High-Level Power Analysis for On-Chip Networks

Noel Eisley and Li-Shiuan Peh
Department of Electrical Engineering
Princeton University, Princeton, NJ 08544.
{eisley, peh}@princeton.edu

ABSTRACT

As on-chip networks become prevalent in multiprocessor systems-on-a-chip and multi-core processors, they will be an integral part of the design flow of such systems. With power increasingly the primary constraint in chips, the tool chain in systems design, from simulation infrastructures to compilers and synthesis frameworks, needs to take network power into account, motivating the need for early-stage communication power analysis.

While there has been substantial research in network performance analysis that enabled critical insights into network design, no power analysis frameworks for networks exist. In this paper, we propose such a framework that takes as input message flows, and derives a power profile of the network fabric, capturing both the spatial variance across the network fabric as well as the temporal variance across application execution time. Our analysis is based on link utilization as the unit of abstraction for network power, with contention among message flows modeled through propagation of overflow areas in link utilization functions. When validated against Orion, a cycle-accurate network power simulator, we show that relative trends in network power are well-preserved. We then demonstrate potential uses of our analysis framework through three case studies, from speedup of multiprocessor and network simulations, to facilitating power-aware communication synthesis and compiler code placement.

Categories and Subject Descriptors: B.8.2 [Performance and Reliability]: Performance Analysis and Design Aids; C.2.0 [Computer-Communication Networks]: General

General Terms: Design, Theory

Keywords: Power analysis, link utilization, systems-on-a-chip (SoC), simulation

1. INTRODUCTION

In application-specific embedded systems, multiprocessor systems-on-a-chip (MPSoCs) are becoming increasingly prevalent as chip complexity increases. The same trend exists in general-purpose processor designs, with several multi-core architecture designs rolled out in industry [35] as well as in academia [30, 19, 21]. While buses were the traditional interconnect fabric in these chips, on-chip networks have become increasingly the de facto communication backbone as the number of processing elements scale up in MPSoCs and multi-core processors [9, 22].

This trend towards multiprocessor architectures has driven the corresponding tool chain in embedded SoC design to factor in the on-chip network fabric. Within the SoC synthesis flow, researchers recently proposed communication synthesis tools that derive network designs that are tailored to the application driver's characteristics under user-specified design constraints [20]. Compilers for multiprocessor chips now factor in network characteristics when determining the placement of programs onto the chip [16, 21]. Simulation infrastructures for multiprocessor chips enable detailed exploration of networking elements alongside processing and memory elements [14, 30].

With power becoming increasingly the first-order constraint in these billion-transistor chips, and the network fabric taking up a substantial portion of chip power budget [15, 4], it is no longer sufficient for these tools to only consider network delay and area. Network power has to be a critical component of compilers, synthesis frameworks, and simulation infrastructures.

One of the first network power models proposed was Orion [31], a cycle-accurate network power-performance simulator where each network component's capacitance is derived based on architectural parameters, and activities at each cycle trigger calculations of network power. While Orion is in use in several simulation infrastructures [14, 28] for network power estimates, its cycle-by-cycle detailed modeling and simulation can further exacerbate the complexity of multiprocessor simulators and stretch already-substantial simulation times even further [23]. As for compilers and synthesis engines where early-stage power comparisons of applications and task graphs are needed, simple energy estimates such as hop counts are typically used, even though they poorly reflect average power, and do not capture the spatial and temporal variance in a network's power profile. Temporal variations in chip power is critical for battery life analysis in embedded SoCs [17], while chip power profiles that capture spatial and temporal power variations help in identifying thermal hotspots [25]. This prompts us to explore high-level network power analysis that can facilitate early-stage comparisons of different network traffic and designs.

While network performance analysis based on queueing, markov and probabilistic models has been studied rigorously in the past [2, 7, 1], enabling early-stage insights into network design tradeoffs, network power analysis has not been explored, to the best of our knowledge. In this paper, we propose a framework for network power analysis that uses link utilization as the unit of abstraction for network utilization and power, capturing power variations both spatially across the network fabric and temporally across application

CASES'04 September 22–25, 2004, Washington, DC, USA.

execution time. Our analysis takes as input message flows (their sources and destinations along with injection rate across time), and transforms these into link utilization functions, taking into account contention among flows. When validated against Orion, our power analysis derives power profiles that match closely with that of Orion across a range of SPEC and MediaBench benchmark traffic traces.

We see our high-level power analysis framework allowing network power to be readily incorporated into simulation infrastructures, communication synthesis frameworks, and compilers. We explore each in turn, demonstrating with simple examples how power analysis can be used to speed up multiprocessor and network simulations, facilitate joint power optimization of application task graph mapping and network design in embedded systems synthesis, and enable power-aware program placement in compilers for tiled chip multiprocessors.

The rest of this paper is organized as follows. In Section 2 we discuss prior related work. Section 3 follows with a discussion of various candidates for abstracting network power, explaining our choice of link utilization as the unit of abstraction. Section 4 presents the details of our power analysis framework, rounding it off with a validation against Orion with actual on-chip network traces. Three case studies outlining potential ways of leveraging our power analysis framework are described in Section 5, and Section 6 concludes the paper.

2. RELATED WORK

Communication power modeling. Circuit-level power models of interconnects estimate the power consumption of local and global wires, taking into account detailed technology parameters [11]. RTL-level models can also be applied to estimate the power consumed by routers in networks. However, these low-level power models require detailed network circuits and design information that is not available at early-stage exploration.

At the next higher level, architectural network leakage and dynamic power models [31, 6] derive power estimates based only on architectural parameters such as network size, topology, number of buffers, ports, packet and flit size etc. In Orion, detailed capacitance equations are derived for various router and link components, assuming specific circuit designs for each component, with underlying transistor sizes and layout scaled based on CACTI [32] and [11]. These equations are then plugged into a cycle-accurate simulator so actual network activity counts will trigger specific capacitance calculations and derive dynamic power estimates. Cycle-accurate power simulations, however, can lead to prohibitively long run-times, and require cycle-by-cycle traffic information, making it hard to incorporate them within communication synthesis and compilers.

Processor power modeling at different levels. At one end, there has been work in instruction-level power modeling [27]. Here the authors empirically determine the current that the processor consumes for each type of instruction by creating loops of a single instruction and then physically measuring the current across the chip. From these values, the power profile of a program can be estimated. As this method requires current values after a chip has already been fabricated, it cannot be factored in at early-stage design. A survey of software power modeling of processor power is available at [18]. At a lower level, there are cycle-accurate architectural power simulators, such as Wattch [5], which models each functional unit of a processor with detailed capacitance equations that are triggered with Simplescalar [36] simulations. These models provide accurate power estimates, but due to the detailed nature of their simulations, can have prohibitively long run-times. Circuit-level power modeling tools similar to those used for communication power analysis can be used as well.

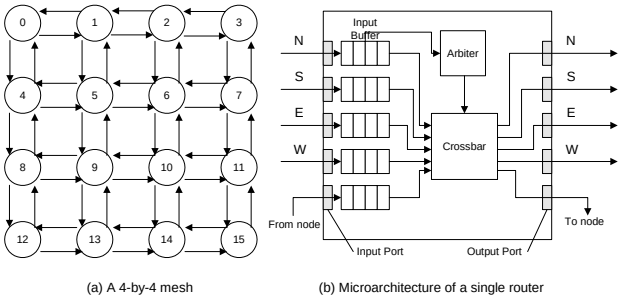


Figure 1: A 4-by4 mesh on-chip network along with the high-level microarchitecture of each router.

Performance analysis. Link (channel) utilization is also derived and used in network performance analysis frameworks to estimate average network delay and throughput [2, 1]. However, as the goal is to identify steady-state network delay and throughput, estimates of average link utilization is sufficient – there is no need to derive the utilization of every link across different points in time. For instance, in [1], mean value analysis is leveraged so a realistic interconnection network and traffic distribution can be modeled with queuing theory. Clearly, this does not suffice for power analysis, where we need to pinpoint when a hotspot occurs, and at which specific part of the network (chip). This need to capture the spatial and temporal variance in network utilization is the key difference between network power and performance analysis.

3. UNIT OF ABSTRACTION

While several different units of abstraction have been proposed for capturing network latency, throughput and energy, none has been explored for network power. Here, we walk through previously proposed units that can potentially be used to estimate network power, and explain their deficiencies. We then propose link utilization as a unit of abstraction for network power.

