Designing Energy-Efficient NoC for Real-Time Embedded Systems through Slack Optimization

Jia Zhan*, Nikolay Stoimenov†, Jin Ouyang‡, Lothar Thiele†,
Vijaykrishnan Narayanan*, Yuan Xie*,§
*The Pennsylvania State University, {juz145,vijay,yuanxie}@cse.psu.edu
†Computer Engineering and Networks Laboratory, ETH Zurich, {stoimenov,thiele}@tik.ee.ethz.ch
‡NVIDIA, jouyang@nvidia.com
§AMD Research, yuan.xie@amd.com

ABSTRACT

Hard real-time embedded systems impose a strict latency requirement on interconnection subsystems. In the case of network-on-chip (NoC), this means each packet of a traffic stream has to be delivered within a time interval. In addition, with the increasing complexity of NoC, it consumes a significant portion of total chip power, which boosts the power footprint of such chips. In this work, we propose a methodology to minimize the energy consumption of NoC without violating the prespecified latency deadlines of real-time applications. First, we develop a formal approach based on network calculus to obtain the worst-case delay bound of all packets, from which we derive a safe estimate of the number of cycles that a packet can be further delayed in the network without violating its deadline—the *worst-case slack*. With this information, we then develop an optimization algorithm that trades the slacks for lower NoC energy. Our algorithm recognizes the distribution of slacks for different traffic streams, and assigns different voltages and frequencies to different routers to achieve NoC energy-efficiency, while meeting the deadlines for all packets.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Design

Keywords

Network-on-Chip, Network calculus, Voltage-frequency scaling

1 Introduction

Contemporary embedded systems and SoCs feature an increasing number of processing elements (PE) and other components, a sign that interconnection will play a more vital role in these chips. Network-on-chips (NoC) is a promising design paradigm for future many-core chips as found by many previous researches [1, 7]. However, the fundamental challenge of using NoCs in many-core embedded systems is that these systems often have very limited resources and stringent processing latency requirements, which places very different constraints than general-purpose processors on NoC design. There are two major differences between embedded systems and general-purpose processors: 1) General-purpose processors are often designed to achieve a high aggregate throughput, and therefore the NoCs for them are allocated sufficient resources to sustain the peak performance. In contrast, embedded systems are designed to provide *just enough* performance to accommodate specific tasks. Thrift is a virtue in designing NoC for those systems, in order for power and area reduction. 2) General-purpose processors care about the overall progress of all tasks running on all cores. In contrast, embedded systems often provide certain guarantees for individual tasks' progress. In the so-called *hard real-time embedded systems*, to provide certain quality-of-service (QoS), each task has an associated maximum allowed communication delay. Reflected on NoC, each network packet needs to be delivered to the destination before a *deadline*; otherwise the corresponding task may not be able to deliver the required quality-of-service, and even causes catastrophic outcomes.

One way to address the conflicting requirements of energy and latency is to leverage the inherent heterogeneity in NoC traffics, and use voltage frequency scaling (VFS) to improve the energy-efficiency of NoC. A lot of previous work [15,19,21] have adopted DVFS to reduce the energy consumption of NoC while still providing high throughput. Heterogeneity can also be utilized to improve the efficiency of NoC. Das *et al.* [9] was the first to propose the idea of ***network slack***, which refers to the number of cycles that a packet can be delayed in the network without affecting execution time. In their work, packets with smaller slacks (those more likely to impact execution time) are prioritized. This slack-based approach improves the throughput of all running tasks. However, the above researches are still focused on designing NoC for general-purpose processors, and aimed at improving the overall throughput. For example, the estimated slack proposed by Das *et al.* [9] does not consider precise deadlines on individual packets and only serves as a hint in assigning priorities to packets. There is no guarantee that a packet will arrive in time before it is needed. Therefore, these approaches cannot be applied to NoC in embedded systems where violating deadlines could be disastrous.

Unlike previous work, we focus on improving NoC energy-efficiency in hard real-time embedded systems by leveraging the heterogeneity in NoC traffics. We propose a design methodology that provides *just enough power* to NoC in order to meet the latency requirements (deadlines) of all traffic streams. Inspired by Das *et al.*'s work [9], we first calculate the worst-case *slacks* for packets of different streams. Then an energy optimization algorithm is proposed that leverages the slacks to allocate differentiated resources (energy) to different portions of the network. Different from their work, the slack calculation must be precise and conservative in order to guarantee the

This work is in part supported by NSF CCF-0903432, CNS-0905365, and SRC grant.

