Software-Directed Power-Aware Interconnection Networks

VASSOS SOTERIOU, NOEL EISLEY, and LI-SHIUAN PEH
Princeton University

Interconnection networks have been deployed as the communication fabric in a wide spectrum of parallel computer systems, ranging from chip multiprocessors (CMPs) and embedded multicore systems-on-a-chip (SoCs) to clusters and server blades. Recent technology trends have permitted a rapid growth of chip resources, faster clock rates, and wider communication bandwidths, however, these trends have also led to an increase in power consumption that is becoming a key limiting factor in the design of such scalable interconnected systems. Power-aware networks, therefore, need to become inherent components of single and multi-chip parallel systems. In the hardware arena, recent interconnection network power-management research work has employed limited-scope techniques that mostly focus on reducing the power consumed by the network communication links. As these limited-scope techniques are not tailored to the applications running on the network, power savings and the corresponding impact on network latency vary significantly from one application to the next as we demonstrate in this paper; in many cases, network performance can severely suffer. In the software arena, extensive research on compile-time optimizations has produced parallelizing compilers that can efficiently map an application onto hardware for high performance. However, research into power-aware parallelizing compilers is in its infancy. In this paper, we take the first steps toward tailoring applications' communication needs at run-time for low power. We propose software techniques that extend the flow of a parallelizing

Extension of Conference Paper. Original work appeared in CASES'05 [Soteriou et al. 2005]. The extensions found in the journal paper submission are the following:

- Updated Related Work in Section 2.1.
- Addition of method for generating software directives under adaptive routing, along with a detailed mathematical model, Sections 4.1 and 4.1.1.
- Characterization and rationale for the use of buffer utilizations at run-time in concert with software directives, Section 5.1.
- Explanation of how buffer utilization is used in adaptive routing Section 5.2.
- Additional results with adaptive routing in Section 6 for all 28 applications of the three architectures considered.
- Full range of power-performance results (all 28 applications) under traffic perturbation in Section 6.6, along with updated relevant discussion. The original conference paper only had a subset of these results.

Authors' address: Vassos Soteriou, Noel Eisley, and Li-Shiuan Peh, Department of Electrical Engineering, Princeton University, Princeton, New Jersey 08544; email: {soteriou,eisley,peh}@princeton.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2007 ACM 1544-3566/2007/03-ART5 \$5.00 DOI 10.1145/1216544.1216548 <http://doi.acm.org/10.1145/1216544.1216548>

ACM Transactions on Architecture and Code Optimization, Vol. 4, No. 1, Article 5, Publication date: March 2007.

compiler in order to direct run-time network power reduction. We target network links, a significant power consumer in these systems, allowing dynamic voltage scaling (DVS) instructions extracted during static compilation to orchestrate link voltage and frequency transitions for power savings during application run-time. Concurrently, an online hardware mechanism measures network congestion levels and adapts these off-line DVS settings to maximize network performance. Our simulations over three existing parallel systems, ranging from very fine-grained single-chip to coarse-grained multi-chip architectures, show that link power consumption can be reduced by up to 76.3%, with a minor increase in latency, ranging from 0.18 to 6.78% across a number of benchmark suites.

Categories and Subject Descriptors: B.9.1 [Power Management]: Low-Power Design [**B.8.2 [Performance and Reliability]: Performance Analysis and Design Aids**]: C.2.0 [Computer-Communication Networks]: General

General Terms: Design, Management, Performance

Additional Key Words and Phrases: Software-directed power reduction, dynamic voltage scaling, interconnection networks, on-chip networks, communication links, simulation

ACM Reference Format:

Soteriou, V., Eisle, N., and Peh, L.-S. 2007. Software-directed power-aware interconnection networks. *ACM Trans. Architect. Code Optim.* 5, 1, Article 5 (March 2007), 40 pages. DOI = 10.1145/1216544.1216548 <http://doi.acm.org/10.1145/1216544.1216548>.

1. INTRODUCTION

Interconnection networks are becoming the *de facto* communication fabric in both single-chip multiprocessors (CMPs) [Taylor et al. 2004; Sankaralingam et al. 2003; Dally and Towles 2001] and multi-chip systems [InfiniBand 2006; Mukherjee et al. 2002], facilitating program parallelism as a means to reduce execution time and to achieve very high, scalable performance. While rapidly improving VLSI technology is allowing the use of additional chip resources along with higher clock rates, performance gains do not arrive without cost. As in the case of uniprocessor systems, interconnected systems, both in the on-chip and chip-to-chip domains, suffer from the effects of ever increasing power consumption, with the interconnection network taking up a sizable portion of the parallel system's power budget. For instance, the on-chip network in the MIT Raw CMP consumes 36% of the entire chip's power budget [Kim et al. 2003]. In board-to-board and multi-chip networks the routers and communication links are already consuming substantial power. In a Mellanox server blade the router and links are estimated to dissipate 15 W out of the total budget of 40 W (37.5%), with the processor allocated the same power budget of 15 W [Mellanox 2006], while 65% of the power budget of the IBM 8-port 12X switch [InfiniBand 2006] is taken up by the communication links (each of the eight ports interfaces 12 2.5 Gbps links that dissipate 2.5 W each, with the entire switch consuming 31 W, on average). In addition, the on-chip router and links of the Alpha 21364 processor consume a substantial 23 W, where the links consume 58% of this allocated power [Mukherjee et al. 2002]. Indeed, the International Technology Roadmap for Semiconductors [ITRS 2005] highlights system power consumption as the limiting factor in developing systems below the 50 nm technology point. To help overcome this barrier, the design of interconnection networks must emphasize power awareness.

A widely recognized power-reduction technique is dynamic voltage scaling (DVS). In uniprocessors, researchers have proposed several methods to explore compile-time DVS scheduling [Saputra et al. 2002; Xie et al. 2003]. These techniques identify periods of program execution slack at various points in a program and appropriately insert DVS instructions in the original code to slow down the processor in order to save power.

In interconnected systems, DVS has also been proposed to reduce the power consumption of on-chip buses [Worm et al. 2002] and chip-to-chip interconnection networks [Shang et al. 2002; Stine and Carter 2004] using run-time hardware-prediction mechanisms to tune voltage and frequency levels on each link according to the projected traffic levels. Though these approaches are simple and provide good power savings, their limited-scope nature cannot accommodate the fluctuating link bandwidth needs of a specific application. As Section 2.2 demonstrates, network performance can be highly variable and unpredictable from one application to the next and, in some cases, severely degraded.

In the software arena, extensive research on compile-time optimizations has produced heavily optimized compilers [Robert et al. 1996; Lee et al. 1998] that expose program parallelism to efficiently map an application onto the parallel architecture, showing good potential for application execution speedup. However, compiler optimizations that address power issues in parallel architectures remain very limited, with recent work in Kadayif et al. [2004] targeting processor power optimization for array-based applications. In short, communication power reduction at the software level must be addressed more thoroughly.

In this paper we take the first steps toward tailoring applications' communication needs at run-time for low power and propose a software-based methodology that extends the parallel compiler flow in order to construct high-performance power-aware interconnection networks by targeting communication links, a significant power consumer in interconnection networks. Our methodology consumes the statically compiled message flow of an application and analyzes the traffic levels for all links in the network over periods of time. By factoring in architecture characteristics, our technique matches DVS link transitions to the expected levels of traffic, generating DVS software directives that are injected into the network along with the network-mapped application. These DVS instructions are then executed at run-time, dynamically adapting link power consumption to actual utilization. Concurrently, a hardware online mechanism measures network congestion levels and fine-tunes the execution of these DVS instructions to handle run-time variabilities that are not precisely captured at compile-time. Our results show that our software-directed approach demonstrates significantly improved power-performance as compared to prior hardware-based approaches reducing link power by up to 76.3% relative to the baseline network configuration, i.e., without DVS links. Network performance incurs a minor increase in network latency ranging from 0.18 to 6.78% across a number of benchmark suites running on three existing networks, spanning very fine-grained single-chip to coarse-grained multi-chip parallel architectures.

Next, Section 2 discusses prior related research, demonstrating the limited-scope nature of existing hardware-driven approaches and motivates the use of software directives to reduce network power, while Section 3 describes the

assumed DVS link model. Section 4 follows with details of our proposed techniques for extracting DVS software directives and Section 5 describes our online DVS hardware mechanism. Section 6 details our simulation setup and results for a range of benchmark suites running on three existing network architectures. Finally, Section 7 concludes the paper.

2. BACKGROUND AND MOTIVATION

2.1 Related Work

As parallel systems become faster and increasingly interconnected, there has been increasing recognition of the need to target the power consumption of the interconnection networks employed by these parallel systems. Several recent studies have modeled and characterized the power profile of network routers and links in a variety of systems. Results over a range of configurations, from clusters [Kim et al. 2003] to servers [Patel et al. 1997] and CMPs [Benini and Micheli 2001; Kim et al. 2003; Wang et al. 2002], emphasize and demonstrate the high power consumed by network routers and communication links.

To a limited extent, prior research has explored the use of power-aware methodologies to reduce link power consumption in interconnection networks. Here we categorize them into *hardware-* and *software-based* power-aware techniques. Hardware-based approaches are further arranged into two main classes according to the type of power-aware link mechanism they employ: dynamic voltage scalable (DVS) and on/off links. The first power-aware interconnection networks proposed by Shang et al. [2002] explored the use of DVS links, where hardware counters measure the levels of past and current network utilization over fixed sampling windows. These online statistics are then compared to fixed thresholds to direct voltage/frequency, i.e., (V, f) , link pair transitions. Later, work by Stine and Carter [2004] demonstrated that under a multi-chip system with synthetic self-similar traffic, in some cases where the network can provide enough bandwidth to meet the application requirements, a statically set link frequency along with adaptive routing can outperform multi-level (V, f) pair DVS links. Furthermore, research by Shin and Kim [2004] proposed an off-line speed link assignment algorithm for energy-efficient networks-on-chips (NoCs) with voltage scalable links. This scheme preassigns fixed voltage and frequency link levels (lower than the maximum levels) *a priori*, given the task graph of an application. This scheme is thus suited for real-time periodic applications mainly run by embedded systems, where designers are able to predict communication delays at design-time. Besides the area of NoCs, researchers [Kaul et al. 2005] devised a DVS scheme for a 6-mm on-chip memory read bus, where the wire supply voltages are dynamically scaled down for typical case conditions, resulting in significant energy reduction while still meeting delay constraints. Finally, DVS policies proposing circuits for implementing DVS in opto-electronic links were demonstrated by Chen et al. [2005].

In the area of on/off communication links, links that switch on/off as a response to network traffic fluctuations in order to save power, Soteriou and Peh [2004] proposed a number of power-aware techniques that depend on

hardware counter measurements obtained from the network during run-time. These statistics are then compared to empirically set thresholds to direct link on/off transitions. In addition, to avoid deadlocks, the researchers devised fully adaptive routing protocols, mapped onto the network topology which was modeled as a connectivity graph. A power-saving strategy for regular interconnection networks, built with high-degree switches, where multiple links connecting adjacent nodes switch on/off, was later demonstrated by Alonso et al. [2005]. Lastly, Kim et al. [2003] proposed and investigated networks comprising DVS-DLS (dynamic voltage scaling-dynamic link shutdown) links that further shutdown when traffic drops to very low levels.

Further, research in the areas of embedded NoCs and multicore systems-on-a-chip (SoCs) demonstrated several software-based techniques that use application profiling to reduce power. In the following three studies, links are fixed to *one* frequency/voltage throughout the entire application run-time. Work by Luo et al. [2003] addressed the joint optimization of variable-voltage processors and communication links under real-time constraints within heterogeneous embedded systems. Similarly, Hu and Marculescu [2004] have proposed a scheduling algorithm that reduces energy in heterogeneous NoCs by scheduling both communication and computation in parallel under real-time constraints, where the required voltages and frequencies are derived from application profiling. Further, Jalabert et al. [2004] have presented the $\times$ pipesCompiler, a tool that uses application profiling to instantiate an application-specific, power-saving NoC for heterogeneous multicore SoCs. The software directives proposed in this paper complement the aforementioned synthesis tools and can allow them to handle power-aware DVS networks.