The network in Figure 1 will be used as a basis for explaining the various units. It shows a 16-node on-chip mesh network, along with a typical router microarchitecture at each hop. To send a message from a source to a destination node, it is usually broken down into packets; each can be independently routed. Packets are then segmented into fixed-sized flits, injected into the router associated with the source node, and transmitted through multiple hops in the network to the destination router and node. A typical router consists of several microarchitectural components – buffers that house flits at input ports, routing logic that steers flits towards appropriate output ports along its way to its destination, arbiters that regulate access to the crossbar, and a crossbar which transports flits from input to output ports.

3.1 Hop count

Hop count, H, the number of hops a packet traverses through on average, is commonly used as an abstraction for network delay, as it approximates zero-load latency, $L0_f$, the minimum delay of a body flit through the network – when there is no contention:

$$L0_f = H \cdot (L_R + L_L)$$

where L_R is the delay through a router, and L_L the delay through the link connecting adjacent routers. L_R can be further decomposed as:

$$L_R = L_{buffer} + L_{arbiter} + L_{crossbar}$$

where L_{buffer} is the latency writing and reading from a buffer,

$L_{arbiter}$ that of the arbitration logic, and $L_{crossbar}$ the crossbar traversal delay[1].

Similarly, hop count readily abstracts flit traversal energy, E_f:

$$E_f = H \cdot (E_R + E_L)$$

where E_R is the energy consumed by each flit within the router, and E_L the energy consumed by a flit when traversing a link between adjacent routers, with E_R further broken down as:

$$E_R = E_{buffer} + E_{arbiter} + E_{crossbar}$$

where E_{buffer} is the energy dissipated writing and reading from a buffer, $E_{arbiter}$ that of the arbitration logic, and $E_{crossbar}$ the crossbar traversal energy.

Clearly, units of abstraction for energy can be combined with that of delay for estimates of average power. H can be multiplied by average per-hop energy and delay to derive average power dissipated – $L0_f \cdot E_f$ in a contentionless network, and $L_f \cdot E_f$ in a realistic network with contention, with L_f, the average flit latency, derived by performance analysis frameworks [7, 2, 1]. However, energy-based units of abstraction cannot reflect the spatial and temporal variance in an actual network power profile and hence are unsuitable as abstractions of network power.

3.2 Channel load

Channel load was proposed as a unit of abstraction for network throughput – the maximum data rate in bits per second (or packets per unit time) that a network can accept per injection port. It is defined as the amount of traffic that must cross a link if each input injects one unit of traffic according to a given traffic pattern [10]. For instance, if there are two flows using the network, one from node 1 to 3 and another from node 2 to 3, the channel load on the link connecting nodes 2 and 3 will be 2. To derive network throughput, the link with the maximum channel load, i.e. the bottleneck channel, is first identified, and network throughput is then derived as an inverse of maximum channel load.

Channel load fails as an abstraction of network power for the following reasons. First, a link with a channel load of, say, two, does not neccessarily consume twice as much power as one with a channel load of one since by definition, a link cannot be used beyond its capacity of one unit. Second, while channel load captures the spatial variance in a network power profile, i.e., it points to which parts of the network are more highly loaded, and hence likely to consume more power, it still does not capture the temporal dimension – it cannot distinguish between the load on the network at different points in time.

3.3 Proposed unit of abstraction – Link utilization

The power dissipated by a network with N nodes, P_N, in a period of time (T cycles) corresponds to the total energy dissipated by all N routers and $4N$ links (assuming a torus, a mesh with wrap-around edges, with unidirectional links) during that period:

$$P_N = \sum_{t=1}^{T}(\sum_{i=1}^{N} E_{R_i}(t) + \sum_{j=1}^{4N} E_{L_j}(t))$$

where $E_{R_i}(t)$ is the energy consumed by router i at time t, and $E_{L_j}(t)$ is the energy consumed by link j at time t.

[1]Since most routers are pipelined, L_R will be the number of pipeline stages multiplied by the pipeline cycle time, rather than the combined delays through the various functions. We illustrate it this way to better show the relation to network energy.

This can be approximated by factoring in individual router component and link utilizations:

$$P_{N'} = \sum_{i=1}^{N}\sum_{p=1}^{P}(U_{buffer_{R_{i_p}}} \cdot E_{buffer_{R_{i_p}}} + U_{arbiter_{R_{i_p}}} \cdot E_{arbiter_{R_{i_p}}} + U_{crossbar_{R_{i_p}}} \cdot E_{crossbar_{R_{i_p}}}) + \sum_{j=1}^{4N} U_{L_j} \cdot E_{L_j}$$

where $U_{x_{R_{i_p}}}$ = percentage of T cycles that component x in port p of router i was used and U_{L_j} = percentage of T cycles that link j was used. For simplicity, the power consumed by the centralized crossbar and arbiter is denoted per port, corresponding to specific arbiter request lines and crossbar input ports.

When we look at an incoming link, we see that whenever a flit traverses a link, it will definitely trigger a buffer write (to store the flit) as well as an arbitration for the crossbar (to secure passage for the flit to move on to the next hop). Similarly, when we explore an outgoing link, we notice that a sequence of network operations must occur immediately preceding a flit traversing the outgoing link. The flit must be read from a buffer, and traverse the crossbar. Hence, the utilization of incoming and outgoing links, U_{L_j} and U_{L_k} subsumes the other utilization ratios, and network power can be approximated with just link utilizations:

$$P_{N''} = \sum_{j=1}^{4N} \cdot U_{L_j} \cdot (E_{L_j} + E_{bufwrite_{R_i}} + E_{arbiter_{R_j}}) + \sum_{k=1}^{4N} \cdot U_{L_k} \cdot (E_{bufread_k} + E_{crossbar_k} + E_{L_k})$$

Besides reflecting network power, link utilization has the additional advantage of readily mapping to message flows, since a message flow can be abstracted as loading a certain amount of utilization on the chain of links between the source and destination.

4. NETWORK POWER ANALYSIS

With link utilization as the unit of abstraction of network power, our analysis framework is tasked with deriving the impact of message flows on individual links' utilizations. In essence, it needs to capture the interactions between message flows injected into the network and derive the utilization of each link across time.

The key steps of our framework are as follows:

1. First, 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.
2. Next, routing maps these injection rate functions onto the links of a network topology, translating them into *link utilization functions*.
3. Sharing of a link between messages is then modeled and captured through the *summation* of link utilization functions.
4. Network contention is reflected through overflows in link utilization functions and handled through *propagation* of this overflow area.
5. Finally, the link utilization functions are split back into constituent message flows.

Our analysis iterates through the above 5 steps until all link utilization functions are within capacity.

Throughout our detailed description of each step of the analysis, we will illustrate the operations involved with the following simple example shown in Figure 2(a). The network is a sixteen-node mesh with each node having an injection port bandwidth of 10Gb/s. There are three message flows on the network: messages A, B, and

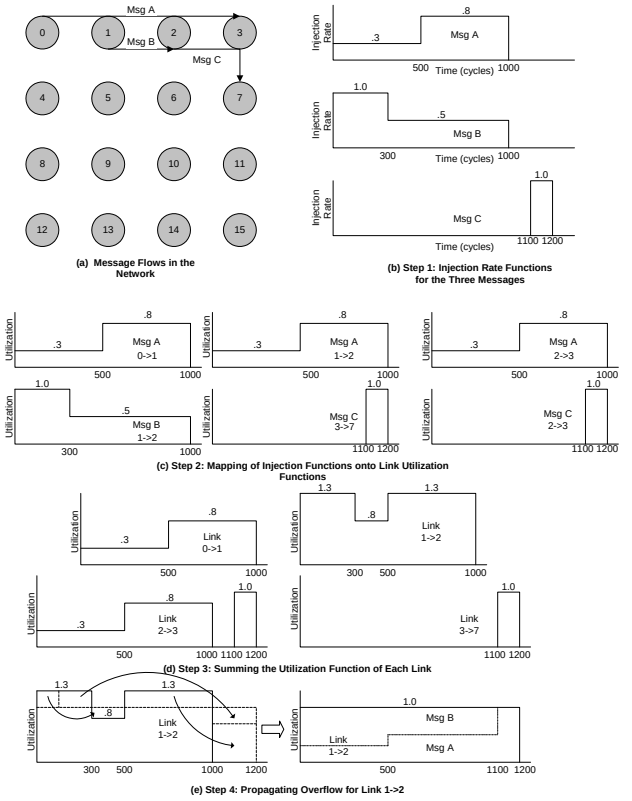


Figure 2: Our simple walkthrough example showing three message flows. The message paths are shown in (a) (with unused links omitted from the diagram). The three injection rate functions are shown in (b). (c) Shows the injection rate functions mapped to link utilization functions. Note that Msg A maps to three links, Msg B to one link, and Msg C to two links - corresponding to the number of hops for each message. The sums of utilization functions for each link are shown in (d). Note that Links $1 \rightarrow 2$ and $2 \rightarrow 3$ are over-utilized. (e) Shows how the excess traffic in link $1 \rightarrow 2$ is propagated until there exists no contention.