DAC '13, May 29 - June 07 2013, Austin, TX, USA

timing-correctness of real-time systems. To solve this problem, we adopt network calculus [3, 4] to predict the worst-case latency of different packets, from which the worst-case slacks can be obtained. Leveraging these slacks, we can progressively reduce the voltages and frequencies of individual routers in the network to reduce energy consumption while still meeting all deadlines. To summarize, the contributions of this work are:

• We develop a formal method based on network calculus to obtain the worst-case slacks of packets in the NoC for hard real-time embedded systems. We improve over previous work [18] by taking virtual channels and heterogeneous router frequencies into consideration.

• We propose an effective algorithm that trades slacks for energy-efficiency of NoC, and thus minimizes the total communication energy while still maintaining timing-correctness. This algorithm adjusts energy and performance by applying voltage and frequency scaling (VFS) to individual routers in the network.

Finally, the voltage-frequency assignments computed using the proposed approach are static, which fits into a design category where a large number of real-time embedded systems fall into. While not considered here, the framework devised in this work can be applied in dynamically reconfigured NoCs by periodically performing voltage/frequency scaling based on run-time network states.

2 Worst-Case Delay Analysis

2.1 Router Architecture

Most of the state-of-the-art NoC researches assume a baseline wormhole router that achieves high energy-efficiency [7]. To provide guaranteed services in NoC, researchers extended the baseline router architecture to either pre-allocate switching time slots for critical packets [10,11], or preserve a virtual channel for each traffic stream [2,20]. Pre-allocating switching time slots eliminates run-time contentions altogether, while preserving virtual channels only prevents head-of-line blocking and still needs proper arbitration schemes for performance guarantee.

In this paper, we assume a baseline wormhole router architecture with five router stages (1. **BW**: Buffer Write; 2. **RC**: Routing Computation; 3. **VA**: Virtual Channel Allocation; 4. **SA**: Switch Allocation; 5. **ST**: Switch Traversal). Recently NoC router architectures with fewer stages were also proposed. However, changing the number of router stages affects only the initial latency in our analysis (refer to details below), and therefore our approach can be applied to router architectures with fewer pipeline stages. In addition, we assume that each traffic stream uses a dedicated virtual channel throughout the network, which is in line with the designs proposed by [2,20]. We do not opt for pre-allocating time-slots for each packet [10,11], because this approach eliminates the flexibility of scaling voltage and frequency to reduce energy consumption.

In this section, the detailed analytic models for the two types of router pipeline stalls are presented, and the worst-case packet delay bound is derived from these models. Table 1 summarizes symbols used in our modeling and analysis.

2.2 Principles of Network Calculus

Network calculus [3] is a theory of deterministic queuing systems for communication networks. In particular, this approach is based on three important concepts:

Arrival Curve: If $A[s,t)$ denotes the number of packets (here we define a packet as a fixed-length basic unit in network traffics; variable-length packets can be viewed as a sequence of fixed-length packets) that arrive in the time interval $[s,t)$, then we say the flow A is constrained by an arrival curve α if and only if for all $s < t$:

$$A[s,t) \leq \alpha(t-s) \tag{1}$$

Table 1: Symbols used for modeling and analysis

Symbols	Description
α	arrival curve
β	service curve
β^{R_i}	overall service curve of router R_i
$\beta^{R_i'}$	ideal service curve of router R_i without back-pressure
$A[s,t)$	the number of packets that arrive during $[s,t)$
$C[s,t]$	the number of packets that can be processed during $[s,t]$
d_{worst}	worst-case packet delay
s_i	number of slots assigned to flow i in the scheduling model
B	VC buffer size
η	router frequency scaling factor
D_i	deadline constraint for flow i
$\otimes$	min-plus convolution e.g. $a \otimes b = \min_{0 \leq s \leq t}\{a(s)+b(t-s)\}$
$\wedge$	infimum e.g. $a \wedge b = min\{a,b\}$
δ_T	burst delay function $\delta_T = +\infty$ if $t > T$, else 0
$\gamma_{r,b}$	affine arrival curve $\gamma_{r,b}(t) = rt + b$ if $t > 0$, else 0
$\beta_{\lambda,T}$	rate-latency function $\lambda[t-T]^+ = \lambda(t-T)$ if $t > T$, else 0
$\overline{f}$	sub-additive closure $\overline{f} = \delta_0 \wedge f \wedge (f \otimes f) \wedge (f \otimes f \otimes f) \wedge ...$