Recently, several relevant works on compiler-directed power-aware networks have been published simultaneously. Work by Chen et al. [2006] explored the use of a proactive power-management technique, where application code is analyzed by the compiler to identify idle periods in order to insert explicit network power-management calls that are executed during network run-time to direct on/off link transitions. The techniques presented, however, have only been applied to highly predictable array-intensive embedded applications, where exact active/idle periods can be extracted; run-time code timing variability and the use of adaptive routing have not been explored. A similar compiler analysis technique was proposed by Li et al. [2005] for communication link power management using DVS links. As this technique was applied to highly regular array-intensive codes, again, run-time variability and run-time adaptation of software directives were not explored. In this paper, we propose the use of buffer utilizations that are measured at run-time to correct and adapt to online network variabilities that cannot be detected at compile-time (see Section 5.1), such as variabilities exhibited in traffic's message-flow timing and adaptive routing.

2.2 Motivation

Though recent dynamically tuned hardware-based power-aware approaches that rely on run-time statistics have exhibited good interconnection network

Fig. 1. Link power savings of limited-scope hardware-directed DVS for three trace benchmarks running on a TRIPS CMP.

power savings [Kim et al. 2003; Shang et al. 2002; Soteriou and Peh 2004; Stine and Carter 2004], they also demonstrate a number of serious limitations. These techniques are limited in scope and are not tailored to the specific application's spatial and temporal variability when running on the network. For good power performance, power-aware policies need to be aware of an application's network usage demands and need to be tuned to the application's fluctuating network bandwidth requirements.

The above hardware-based techniques depend on statistics obtained during application run-time that are then compared against thresholds to direct power-aware decisions. However these statistics are short-lived and are measured over a limited number of system cycles or sampling windows, reflecting only short-term temporal traffic variability. These statistics are also obtained locally at each router and, therefore, do not reflect the spatial variability across the entire network topology. Lastly, the thresholds are fixed and empirically set and are not based upon traffic behavior indicators.

The original work on interconnection networks with DVS links by Shang et al. [2002] presents good power savings with synthetic self-similar traffic while sustaining high performance. However, since, under the proposed methodology, the thresholds are set empirically and are fixed, performance can suffer severely when faced with a traffic pattern that differs from that for which it is tuned. To demonstrate this, we applied traffic traces from the TRIPS CMP [Sankaralingam et al. 2003] to the exact implementation of Shang et al. [2002], using the same threshold levels and sampling window sizes of the original work. We assumed the DVS link model described in Section 3.

While high link power savings averaging 74.4% are demonstrated (see Fig. 1), Fig. 2 shows the overall severe impact on network performance. With short sampling window sizes of ten cycles, latency penalties can increase to more than double (100.7%) when compared to the original network delay without

Fig. 2. Network latency penalty of limited-scope hardware-directed DVS for three trace benchmarks running on a TRIPS CMP.

DVS. This is because the short sampling windows are not able to distinguish short-term traffic fluctuations from long-term ones. Latency penalties are 47.4% at the minimum and 62.1%, on average, for all configurations and benchmarks. With longer sampling windows, latency penalties and power savings tend to decrease slightly, as links do not toggle (V, f) pairs as often. For instance, threshold set 1 with a sampling window size of 10,000 cycles (see Fig. 1) presents the smallest link power savings of 71.2% over all of our experiments. It also yields the shortest network latency increase of 47.4%.

The prerequisite of the power-aware techniques proposed in this paper is that they depend on traffic profiling. They, therefore, need to possess an advance knowledge of the network traffic's spatio-temporal behavior in order to extract software directives, which, in turn, dictate the network's power-aware responses at run-time. Note though that we include a hardware mechanism that measures online network usage, throttling DVS directives to tune links to actual observed traffic when short-term network congestion is detected (see Section 5). Despite the requirement of profiling, these techniques present a number of important advantages as compared to the above hardware-based approaches:

- **Global view of traffic:** First, our approach has an advance global (collective) view of the network via the estimation of all link utilization levels that are carried out for each link individually, covering the entire application running on the network. Our approach is, therefore, able to “see” the entire network traffic's spatial and temporal variability that is unique for each application, directing DVS link transitions at each link independently for excellent power-performance during run-time.
- **Threshold customization:** Our approach automatically picks customized thresholds, unique to each application, based on profiling of the parallelized application itself.
- **Architectural-specific customization:** Our methodology accounts for network configuration variables, such as network size and buffer capacity, routing type

Table I. Multi-Level Discrete DVS Link Model (V, f) Pairs

$(V, f)_0 \leftarrow (2.50 \text{ V}, 1.00 \text{ GHz})$	$(V, f)_1 \leftarrow (2.38 \text{ V}, 0.95 \text{ GHz})$
$(V, f)_2 \leftarrow (2.27 \text{ V}, 0.90 \text{ GHz})$	$(V, f)_3 \leftarrow (2.15 \text{ V}, 0.85 \text{ GHz})$
$(V, f)_4 \leftarrow (2.02 \text{ V}, 0.80 \text{ GHz})$	$(V, f)_5 \leftarrow (1.93 \text{ V}, 0.76 \text{ GHz})$
$(V, f)_6 \leftarrow (1.84 \text{ V}, 0.72 \text{ GHz})$	$(V, f)_7 \leftarrow (1.75 \text{ V}, 0.68 \text{ GHz})$
$(V, f)_8 \leftarrow (1.66 \text{ V}, 0.64 \text{ GHz})$	$(V, f)_9 \leftarrow (1.57 \text{ V}, 0.60 \text{ GHz})$

(deterministic or adaptive), and individual link architecture characteristics, such as maximum assigned link frequency and bandwidth, in deriving DVS software directives. Our methodology can, therefore, be applied to various network types, such as heterogeneous systems (e.g. SoCs) with links having different assigned bandwidths.

Because of the above advantages of our software-directed methodology, the results of Sections 6.5 and 6.6 show high resilience to fluctuating link bandwidth requirements resulting from high variabilities in the parallelized application's spatial and temporal distributions. The results demonstrate excellent power-performance results when our software-directed power-aware methodology is applied to three existing parallel architectures, ranging from fine-grained on-chip to coarse-grained multi-chip implementations.

3. DVS LINK MODEL

Chip-to-chip parallel [Wei et al. 2000] and serial links [Kim and Horowitz 2002], which automatically and continuously adjust their frequency at a minimum voltage, have already been demonstrated. The variable-frequency serial link has a supply voltage which varies from 0.55 to 2.5 V, dissipating 21 mW at 1 Gbps and up to 197 mW at 3.5 Gbps, providing up to 90% power reduction. Though this link was designed for off-line frequency settings and not for both dynamic voltage and frequency settings, the link architecture can be extended to accommodate DVS [Shang et al. 2002; Chen et al. 2005].

In this paper, we construct a realistic multi-level DVS model, where the serial link can take only a range of 10 discrete frequency levels and corresponding voltage levels. The maximum voltage-frequency pair of the serial link is 1 GHz at 2.5 V and can be scaled down to 0.6 GHz at 1.57 V. Though previous research [Shang et al. 2002] has suggested a range of frequencies from 1 GHz to 125 MHz that yields up to 10X power improvement, the latter frequency level increases the traversal time of a flit (the term flit is an abbreviation for “flow control unit,” a fixed-size segment of a packet) crossing a link by a factor of 8X. As a link has to go through all (V, f) transition steps sequentially, requiring a considerable number of cycles, transitioning to the maximum (V, f) level, in case of an abrupt increase in link traffic, can have a serious negative impact on performance. In our model, even though the minimum frequency is restricted to 0.6 GHz, our model exhibits considerable power savings of up to 76.33%. Since frequencies and voltages are compacted in the upper (V, f) range, more fined-grained frequency levels can be considered. This allows the discrete link frequencies to be fine-tuned to the expected traffic levels. Table I

shows the (V, f) voltage-frequency pairs of our DVS link. Dynamic link power is estimated using:

$$P_{link} = C_{load} V_{dd}^2 f_{link} \quad (1)$$

where P_{link} is the power consumed by the link, C_{load} is the load capacitance, V_{dd} is the supply voltage, and f_{link} is the link frequency. The voltage and link frequency transitions occur separately. When the link down-ramps (V, f) , the frequency is reduced first, followed by voltage. When the link up-ramps (V, f) , the voltage increases first, and then the frequency. During frequency transitions network traffic (packets) cannot cross the link, but during voltage transitions traffic can cross the link. It takes 20 clock cycles to transition between any two sequential discrete frequency steps and 100 cycles to transition between any two sequential discrete voltage steps [Chen et al. 2005]. In other words, the model requires a total of 1080 clock cycles to traverse the entire range of discrete frequency and voltage levels. The energy consumed during transitioning is [Burd and Brodersen 2000]:

$$E_{link-trans} = (1 - \eta) C_{filter} |V_a^2 - V_b^2| \quad (2)$$

where η is the efficiency typically taken to be 90% and C_{filter} is the filter capacitance, assumed to be 5 pF [Kim and Horowitz 2002]. In our experiments, we considered both the dynamic and overhead transitioning link energies in estimating power savings.

4. SOFTWARE DIRECTIVES GENERATION

A parallelizing compiler such as that by Lee et al. [1998] and Robert et al. [1996] takes as input sequential code, performs temporal and spatial partitioning of the code into code segments, and then distributes these segments onto computational nodes. For correct code execution, the compiler orchestrates inter-node communication by synchronizing `send()` and `receive()` message-passing operations. Each node communicates with others through a communication fabric. As the number of nodes scales, networks become the fabric of choice, with each node interfacing to an associated router. Figure 3 shows an example of a code snippet being partitioned into three segments, with each code segment mapped onto a computational node.

Our power-aware methodology extends this flow statically, generating DVS software directives immediately after code partitioning and scheduling. DVS directives generation is achieved in three phases, as Fig. 3 shows. In the first phase, our technique uses LUNA [Eisley and Peh 2004], a framework that was originally proposed to analyze network power consumption, as a base. LUNA factors in network architecture parameters, such as network size, the type of the routing protocol (deterministic or adaptive), and the compiler-generated communication code streams to periodically estimate average link utilization levels across all network links, paced by a sampling window of T_w cycles. For this to work, message flows need to contain network injection time stamps. In CMP architectures, such as Raw [Taylor et al. 2004], the hardware is fully exposed to the compiler by exporting a cost model for communication and computation. The

Fig. 3. Overview of proposed software directives.

Fig. 4. Conceptual hardware design of a network router with software-directed dynamic voltage scalable links (one link shown). Note that the buffer utilization counters are connected to the routing decision circuitry only in the case of adaptive routing; this connection does not apply to routers with deterministic routing (see Section 5.2).

Rawcc compiler [Lee et al. 1998] explicitly manages all communication through the interconnect statically at compile-time, providing cycle-by-cycle message flow scheduling and timing information that can be used by LUNA. Sequencing is exact, but as a result of dynamic events there are some disturbances in run-time flows, with a 5% probability of occurrence [Lee et al. 1998]. An advantage of our methodology, as we show in Section 6.6, is that message flow-timing information does not have to be exact and can tolerate fairly large disturbances. Where static compilation is unable to analyze message flows to estimate link utilization, one can profile applications to obtain timing estimates. Saputra et al. [2002] and Xie et al. [2003], and Hu and Marculescu [2004] used this profiling approach in uniprocessors and embedded systems, respectively.

In phase 2, LUNA's link utilization estimates are normalized to the link bandwidths, and by considering the multi-level discrete DVS model of Section 3, DVS instructions are generated for each link individually using the proposed DVS software directives algorithm of Section 4.2.