C. Message A is injected at node 0, destined for node 3, and uses 3Gb/s for the first 500 cycles, followed by 8Gb/s for the next 500 cycles. Message B's source and destination are nodes 1 and 2 respectively, and it injects the full 10Gb/s for the first 300 cycles, and 5Gb/s for the next 700 cycles. Message C's source and destination nodes are 2 and 7 respectively, and it injects at the full 10Gb/s, though only from cycle 1100 to 1200.

4.1 Step 1: Message Flows as Injection Rate Functions

An injection rate function is a piecewise-constant function over time, where injection rate u_i is the amount of data injected by a message per unit time expressed as a percentage of the injection port bandwidth, so $0 \leq u_i \leq 1$. For each message flow M_j, it is mapped onto an injection rate function f_{M_j} as a set of pairs, (t_i, u_i), each pair indicating that the message injects at a rate of u_i from time t_i to time t_{i+1}:

$$f_{M_j} = \{(t_0, u_0), (t_1, u_1), (t_2, u_2), ..., (t_{p-2}, u_{p-2}), (t_{p-1}, u_{p-1})\}$$

where p is $|f_{M_j}|$, the number of pairs in function f_{M_j}, or the number of distinct phases where the message flow injects at different rates.

In our walkthrough example, Figure 2(b) shows the three messages mapped as injection rate functions, with
$f_{M_A} = \{(0, 0.3), (500, 0.8), (1000, 0.0)\}$,
$f_{M_B} = \{(0, 1.0), (300, 0.5), (1000, 0.0)\}$, and
$f_{M_C} = \{(0, 0.0), (1100, 1.0), (1200, 0.0)\}$.

4.2 Step 2: Mapping onto Link Utilization Functions

Given a network topology, the routing algorithm determines how injection rate functions are mapped onto individual links of the network as it decides which links a message flow will travel through. Our framework takes as input the specific topology and routing protocol of the target chip platform, and the analysis applies to networks with any topology and routing. While we will focus on deterministic dimension-ordered X-Y routing[2] in meshes in our discussion due to its simplicity, we will discuss how our model handles more sophisticated topologies and routing protocols in Section 4.6.

Basically, for each message whose route goes through link i, the message's injection rate function is mapped onto link i's utilization function. Continuing our walkthrough example, f_{M_A} will be mapped onto links $0 \rightarrow 1$, $1 \rightarrow 2$, and $2 \rightarrow 3$'s utilization functions, while f_{M_B} will be mapped onto link $1 \rightarrow 2$'s utilization function, and f_{M_C} will be mapped onto links $2 \rightarrow 3$ and $3 \rightarrow 7$'s utilization functions, as shown in Figure 2(c).

4.3 Step 3: Summation of Link Utilization Functions

Next, the multiple link utilization functions on a single link are summed to reflect the sharing of the link among these multiple message flows. To sum two functions, f_{M_j} and f_{M_k}, we must first transform them to introduce redundancy:

First, we gather in set T the number of unique times t_i in the sets of both functions:

$$T_{f_{M_i}} = \{t_0, .., t_{p-1}\}, where\ p = |f_{M_i}|$$

$$\begin{aligned} T_{f_{M_j}+f_{M_k}} &= T_{f_{M_j}} \bigcup T_{f_{M_k}} \\ &= \{T_0, T_1, ...T_{P-1}\}, where\ P\ is |T_{f_{M_j}+f_{M_k}}| \end{aligned}$$

Then, each function is transformed so the number of pairs in the function is $|T_{f_{M_j}+f_{M_k}}|$:

$$f'_{M_j} = \{(T_0, f_{M_j}(T_0)), (T_1, f_{M_j}(T_1)), ..., (T_{P-1}, f_{M_j}(T_{P-1}))\}$$

$$f'_{M_k} = \{(T_0, f_{M_k}(T_0)), (T_1, f_{M_k}(T_1)), ..., (T_{P-1}, f_{M_k}(T_{P-1}))\}$$

$$where f_{M_i}(t) = u_k, if(t_k \leq t < t_{k+1})$$

Now, we can simply express link utilization f_{L_i} as the sum of message injection rate functions that use the link, where summation is defined as:

$$\begin{aligned} f_{M_j} + f_{M_k} &= f'_{M_j} + f'_{M_k} \\ &= \{(T_0, f_{M_j}(T_0) + f_{M_k}(T_0)), (T_1, f_{M_j}(T_1) + f_{M_k}(T_1)), \\ &\quad ..., (T_{P-1}, f_{M_j}(T_{P-1}) + f_{M_k}(T_{P-1}))\} \end{aligned}$$

This summed link utilization function can be further reduced if there exists redundancies:

$$f_{L_i} \equiv reduce(f_{L_i}), where$$
$$reduce(\{(t_i, u_i), (t_{i+1}, u_{i+1})\}) = (t_i, u_i)\ when\ u_i = u_{i+1}$$

[2]This simple routing algorithm first routes packets completely in the x-dimension, before moving them in the y-dimension towards their destinations.

In other words, if two consecutive segments have the same utilization, they can be represented as a single segment. Hence, the summation of two functions does not neccesarily lead to a link utilization function that has more pairs.

The assumption here is that the message flows will be multiplexed on the link, and sharing occurs in a round-robin fashion among flows. This is an approximation of the actual multiplexing of message bit streams in a network – either on a packet-by-packet basis in packet-switched networks, or on a flit-by-flit basis in virtual-channel networks [8].

Following our walkthrough example, the summation of the two link utilization functions on link $1 \rightarrow 2$ goes through the following steps (summarized in Figure 2(d)):

$$\begin{aligned} T_{f_{M_A}} &= \{0, 500, 1000\} \\ T_{f_{M_B}} &= \{0, 300, 1000\} \\ T_{f_{M_A}+f_{M_B}} &= \{0, 300, 500, 1000\} \\ f'_{M_A} &= \{(0, 0.3), (300, 0.3), (500, 0.8), (1000, 0.0)\} \\ f'_{M_B} &= \{(0, 1.0), (300, 0.5), (500, 0.5), (1000, 0.0)\} \\ f_{M_A} + f_{M_B} &= \{(0, 1.3), (300, 0.8), (500, 1.3), 1000, 0.0)\} \end{aligned}$$

It shows that from time 0 to 300, link $1 \rightarrow 2$ is loaded beyond its capacity, at 1.3, from time 300 to 500, it is loaded to 80% capacity, and from 500 until time 1000, it is again over-capacity at 1.3. Note that link $2 \rightarrow 3$ is also loaded by two messages, A and C, but we omit the explicit walkthrough of this step in the interests of brevity. The resulting summation can be seen in Figure 2(d).

4.4 Step 4: Overflow Propagation of Link Utilization Functions

Through the summation of link utilization functions, we can then easily see contention in links – whenever the summed link utilization exceeds 1.0, flows are contending for the link, and queuing will have to occur to ease the overflow. To model the impact of this contention on link utilization and thus network power, we observe that the total area under a link utilization function corresponds to the total amount of data sent over the link and must remain unchanged assuming no packet drops (Section 4.6 will touch on how our model can be extended for packet drops). Hence, upon overflow in link utilization functions, our analysis framework propagates the overflow as follows to reflect the impact on network power:

$$\begin{aligned} prop(f_{L_i}) = \forall(t_i, u_i), u_i > 1 \Rightarrow \\ (f_{L_i} - \{(t_i, u_i)\} - \{(t_{i+1}, u_{i+1})\}) \cup \{(t_i, 1.0)\} \\ \cup \{(t_{i+1}, (u_i - 1) \cdot (t_{i+1} - t_i)/(t_{i+2} - t_{i+1}) + u_{i+1})\} \end{aligned}$$

Essentially, the overflow area is propagated forward, or "spilt over" to underflow areas in the link utilization graph, reflecting that the excess packets will be queued and subsequently sent when the link has enough bandwidth to accomodate them. The power profile of the network will thus reflect full utilization (maximum power) as long as the propagated overflow load on the link exceeds link capacity when coupled with the current link utilization levels.

