks and six workloads from the PARSEC suite [2].

Figures 13(a)-(d) show the results with the application suite. Having a faster network with congestion management, reduces the average load/store latency and directly impacts system performance in terms of instructions per cycle (IPC). On an average, there is 35.7% (up to 37.16% with *facesim*) reduction in latency (shown in Figure 13(a)), which translates to 21.2% (up to 23.1% for *sjas*) improvement in IPC (Figure 13(b)) across all benchmarks. Correspondingly, there is on an average 15.17% (up to 19.57% for *sap*) reduction in network power (Figure 13(c)) for these applications. We find the IPC improvements to be greater for commercial benchmarks (21.70%) compared to PARSEC benchmarks (20.50%) because of the bursty characteristic and network-latency critical nature of the commercial workloads. Figure 13(a), also, shows the reduction in network latency when using a brute force frequency boosting approach (*BaseCase+TS*). The difference in benefits with *FreqTune* and *BaseCase+TS* scheme is solely due to congestion management offered by our scheme. On an average, 10.8% latency difference between *FreqTune* and *BaseCase+TS* schemes is due to the congestion management technique employed in the former.

Figure 13(c) shows the overhead due to the controllers we deploy in the network for the *FreqTune* scheme.*BaseCase +TS* is omitted in this plot since it always consumes 25% more power, and thus, incurs higher network power. We find that with a modest 2.4% increase in power overhead, we get up to 23.1% increase in IPC. Further, the overheads in power



(a) Contour Plot



(b) Comparison with PowerHerd (global power budget=33W)

Figure 14: (a) Contour Plot of a snapshot during SPECjbb's operation with Column-wise Scheme and (b) Comparison with PowerHerd.

consumption due to the controllers are counter-balanced by the 15.17% savings in network power.

Even with the column-wise scheme, there is on an average 16% reduction in latency, translating to 12% improvement in IPC and 7.23% reduction in power (Figure 13(d)). Similar to Figure 6(f), Figure 14(a) shows the contour plot of router frequencies with the column-wise *FreqTune* in an 8x8 network with the SPECjbb benchmark. Since we use the column-wise scheme, all the routers in a half-column have the same average frequency and the routers around a congested router (shown as dark-red color in the contour plot) operate at a lower frequency (blue color), demonstrating that the routers and the entire network work in concert as per the *FreqTune* mechanism.

Finally, we compare our proposed techniques with PowerHerd [28] in Figure 14(b), which handles peak power management by distributed throttling of flits as opposed to our design of distributed throttling/boosting of router frequencies. Since PowerHerd is aimed only at peak power management, it suffers in performance compared to even the *BaseCase* scenario. For this comparison, we use a similar experimental environment as used for the PowerHerd work, i.e. we use an 8x8 torus network with a global power constraint of 33W and our area and power numbers are scaled

to 110nm. On an average, *FreqTune* reduces the latency by 38.5% compared to PowerHerd with UR traffic and also stays within the global power constraint. In addition, we did a quantitative comparison of our network congestion management scheme, *FreqTune*, with a recently proposed congestion management scheme, RCA [8], where the congestion information is propagated across the network to improve the ability of adaptive routers to spread network load. For comparisons with RCA, we used a simulation setting similar to the one used in RCA, i.e. 8 VCs with 5 flits per VC, 1-6 flits/packet. Although the comparison results with RCA are not included here due to space limitations, our scheme provides better performance (13.6% less latency) than RCA in addition to managing power since RCA is only aimed at congestion management and our proposed techniques simultaneously manage performance and power.

5. RELATED WORK

We summarize the prior work in two sub-sections: on-chip networks and frequency scaling techniques.

5.1 On-Chip Networks

Network-on-Chip is widely viewed as a de-facto solution to wire-delay problems with future technology scaling. However, due to the resource constrained nature of an NoC substrate, most researches have focused on two major themes; improving the performance and reducing power consumption. For improving performance, researches have proposed the use of virtual channels and path speculation [26], look-ahead routing [7], smart pipelines [23] and dynamic traffic distribution [8, 15, 14] to reduce contention, improve throughput and provide fault-tolerance. Our approach is different from all these works since, we use dynamic frequency modulation of routers to manage congestion and improve throughput. Recently, Moscibroda et.al. show that a bufferless routing approach [22] leads to lower power consumption without significantly sacrificing performance in on-chip networks. However, their proposed bufferless routing schemes delivers reasonable performance only when the injection rate into the network is low. On the other hand, our proposed techniques adapts to the load in the network to maximize performance and minimize power at low as well as high injection rates. Hence, our scheme is applicable to a more general purpose on-chip environment.