A detailed conceptual design of a software-directed router with DVS links is depicted in Fig. 4. As described in Section 4.2, the software directives are generated based upon LUNA's sampling intervals of T_w cycles, where these T_w -based intervals form the execution time stamps and thus set the periodicity at which the directives are to be executed at network run-time. During the online phase, therefore, these directives are executed at times relevant to a hardware clock counter, i.e., at every T_w . Thus, unlike recently proposed methods [Chen et al. 2006] the software directives are not inserted *in-line* with the application code, but are used in *parallel* with the application code; in other terms, the software directives executions are synchronized with the clock counter and not with the processor instructions. Thus, at each node, custom directives for each link are written into a dedicated FIFO memory whenever a parallelized code segment is scheduled to run on the associated node processor. The directives can be written into the FIFO memory by the operating system and can be saved/restored as a part of the context by the OS, if the application is swapped out from the system to run a second application. In this case, the OS writes

into the FIFO memory the directives applicable to the second application, while the original DVS directives are stored in the process context block of the first application. Note that as the process switches occur infrequently, the OS context switch overheads impacting the overall system are expected to be relatively small.

The node processor is responsible for resetting the clock counter when the application starts executing, while the DVS controller polls these directives from the FIFO memory periodically, paced at T_w intervals, to set the voltage and frequency levels of the outgoing DVS link. Since the execution of these directives is independent of the current node's processor, the pipeline of the processor is unaffected and no additional latencies are incurred. In short, instructions are executed by the main processor core while DVS directives are acted upon by the DVS controller, which is a state machine within the router. The FIFO memory can be of a reasonable size; for instance, if T_w is 100,000 cycles, then a 100-entry FIFO memory can hold enough directives to last for 10 million cycles of application run-time. There are two DVS directives per T_w (see Section 4.2), one that reaches an intermediate (V, f) level within the current T_w and another that settles on the target (V, f) at the end of the current T_w . Hence, each entry of the software directives FIFO memory contains two (V, f) directives for each sampling period of T_w cycles: it is 8 bits wide since each directive can be represented by 4 bits, given our 10-level discrete DVS link model (described in Section 3). Memories in network routers, in the form of caches, have also been used in on-chip architectures, such as Raw, to hold instructions that direct run-time packet switching in a static network [Taylor et al. 2004]. These instructions are similarly created during static compilation.

In the third phase (Fig. 3), queueing theory principles are used to translate already estimated link utilizations into router output buffer utilizations. A histogram is then constructed, which shows the distribution of the entire network's output buffer utilization by aggregating all the individual estimated output buffer utilization levels. These statistics are used to set router thresholds, which are stored in a threshold memory at each router. The DVS controller shown in Fig. 4 takes into account the output buffer utilization statistics of the buffer counter to throttle DVS instructions when short-term contention exceeding these threshold levels is detected in order to maximize network performance. In the case of adaptive routing, the buffer utilization statistics are also used to reach a routing decision by picking the least congested output port. Section 5 provides detailed descriptions of the proposed hardware mechanisms. We next explain each phase in detail.

4.1 Phase 1: Link Utilization Estimation

To capture spatial and temporal message flow variability in order to explore power savings, we use LUNA to estimate link utilizations across the network [Eisley and Peh 2004]. LUNA is a high-level network power analysis tool whose accuracy was shown to be within 5.9% of cycle-level simulators [Wang et al. 2002], with a run-time that is up to 360X faster. These attributes make LUNA suitable for compiler-directed network power analysis. LUNA abstracts

Fig. 5. A walk-through example showing the five steps of link utilization calculations using LUNA with deterministic XY routing. Time (t) is in terms of absolute router clock cycles.

network power through link utilizations, capturing the effect of contention among message flows in its estimation of utilization across time for each link in the network. Based on these estimates, we then create DVS software directives in phase 2 (Section 4.2) that leverage unused link capacity as power-saving opportunities.

There are five key steps in LUNA, explained in Fig. 5. Note that we show traffic only across router nodes 0, 1, 2, and 5 for clarity (see Fig. 5a).

Step 1. In the first step, message flows are captured as *injection rate functions*, with the injection rate of the message expressed as a percentage of the injection port bandwidth over time. Figure 5b shows the injection rate functions of message flows A and B over the first 1,000 clock cycles.

Step 2. During this step, routing maps the injection rate functions of step 1 onto links of a network topology, translating them into *normalized link utilization functions*, $F(Msg_j, t)_i$, with values between 0 (no traffic) to 1.0 (link saturated). LUNA was modified to support both deterministic XY (or static) routing, where routing messages fully traverse the X dimension before traversing the Y dimension toward their destination node and adaptive routing. Both routing functions are minimal, which means that they route progressively¹ within a minimum rectangle²: given the source n_s and destination n_d nodes,

¹In progressive routing, every routing step leads a packet one hop closer to its destination.

²A minimum rectangle applies to 2D tori, where the topology wraps at the edges causing any n_s - n_d combination to produce four rectangles that entirely cover the topology. The minimum rectangle is the rectangle that possesses the smallest area. In meshes, wraparound links are absent and, as a result, the only rectangle formed by n_s - n_d is also the minimum rectangle.

four rectangles contain n_s and n_d as their diagonally opposite vertices. The minimum rectangle is the one with the minimum diagonal distance between n_s and n_d . Since both functions route progressively within a minimum rectangle they both exhibit the same hop count, q , for every packet routed along every possible $n_s \rightarrow n_d$ path³ between routers n_s and n_d . In deterministic routing, this route is predetermined (exhausting the horizontal and then the vertical dimension, i.e., XY) while in adaptive routing, this route can take any staircase form, where horizontal and vertical packet traversals may alternate, each time bringing the packet a hop closer to its destination.

In this example, we demonstrate XY deterministic routing, while Section 4.1.1 explains how step 2 is modified for adaptive routing. Only step 2 is different for the two routing protocols as message routing determines message injection rate mapping, with steps 1 and 3–5 remaining identical. LUNA concurrently considers (1) the packet size in terms of flits and (2) the source-destination router coordinates of each packet (see phase 1, Fig. 3). Packet types, whether data, control, or acknowledgment, and packet contents are ignored since only the “volume” of traffic in terms of flits is required by LUNA. Figure 5c shows how message flows traverse links $0 \rightarrow 1$, $1 \rightarrow 2$, and $2 \rightarrow 5$ for the same time duration of 1,000 clock cycles. $F(Msg_A, t)_{Path(0 \rightarrow 1)}$ and $F(Msg_A, t)_{Path(1 \rightarrow 2)}$ correspond to the normalized link utilization functions for message A over links $0 \rightarrow 1$ and $1 \rightarrow 2$, respectively. Similar notations apply for message B, as shown in Fig. 5c.

Step 3. Next, link utilization functions are *superimposed* and *summed*, reflecting the sharing of links among multiple message flows. In our example, functions $F(Msg_A, t)_{Path(1 \rightarrow 2)}$ and $F(Msg_B, t)_{Path(1 \rightarrow 2)}$ are added during $0 \leq t \leq 1,000$ to produce $F(Msg_A + Msg_B, t)_{Path(1 \rightarrow 2)}$. Figure 5d shows that this summation actually detects traffic contention or overflow over link $1 \rightarrow 2$ between cycles 0–300 and 600–1,000 since the normalized utilization rate of 1 is exceeded (100% link bandwidth capacity).

Step 4. To account for link contention, LUNA *propagates* this overflow area as depicted in Fig. 5e. Intuitively, this overflow area corresponds to the number of bits that need to be transported later as they currently exceed the link capacity (bandwidth).

Step 5. Finally, the link utilization functions are *split* back into *constituent* message flows, reflecting how individual messages are affected by the contention. Fair arbitration is assumed in splitting the link utilization among the message flows, as shown in Fig. 5f.

4.1.1 Adaptive Routing: Modification of LUNA’s Step 2. Adaptive routing routes progressively within a minimum rectangle, $Min_Rectangle(n_s, n_d)$, in a staircase manner; any possible path, $Path(n_s \rightarrow n_d)$, within this rectangle between source-destination nodes n_s and n_d can be traversed. Both static and adaptive routing exhibit the same minimal hop count q between n_s and n_d . The number of different paths that can be formed between any $n_s \rightarrow n_d$ router pair

³The terms “route” and “path” are used interchangeably in the paper, as well as the terms “router” and “node.”

Fig. 6. Example showing a single message Msg being routed from source node $n_s = 3$ to destination node $n_d = 2$: (a) $r_{j \rightarrow j+1}$ next hop traversal probabilities for each link in the minimum rectangle formed by $Min_Rectangle(n_s, n_d)$ for adaptive routing; (b) $F(Msg, t)$ mapped injection rate function for deterministic XY routing with routing path $Path(3 \rightarrow 4 \rightarrow 5 \rightarrow 2)$; (c) $P_{Path(n_s \rightarrow n_d)_{(0,1,2)}}$ path probabilities for every link lying in the minimum rectangle ($N = 3$); and (d) $f(Msg, t)_{Path(n_s \rightarrow n_d)_{(0,1,2)}}$ subset link utilization functions for each routing path for adaptive routing. Time (t) is in terms of absolute router clock cycles.

is equal to the number of progressive direction combinations between n_s and n_d . These combinations cover all links in the rectangle formed between n_s and n_d . In general, if the Manhattan distance between n_s and n_d consists of q link traversals and k_N, k_S, k_E, k_W where $q = k_N + k_S + k_E + k_W$ are the number of traversals in the north, south, east, and west directions, respectively, then the number of different paths within $Min_Rectangle(n_s, n_d)$ is ${}_q C_{k_N, k_S, k_E, k_W} = \frac{q!}{k_N! k_S! k_E! k_W!}$. However since we account for progressive adaptive routing within a minimum rectangle, then only two distinguishable and progressive directions are considered and, hence, ${}_q C_{k_a, k_b} = \frac{q!}{k_a! k_b!}$ where a designates either the east or west direction and b designates either the north or south direction, i.e., $a = \{E, W\}$ and $b = \{N, S\}$. The number of paths between $n_s \rightarrow n_d$ is denoted here as $N = {}_q C_{k_a, k_b}$. For instance, referring to Fig. 6a, the total number of paths between $n_s \rightarrow n_d$ (nodes 3 and 2) is $N = 3$ (${}_q C_{k_a=E, k_b=N} = \frac{(1+2)!}{1! 2!} = 3$). These three individual paths are shown in Fig. 6c.