In our walkthrough example, targeting the contention from time 0 to 300 when the link $1 \rightarrow 2$ is loaded to 130% of its capacity,

$$\begin{aligned} &prop(f_{M_A} + f_{M_B}) \\ &= \{(0, 1.0), (300, (0.3 \cdot (300/200) + .8), (500, 1.3), (1000, 0.0)\} \\ &= \{(0, 1.0*), (300, 1.25), (500, 1.3), (1000, 0.0)\} \\ &prop(prop(f_{M_A} + f_{M_B})) \\ &= \{(0, 1.0), (300, 1.0), (500, 1.4), (1000, 0.0)\} \\ &prop(prop(prop(f_{M_A} + f_{M_B}))) \\ &= \{(0, 1.0), (300, 1.0), (500, 1.0), (1000, 1.0), (1200, 0.0)\} \end{aligned}$$

which, when reduced, is equivalent to $\{(0, 1.0), (1200, 0.0)\}$.

4.5 Step 5: Splitting link utilization functions back into message flows

The effect of link contention and overflow propagation on individual message flows now needs to be considered. As we assume round-robin arbitration, the resultant link utilization function, $prop(f_{L_i})$ has to be fairly split among the messages. In our implementation, the algorithm actually splits the messages as they are being propagated. We see in the right half of Figure 2(e), how the traffic is split between Msg A and Msg B (the dotted line divides the two components). For example, from time 0 to time 500, Msg A uses 30% of the bandwidth of link $1 \rightarrow 2$, and Msg B uses 70% of the bandwidth. Again, since there are no packet drops, the area under each message's utilization function needs to be the same before and after the split:

$$\begin{aligned} &\sum_{i=0}^{|f_{M_A}|-1} ((t_{i+1} - t_i) \cdot f_{M_A}(t_i)) = \\ &\sum_{i=0}^{|prop(f_{M_A})|-1} ((t_{i+1} - t_i) \cdot prop(f_{M_A})(t_i)) \end{aligned}$$

This wraps up a single iteration of our analysis framework, which continues through all five steps till no contention points exist, i.e until $\forall f \in F, \forall(t_i, u_i) \in f, u_i \leq 1.0$.

But notice that since we modified the injection rate of Msg A, it now conflicts with Msg C on link $2 \rightarrow 3$. Thus, we have to repeat the steps again (this is not shown in the figure due to space limitations). Msg A is now $\{(0, .3), (500, .5), (1100, 1.0), (1200, 0.0)\}$, and Msg C is still $\{(0, 0.0), (1100, 1.0), (1200, 0.0)\}$. Summing these functions we get $\{(0, .3), (500, .5), (1100, 2.0), (1200, 0.0)\}$. After propagation, we get $\{(0, .3), (500, .5), (1100, 1.0), (1300, 0.0)\}$. Msg A is thus again modified, and it is now equal to $\{(0, .3), (500, .5), (1300, 0.0)\}$ (after reduction), and Msg C is modified to $\{(0, 0.0), (1100, .5), (1300, .5)\}$. Now there are no points of contention in the network, so we are done.

4.6 Discussion

While several assumptions were made to simplify the description of the framework, many are not neccessary for the correctness of our analysis. Here, we discuss how major practical concerns can be factored into our framework.

Routing protocols. Our analysis, as described, assumes that routing is deterministic, i.e., that a single path is used for message flows between specific source-destination pairs. Most on-chip networks adopt deterministic routing protocols due to tight area, delay and power constraints. A deterministic routing protocol makes it possible to map message injection rate functions onto specific links directly – the injection rate functions should be mapped onto all links that lie along the fixed route (Step 2). However, our analysis can function with non-deterministic routing algorithms such as oblivious and adaptive routing protocols. Oblivious routing protocols pick a route irregardless of network utilization, choosing to randomly route flows along different minimal paths for load balancing [24]. This can be modeled within our framework by first evenly dividing the utilizations of a message injection rate function into multiple injection rate functions for each minimal path, and mapping that fraction of the injection rate function onto the links along the path. Adaptive routing, where the route is chosen at run-time depending on actual traffic congestion, is trickier. A way to abstract this is to select the least-utilized path when mapping (Step 2), where path utilizations are calculated by summing link utilizations for all links along the path, averaging it across the timeline, given message flows that have been mapped. This greedy algorithm reflects the greedy nature of actual implementations of

adaptive routing protocols – whenever a route is needed, the current load situation on the network will be consulted and used to prescribe the least-loaded path.

Buffers. Our analysis currently does not take into account the effect of buffering at each individual router. When there is contention, we model the buffering to occur at the souce node; that is, if a node injects data into the network, but one of the links down the path of the message is congested, the effects of the throttling of traffic are propagated backwards all the way to the source. Thus, the contribution of a message to the utilization function of each link along the path is the same. However, our model can be readily extended to include the effects of buffering as follows. Once we calculate the propagated utilization functions of each message which passes through a particular link (Step 4), instead of then updating each link along a particular message's path with this new utilization function, we would time delay the function upstream from this link. This is because when a message is blocked somewhere along its path, it is possible for body flits upstream to continue traversing links, up to the blocked link, and until input buffers become full (assuming certain flow controls such as virtual cut-through [10]). Thus, link utilization and network power will be higher for those few cycles before the buffers fill up. We will incorporate this extension into our framework, though it should be noted that in many cases, early in the design phase, the number of buffers (or perhaps even the flow control) is not known.

Topologies. For the analysis we have described here, we assumed a 2D mesh or torus topology and a homogeneous network (that is, all of the links have the same bandwidth). However, our model is readily applicable to heterogeneous networks of any topology, though its current implementation only supports 2D mesh and torus networks. Instead of thinking of the abstraction of power as link utilization, we would view it as normalized total bandwidth. That is, we are abstracting the amount of data that is currently being transferred through the network. In the homogeneous case, these two abstractions are equal, since for every link, 100% link utilization represents the same absolute bandwidth. But in a heterogeneous network, 100% utilization on one link may be consuming more or less power than 100% utilization of another link. Thus, we would make the subtle change in metric from link utilization to total bandwidth (or, alternatively, network utilization). We will also extend our implementation of the framework to accept arbitrary network topologies.

Packet drops. In our analysis, we assume that packets are never dropped, since most on-chip networks cannot afford packet drops, unlike in TCP/IP [10]. To extend our model for scenarios where packets can be dropped upon congestion, we will have to change the way overflow link utilizations are handled (Step 4) – the area before and after a propagation now need not be equal. A way to factor in packet drops is to lower the overflow area after propagation by the packet drop probability, so if there is a 10% probability of packet drops, only 90% of the overflow area will be propagated onwards.

4.7 Validation

4.7.1 Experimental setup

We validate our results by comparing against simulation results from Orion, a cycle-accurate network power-performance simulator. All parameters for Orion remain constant across validation runs, and are summarized in Table 1. Dimension-ordered X-Y routing is assumed. The process technology is 0.1μm.

For our validations, we use traces from SPEC [37] and MediaBench [34] benchmarks executed on TRIPS [21], a chip multiprocessor which consists of 4 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 network. These traces are used to drive Orion's simulations to reflect realistic network communication. Furthermore, each traffic trace has associated with it a certain average number of flits per message (e.g., trace `ammp 2` in Table 2 injects 2 flits per packet).

Parameter	Flag	Value
Array Size	-A	5
Dimension	-c	2
Input Buffer Size	-B	64 (flits)
Output Buffer Size	-O	4 (flits)
Flit Size	-F	2 (=128 bits)
Link Length	-L	3000 (nm)
Routing	-R	0 (xy-routing)

Table 1: Parameters used for all validation experiments with Orion

To give an idea of the traffic traces, each line in the trace file represents a single message which is sent in the network. The time at which the message is injected into the network is given, as well as the source and destination nodes, as well as the number of flits in the message. So to parse this trace file into a set of message injection rate functions for our analysis, we sample the trace file at a rate, say every 2000 cycles, and calculate the average injection rate for each message during this period of time. For example, say between time 2000 (cycles) and 4000, we calculate from the trace file that message m_0 sends 100 messages of size 5 flits each. Then the total amount of data injected by this message is 500 flits. The maximum number of flits that this message *could* inject into the network is 2000 (2000 cycles * 1 flit/cycle). Thus, we say that from t_{2000} to t_{4000}, the injection rate is $.25 = 500/2000$.