Service Curve: If $C[s,t)$ denotes the number of packets that can be processed by a router or a whole network over the time interval $[s,t)$, and C is bounded by a service curve β if and only if for all $s < t$:

$$C[s,t) \geq \beta(t-s) \tag{2}$$

Delay Bound: Assume a packet stream, constrained by an arrival curve α, traverse a system that offers a service curve β. Then the worst-case packet delay d_{worst} can be bounded as:

$$d_{worst} \leq \sup_{t \geq 0}\{\inf\{\tau \geq 0 : \alpha(t) \leq \beta(t+\tau)\}\} \tag{3}$$

An example is shown in Figs. 1a and 1b for a single router, where we show an affine arrival curve $\gamma_{r,b}$, defined by: $\gamma_{r,b}(t) = rt + b$ for $t > 0$, and $\gamma_{r,b} = 0$ otherwise, and a rate-latency service curve $\beta_{\lambda,T}$, defined by: $\beta_{\lambda,T}(t) = \lambda[t-T]^+ = \lambda(t-T)$ for $t > T$, and $\beta_{\lambda,T} = 0$ otherwise. The arrival curve $\gamma_{r,b}$ implies that the source can send at most b packets at once, but no more than r packets/cycle in the long run, while the service curve $\beta_{\lambda,T}$ implies a pipeline delay T for a packet to traverse a router and an average service rate of λ packets/cycle. As shown in Fig. 1b, the worst-case delay bound d is the maximum horizontal distance between arrival curve and service curve.

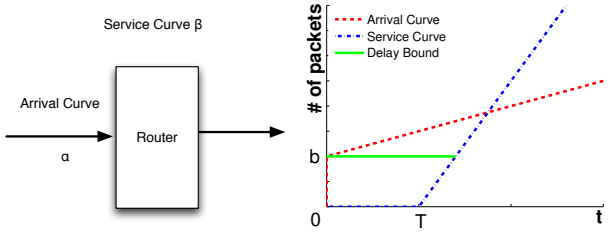
When extended to multiple interconnected components, as shown in Fig. 1c, the end-to-end packet delay becomes more unpredictable. Fig. 1d is the worst-case delay analysis for one flow f_2. As we can see, arrival curve α remains as the given injection pattern, while service curve is now the concatenation of all the routers that f_2 traverses from source to destination, which can be calculated through server concatenation [3]. For instance, the concatenation of two routers with service curve β^{R_1} and β^{R_2} is:

$$\beta^{R_{\{1,2\}}} = \beta^{R_1} \otimes \beta^{R_2} = \min_{0 \leq s \leq t}\{\beta^{R_1}(s) + \beta^{R_2}(t-s)\} \tag{4}$$

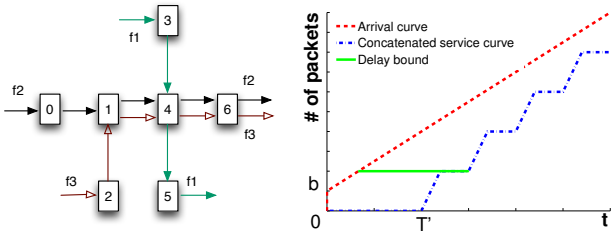
So far we only consider the case where routers with infinite buffers provide service to a single flow. In reality, the router is designed with a finite buffer size that exerts back-pressure, and many flows may share routers in NoC experiencing reduced service quality. Both factors introduce additional stalls (flow-control stall and switch-contention stall). In the rest of this section, we consider the additional stalls resulted from back-pressure and resource sharing in the worst-case delay analysis.

2.3 Flow-Control Stall

With credit-based flow control [8], the upstream router keeps a count of the number of free buffers in each virtual channel downstream. No packets will be forwarded if their intended



(a) single router (b) delay bound for a router



(c) multiple flows (d) delay bound for a flow

Figure 1: Delay bound from network calculus

buffers are full, until the downstream buffer forwards a packet and sends a credit back to the upstream router. Here we adapt Chang *et al.*'s work [5] to derive the worst-case latency bound under the back-pressure of credit-based flow control. For simplicity, we consider two adjacent routers R_1 and R_2 in our demonstration, and the results can be easily applied to the case when more routers are involved. Fig. 2 shows a graphical view of the two-router case.