Only step 2, the mapping of the injection rate functions onto link utilization functions in LUNA's five-step chain differs between deterministic and adaptive routing. Once the injection rate functions are calculated, based on the choice of routing, LUNA continues with identical calculations for the two routing cases from step 3–5 in summing and propagating the injection rate functions. Step 2 depends on the routing protocol as routing affects the path that each message takes within $Min_Rectangle(n_s, n_d)$ and, therefore, the mapping of its route onto the network. For instance, Figure 6a, b show a single message, Msg , injected at router 3 and destined for router 2, where, in deterministic routing, this message takes the route $Path(3 \rightarrow 4 \rightarrow 5 \rightarrow 2)$, captured by the single link utilization function $F(Msg, t)_{Path(3 \rightarrow 4 \rightarrow 5 \rightarrow 2)}$. With adaptive routing there

are multiple paths within $Min_Rectangle(n_s, n_d)$ with each such path potentially carrying a portion of the aggregate message Msg . The superimposition of all these paths spans all links in the minimum rectangle. Thus, in adaptive routing $F(Msg, t)_{Min_Rectangle(n_s, n_d)}$ is split into multiple *link utilization subfunctions* $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$, with each such subfunction representing a particular route $Path(n_s \rightarrow n_d)_i$. For each of the N distinct routing paths in a minimum rectangle formed between n_s and n_d there is a corresponding and unique injection rate subfunction $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$. $F(Msg, t)_{Min_Rectangle(n_s, n_d)}$ is the summation (superset) of all injection rate subfunctions for message Msg between routers n_s and n_d , i.e., $F(Msg, t)_{Min_Rectangle(n_s, n_d)} = \sum_i f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$. The injection rate subfunctions, $f(Msg, t)_0, f(Msg, t)_1, \dots, f(Msg, t)_{[N-1]}$, satisfy the following condition for every message (each numbered subscript here denotes a distinct path in the minimum rectangle; path notations are omitted for brevity):

$$F(Msg, t) = f(Msg, t)_0 + f(Msg, t)_1 + \dots + f(Msg, t)_{[N-1]} \quad (3)$$

Figure 6d shows how $F(Msg, t)_{Min_Rectangle(n_s, n_d)}$ from Fig. 6b is split into $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$ where $i = \{0, 1, 2\}$, with each such subfunction representing a distinct path between nodes $n_s = 3$ and $n_d = 2$. During network runtime, adaptive routing is ultimately carried out on a per-node basis, where the least congested progressive next-hop route is taken to balance network traffic among the various routing paths in hopes of achieving higher performance. Since each possible path cannot be known a priori, we emulate adaptive routing by assigning *next hop traversal probabilities* $r_{j \rightarrow j+1}$ at each connecting link that lies in $Min_Rectangle(n_s, n_d)$. Figure 6a shows these $r_{j \rightarrow j+1}$ assignments. $r_{j \rightarrow j+1}$ lies within $[0.5, 1]$ as we assume progressive routing, so there are at most two possible next-hop routers for 2D topologies⁴ emanating from a current router with each such next-hop router having a probability $r_{j \rightarrow j+1} = 50\%$ of being reached. Routers that lie at the two minimum rectangle perimeter edges touching n_d carry a $r_{j \rightarrow j+1} = 1$ since there is only one next-hop router to be reached.

Since each path does not carry the same distribution of $r_{j \rightarrow j+1}$ values, this means that each possible $Path(n_s \rightarrow n_d)_i$ does not carry the same probability of being traversed. This needs to be reflected upon the injection rate subfunctions $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$. We define the set consisting of $p_{Path(n_s \rightarrow n_d)_i}$ as the *path probabilities*, i.e., the probability of each possible path being traversed by a message stream between nodes n_s and n_d , one for each $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$. Each $p_{Path(n_s \rightarrow n_d)_i}$ represents the ratio of traffic carried by the corresponding $f(Msg, t)_i$, and has a value equal to the product of all the $r_{j \rightarrow j+1}$ of those links that lie on its path. In other words $\forall n_{ij} \in Path(n_s \rightarrow n_d)_i \exists r_{s \rightarrow s+1}, r_{s+1 \rightarrow s+2}, \dots, r_{d-1 \rightarrow d}$, where $p_{Path(n_s \rightarrow n_d)_i} = \prod_{s=0}^{d-1} r_{s \rightarrow s+1}$. For instance, the uppermost graph of Fig. 6c shows that $p_{Path(3 \rightarrow 4 \rightarrow 5 \rightarrow 2)} = r_{3 \rightarrow 4} \times r_{4 \rightarrow 5} \times r_{5 \rightarrow 2} = 0.5 \times 0.5 \times 1.0 = 0.25$. This means that $f(Msg, t)_{Path(3 \rightarrow 4 \rightarrow 5 \rightarrow 2)}$ will have a probability of 0.25 of being traversed.

⁴The scheme can be readily extended for higher dimension networks.

Thus, $f(Msg, t)_{Path(3 \rightarrow 4 \rightarrow 5 \rightarrow 2)}$'s utilization is $0.25 \times F(Msg, t)_{Min_Rectangle(n_s=3, n_d=2)}$ as Fig. 6c and d show. Between cycles 0 and 600 the utilization levels fall from 0.4 to $0.4 \times 0.25 = 0.1$. Similarly, on the interval 600–1,000, we see the utilization ratio 0.8 scaling to 0.2. Note that adding all $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$, where $i = \{0, 1, 2\}$, of Fig. 6d will sum to part (a) of the same figure, i.e. $\sum_i f(Msg, t)_{Path(n_s \rightarrow n_d)_i} = F(Msg, t)_{Min_Rectangle(n_s, n_d)}$. This is because $\sum_{i=0}^{N-1} p_{Path(n_s \rightarrow n_d)_i} = 1$, which verifies the correctness of the method, i.e., traffic is neither lost or created and adds up to the original $F(Msg, t)_{Min_Rectangle(n_s, n_d)}$ traffic.

As mentioned, the number of paths between $n_s \rightarrow n_d$ grows as $N = {}_q C_{k_a, k_b}$. When the network is relatively large, this may increase LUNA's computational time. For instance in an 8×8 mesh topology, when a corner router sends a message stream to the opposite diagonal corner (though because of parallel compiler optimizations targeting code spatial locality such probability of occurrence is very small) of the topology this can result in $N = \frac{(7+7)!}{7!7!} = 3432$ different paths. This is impractical; to reduce the number of calculations, we select a subset of such paths. This can result in overestimation or underestimation of some $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$. However, this impact is small as Manhattan distances are usually short and produce a small N . Thus, the number of $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$ is relatively small. Typically we choose $5 \leq M \leq 10$, where $M \leq N$ and M is the subset of paths out of the total number of possible paths N . The set of M paths chosen consists of the paths at the outer perimeter of the minimum rectangle, which have the largest path probabilities, $p_{Path(n_s \rightarrow n_d)_i}$, the diagonal across the rectangle connecting the two routers, and equally spaced paths between each side of this diagonal and the facing rectangle perimeter. The p_i probabilities (we drop the path subscript from $p_{Path(n_s \rightarrow n_d)_i}$ and $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$ for clarity) need to be normalized over the M paths to determine the ratio of traffic each $f(Msg, t)_i$ now takes. Since $\sum_{i=0}^{N-1} p_i = 1$, we let $0 < W = \sum_{i=0}^{M-1} p_i \leq 1$, define a weighing factor for all $f(Msg, t)_i$:

$$F(Msg, t) = \frac{p_0}{W} f(Msg, t)_0 + \frac{p_1}{W} f(Msg, t)_1 + \dots + \frac{p_{[M-1]}}{W} f(Msg, t)_{[M-1]} \quad (4)$$

This correctly estimates the relative proportion of traffic captured by each $f(Msg, t)_{Path(n_s \rightarrow n_d)_i}$ consistently as $\sum_{i=0}^{N-1} p_{Path(n_s \rightarrow n_d)_i} = \sum_{i=0}^{M-1} \frac{p_{Path(n_s \rightarrow n_d)_i}}{W} = 1$, in full agreement with the original Eq. (3). Note that the calculation of all injection rate functions under adaptive routing offers a conjecture of how traffic will be routed during network run-time. It is obviously impossible to predict the exact paths every message stream will take beforehand during static link utilization estimations. As routing decisions occur on-the-fly there are possible mismatches in routing between predicted paths and actual paths taken. At each hop, the routing decision chooses the least congested output link (out of the two that conform to progressive routing) based on the run-time measurement of output buffer utilizations, as will be described in Section 5.2. Though the above routing mismatches can occur, the online hardware mechanism of Section 5 helps adapt to these inconsistencies and maintain high network power-performance. Results in Section 6 show the superiority of adaptive versus static routing.

Fig. 7. Example of software-directives generation.

4.2 Phase 2: Software-Directives Extraction

Estimated link utilizations from phase 1 are used as inputs in phase 2 to generate DVS software directives for each network link individually. Though phase 2 is carried out independently for each link, the link utilization estimates from phase 1 are based on global information of the message flows across an entire application, unlike previous limited-scope hardware methods [Shang et al. 2002; Soteriou and Peh 2004] that only used local information.

Figure 7 sketches an overview of phase 2. Intuitively, the process of generating software directives works as follows: given the average link utilizations generated by LUNA in phase 1 over window intervals T_w , the method first maps these utilization levels to the closest upper discrete link voltage/frequency levels that can support the required bandwidth (Fig. 7a). Then, at each sampling window, starting from the voltage/frequency setting at the beginning of the sampling window, the algorithm lowers voltage/frequency as long as it can return to the voltage/frequency setting required at the start of the next sampling window. Here, voltage/frequency transition delays come from our DVS link model (Fig. 7b). We term this lowest voltage/frequency level the intermediate (V, f) target and the voltage/frequency setting just before the start of the next window as the final (V, f) target. Finally, these two directives are entered for each T_w to create a T_w -based list of DVS software directives that are executed at run-time (Fig. 7c).

In exploring opportunities for (V, f) link reduction, our methodology works by exploiting excess link resources that reside on two axes: time and remaining unutilized link capacity on the horizontal and vertical axes, respectively (see Fig. 7a). When a link's operating frequency is reduced, the time needed by a flit to cross that link is proportionally stretched over the horizontal axis. As our T_w -based approach is designed to satisfy the condition of prohibiting program execution spilling over the subsequent T_w segment, this link now has to transport the same volume of traffic (number of flits) at a slower pace within

```

1: for each link in the network do
2:   for each  $T_w$  do
3:      $\{prevIdx, curIdx, nextIdx\} \leftarrow$  previous, current, and next segment frequency indices
4:     // calculate the number of flits that must be sent
5:      $numFlitsToSend \leftarrow$  current link utilization  $\times T_w$ 
6:     // find lowest acceptable target frequency index for the current time window,
7:     // such that this index is greater than or equal to  $curIdx$  given above
8:      $curCapacity = -1$  // ensure while loop is executed at least once
9:     while  $steadyTime < 0$  or  $curCapacity < numFlitsToSend$  do
10:       $downTime \leftarrow (df + dv) \times (curIdx - prevIdx)$ 
11:       $upTime \leftarrow (df + dv) \times (curIdx - nextIdx)$ 
12:       $steadyTime \leftarrow windowTime - (downTime + upTime)$ 
13:      // Find the number of flits the link can transmit during this time
14:      // window given the frequency and voltage transitions
15:       $curCapacity \leftarrow \sum_{i=prevIdx}^{curIdx} dv \frac{f_i}{f_0} + steadyTime \frac{f_{curIdx}}{f_0} + \sum_{i=nextIdx}^{curIdx} dv \frac{f_i}{f_0}$ 
16:      if  $flitCapacity < numFlitsToSend$  or  $steadyTime < 0$  then
17:         $curIdx \leftarrow curIdx - 1$ 
18:      end if
19:    end while
20:  end for
21: end for

```

Fig. 8. Software directives generating algorithm.

the current T_w segment. This translates to an increase in the link's bandwidth utilization (or a decrease in the remaining link transport capacity) seen on the vertical axis within this T_w segment. The methodology recurses over consecutive (V, f) steps to determine the specific (V, f) link level, which is just enough to satisfy the above condition, at which point a DVS directive reflecting the calculated (V, f) target level for the current T_w is generated.

Mathematically, each link utilization profile is modeled as a discrete-time function, $U[T_w n]$, where $n = 0, 1, 2, \dots, N$ and T_w is the sampling period, or window size. As Fig. 7a shows, $U[t]$ is a step function and is continuous on the interval $[T_w n, T_w(n+1)]$ such that $U[t] = U[T_w n]$ for $t \in [T_w n, T_w(n+1)]$. Its amplitude is the average measured link bandwidth requirement, normalized so that 0 indicates zero utilization and 1 maximum capacity. The amplitude of each $U[T_w n]$ is matched to the next higher discrete frequency f_k , where k is the frequency index ranging from 0 to 9. f_0 denotes full link frequency (1 GHz) and f_9 the smallest available frequency (0.6 GHz).

To create (V, f) pair directives, we apply the algorithm of Fig. 8 to each network link individually to extract the intermediate and final target DVS instructions for every T_w . This is depicted in Fig. 7b and c. Though individual directives are created for each T_w , the calculations carried out in the algorithm need to consider the beginning (end of $T_w(i-1)$) and the end (beginning of $T_w(i+1)$) link utilization levels of $T_w(i)$, between $t = T_w$ and $t = 2T_w$, in our example. Because the calculation at a current window depends on that from the previous window, directives are created in order with respect to time.