To evaluate the effectiveness of our analysis in capturing relative power trends, we first normalize both the power profiles from Orion and our analysis such that the range of the function lies between 0 and 1 (i.e. the minimum of the function is exactly 0, and the maximum is exactly 1). We then take the absolute value of the difference between each pair of points at each point in time, and average these across the entire simulation time:

$$Err_{rel} = \left(\sum_{t=0,T,2T,\ldots,T_f} |P_{Orion}(t) - P_{Analysis}(t)| \right) / T_f$$

where T is the sampling period, T_f is the total simulation time in number of sampling periods, $P_{Orion}(t)$ is the normalized power estimate given by Orion for the period t, and $P_{Analysis}(t)$ that given by our analysis for the period t.

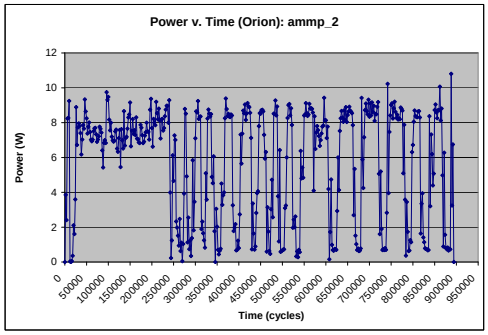
This means that a relative error, Err_{rel}, of, for example, .053, means that corresponding points in $P_{Orion}(t)$ and $P_{Analysis}(t)$ differ, on average, by 5.3%. There is a need to normalize the power profiles as the units are different. Our model's power profile are in units of links utilized, while Orion's is in units of Watts. Furthermore, the power profile given by Orion is dependent on parameters such as link length and on technology-dependent factors such as interconnect capacitances and transistor sizes.

4.7.2 Effectiveness in capturing variance in power profile

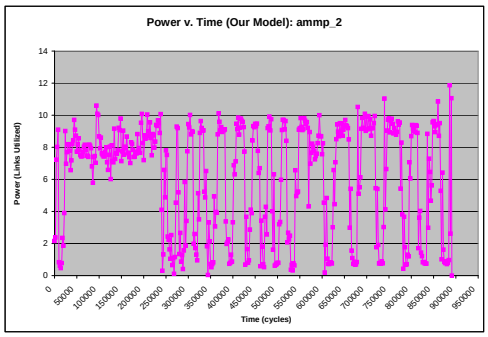
Table 2 shows Err_{rel} for eight benchmarks, while Figure 3 shows the power profiles generated by both Orion and our framework for the `ammp` benchmark. From Figure 3, it is clear that the power profiles match well, resulting in the small relative error of 1.9%.

Bench	adpcm 2	ammp 2	art 2	compress 3	dct 3	equake 3	gzip 3	hydro2d 3
Err_{rel}	.053	.019	.089	.074	.059	.008	.01	.023

Table 2: Average relative error between Orion's power estimates and our analysis' estimates for various benchmarks.



(a)



(b)

Figure 3: Results from Orion and from our model for the ammp benchmark and 2 flits per packet

4.7.3 *Effect of contention on framework's accuracy*

Here, we increase the traffic load of the `ammp` benchmark from 2 flits/message to 4.5 flits/message to see the effect of higher contention on the accuracy of our analysis. At 4.5 flits/message, relative error increases from 1.9% to 5.9%. This increase is expected, for the following reason: as traffic load increases, contention increases in the network, causing buffers to back up. As our analysis currently does not take buffering delays into account, deviation from actual behavior is to be expected. Section 4.6 discusses how buffering can be readily incorporated into our framework.

While we do not show the power profile derived by our framework for `ammp` at 4.5 flits/packet due to space limitations, the noticable differences are few. A small number of the peaks and valleys are slightly changed, relative to one another, but the overall peak power remains unchanged.

4.7.4 *Effect of sampling period*

Here we explore how the accuracy of our framework is affected by changing the sampling rate of both Orion and our model. Table 3 shows results for the `adpcm` trace sampled every 200, 600, 2000, 6000, and 20000 cycles. We see that as we increase the sampling frequency (and shorten the sampling period), our accuracy drops, as compared to Orion's estimates. This is because as we shorten the sampling period, the benefits of Orion's cycle-accurate modeling become more apparent, while the errors introduced by our framework's simplifications grow more glaring. Conversely, when

Period (cycles)	200	600	2000	6000	20000
Err_{rel}	.144	.111	.053	.007	.012
Run Time (s)	780	66	11	3	1

Table 3: Average error between Orion and our analysis, as well as running times for the `adpcm` benchmark at five different sampling periods

Benchmark	Orion(s)	Our Model(s)	Speedup
adpcm 2	263	11	24
ammp 2	215	2.9	74
art 2	178	7	25
compress 3	263	8.3	32
dct 3	220	1.5	147
equake 3	250	3.8	66
gzip 3	209	4.4	48
hydro2d 3	200	2.1	95

Table 4: Run times for Orion and our model for various benchmarks and traffic load (flits per packet)

we lengthen the sampling period, Orion's cycle-accurate power estimates are averaged over a longer time frame as well, so small mismatches in our model's power profile are glossed over. A deviation from this trend is the 20000-cycle sampling period – because the calculations of Err_{rel} are normalized to the peak power, it is possible, as we increase the sampling period (and smooth the power profile), that the peak power occurs at a different time period, altering the accuracy calculation, which has happened here. One might notice that as we increase the sampling period, both accuracy and run time improves, seemingly indicating that we should use as large a sampling period as possible. While we can decrease sampling frequency as much as possible to improve our framework's accuracy, our analysis will then approach average power analysis, losing valuable spatial and temporal trends. Our framework allows users to tweak this balance between accuracy and runtime, for instance, through the use of varying granularities in power analysis – long sampling periods to scan for times of peak network power, and zooming in on interesting regions with fine granularity.

As it is currently implemented, decreasing the sampling period to 200 cycles and beyond will lead to our framework losing the time advantage versus Orion (see Table 4), as the Orion implementation is optimized for event-driven rather than cycle-by-cycle simulation. While we are investigating ways of optimizing the implementation of our analysis framework, our case studies illustrate that such sampling frequency is sufficient for most early-stage power analysis.

4.7.5 *Comparison of run times*

We compare the run times of Orion and our implementation of the proposed power analysis framework for the traces of various benchmarks in Table 4. We can see that our model can provide between one and two orders of magnitude speedup over Orion, which is a significant improvement. Note, however, that these speedups were obtained using a granularity of 2000 cycles per sample. Depending on the needs of the user, these speedups could be greater (for coarser granularity), or lower (for finer granularity) as we see in Section 4.7.4.

Quantization	Run time(s)	Speedup
Orion	180	1
Our Model	6.8	27
.01	1.4	129
.02	1.0	180
.05	0.5	360

Table 5: Running times for different quantizations of `art`'s injection rate functions on our power analysis framework.

5. CASE STUDIES

Here, we outline three potential ways of leveraging our power analysis framework, presenting case studies that illustrate with simple examples how we see our power analysis (1) enabling speedup of multiprocessor simulators, (2) allowing power-aware communication synthesis of embedded systems, and (3) facilitating communication power analysis of programs in compilers.

5.1 Usage I: Network Simulation Speedup

While cycle-accurate simulators such as SimpleScalar [36] have been the foundation of numerous processor architecture evaluations, as applications and architectures increase in complexity, simulation time is exploding and designers and researchers are questioning the necessity of cycle accuracy [23]. The advent of multiprocessor architectures, and the use of on-chip networks as the communication fabric for these processor cores further aggravates simulator complexity. Now, instead of modeling a single, central bus, numerous routers and links operating independently with their own pipelines have to be considered. This triggered the proposal of simulation frameworks where modeling fidelity can be tuned so designers can zoom in on a particular time window of interest, explore that with cycle-accuracy, while simulating the remaining time at a high level [28]. We see our power analysis factoring into these higher-level simulation frameworks, enabling tradeoffs of network power model fidelity with model complexity (or speed).