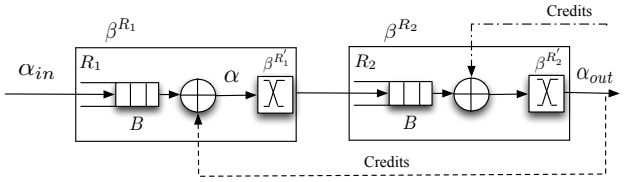


Figure 2: Analysis of credit-based flow control

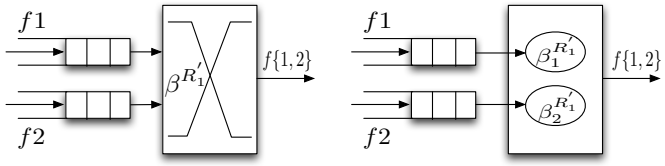
Let α_{in} be the generic input process to router R_1 while α be the effective input process to the internal crossbar of the router, which is the outcome of both α_{in} and back-pressure. α_{out} is the output process of router R_2. Suppose the overall service curve β^{R_2} of R_2 seen by R_1 is known, and the ideal service curve (without back-pressure) of R_1 is $\beta^{R_1'}$ (provided by the crossbar), then according to [5], the overall service curve β^{R_1} of R_1 considering back-pressure is given as:

$$\beta^{R_1} = \beta^{R_1'} \otimes \overline{(I_B \otimes (\beta^{R_1'} \otimes \beta^{R_2}))} \tag{5}$$

where B is the buffer size, and I_B is defined as $I_B(t) = \infty$ for $t > 0$ and $I_B(0) = B$. The horizontal bar is the operation for sub-additive closure. In this way, we can derive the service curve of each router and then recursively concatenate them based on Equation (4) from destination to source to get the concatenated service curve for all routers along a flow's path.

2.4 Switch-Contention Stall

Packet stall can happen at switch allocation stage, when all front packets in different virtual channels compete for the same crossbar input or output port. Here we model a generic switch arbiter that allocates time slots to different input ports according to their priorities. In Fig. 3a, we show an example where two flows arrive at a router and compete for the same output link to illustrate service curves experienced by each flow.



(a) (b)

Figure 3: Analysis of switch contention

The length of individual slot $s_i (i = 1, 2)$ assigned for flow f_i is proportional to the relevant priorities of incoming flows and should only take values that are multiples of a cycle length, and the corresponding total cycle length is $s = \sum_i s_i$. Assume the full ideal service curve of the router is $\beta^{R_1'}$, then the partial ideal service curve for f_i, as shown in Fig. 3b, is proportional to the slot distribution:

$$\beta_i^{R_1'} = \frac{s_i}{s} \beta^{R_1'} \otimes \delta_{s-s_i} \tag{6}$$

where δ_{s-s_i} is the delay bound when stream f_i just missed its slots in the worst case and has to wait for the next round.

As for stream f_i, (5) can be re-written as:

$$\beta_i^{R_1} = \beta_i^{R_1'} \otimes \overline{(I_B \otimes (\beta_i^{R_1'} \otimes \beta_i^{R_2}))} \tag{7}$$

Then by plugging Equation (6) into (7) we can derive the allocated service curve for a specific stream at each router it traversed, and the concatenated service curve according to (4). Finally, the worst-case delay bound can be obtained by applying the principles of network calculus. In this paper, we assume a ***round-robin*** arbiter for every router in the network, which implies that *the priorities of each flow are proportional to their arrival rates to a specific router*. While not considered here, such results can be obtained in a similar way for other scheduling policies like Fixed Priority (FP), Rate Monotonic (RM), and Earliest Deadline First (EDF).

3 Slack Optimization for Saving Energy

Applying the methodology in the previous section, we are able to bound the worst-case packet delay for individual application streams, hence obtaining *worst-case slack*—the time interval between the delay bound and its pre-specified transmission deadline, which indicates the number of cycles that a packet can be further delayed in the network. Based on this idea, we propose an energy-aware voltage and frequency scaling approach to optimize network energy-efficiency under deadline constraints.