4.2.1 Algorithm Details. The algorithm begins by translating the amplitude of each link utilization function $U_j[T_w i]$ for each time segment i into the smallest index k such that $\frac{f_k}{f_0} \geq U_j[T_w i]$. This k is termed $curIdx$. Similarly, $prevIdx$ (beginning of T_w) and $nextIdx$ (end of T_w) for $U[T_w(i-1)]$ and $U[T_w(i+1)]$, respectively, are determined, to meet the utilization targets at

the two ends of T_w . The number of flits that will traverse a link j is the product of the current utilization level (flits/cycle) and window size (cycle count), $numFlitsToSend_j = T_w \times U_j[T_w]$. Figure 7a depicts $numFlitsToSend_j$ as the shaded area between $t = T_w$ and $t = 2T_w$. All these flits captured by $numFlitsToSend_j$ must be able to traverse the link within this same time duration, that is, T_w , with our calculated lower-frequency (V, f) targets.

To create DVS directives, the algorithm tracks the number of discrete (V, f) step-downs and step-ups relative to the starting and ending link utilizations of the current T_w sampling window. As an example, consider the time $t = T_w$ to $t = 2T_w$ of Fig. 7b, where the frequency and voltage are reduced by three steps from $(V, f)_1$ to the intermediate target of $(V, f)_4$, and then they are increased by one step to a final target of $(V, f)_3$, prior to $2T_w$. Specifically, given $prevIdx$, $curIdx$, $nextIdx$, the step-down count from the beginning of a window is $\max(curIdx - prevIdx, 0)$; the step-up count to the end of the segment is $\max(curIdx - nextIdx, 0)$.

To determine the final number of (V, f) step-ups/-downs within the current T_w the algorithm begins at f_{curIdx} discrete frequency level, which can accommodate $U[T_w]$, that is $\frac{f_{curIdx}}{f_0} \geq U[T_w]$, and keeps recursing, with each recursion replacing $CurIdx$ with $CurIdx - 1$ (i.e., next higher discrete-level link frequency), until both: (1) there is just enough time for the calculated number of (V, f) hoppings (i.e., the horizontal component is satisfied), and (2) the utilized link bandwidth can fit into the link's maximum capacity of 1.0 (i.e., the vertical component is satisfied), and $numFlitsToSend$ can be sent within T_w . The horizontal component is measured via $steadyTime$ and the vertical via $flitCapacity$. $steadyTime$ is directly affected by dv and df , the voltage and frequency transition delays, with flits unable to traverse a link during df (see Section 3). These are in terms of nominal router cycles with respect to $f_0 = 1$ GHz, and, the time to step down is $downTime = numStepsDown \times (dv + df)$. Similarly the time to step-up is $upTime = numStepsUp \times (dv + df)$. The horizontal component is measured via $steadyTime = T_w - (downTime + upTime)$ and it is the time spent at the intermediate target frequency and voltage for the current T_w .

The used link capacity $flitCapacity$ is determined by the cumulative effects of up and down (V, f) link transitions. With each (V, f) step, the link's throughput changes directly with frequency and this effect is taken into account to determine whether $flitCapacity \geq numFlitsToSend$ is satisfied within T_w . At each recursive step, the algorithm calculates $flitCapacity$ under the given (V, f) transitions by multiplying each frequency that the link operates at during the current segment by the number of cycles the link spends at that frequency, and summing these numbers. For our example, Fig. 7b depicts this frequency-time product as the diagonally striped area, entitled "steady state."

The intermediate target index decrements by 1, equivalently reducing the number of step-ups/-downs in the current segment, until $steadyTime \geq 0$ and $flitCapacity \geq numFlitsToSend$. Following our example, Fig. 7b shows that in order to reach the intermediate (V, f) target, there must be three step-down hops from $(V, f)_1$ to $(V, f)_4$ and one step-up to meet the final target of $(V, f)_3$, after which the available $steadyTime$ is exhausted and no more (V, f) transitions can occur. Continuing with our example of Fig. 7c, once these two voltage-frequency

Fig. 9. Wormhole router microarchitecture, depicting the physical parameters of an M/D/1 queueing system.

targets are calculated, software directives representing these intermediate and final $(V, f)_k$ levels are created. The algorithm then recurses over the remaining T_w segments to create further two $(V, f)_k$ instructions for each T_w , spanning the entire application duration and all network links.

4.3 Phase 3: Output Buffer Utilization Estimation

In this phase, we make use of queueing theory principles to translate the estimated link utilizations of phase 1 and the target link operating frequencies of phase 2 into output buffer utilization histograms. The goal here is to derive statistics to set application-specific thresholds, which guide our online DVS mechanism of Section 5.

We model each output buffer as an M/D/1 queue and estimate the utilization of each such buffer and for each sampling period T_w . Under standard queueing notation, this refers to a queue that has a Poisson flit arrival rate with average value λ , a deterministic flit service rate of μ , and a single server (link). The service rate is considered to be deterministic since phases 1 and 2 provide information concerning the average link utilizations and operating frequencies over the entire application span. Figure 9 depicts the microarchitecture of a wormhole router with an output link connecting sender (upstream) and receiver (downstream) routers. λ is the rate of traffic on the downstream side of the crossbar. The router operating frequency f_r is constant (1 GHz) with $f_r \geq f_l$, where f_l is the link frequency and is variable over 10 discrete (V, f) levels, as described in Section 3. The following equation is used to determine the average number of occupied buffers (equivalently customers in a queue) for our M/D/1 queueing system [Kleinrock 1975]

$$\bar{N} = \frac{2\rho - \rho^2}{2(1 - \rho)} \quad (5)$$

where $\rho = \frac{\lambda}{\mu}$. This system assumes a constant service rate, however, with a DVS link the service rate varies since the software directives can set any of $(V, f)_{0 \rightarrow 9}$ level pairs with each T_w . To account for this, we parameterize $\mu_{i,j} = \frac{f_{i,j}}{f_0}$, where $f_{i,j}$ is the intermediate target frequency value for link j and time $T_w i$, and f_0 is again the maximum frequency, using information from phases 1 and 2.

Similarly $\rho_{i,j} = \frac{\lambda_{i,j}}{\mu_{i,j}}$. Equation (5) becomes

$$\overline{N}_{i,j} = \frac{2\rho_{i,j} - \rho_{i,j}^2}{2(1 - \rho_{i,j})} \quad (6)$$

In this model, we assume that the output and input buffers have enough capacity to prevent overflows and we use LUNA's estimation of link utilization (per T_w) as the value of $\lambda_{i,j}$. Under wormhole credit-based flow control (see Section 6.1), a flit that has permission to traverse the crossbar must have reserved a position at the output buffer of the current router and a position at the input buffer of the downstream router.

We calculate the average network buffer utilization $\overline{BU}$ by averaging all $\overline{N}_{i,j}$, normalized with respect to the buffer size, over all combinations of network links and sampling periods. Specifically

$$\overline{BU} = \frac{1}{|L|} \sum_{i \in L} \frac{1}{|T|} \sum_{j \in T} \frac{\overline{N}_{i,j}}{|B|} \quad (7)$$

where L and T are the set of links in the network and set of all sampling periods under consideration, respectively, with $|L|$ and $|T|$ denoting the cardinality of these sets. $|B|$ is the buffer size in terms of flits. Next, Section 5 will describe how $\overline{BU}$ determines thresholds that help our online DVS mechanism tune (V, f) pair transitions to maximize network performance.

5. ONLINE DVS HARDWARE MECHANISM

In this section, we describe our online DVS mechanism and its interaction with DVS software directives. The online DVS mechanism has a dual function: (1) it reacts to run-time variabilities in the traffic profile, which can arise as a result of averaging effects or inaccuracies of LUNA and/or compiler-time scheduling/profiling inconsistencies (Section 5.1), and, under the adaptive routing protocol, (2) it orchestrates next-hop routing decisions by choosing the least congested progressive output port (based upon online output buffer utilization measurements) at every network router (Section 5.2).

5.1 Output Buffer Thresholds

To detect online traffic variability, we compare LUNA's statically estimated link utilization levels to the network utilization at run-time. If the latter is greater than the former, DVS directives are throttled to reduce network contention. To direct this online mechanism, we make use of statistics collected via hardware counters.

An obvious way of gathering run-time statistics is tracking link utilizations directly in hardware. However, with practical flow control methods, link utilization only tracks resource utilization well at low- to mid-network traffic levels. When the network is congested, or when the link's (V, f) level is currently set below the required bandwidth, traffic tends to get buffered in input and output buffers and link utilization leans to zero, making it an unsuitable metric [Shang et al. 2002; Soteriou and Peh 2004]. Per-port output buffer utilization, which is

the number of buffers occupied per unit time for a given output link, is a better proxy

$$BU_{p_{out}}[n] = \frac{\sum_{t=-M}^1 F[n-t]}{|B| \times M}, \quad 0 \leq BU_{p_{out}} \leq 1 \quad (8)$$

where n is the sampling time at which we measure the output buffer utilization of output port p_{out} , t is a dummy timing index that spans over the past M router cycles, and $F[n-t]$ is the number of output buffers occupied at time $n-t$. $|B|$ is the output buffer size in terms of flit occupancy and M is the sampling moving window size in terms of router clock cycles. Essentially, $BU_{p_{out}}[n]$ is the average buffer occupancy over the past M router cycles, measured at sampling time n for each output port at a router. $BU_{p_{out}}[n]$ is calculated at every router clock cycle. In our experiments, we set $M = 300$ cycles for two reasons: to detect recent network contention levels and to keep the hardware compact. It is critical to keep in mind the hardware overhead involved in gathering statistics. All of our proposed statistics only require simple hardware counters.⁵

We use LUNA's average network buffer utilization estimation $\overline{BU}$ from Section 4.3 to set thresholds. Our online DVS mechanism compares these thresholds against $BU_{p_{out}}[n]$ when optimizing the execution of software directives to maintain high network performance. When localized link congestion is detected, DVS(V, f) transitioning is backed off or postponed by the mechanism. Unlike previous limited-scope methodologies [Shang et al. 2002; Soteriou and Peh 2004], thresholds are not set based upon some empirical value, but are customized according to the application characteristics. Using LUNA's statistics from Section 4.3, in particular, the average number of occupied output buffers $\frac{\overline{N}_{i,j}}{|B|}$ for every sampling period $T_w i$ for all links j , we are able to draw a histogram of the network's output buffer utilization profile. In this histogram the x-axis shows the normalized output buffer occupancy and the y-axis shows the number of occurrences. The aggregate sum of all occurrence values along the y-axis equals the product of the number of all sampling periods $|T|$ and the total number of network links $|L|$. The derived output buffer utilization profile, though application-dependent, approximates a Gaussianlike distribution with most applications. Using this histogram we can estimate the standard deviation $\overline{\sigma}_{BU}$, for which we base our thresholds to capture the outlier cases of higher buffer utilizations when network contention is likely to occur. Actual run-time utilization measurements of Fig. 10 show histograms of the four output buffers at a randomly chosen router in the 5×5 TRIPS CMP architecture. Though, in this example, some of the histograms skew to the left, the purpose of $\overline{\sigma}_{BU}$ is to capture the outlier cases (closer to $BU_{p_{out}}[n] \simeq 1$) of higher buffer utilizations at which network congestion is most likely to occur. To capture these outlier points, we make use of the following three threshold levels; an explanation of their use follows

$$Th_{BU_{higher}} = \overline{BU} + \alpha \overline{\sigma}_{BU} \quad (9)$$

⁵The power consumed by counters is ignored, as similar hardware has been shown to consume little CMOS area with negligible power [Shang et al. 2002; Soteriou and Peh 2004].