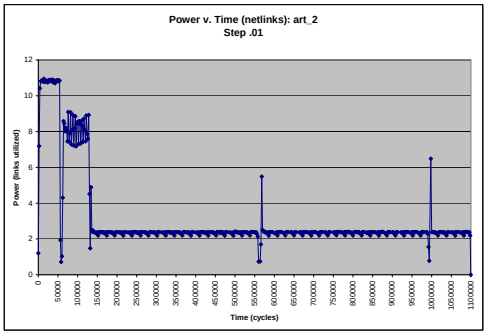
Since the speed of our power analysis depends on the complexity of the injection rate functions (i.e., the number of pairs in the function), simplifying injection rate functions improves model speed. Simplification involves merging of pairs in the injection rate function. This has to be carefully done to minimize information loss which can lead to poor model accuracy. We propose quantizing link utilizations, with pairs that fall within the same range of link utilizations merged as follows:

$$merge(\{(t_0, u_0), (t_1, u_1)\}) = \{(t_0, \frac{(t_1 - t_0) * u_0 + (t_2 - t_1) * u_1}{t_2 - t_0}\}$$

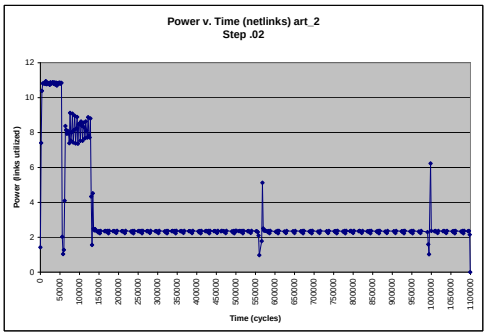
where t_2 is the time corresponding to the pair after t_1.

With the same simulator setup as in Section 4.7, we evaluated three different quantization granularities with the `art` TRIPS benchmark – .01, .02 and .05 units of link utilization (100, 50 and 20 levels respectively). Figure 4 plots the power profiles, while Table 5 shows the running times.

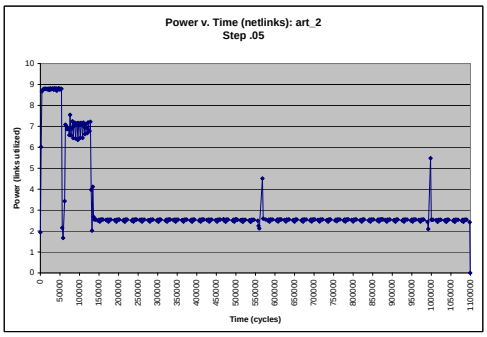
Comparing the running times of our model to the non-quantized running time for this benchmark, we see that we can obtain approximately another order of magnitude of speedup. Note though that as we increase the quantization granularity to .05, the resulting power profiles become smoother. For instance, the peak towards the end of the trace is at about 7 units of link utilization in the baseline, at about 6.5 for steps of .01, at about 6 for steps of .02, and at about 5.5 for steps of .05. Similarly, the other peaks in the trace are drawn towards the average. This is the reason for the higher relative error, and is expected since the inputs to our model are smoothed to enable higher-level modeling. The power profiles show that trends in the power profile remain intact though.



(a)



(b)



(c)

Figure 4: Art benchmark simplified into (a) 100, (b) 50, and (c) 20 quantization levels.

5.2 Usage II: Communication Power Synthesis of Embedded Task Graphs

As communication synthesis moves from buses and dedicated links to networks-on-chip as the underlying fabric, network power becomes increasingly an integral constraint in such synthesis frameworks [22, 33, 13, 20]. Current communication synthesis frameworks such as that by Murali et al. [20, 14] separate the mapping phase where tasks are placed onto nodes, from the network synthesis phase which takes the task mapping, and investigates different network designs' delay, area, and power to improve synthesis's tractability. In the mapping phase, a simple metric – hop count is used to abstract network power, while in the network synthesis phase, detailed cycle-accurate power models (Orion) and circuit-level power estimation tools are used.

With our power analysis framework, we see the high-level power estimates making it possible to co-optimize the two phases for power, exploring energy-efficient task graph mapping and network design tailored to that mapping. Designers will be able to quickly evaluate the power impact of different task mappings on a variety of network topologies and architectures. To use our analysis framework, the communication synthesis framework first generates the injection rate functions corresponding to the communication flows between tasks in the task graph, before mapping them onto link utilization functions based on the topology and architecture and generating a power profile of the task graph using our framework. This profile can then be used to evaluate different task mappings of the same graph onto the same network; or, to compare mappings of the same task graph onto different networks, facilitating co-optimization across the two phases.

Here, we walk through a simple example illustrating how network power analysis can readily distinguish between two power profiles as a result of different task mappings on a specific network even when they appear equally energy-efficient when using hop count as the metric. The task graph we use is presented in Figure 5(a). Each circle represents a task, identified by a letter inside the circle. Each task has a time associated with it which represents the number of cycles the task is expected to take. Each directed link has associated with it the number of flits which the task must send to the next task, after it has completed, and before the next task can commence. We assume that each link in the network, shown in Figure 5(b), has a bandwidth of one flit/cycle, so that the number associated with each link in the task graph is also equal to the number of cycles that it takes to send the data to the next task. Finally, the period of the task graph is the time by which the entire set of tasks must complete because it will begin again at the specified time, and is 3000 time units for this task graph. The task graph is looped 5 times.

We consider the mapping as shown in Figure 5(b). The letter(s) inside each node represents the task(s) which is(are) located on that node. For example, node 1 contains two tasks: Task C and Task X. We refer to this mapping as Mapping I. Mapping II is almost identical to Mapping I, except that Task G is moved from node 12 to node 14. Note that this network is a 4-by-4 mesh topology, as we saw in Figure 1.

In order to input a task graph into our analysis framework, we need to generate message injection rate functions. Each directed link in the task graph corresponds to exactly one message, because the source task and destination task are each mapped onto source and destination nodes in the network. In the case that both tasks are mapped onto the same node, then we omit that particular link in our model. We generate the injection rate function for each task T_i as follows:

$$f_{T_i} = \{(0,0), (t_{cur_i}, 1.0), (t_{cur_i} + t_{comm_i}, 0)\}$$

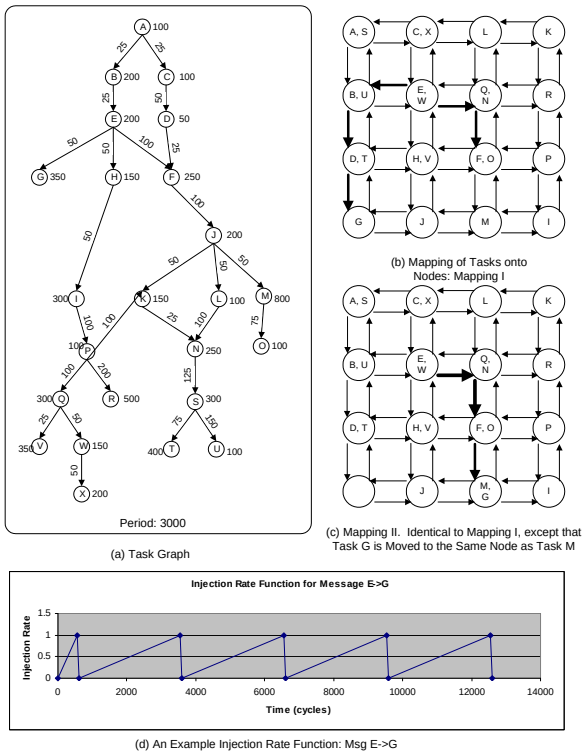


Figure 5: A sample task graph and mapping onto a 4-by-4 mesh network

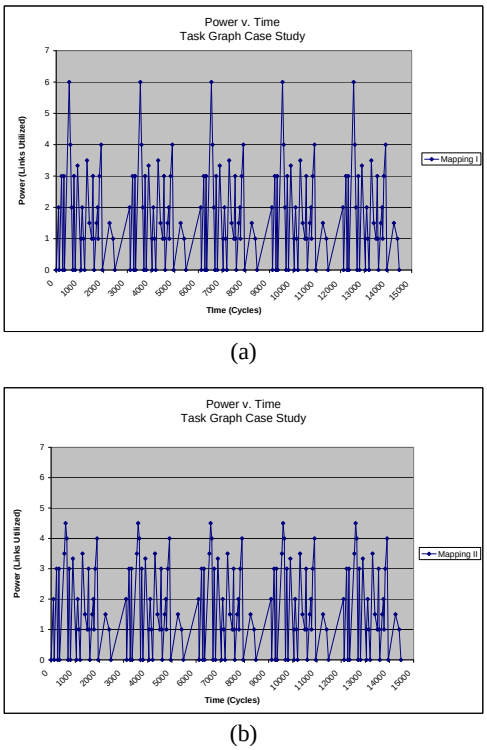


Figure 6: Power profiles for two mappings showing that Mapping II is estimated to have a lower peak power than Mapping I.

where t_{cur} is the maximum time of the sum of all paths from the first task $T1$ to T_i's completion, and t_{comm} is the communication time of task T_i. This function is then repeated as many times as the task graph loops. As an example, consider the injection rate function for the message from task E to task G, in Figure 5(d). If we sum the time from the top of the task graph down to node E (including the time for task E to complete), we get 550 cycles. Thus, task E should begin injecting data at 550 cycles, and, for this message, will do so until time 600 (the weight of the edge from task E to task G is 50, meaning that it needs to send 50 flits of data). This pattern repeats every 3000 cycles, since that is the period of the task graph as we have defined it.