An activity based power management scheme was recently implemented in the routers of the Intel 80-core chip [11]. For this chip, power reduction was achieved by de-activating ports in the on-chip router and putting individual queues in the ports to sleep or clock-gating them based on activity inside the router. Ours approach is orthogonal to this idea and can be implemented on top of this design for performance and power management as well.

5.2 DVFS and Time-Stealing Techniques

Several works have proposed DVFS in processors [5, 35, 27] for power management. Recently, work done by Kim et.al [17], has shown the possibility of on-chip voltage regulators. Prior works have proposed DVFS for links [29, 12] to manage power in off-chip networks. In PowerHerd [28], throttling a flit traversal in a router has been shown as a means to manage peak power constraints in off-chip networks. Also, in ThermalHerd [30], a collaborative run-time thermal management scheme is proposed for on-chip networks that uses distributed throttling and thermal correlation based routing to tackle thermal emergencies.

Time-stealing is a technique that has been widely used in many works to handle Process Variation [20, 32, 19]. Our, work on the other hand, extends the concept of time-stealing to exploit the imbalance in router pipeline stages to overclock the router at frequencies above the nominal frequency.

Our proposed schemes are quite different from all related works discussed here in the sense that we propose an adaptive frequency router which adapts to load around its vicinity. We do distributed throttling of routers by frequency tuning in on-chip networks to manage power and optimize performance. Moreover, our proposed schemes manage congestion in the network, and thereby, give additional throughput gain. To the best of our knowledge, no prior research has proposed any technique to tune a router's frequency dynamically and adapt to the load conditions.

6. CONCLUSIONS

Since both high performance and low power consumption are essential in designing on-chip interconnects for CMP/SoC architectures, in this paper, we propose a variable frequency router architecture for dynamically controlling the performance and power behavior of on-chip interconnects by effective congestion management. Towards this end, we propose three dynamic congestion management techniques, called *FreqBoost*, *FreqThrtl* and *FreqTune*, show their implementation details, and conduct a comprehensive experimental evaluation using synthetic as well as application benchmarks to show the efficacy of our proposal. *FreqBoost* gives the highest performance throughout but consumes more power at low load. *FreqThrtl* gives the best power behavior across all injection rates. *FreqTune* is a hybrid technique that uses *FreqBoost* at low load to enhance performance and *FreqThrtl* during high load to minimize network congestion and power consumption. The frequency tuning techniques are enabled by clock scaling through DVFS and frequency boosting through time stealing.

The novelty of these schemes are that they can be applied independently for optimizing either performance (through *FreqBoost*) or power (through *FreqThrtl*) or both. In the absence of any time-stealing approach due to balanced router pipeline design, one can still use DVFS for congestion management and get benefits both in performance and power. Furthermore, these techniques are more effective than using adaptive routing, simple router frequency scaling for performance enhancement, and peak power reduction techniques. We believe all these results make a strong case for using variable frequency routers for future on-chip networks.

7. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their reviews and comments in improving this paper. This work is supported in part by National Science Foundation (NSF) grants CCF-0702617, CNS-0916887 and CCF-0903432.

8. References

[1] 65 nm PTM Technology Model, http://www.eas.asu.edu/ ptm/.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[3] S. Borkar. Networks for Multi-core Chips:A contrarian view. In *Special Session at ISLPED 2007*.

[4] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, 19(4):23–29, 1999.

[5] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *7th Intl. Symp. High Performance Computer Architecture*, 2001.

[6] R. Das, A. K. Mishra, C. Nicopoulus, D. Park, V. Narayanan, R. Iyer, M. S. Yousif, and C. R. Das. Performance and Power Optimization Through Data Compression in Network - on -Chip Architectures. In *Proceedings of the 14th Intl. Symp. on High-Performance Computer Architecture*, 2008.

[7] M. Galles. Scalable Pipelined Interconnect for Distributed Endpoint routing: The SGI SPIDER Chip. In *Symposium on High Performance Interconnects (Hot Interconnects)*, pages 141–146, 1996.

[8] P. Gratz, B. Grot, and S. Keckler. Regional Congestion Awareness for Load Balance in Networks-on-Chip. In *Proceedings of the 14th International Symposium on High-Performance Computer Architecture (HPCA)*, February 2008.

[9] M. Hashimoto, T. Yamamoto, and H. Onodera. Statistical Analysis of Clock Skew Variation in H-Tree Structure. In *ISQED '05: Proceedings of the 6th International Symposium on Quality of Electronic Design*, pages 402–407. IEEE Computer Society, 2005.