Fig. 10. Histograms showing output buffer utilization profiles of the four output buffer ports at a randomly chosen router in a 5×5 TRIPS mesh inter-ALU interconnection network running the bzip2 benchmark. Each buffer utilization occurrence is measured across 500 cycles of simulation time with the architecture configuration parameters shown in Table II

$$Th_{BU_{high}} = \overline{BU} + \beta \overline{\sigma}_{BU} \quad (10)$$

$$Th_{BU_{low}} = \gamma \overline{BU} \quad (11)$$

where the above constants are set as follows:

$$0 < \beta < \alpha, \quad \gamma < 1 \quad (12)$$

The various thresholds act as follows: when $BU_{pout}[n] > Th_{BU_{high}}$ and $BU_{pout}[n] < Th_{BU_{higher}}$, the algorithm postpones software directives for a retry period t_{retry} . We set $t_{retry} = 240$ cycles in our experiments, $2 \times (dv + df)$, so as to facilitate a near-future retry test for DVS directives execution. When $BU_{pout}[n] > Th_{BU_{higher}}$, the algorithm ignores software directives, and the link transitions to a higher voltage/frequency. If any of the above cases occur and if t_{retry} has elapsed, the link transitions to a lower (V, f) pair approaching the intermediate (V, f) target only if $BU_{pout}[n] < Th_{BU_{low}}$.

Essentially DVS software directives acts as **recommendations** for setting (V, f) target pairs for every T_w , while the online mechanism acts as the final **decider** of setting these levels according to the various conditions just described. The directives present a *lower bound* for (V, f) targets for power optimization, while the online mechanism allows (V, f) pairs to float above this lower bound, conservatively delaying/ignoring power-savings opportunities in favor of performance.

Figure 11 exhibits the behavior of a software-directed network. It shows the (V, f) transitions for two consecutive T_w windows. In the upper part, the link transitions from $(V, f)_1$ to reach the intermediate target $(V, f)_4$. At $(V, f)_3$, $BU_{pout}[n] > Th_{BU_{high}}$ postponing a further down-ramp for a time duration of t_{retry} . When this time has elapsed, it tests for $BU_{pout}[n] < Th_{BU_{low}}$, which is not satisfied, postponing $(V, f)_{3 \rightarrow 4}$ for another t_{retry} . At the next t_{retry} expiration, $BU_{pout}[n]$ satisfies the threshold test and so $(V, f)_{3 \rightarrow 4}$ transitioning is performed. Also note that enough *steadyTime* is present with respect to time $t = T_w$ for performing this transition. At $t_{ramp-up}$ the link starts up-ramping $(V, f)_{4 \rightarrow 3}$ to meet the final $(V, f)_3$ target at the end of $t = T_w$.

In the lower part of Fig. 11, the link starts from $(V, f)_3$ to try to reach the final (equals to the intermediate) $(V, f)_5$ target. From the start $BU_{pout}[n] > Th_{BU_{high}}$ therefore postponing the $(V, f)_{3 \rightarrow 4}$ software directive for a t_{retry} period. When

Fig. 11. Online DVS (V, f) transitioning example.

t_{retry} has elapsed, $BU_{pout}[n]$ is consulted, $BU_{pout}[n] > Th_{BU_{higher}}$ and the link back-offs up-ramping $(V, f)_{3 \rightarrow 2}$. A t_{retry} is elapsed where $BU_{pout}[n] > Th_{BU_{low}}$, setting another t_{retry} at which $BU_{pout}[n] < Th_{BU_{low}}$, and the link down-ramps $(V, f)_{2 \rightarrow 3}$, and subsequently, $(V, f)_{3 \rightarrow 4}$. The link does not reach the target $(V, f)_5$ level since it has exhausted *steadyTime* ($t_{remain} < dv + df$) with respect to $t = 2T_w$. The mechanism will then try to reach the (V, f) intermediate and final target levels within the next T_w (recursive behavior). This example demonstrates the adaptive nature of the online mechanism. It tunes the links to maintain performance over short-term changes in network contention and, at the same time, it attempts to meet the target (V, f) levels to lower power consumption.

5.2 Adaptive Routing Using Online Buffer Utilization Measurements

The output buffer utilization levels ($BU_{pout}[n]$ of Eq. 8) measured at every router, are used dually under adaptive routing: (1) to throttle DVS directives when high contention levels are detected, as in the case of static routing, and (2) to orchestrate per-hop routing decisions. This section provides details for the latter case.

As described in Section 4.1.1, software directives statically generated for run-time adaptive routing need to synchronize with on-the-fly per-hop adaptive routing decisions. With progressive adaptive routing, there are either one or two output link alternatives to be considered at each router. The link with the lower output buffer utilization is chosen for a next-hop packet traversal. For instance, at the time n of reaching a routing decision at a router, if the east port's output

Table II. Configurations of Simulated Architectures

Architecture	Network size	Pipeline length	Input/output buffer size (#flits)	Packet size (flit count)
Raw	4×4	$5/1^a$	5/5	1
TRIPS	5×5	5	10/10	3
Alpha 21364	8×8	13	128/128	16 and 80

^aIn Raw, the static router is a five-stage pipeline. The router decodes switching instructions to set up a path in advance (see Section 6.2). Once the path is established, every flit encounters a unit delay in the sender and receiver router ALUs in ALU to ALU communication, plus the link delay.

buffer utilization is greater than the south port's buffer utilization ($BU_{p_{east}}[n] > BU_{p_{south}}[n]$) and assuming that both routing directions are progressive (see Section 4.1.1), then the south link will be chosen as it exhibits lesser contention. This process will be repeated at the next downstream router until the packet reaches its destination and is finally ejected from the network, i.e., routing decisions are carried out on a per-hop basis.

6. EXPERIMENTAL SETUP AND RESULTS

6.1 Simulator Setup

To evaluate power-latency tradeoffs of our approach, we simulated parallelized code running on three existing network architectures with software-directed DVS links. These architectures are the Raw [Taylor et al. 2004] and TRIPS [Sankaralingam et al. 2003] single-chip multiprocessors (CMPs), and an Alpha 21364-based multi-chip server [Mukherjee et al. 2002]. Details of these architectures are provided in subsequent subsections. Our simulator models an event-driven wormhole switching network with credit-based flow control at the flit level [Duato 1997], extending upon PoPNet [Shang 2002], a publicly available simulator. The simulator supports deterministic and adaptive routing protocols, described in Sections 4.1 and 4.1.1.

The simulator supports k-ary 2-mesh topologies with 1 GHz multi-stage pipelined router cores, each with two virtual channels. Packets are composed of 32-bit flits with each flit transported in 1 link cycle over links of 32 Gbps bandwidth. Each router consists of eight unidirectional channels (four incoming and four outgoing). Table II provides a summary of our simulated network architectures.

In all our experiments, we set $Th_{BU_{higher}} = \overline{BU} + \frac{5}{4}\overline{\sigma}_{BU}$, $Th_{BU_{high}} = \overline{BU} + \frac{3}{4}\overline{\sigma}_{BU}$ and $Th_{BU_{low}} = \frac{3}{4}\overline{BU}$. t_{retry} is set to 240 cycles, which is $2 \times (dv + df)$ and $T_w = 20,000$ cycles. Each simulation is run for the entire trace length measuring up to 10 s of millions of cycles. The metrics considered are latency and power consumption. Latency spans the injection of the head flit of a packet until its tail flit is ejected from the destination router. Link power savings is the ratio of the aggregate power consumption across all links in the network with DVS, divided by the power consumption of all links operating at full frequency (no DVS). The link power savings shown in the subsequent results translate to overall system power savings of approximately 10 to 15% in chip-to-chip parallel systems and to

Fig. 12. Link power savings for a suite of benchmarks running on a Raw CMP.

approximately 10 to 12% in on-chip parallel systems. For instance, in a Mellanox server blade [Mellanox 2006] the link circuitry dissipates 15 W out of the total budget of 40 W (37.5%). As our results from chip-to-chip systems show possible link power savings of up to 38%, the combination of these numbers translate to overall potential system power savings of up to 14.25% when our power-saving methodologies are applied to a Mellanox server blade.

6.2 Raw Architecture with Raw VersaBench Applications

The Raw CMP [Taylor et al. 2004] comprises 16 identical tiles, each with its own pipelined RISC processor, memory, computational resources, and programmable routers, with each tile interconnected to its closest neighbor in a 4×4 mesh array. It uses an ISA, where all the raw hardware resources, including interconnect wire delays, are fully exposed to the software interface, allowing the compiler to optimize program execution by mapping and scheduling parallelized code onto each tile.

An interesting feature of this architecture is the static network that allows the implementation of compiler-directed routing among tiles. This network provides ordered, flow-controlled and reliable transfer of single-word operands and data streams between functional units. The static router at each tile has its own instruction memory and is thus programmable by the compiler. This memory holds a corresponding switching instruction for each operand to be sent on the network, with instructions programmed statically in advance during compile-time and then cached in the memory. Thus, the static routers collectively configure the entire network on a cycle-by-cycle basis.

To evaluate the effectiveness of our software-directed DVS methodology, we ran binaries compiled by the Raw compiler on the Raw cycle-accurate simulator, which accurately matches hardware timing, and extracted communication traces from Raw's static network. These traces contain all the information required by our methodology: the router switching time stamp of operands, and operand source and destination tiles.

Figure 12 shows the link power savings and Fig. 13 shows the corresponding network latency penalties of nine benchmarks running on a Raw CMP for deterministic (static) and adaptive routing. This suite includes a mix of

Fig. 13. Network latency penalty for a suite of benchmarks running on a Raw CMP.

streaming (streams), bit-level (802.11a_encoder), SPECINT2000 [SPEC 2006] (164.gzip), and MediaBench [Lee et al. 1997] (adpcm) benchmarks. With static routing, we observe high link power savings across all benchmarks, 49.4%, on average, with 2.8% network latency penalty, on average. With adaptive routing the network latency penalty improves to an average of 2.65% while power savings increase to an average of 56.8%. This combination of lower network latency and higher link power savings under adaptive versus static routing occurs across most benchmarks tested, with an exception of *fir* and *164.gzip*, because of possible minor mismatches between statically generated DVS directives for adaptive routing and actual online routing decisions. For all other benchmarks, the power-latency improvements of adaptive versus static routing is because of packet routing over less congested paths. Lesser power savings and corresponding impact on latency were observed for *fir*, as higher network resource utilization was observed, showing our methodology efficiently adapting to increased network usage demands. The relative latency increase—relative power savings product, $L_{rel} \times P_{rel}$, a common metric for capturing power-delay [Alonso et al. 2005], is 0.52 with static routing and 0.44 under adaptive routing (without DVS $L_{rel} \times P_{rel} = 1$), indicating excellent power savings with little performance impact.

6.3 TRIPS Architecture with SPEC and MediaBench Benchmarks

To further evaluate the effectiveness of our proposed power-aware methodology, we obtained network traces from the TRIPS CMP [Sankaralingam et al. 2003]. The TRIPS CMP consists of four large, coarse-grained element cores each of which is an instantiation of the grid processor architecture (GPA) containing an ALU execution array and local L1 memory tiles interconnected via a 5×5 mesh network.

TRIPS network packets carry data (operands for instructions or addresses to memory) and status information associated with them. Network traces were obtained from simulations of a suite of sixteen SPEC [SPEC 2006] and MediaBench [Lee et al. 1997] benchmarks. The traces are, in general, very bursty, exhibiting high temporal variance along with spatial injection variability among routers. Large bursts of packets are injected at times (see top-most

Fig. 14. Link power savings for a suite of benchmarks running on a TRIPS CMP.

Fig. 15. Network latency penalty for a suite of benchmarks running on a TRIPS CMP.

left histogram in Fig. 20), and no packets are injected at other times, a scenario which presents interesting opportunities for power optimization.