Having generated the injection rate functions for each message in the task graph, we then use these functions as the input to our model, and the power profiles are presented in Figure 6 for both mappings. Note the period of the functions is 3000, the same as the period of the task graph. Comparing these two functions, we can see that the peak power of Mapping II should be less than that of Mapping I. Conceptually, this is because by moving Task G from one node to another, we create contention between two messages: message $E \rightarrow G$ and message $E \rightarrow M$; this contention is illustrated in Figures 5(b) and 5(c). In Figure 5(b), two message paths are highlighted (bold arrows): $E \rightarrow G$ and $E \rightarrow F$. As we then see in Figure 5(c), after task G is moved, these two message paths now overlap. The links cannot be utilized at a bandwidth of greater than 1, so the power is reduced, but it takes more time to complete the transfer of all data.

When we compare this to the hop count metric used in current synthesis frameworks [20], which does not take into account temporal variations, nor contention in the network, it cannot distinguish between these two mappings. Though Task 7 is moved, the number of hops for the message between Task E and Task G remains unchanged (nor does its bandwidth). So, with this simple example, we illustrate how our power analysis framework can be leveraged within communication synthesis and how it enables insights that cannot be obtained currently with the use of simple energy-based metrics.

5.3 Usage III: Program Communication Power Analysis

In most recent proposals of chip multiprocessors, where there are tiles of processing elements across the chip, the compiler is responsible for placing instructions onto processing elements which can range from single ALUs [21] to single-issue pipelines [30] and coarser-grain processors [19]. In these architectures, on-chip network power is becoming increasingly significant, taking 36% of chip power on average in the MIT RAW chip multiprocessor [15]. It is thus important for the compilers of such architectures to consider network power during program analysis, when making placement decisions.

Here, we walk through a simple sample program and show how our power analysis can be factored into the MIT RAWCC compiler [16]. The RAWCC compiler is designed to compile sequential programs to the RAW chip multi-processor. It essentially has three phases: the high-level program analysis and transformation stage; the space-time scheduler; and the code generation phase. It is in the basic-block orchestration sub-phase of the space-time scheduling where we see our model to be of use. Currently, the metric for scoring a particular placement is the number of communication hops. But as we showed earlier, hop count is not an adequate metric for abstracting high-level power estimates. We present a simple example to illustrate this inadequacy below.

Consider the code in Figure 7(a). This is just a hypothetical kernel designed to demonstrate how our power analysis framework can be used – the mapping is not an actual mapping by RAWCC. In this case we map the code onto four nodes of a 16-node network, as shown in Figure 7(b). We then see the code which is mapped onto each individual node, including the communication code, in Figure 7(c). We assume each instruction takes one cycle, except for additions which take two cycles, and multiplications, which take four, as in [16]. We also assume that the links have a bandwidth of one word per cycle, and that the size of data being operated on is three words, which means that it takes three cycles to transmit this data across a link. Finally, this code kernel loops 10 times.

To generate the injection rate functions, we need to analyze the communication patterns of each node, as is done in RAWCC. For instance, Node 0 injects three flits of data at time 1 and every 5 cycles after that. It also injects three flits of data at time 2 and every 5 cycles after that. We similarly analyze the communication code of the other nodes in the network, and generate injection rate functions for the other messages in the network (see Figure 8). We then input these injection rate functions into our model, and generate the overall power function for the network seen in the top half of 7(d).

To illustrate how our analysis can be leveraged, we compare the power profiles of the same piece of code with a slightly different mapping: we move the code on Node 1 to Node 3, and we move the code on Node 2 to Node 4. The path for the message from 0 to 5 remains unchanged. In this new mapping, as seen in Figure 7(c), there is no contention, and we will thus expect a higher peak power. As we see in the right half of 7(e), this is indeed the case. Our analysis shows a higher peak power for Mapping B, as well as larger fluctuations in the power profile as flits that are stalled in Mapping A are subsequently transmitted during idle cycles (overflow propagation of contention). These two mappings also illustrate the deficiencies of relying on just hop count as a high-level estimation of power, as both mappings have the same hop count, but dramatically different power profiles.

6. CONCLUSIONS

Network power analysis is critical for early-stage power characterization of different application traffic behaviors and alternative network architectures. In this paper, we propose a power analysis framework that takes as input the message flows across nodes over time, and outputs a power profile of the network across the chip and over time.

When evaluated on a range of chip multiprocessor benchmark traffic traces, our framework is shown to generate power profiles that match that of architectural cycle-accurate network power simulators. Armed with our power analysis framework, we show how simulation infrastructures can run at varying fidelity, speeding up simulation time; how power analysis can help prune the design space of communication synthesis, allowing it to jointly optimize application task graph mapping with network synthesis; and how compilers for chip-multiprocessors can leverage our power analysis framework for power-aware placement of instructions.

We are in the process of preparing our power analysis framework for release, so it can be incorporated into simulation infrastructures, compilers and synthesis frameworks, as well as other new potential uses.

Acknowledgments

We would like to thank the UT Austin TRIPs research group for the network traffic traces, and Hang-Sheng Wang of Princeton for his technical support in our use of Orion. This work is partially supported by the MARCO Gigascale Systems Research Center,

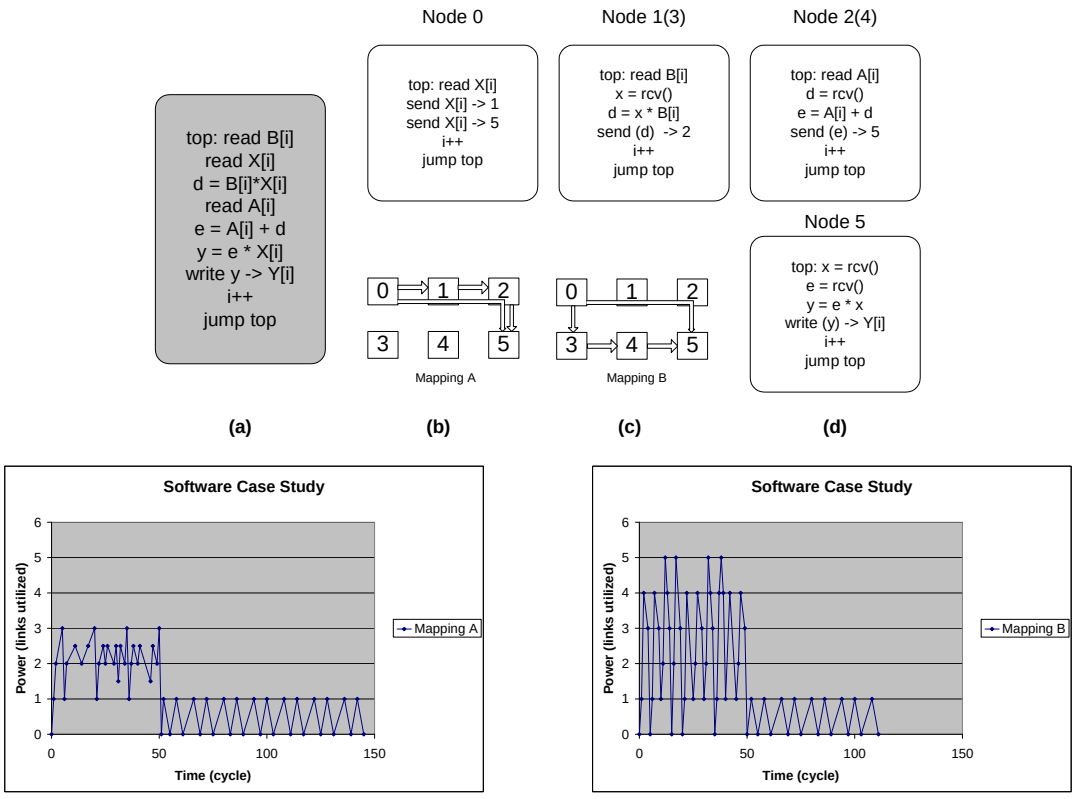


Figure 7: An example code segment and how it might be mapped using, for example, RAWCC, and the estimated power over the course of N loops of this kernel