3.1 Frequency and Voltage Scaling

The existence of packet slack implies that we can still achieve the required performance while lowering the operating frequency of some routers instead of making them run at a homogeneous high speed. The supply voltage, in the meantime, can be reduced together with the frequency to reduce energy.

Voltage-frequency islands (VFI) [14] have been adopted for achieving fine-grain system-level power management. A fine granularity partitioning could assume that each module in the design belongs to a different island [16] for best flexibility, or find the optimum partitioning via island merging [17] for energy savings. Here we are doing static voltage-frequency assignment and model these routers to be able to run at its own voltage and frequency. For completeness, we also explore the energy gain of operating all routers at homogeneous voltage and frequency.

In order for the network routers to operate at different frequencies, they should communicate in a Globally Asynchronous

Locally Synchronous (GALS) mechanism. To address synchronization latency, we adopt the fast synchronizer proposed by Dally *et al.* [6] which adds only half cycle of synchronization delay that can be well absorbed in the buffer write stage.

Note that if we assign a lower frequency to a router, its service curve is affected accordingly. Specifically, the packet service rate will decrease and the time it takes to traverse a router will increase. For example, for a router R_k with rate-latency service curve $\beta^{R_k}_{\lambda,T}(t) = \lambda[t-T]^+$ as discussed in Section II, when its operating frequency is scaled by a factor η, its new service curve is modeled as:

$$\beta^{R_k}_{\eta\lambda,T/\eta}(t) = \eta\lambda[t - T/\eta]^+ \tag{8}$$

Then the worst-case packet delay should be updated to check if there is remaining slack time for further optimization. Our energy optimization algorithm is described below in detail.

3.2 Energy-Aware Heuristic Search Algorithm

There are two straightforward ways to trade slack for energy savings. One is to simply scale down all the routers simultaneously by the same factor, which will keep them running at homogeneous frequency and voltage. The other is through exhaustive search to find out the optimum assignment. However, the former approach is not flexible enough for adjustment, because the degree to which the network speed can be reduced is limited by the packet flow with minimum slack. The latter is time-consuming and not scalable for a large network with multiple voltage-frequency levels.

Therefore, we propose an energy-aware heuristic search (EHS) algorithm, which can find an efficient solution leveraging network heterogeneity and avoiding exhaustive search at the same time. Specifically, we abstract a NoC energy model and integrate it into our worst-case delay analysis framework to automatically generate the frequency-voltage assignments.

3.2.1 Energy Models

The set of nodes in the network is denoted by $T = \{0, 1...N-1\}$. The supply voltage-frequency pairs of each node $i \in T$ are given by (V_i, f_i). Then the sum of dynamic and static energy consumption associated with node i is:

$$E(V_i, f_i) = E_d(V_i, f_i) + E_s(V_i, f_i) \tag{9}$$

The dynamic energy part can be calculated through:

$$E_d(V_i, f_i) = M_i * E_p(V_i, f_i) \tag{10}$$

where M_i is the total number of packets that traverse node i during execution, and $E_p(V_i, f_i)$ is the energy consumption when a packet traverses node i:

$$E_p(V_i, f_i) = E_{buffer} + E_{switch} + E_{link} \tag{11}$$

where E_{buffer}, E_{switch}, and E_{link} represent the energy dissipated at input buffers, switch and link and are found experimentally using ORION 2.0 [13].

The static energy $E_s(V_i, f_i)$ part is defined as:

$$E_s(V_i, f_i) = P_s(V_i, f_i) * t \tag{12}$$

where t is the system execution time, while $P_s(V_i, f_i)$ is static power for node i and can be obtained as:

$$P_s(V_i, f_i) = I^i_{static} * V_i \tag{13}$$

where I^i_{static} is the leakage current for node i and can also be extracted from ORION 2.0 [13].