Figure 14 shows the link power savings and Fig. 15 shows the corresponding impact on network latency of the 16 benchmarks running on TRIPS for both static and adaptive routing. Under static routing, we again observe high link power savings across all benchmarks, 70.2% on average, with 1.16% latency penalty on average. Corresponding values under adaptive routing are 69.3% power savings, a slight drop as compared to static routing and 0.88% increase in latency. The worst-case latency penalty (static routing), 5.22% was observed with the mpeg2encode benchmark. The $L_{rel} \times P_{rel}$ product is 0.301 under static routing and 0.31 for adaptive routing, indicating excellent power savings with little performance impact. It is interesting to note that though adaptive routing improves performance, the decrease in power savings under adaptive routing as opposed to static routing leads to a smaller energy-delay product than that produced under static routing.

6.3.1 Discussion. The TRIPS traces, for both static and adaptive routing, exhibit higher power savings and smaller impact on performance as compared to the Raw traces because of a couple of reasons. First the Raw static network exhibited a higher utilization ratio as compared to the TRIPS network, therefore providing fewer opportunities for (V, f) pair down-ramping for further power reduction. Second, the TRIPS traffic exhibited greater spatial and temporal variance than the Raw traffic, with less frequent bursts (greater gaps) of

Fig. 16. Network link power savings for SPLASH-2 benchmarks.

Fig. 17. Network latency penalty for SPLASH-2 benchmarks.

injected traffic, therefore providing greater opportunities for power optimization along with smaller latency increases.

6.4 Alpha 21364 Architecture with SPLASH-2 Benchmarks

Finally, we apply the proposed software directives to a chip-to-chip network. We ran three benchmarks from the SPLASH-2 suite [Woo et al. 1995] on the RSIM [Pai et al. 1997] shared-memory cache-coherent multiprocessor infrastructure, modeling Alpha 21364 processor nodes and cache coherence models, and collected their traffic traces. Note that unlike the previous two studies, the multiprocessor architecture is not modeled using its original simulator, so clearly, the traces will not precisely match that of Alpha 21364. We tried our best to mimic published Alpha 21364 parameters with RSIM parameters.

Figures 16 and 17 show the link power savings and the corresponding latency penalties for the three tested SPLASH-2 benchmarks, respectively, with static and adaptive routing. Network link power savings are 20.75% and 30.71%, on average, for static and adaptive routing, respectively. Corresponding latency penalties are 5.54% and 3.71%, on average, for static and adaptive routing, respectively. The $L_{rel} \times P_{rel}$ is 0.836 for static and 0.625 for adaptive routing indicating good power-performance responsiveness of our methodology.

6.4.1 Discussion. Here we observe smaller power savings than those observed in the two on-chip architectures for both static and adaptive routing. This is due to a number of reasons. The SPLASH-2 benchmarks are designed to evaluate off-chip shared-address memory architectures where the packet

size is considerably larger than typical packet sizes in on-chip architectures. Network traffic consists of either 16-flit packets, containing control information such as requests for a cache line and coherence protocol actions, or 80-flit packets containing replies to requests that can carry contents of cache lines. Traffic patterns were also observed to be considerably less bursty and more uniform than in on-chip traffic, translating to decreased spatial and temporal variability. In summary the SPLASH-2 network traffic imposes increased communication demands onto the network links with sparser opportunities for lowering power. Though smaller power savings were observed here, we see our power-aware policies adapting to a wide spectrum of applications and network utilization demands, lowering power consumption while maintaining high interconnection network performance. It must also be noted that in relative proportions, adaptive routing benefits the 8×8 mesh 21364 architecture as compared to the substantially smaller 4×4 Raw and 5×5 TRIPS architectures. In conclusion, the results suggest that adaptive routing can harness greater power-performance benefits when applied to larger architectures. These architectures contain longer routing paths, thus greater path traversal diversity is explored under adaptive routing, leading to higher maintainable performance as compared to static routing, where packet link traversals are predetermined and path diversity is, therefore, restricted.

6.5 Discussion: Threshold Perturbation

Here, we demonstrate the relative invariance of our techniques to variations in thresholds and sampling windows when applied to a range of benchmarks, concurrently assessing our approach's resilience to four combinations of thresholds and three LUNA sampling windows (T_w). Static routing is used. Referring to Eq. (9–12), we use three (α, β, γ) 3-tuples, $(1, \frac{1}{2}, \frac{3}{4})$, $(\frac{5}{4}, \frac{3}{4}, \frac{3}{4})$ and $(\frac{5}{4}, \frac{3}{4}, \frac{1}{2})$ correspondingly for sets 1 to 3. For each set, T_w is placed at 5,000 (5k), 20,000 (20k), and 50,000 (50k) system cycles. Set 1 presents more aggressive behavior with greater power savings expected along with higher impact on latency. Set 2 presents more responsive behavior with smaller impact on power and latencies expected. Set 3 uses the same α and β as set 1, however with a smaller γ . In this latter case, (V, f) backing-off is prolonged since the traffic has to settle at a higher level before the link can retry to transition to a lower (V, f) level.

It is clear from the Fig. 18 that consistent (to each application) high power savings, 73.8%, on average, can be achieved for the three TRIPS applications, even with deterministic (static) routing. Foremost, Fig. 19 shows consistent (to each application) low latency impact, 2.74%, on average. It is evident that the latency penalty and power savings are almost invariant to T_w —the sampling period does not affect power-performance. Relatively high consistency in power-performance results is also observed with the three combinations of thresholds, with very small variance observed. This is because the thresholds are customized to each application, and are based upon the expected average of output buffer utilization $\overline{BU}$ and its standard deviation $\overline{\sigma}_{BU}$, capturing only the outlier cases of higher buffer utilizations at which network congestion is most likely to occur. These results stand in stark contrast to the behavior of hardware limited-scope approaches as shown in Section 2.2 (see Figs. 1 and 2).

Fig. 18. Link power savings of software-directed DVS for three trace benchmarks from the TRIPS CMP with static routing.

Fig. 19. Network latency penalty of software-directed DVS for three trace benchmarks from the TRIPS CMP with static routing.

Fig. 20. Profile of perturbed mpeg2encode from the TRIPS CMP.

6.6 Discussion: Traffic Perturbation

Though parallelizing compilers such as Rawcc [Lee et al. 1998] statically schedule instructions across the network in both space and time, the estimated message flow timing information is not always exact. Because of the presence of dynamic events such as data dependencies, dynamic memory references, and I/O operations, some of the message flows may not be routed at preset times.

We evaluate the resilience of our proposed technique to inaccuracies in message flow injection/arrival timings by artificially perturbing the entire application flow. We note that the traffic perturbation presented here does not directly model the true reactive or dynamic effects caused by data dependencies among the various routing messages that cause “time shifting” in routing traffic, but emulates this time-shifting effect through artificial time-domain traffic scrambling. Since the methodology of extracting software directives (see Section 4) and the online hardware mechanism’s (see Section 5) adaptivity to traffic variations both depend on the volume of routing traffic and do not require the knowledge of the actual data contained in packets, time-shifting caused by data dependencies is unnecessary, though time shifting caused by data dependencies would probably improve the accuracy of results presented here.

Fig. 21. Profile of perturbed 164.zip from the Raw CMP.

We consider the compiler-derived injection time of each injected packet burst (i.e., each injection burst is shuffled to another injection time in its entirety) as the mean value of a normal distribution and we change this time within $\pm 3\sigma_d$ (standard deviation flow displacement). We test three cases where σ_d can take a value of 10, 100, and 1,000 cycles. Note that $\sigma_d = 0$ indicates no timing perturbation. We set $T_w = 5,000$ cycles, to allow minimal *steadyTime* for our DVS online mechanism to adjust to any expected traffic bursts. Note that these settings present a highly challenging message flow timing inexactness—in reality, message shuffling of such a magnitude is expected to occur infrequently [Lee et al. 1998]. We apply this scenario to all 28 benchmarks used in this paper under adaptive routing: 16 running on the TRIPS CMP architecture, 9 running on the Raw CMP, and the 3 SPLASH-2 benchmarks running on an Alpha 21364 mesh interconnected 8×8 network.

Figures 22 and 23 correspondingly show the impact of message flow perturbation upon TRIPS’s network latency and link power savings. The general trend, with ammp and art forming exceptions, is that with increased traffic perturbation latencies increase while power savings drop minimally. Under most benchmarks the latency penalties are greatest at $\sigma_d = 100$ cycles, while at $\sigma_d = 1,000$ cycles network latency penalties are still larger than when traffic perturbation is not applied or at $\sigma_d = 10$ cycles, but smaller than the latency

Fig. 22. Network latency penalty for TRIPS CMP perturbed traffic (adaptive routing).

Fig. 23. Network link power savings for TRIPS CMP perturbed traffic (adaptive routing).

penalties observed at $\sigma_d = 100$ cycles. Figure 20, which shows the mpeg2encode benchmark running on the TRIPS CMP, helps to explain this phenomenon: the original traffic flow possesses high temporal variability with up to 80-packet injection bursts. As these bursts are displaced with a larger σ_d , DVS software directives with out-of-date (V, f) (or better, directives with “wrong” execution time stamps and corresponding (V, f) level targets) target levels are executed causing larger latency penalties. As the online back-off mechanism continuously tries to recover from congestion by backing-off to higher (V, f) levels, power savings are also slightly reduced. With $\sigma_d = 1,000$ cycles, though, some of the traffic injection bursts are displaced in such a way that the overall traffic smooths out. The number of back-offs is reduced, leading to a slightly better power-performance than the one observed with $\sigma_d = 100$ cycles as Fig. 20 shows. Interestingly, applications mgrid and tomcatv with traffic perturbation of $\sigma_d = 1,000$ cycles show latency improvements as compared to their original traffic network injection profiles, i.e., with $\sigma_d = 0$ cycles. However, these latency improvements are observed along with lower link power savings. Average network latency penalties and link power savings pairs across all benchmarks are at (1.51% and 69.1%), (3.27% and 69.1%), and (2.20% and 68.7%) for $\sigma_d = 10, 100$, and 1,000 cycles, respectively.

Similar trends are seen with the benchmarks running on the Raw CMP architecture: as traffic is displaced with a greater σ_d , network latency penalties increase, with most of them peaking at $\sigma_d = 100$ cycles and slightly dropping with $\sigma_d = 1,000$ cycles. This can be explained using the fact that at $\sigma_d = 1,000$

Fig. 24. Network latency penalty for Raw CMP perturbed traffic (adaptive routing).

Fig. 25. Network link power savings for Raw CMP perturbed traffic (adaptive routing).

cycles traffic injection levels are smoothed over time as injection peaks spread out temporarily, filling inactivity gaps with short injected traffic bursts as compared to when $\sigma_d = 100$ cycles; this causes fewer voltage/frequency back-offs and therefore the number of (V, f) transitions which introduce overheads is reduced, leading to better network performance. For example, this traffic spreading effect of the 164.gzip application running on the Raw CMP with $\sigma_d = 1,000$ cycles is shown in Fig. 21, bearing traffic temporal behavior similarities to those shown in Fig. 20, which depicts a profile of TRIPS's mpeg2encode perturbed benchmark. Interestingly, applications 802.11a_encoder, 8b_10b_encode, adpcm and fft exhibit lower network latencies at $\sigma_d = 1,000$ as compared to those at $\sigma_d = 0$ cycles, i.e., the original traffic with no perturbation. Link power savings also drop progressively with larger σ_d s, as the online back-off mechanism throttles DVS software directives to recover from congestion. Figures 24 and 25 show the impact of traffic perturbation upon the network latency penalty and link power savings, respectively. Average latency penalties and link power savings pairs across all benchmarks are at (4.60% and 56.24%), (6.86% and 56.02%), and (2.79% and 55.24%) for $\sigma_d = 10, 100$, and 1,000 cycles respectively.