[10] K. Hausman, G. Gaudenzi, J. Mosley, and S. Tempest. US Patent 4978927 - Programmable Voltage Controlled Ring Oscillator. 1990.

[11] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz Mesh Interconnect for a Teraflops Processor. volume 27, pages 51–61, Sept.-Oct. 2007.

[12] E.-J. Kim, G. Link, K. H. Yum, V. Narayanan, M. Kandemir, M. J. Irwin, and C. Das. A Holistic Approach to Designing Energy-Efficient Cluster Interconnects. In *IEEE Trans. on Computers*, volume 54, pages 660–671, June 2005.

[13] J. Kim, W. J. Dally, S. Scott, , and D. Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. In *35th International Symposium on Computer Architecture (ISCA)*, 2008.

[14] J. Kim, D. Park, C. Nicopolous, N. Vijaykrishnan, and C. R. Das. Design and Analysis of an NoC Architecture from Performance, Reliability and Energy Perspective. In *ANCS'05: Proceedings of the 2005 Symp. on Arch. for Networking and Comm. Systems*, 2005.

[15] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das. A Low Latency Router Supporting Adaptivity for On-Chip Router. In *42nd Design Automation Conference (DAC)*, 2005.

[16] M. M. Kim, J. D. Davis, M. Oskin, and T. Austin. Polymorphic On-Chip Networks. In *Proc. of the 35th International Symposium on Computer Architecture, ISCA*, 2008.

[17] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators. In *Proceedings of the 14th International Symposium on High-Performance Computer Architecture (HPCA)*, February 2008.

[18] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. K. Jha. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *25th International Conference on Computer Design (ICCD)*, 2007.

[19] X. Liang and D. Brooks. Mitigating the Impact of Process Variations on CPU Register File and Execution Units. In *39th Intl. Symposium on Microarchitecture(MICRO), 2006.*

[20] X. Liang, G.-Y. Wei, and D. Brooks. ReVIVaL: A Variation Tolerant Architecture using Voltage Interpolation and Variable Latency. In *35th International Symposium on Computer Architecture (ISCA), 2008.*

[21] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *Computer*, 35(2):50–58, 2002.

[22] T. Moscibroda and O. Mutlu. A Case for Bufferless Routing in On-Chip Networks. In *36th International Symposium on Computer Architecture (ISCA)*, 2009.

[23] R. Mullins, A. West, and S. Moore. Low-Latency Virtual-Channel Routers for On-Chip Networks. In *ISCA '04: Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.

[24] N. Muralimanohar and R. Balasubramonian. The Effect of Interconnect Design on the Performance of Large L2 Caches. In *3rd IBM Watson Conference on Interaction between Architecture, Circuits, and Compilers (P=ac2),2006.*

[25] N. Muralimanohar and R. Balasubramonian. Interconnect Design Considerations for Large NUCA Caches. *ISCA '07: Proceedings of the 34th Annual International Symposium on Computer Architecture*, page 369, 2007.

[26] L.-S. Peh and W. J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Comp. Architecture*, page 255, 2001.

[27] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *8th Intl. Symp. on High-Performance Computer Arch.*, 2002.

[28] L. Shang, L.-S. Peh, , and N. K. Jha. Powerherd: Dynamic satisfaction of peak power constraints in interconnection networks. In *Association Computing Machinery (ACM) Int. Conf. Supercomputing*, 2003.

[29] L. Shang, L.-S. Peh, and N. K. Jha. Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks. In *9th Int. Symp. High Performance Computer Architecture*, 2003.

[30] L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha. Thermal Modeling, Characterization and Management of On-Chip Networks. In *MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, 2004.

[31] S. Tam, R. Limaye, and U. Desai. Clock Generation and Distribution for the 130-nm Itanium 2 Processor with 6-MB On-Die L3 Cache. In *IEEE Journal of Solid-State Circuits*, volume 39, pages 636–642, April 2004.

[32] A. Tiwari, S. Sarangi, and J. Torrellas. ReCycle: Pipeline Adaptation to Tolerate Process Variation. In *34th Annual International Symposium on Computer Architecture (ISCA), 2007.*

[33] S. Vangal, J. Howard, and et. al. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In *IEEE International Solid-State Circuits Conference, ISSCC*, 2007.

[34] H. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A Power-Performance Simulator for Interconnection Networks. In *ACM/IEEE MICRO*, Nov 2002.

[35] F. Xie, M. Martonosi, and S. Malik. Compile-time Dynamic Voltage Scaling Settings: Opportunities and Limits. In *PLDI:Conference on Programming Language Design and Implementation*, 2003.