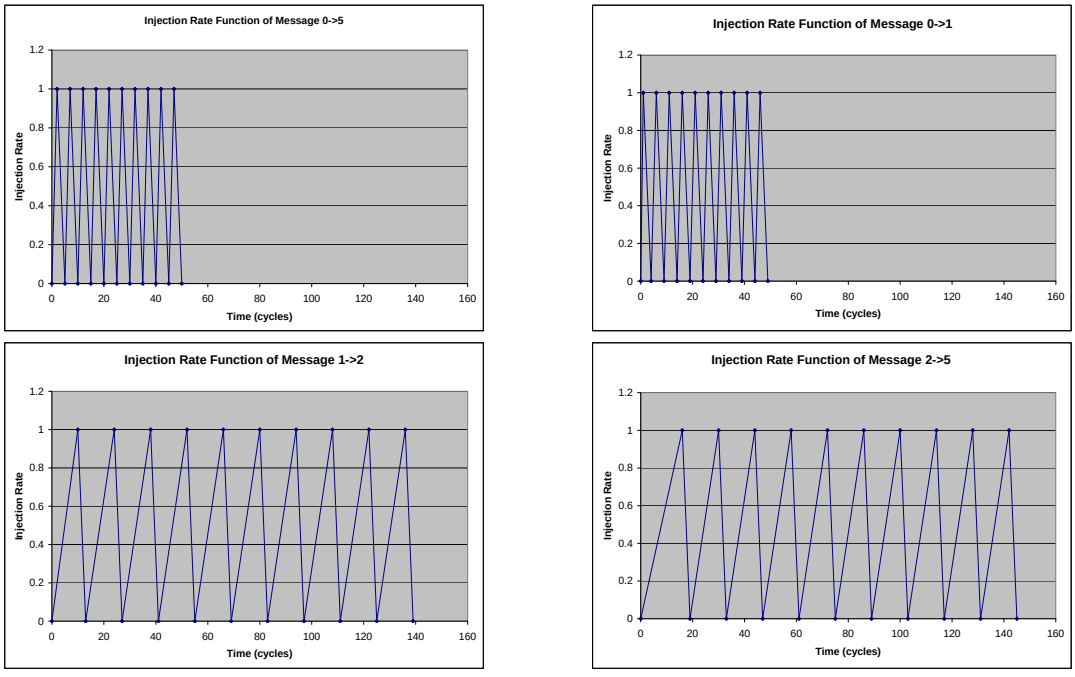


Figure 8: The injection rate functions for each message in the network

NSF grants CCR-0237540 (CAREER) and ANI-0305617, as well as Princeton University's University Research Board grant.

7. REFERENCES

[1] V. S. Adve, M. K. Vernon, "Performance Analysis of Mesh Interconnection Networks with Deterministic Routing," *IEEE Trans. on Par. and Dist. Syst.*, vol. 5, no. 3, pp. 225-246, March 1994.

[2] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Trans. on Par. and Dist. Syst.*, vol. 2, no. 4, pp 398-412, October, 1991.

[3] R. Barua, W. Lee, S. Amarasinghe, A. Agarwal, "Compiler Support for Scalable and Effi cient Memory Systems," *IEEE Trans. on Computers*, vol. 50, no. 11, pp. 1234-1247, November, 2001.

[4] L. Benini, G. De Micheli, "Powering networks on chips," presented at the Int. Symp. System Synthesis, Invited Talk, Montreal, Canada, 2002.

[5] D. Brooks, V. Tiwari, M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proc. of ISCA*, pp. 83-94, 2002.

[6] X. Chen, L-S. Peh, "Leakage power modeling and optimization in interconnection networks," in *Proceedings of the 2003 international symposium on Low power electronics and design*, pp. 90-95, 2003.

[7] W. J. Dally, "Performance Analysis of *k*-ary *n*-cube Interconnection Networks," *IEEE Trans. on Computers*, vol. 39, no. 6, pp. 775-785, June 1990.

[8] W. J. Dally, "Virtual Channel Flow Control," *IEEE Trans. on Par. and Dist. Syst.*, vol. 2, no. 2, pp. 194-205, March, 1992.

[9] W. J. Dally, B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proceedings of the 38th Design Automation Conference*, Las Vegas, NV, June 2001.

[10] W. J. Dally, B. Towles, "Principles and Practices of Interconnection Networks," Morgan Kaufmann, 2003.

[11] R. Ho, K. Mai, M. Horowitz, "The Future of Wires," in *Proceedings of the IEEE*, April 2001, pages 490-504.

[12] R. Ho, K. Mai, M. Horowitz, "Effi cient On-Chip Global Interconnects," *IEEE Symposium on VLSI Circuits*, June 2003.

[13] J. Hu, Y. Deng, R. Marculescu. "System-Level Point-to-Point Communication Synthesis Using Floorplanning Information," ASP-DAC/VLSI, January, 2002.

[14] A. Jalabert, S. Murali, L. Benini, G.De.Micheli, "XpipesCompiler: A Tool for Instantiating Application Specifi c Networks on Chip," in *Proc. of International Conference on Design and Test Europe*, pp. 884-889, 2004.

[15] J. S. Kim, M. B. Taylor, J. Miller, D. Wentzlaff, "Energy Characterization of a Tiled Architecture Processor with On-Chip Networks," in *Int. Symp. on Low Power Electronics and Design*, pp. 424-427, August 2003.

[16] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, S. Amarasinghe, "Space-Time Scheduling of Instruction-Level Parallelism on a Raw Machine," in *Proc. of the 8th Int. Conf. on Arch. Support for Programming Languages and Operating Syst.*, October, 1998

[17] J. Luo, N. K. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in *Proceedings of the 38th Conference on Design Automation*, pp. 444-449, 2001.

[18] E. Macii, M. Pedram, F. Somenzi, "High-Level Power Modeling, Estimation and Optimization," *IEEE Trans. on CAD of Integrated Circuits and Systems.*, 1998.

[19] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, M. Horowitz, "Smart Memories: A Modular Reconfi gurable Architecture," in *Proc. of the International Symp. on Comp. Arch.*, June, 2003.

[20] S. Murali, G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," *Design, Automation and Test in Europe Conference and Exhibition Volume II*, February, 2004.

[21] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, C. R. Moore, "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture," in *Proc. of the International Symp. on Comp. Arch*, pp. 422-433, June, 2003.

[22] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, J. Rabaey, S. Malik, A. Sangiovanni-Vincentelli, "Addressing the System-on-a-Chip Interconnection Woes throughCommunication Based Design" Invited Paper, DAC 2001.

[23] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, B. Calder, "Discovering and Exploiting Program Phases," *IEEE Micro: Micro's Top Picks from Computer Architecture Conferences*, December 2003

[24] A. Singh, W. J. Dally, A. K. Gupta, B. Towles, "GOAL: A Load-balanced Adaptive Routing Algorithm for Torus Networks," in *Proc. of the International Symp. on Comp. Arch.*, pp. 194-205, June, 2003.

[25] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, "Temperature-Aware Microarchitecture," In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003

[26] M. B. Taylor, W. Lee, S. Amarasinghe, A. Agarwal, "Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures," in *Proc. of the 9th Int. Symp. on High-Speed Comp. Arch.*, p. 341, February 2003.

[27] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: A First Step towards Software Power Minimization," *IEEE Trans. on VLSI Syst.*, vol. 2, no. 4, pp. 437-445, December, 1994.

[28] M. Vachharajani, N. Vachharajani, D. Penry, J. Blome, D.I. August, "Microarchitectural Exploration with Liberty," in *Proceedings of the 35th Int. Symp. on Microarchitecture*, November 2002.

[29] K. S. Vallerio and N. K. Jha, "Task Graph Transformation to Aid System Synthesis," in *Proc. Int. Conf. on Circuits and Systems*, vol. 4, pp. 695-698, May 2002.

[30] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, A. Agarwal "Baring It All to Software: Raw Machines," *IEEE Computer*, pp. 86-93, June 1997.

[31] H-S. Wang, X. Zhu, L-S. Peh, S. Malik, "Orion: A Power-Performance Simulator for Interconnection Networks," in *Proc. of MICRO 35*, November, 2002.

[32] S. Wilton, N. Jouppi, "Cacti: An enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, May 1996. pp. 677-688.

[33] T-Y. Yen, W. Wolf, "Communication synthesis for distributed embedded systems," in *Proc, of the 1995 IEEE/ACM international conf. on Computer-aided design*, pp. 288-294, November, 1995.

[34] http://www.eecs.umich.edu/mibench

[35] http://www.intel.com/products/server/processors/server/itanium/

[36] http://www.simplescalar.com

[37] http://www.spec.org