Finally, similar latency-power trends (Figs. 26 and 27) with traffic perturbation to those seen under the TRIPS and Raw CMPs are observed with the SPLASH-2 benchmarks running on the 21364 network architecture. Average latency penalties and power savings pairs across all benchmarks are at (3.89% and 29.66%), (6.22% and 28.80%), and (4.50% and 27.72%) for $\sigma_d = 10, 100$, and

Fig. 26. Network latency penalty for SPLASH-2 perturbed traffic (adaptive routing).

Fig. 27. Network link power savings for SPLASH-2 perturbed traffic (adaptive routing).

1,000 cycles, respectively. Note that even though, under all three architectures, network latencies increase and link power savings drop slightly with increased levels of traffic perturbation, results for latencies here are up to an order of magnitude better as compared to previously proposed hardware methods (discussed in Section 2.2). These results verify the online software directive back-off mechanism's positive impact on sustaining good network power-performance.

7. CONCLUSIONS AND FUTURE WORK

This paper proposes software-driven power-aware techniques to address the critical issue of interconnection network power consumption. These techniques form an extension to the parallelizing compiler flow, statically generating DVS instructions that later direct DVS link (V, f) settings during application run-time. The proposed approach presents a number of advantages over recent related work, as it handles arbitrary network flows and adapts to compile-time software directives inaccuracies using hardware mechanism adaptations to match run-time traffic flow variability. Thus, DVS directives are in a position to tailor power-performance of a running application, fine-tuning (V, f) transitions to match message flow network utilization requirements. Experimental results show power reduction levels of up to 76.3%, with a minute increase in latency, 6.78% at most, for a spectrum of benchmark suites running on three existing network architectures.

This paper concentrates on investigating the impact on network link power-performance of parallel systems. Future interesting avenues include the exploration of the impact of software directives upon the power-performance of entire on-chip multicore or chip-to-chip parallel systems. Software directives can also be used to control power-performance of additional network

components, such as buffers, and crossbar and arbitration circuitry to construct fully power-aware interconnection networks. In addition, application-specific tuning that will capture the reactive or dynamic effects caused by data dependencies among the various routing messages that can cause “time-shifting” in routing traffic can also be explored. Software-directed power-aware networks can be further explored by investigating the impact of software directives upon future multiprogrammed on-chip networks, where multiple applications are to run concurrently, sharing network links, buffers, switching and arbitration resources. This will present new challenges, as accurate application of run-time sharing and network router and link resource superimposition must be captured, accounted for, and predicted so as to maximize power-performance. This will require the cooperation of a power-aware multitasking operating system, where multithread-level scheduling knowledge will be used synergistically with software directives to explore opportunities for interconnection network power-performance optimizations.

With parallel compilers already presenting sophisticated performance enhancements but fairly limited in power-aware optimizations, and operating systems having limited power-aware scope, there remain numerous opportunities for further power-performance exploration in parallel systems, both in the inter- and intra-chip domains.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers for their valuable suggestions and comments. The authors also wish to thank the TRIPS Team at the University of Texas at Austin for supplying the traffic traces of various benchmarks, the MIT Raw Team and especially Michael Taylor for supplying the Raw simulator and the VersaBench applications, Amit Kumar of Princeton University for his help on compiling the Raw applications and for running SPLASH-2 benchmarks on RSIM, and Kevin Ko for detailed proofreading of this paper. This work was supported in part by the National Science Foundation (NSF) under contracts CCR-0237540 (CAREER), CCR-0324891 (ITR) and CNS-0305617 as well as the MARCO Gigascale Systems Research Center.

REFERENCES

- ALONSO, M., MARTÍNEZ, J.-M., SANTONJA, V., LÓPEZ, P., AND DUATO, J. 2005. Power saving in regular interconnection networks built with high-degree switches. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium*. 5b–11b.
- BENINI, L. AND MICHELI, G. D. 2001. Powering networks on chips: Energy-efficient and reliable interconnect design for SoCs. In *Proceedings of the 14th International Symposium on System Synthesis*. 33–38.
- BURD, T. D. AND BRODERSEN, R. W. 2000. Design issues for dynamic voltage scaling. In *Proceedings of the 5th International Symposium on Low Power Electronics and Design*. 9–14.
- CHEN, G., LI, F., AND KANDEMIR, M. 2006. Compiler-directed channel allocation for saving power in on-chip networks. In *Proceedings of the 33rd Annual Symposium on Principles of Programming Languages*. 194–205.
- CHEN, X., PEH, L.-S., WEI, G.-Y., HUANG, Y.-K., AND PRUCNAL, P. R. 2005. Exploring the design space of power-aware opto-electronic network systems. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. 120–131.

- DALLY, W. J. AND TOWLES, B. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference*. 684–689.
- DUATO, J. 1997. A theory of fault-tolerant routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems* 8, 8 (Aug.), 790–802.
- EISLEY, N. AND PEH, L.-S. 2004. High-level power analysis for on-chip networks. In *Proceedings of the 7th International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. 104–115. LUNA, available [online]: <http://www.princeton.edu/~eisley/LUNA.html>.
- HU, J. AND MARCULESCU, R. 2004. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. Vol. 1. 234–239.
- InfiniBand 2006. InfiniBand Trade Alliance. The InfiniBand Architecture. Available [online]: <http://www.infinibandta.org>.
- ITRS. 2005. Semiconductor Industry Association. International Technology Roadmap for Semiconductors, available [online]: <http://www.itrs.net/Common/2005ITRS/Home2005.htm>.
- JALABERT, A., MURALI, S., BENINI, L., AND MICHELI, G. D. 2004. $\times$ pipesCompiler: A tool for instantiating application specific networks on chip. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Vol. 2. 884–889.
- KADAYIF, I., KANDEMIR, M., AND KOLCU, I. 2004. Exploiting processor workload heterogeneity for reducing energy consumption in chip multiprocessors. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Vol. 2. 1158–1163.
- KAUL, H., SYLVESTER, D., BLAAUW, D., AUSTIN, T., AND MUDGE, T. 2005. DVS for on-chip bus designs based on timing error correction. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Vol. 1. 80–85.
- KIM, E. J., YUM, K. H., LINK, G. M., VIJAYKRISHNAN, N., KANDEMIR, M., IRWIN, M. J., YOUSIF, M., AND DAS, C. R. 2003. Energy optimization techniques in cluster interconnects. In *Proceedings of the 8th International Symposium on Low Power Electronics and Design*. 459–464.
- KIM, J. AND HOROWITZ, M. 2002. Adaptive supply serial links with sub-1V operation and per-pin clock recovery. In *Proceedings of the International Solid-State Circuits Conference*. 1403–1413.
- KIM, J. S., TAYLOR, M. B., MILLER, J., AND WENTZLAFF, D. 2003. Energy characterization of a tiled architecture processor with on-chip networks. In *Proceedings of the 8th International Symposium on Low Power Electronics and Design*. 424–427.
- KLEINROCK, L. 1975. *Queueing Systems*. Vol. 1. Wiley (Interscience). New York.
- LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the 30th International Symposium on Microarchitecture*. 330–337.
- LEE, W., BARUA, R., SRIKRISHNA, D., BABB, J., SARKAR, V., AND AMARASINGHE, S. 1998. Space-time scheduling of instruction-level parallelism on a Raw machine. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*. 46–57.
- LI, F., CHEN, G., AND KANDEMIR, M. 2005. Compiler-directed voltage scaling on communication links for reducing power consumption. In *Proceedings of the International Conference on Computer Aided Design*. 456–460.
- LUO, J., PEH, L.-S., AND JHA, N. K. 2003. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. In *Proceedings of the Design Automation and Test in Europe Conference and Exhibition*, Vol. 1. 1150–1151.
- MELLANOX. 2006. Mellanox Technologies Performance, Price, Power, Volume Metric (PPPV). Available [online]: <http://www.mellanox.com/products/shared/PPPV.pdf>.
- MUKHERJEE, S. S., BANNON, P., LANG, S., SPINK, A., AND WEBB, D. 2002. The Alpha 21364 network architecture. *IEEE Micro* 22, 1 (Feb.), 26–35.
- PAI, V. S., RANGANATHAN, P., AND ADVE, S. V. 1997. RSIM: An execution-driven simulator for ILP-based shared-memory multiprocessors and uniprocessors. *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter* 35, 11 (Oct.), 37–48.
- PATEL, C., CHAI, S., YALAMANCHILI, S., AND SCHIMMEL, D. 1997. Power-constrained design of multiprocessor interconnection networks. In *Proceedings of the 15th International Conference on Computer Design*. 408–416.

- ROBERT, R. W., FRENCH, S., WILSON, C. S., AMARASINGHE, S. P., ANDERSON, J. M., TJIANG, S. W. K., LIAO, S.-W., TSENG, C.-W., HALL, M. W., LAM, M. S., AND HENNESSY, J. L. 1996. SUIF: An infrastructure for research on parallelizing and optimizing compilers. *ACM SIGPLAN Notices* 29, 12 (Dec.), 31–37.
- SANKARALINGAM, K., NAGARAJAN, R., LIU, H., KIM, C., HUH, J., BURGER, D., KECKLER, S. W., AND MOORE, C. R. 2003. Exploiting ILP, TLP and DLP with the polymorphous TRIPS architecture. In *Proceedings of the 30th International Symposium on Computer Architecture*. 422–433.
- SAPUTRA, H., KANDEMIR, M., VIJAYKRISHNAN, N., IRWIN, M. J., HU, J. S., HSU, C.-H., AND KREMER, U. 2002. Energy-conscious compilation based on voltage scaling. In *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems*. 2–11.
- SHANG, L. 2002. PoPNet simulator. Available [online]: <http://www.princeton.edu/~lshang/popnet.html>.
- SHANG, L., PEH, L.-S., AND JHA, N. K. 2002. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*. 79–90.
- SHIN, D. AND KIM, J. 2004. Power-aware communication optimization for networks-on-chips with voltage scalable links. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. 170–175.
- SOTERIOU, V. AND PEH, L.-S. 2004. Design-space exploration of power-aware on/off interconnection networks. In *Proceedings of the 22nd International Conference on Computer Design*. 510–517.
- SOTERIOU, V., EISLEY, N., AND PEH, L.-S. 2005. Software-directed power-aware interconnection networks. In *Proceedings of the 8th International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. 274–285.
- SPEC. 2006. The Standard Performance Evaluation Corporation. Available [online]: <http://www.spec.org/>.
- STINE, J. M. AND CARTER, N. P. 2004. Comparing adaptive routing and dynamic voltage scaling for link power reduction. *Computer Architecture Letters* 3, 1 (June), 4–8.
- TAYLOR, M. B., LEE, W., MILLER, J., WENTZLAFF, D., BRATT, I., GREENWALD, B., HOFFMANN, H., JOHNSON, P., KIM, J., PSOTA, J., SARAF, A., SHNIDMAN, N., STRUMPEN, V., FRANK, M., AMARASINGHE, S., AND AGARWAL, A. 2004. Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In *Proceedings of the 31st International Symposium on Computer Architecture*. 2–13.
- WANG, H., ZHU, X., PEH, L.-S., AND MALIK, S. 2002. Orion: A power-performance simulator for interconnection networks. In *Proceedings of the 35th International Symposium on Microarchitecture*. 294–305.
- WEI, G.-Y., KIM, J., LIU, D., SIDIROPOULOS, S., AND HOROWITZ, M. A. 2000. A variable-frequency parallel I/O interface with adaptive power-supply regulation. *Solid-State Circuits* 35, 11 (Nov.), 1600–1610.
- WOO, S. C., OHARA, M. O., TORRIE, E., SINGHAND, J. P., AND GUPTA, A. 1995. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*. 24–36.
- WORM, F., LENNE, P., THIRAN, P., AND MICHELI, G. D. 2002. An adaptive low-power transmission scheme for on-chip networks. In *Proceedings of the 14th International Symposium on System Synthesis*. 92–100.
- XIE, F., MARTONOSI, M., AND MALIK, S. 2003. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. 49–62.

Received March 2006; revised July 2006; accepted August , 2006