Thus, combining Equation (9), (10) and (12), the total NoC energy consumption for an application can be expressed as:

$$E = \sum_{i=0}^{N-1} (M_i * E_p(V_i, f_i) + P_s(V_i, f_i) * t) \tag{14}$$

3.2.2 Algorithm Description

If node i is scaled from the current voltage-frequency level (V_i^k, f_i^k) to the next lower level (V_i^{k+1}, f_i^{k+1}), then energy reduction can be expressed as:

$$\begin{aligned}\Delta E_i = \sum_{i=1}^{N} (M_i * (E_p(V_i^k, f_i^k) - E_p(V_i^{k+1}, f_i^{k+1})) \\ + P_s(V_i^k, f_i^k) * t^k - P_s(V_i^{k+1}, f_i^{k+1}) * t^{k+1})\end{aligned} \tag{15}$$

Under deterministic routing, the only undetermined variable is the system execution time t^k. Assume the set of application streams is denoted by $S = \{s_1...s_m\}$. The number of packets injected by s_j is M_{S_j} with average injection rate r_{s_j}. Then t^k can be approximated from the slowest stream:

$$t = \max_{s_j \in S} \{M_{s_j}/r_{s_j}\} \tag{16}$$

This is because the end-to-end packet delay is negligible compared to t, especially when the packet number is large.

At the same time, the service curve β^{R_i} is modified based on Equation (8) if scaling i and the new stream delay d_{s_j} is calculated via the worst-case delay analysis in section II. The accumulated slack cost after scaling is represented as:

$$\Delta d_i = \sum_{s_j \in S} \Delta d_{s_j} \tag{17}$$

where Δd_{s_j} is the reduced slack for stream s_j.

Our heuristic search algorithm uses $\Delta d_i/\Delta E_i$ as a measure of the slack cost and the energy gain if we adjust the voltage-frequency level of a router i. The pseudo-code below outlines this algorithm. Specifically, the algorithm iterates through all routers in the network. At each iteration, it generates a list containing the slack costs and the energy gains for all routers in the network, picks the router i with lowest $\Delta d_i/\Delta E_i$ without causing deadline violations, and steps down the router's voltage and frequency. Finally, the algorithm terminates when there is no router in the network of which the voltage and the frequency can be further reduced without causing timing violations.

Algorithm 1: Energy-aware heuristic search algorithm

```
Result: Frequency-voltage assignment for all nodes
Initialize: Flag = 0;
while Flag == 0 do
    Flag = 1;
    for i ← 0 to N − 1 do
        if f_i is at its lowest level then
            continue;
        else
            /* worst-case delay analysis                    */
            Calculate d_{s_j}(∀s_j ∈ S), Δd_i and ΔE_i if scaling
            down f_i by one level, and insert them into a list L
            as one element ;
        end
    end
    Sort L in ascending order of Δd_i/ΔE_i, associated with the
    original index i;
    for each entry in the list L do
        if d_{s_j} < D_{s_j}(∀s_j ∈ S) then
            Scale down f_i by one level; Flag = 0; break;
        end
    end
end
```

For an N-node NoC with k voltage-frequency levels for each node, the algorithm complexity of our EHS algorithm is $(k-1) * N^2 logN$, compared to k^N for exhaustive search.

4 Experiments

4.1 Experimental Setup

We implemented a cycle-accurate network simulator based on the booksim 2.0 simulator [12], with dynamic and leakage power numbers extracted from ORION 2.0 [13] .

We also analyze the timing behavior of some video applications and characterize the arrival curves for packet streams. Table 2 shows the simulator and benchmark configurations.

Table 2: Simulation Parameters

Baseline Network Configuration			
Topology	2D mesh	Phit width	128bits
Size	4×4=16	Frequency	2GHz
VC #	3	Voltage	1.5v
Buffer depth	4 flits	Routing	dimension-order
Video Application Configuration			
MJPEG		PiP (HR)	PiP (LR)
Frame	352×240	Frame	704×576
Period	90,000	PE service	226.57
Throughput	307.2KB	Macroblock #	1584
JPEG size	8Kb	Rate: 25 frames/s	Rate: 12.5 frames/s
Deadline Constraint (cycle)			
$D_1 = 50$		$D_2 = 95$	$D_3 = 50$

Motion-JPEG (MJPEG) decoder: The MJPEG decoder is a video codec in which each video frame is compressed as a JPEG image. The video of 352 × 240 pixels is spilt into JPEG image size of 8 Kb. The maximum throughput is 307.2 KB per invocation with a period of 90,000 cycles.

Picture-in-picture (PiP): We use two sets of video clips: Regular clips with moderate to high motion content and clips displaying still images. These two sets characterize the two streams high-resolution (HR) and low-resolution (LR). Incoming streams have the same frame resolution of 704 × 576 pixels but will be down-scaled for LR, and each frame consists of 1584 macroblocks. Frames are read at a constant rate of 25 frames/s for HR and 12.5 frames/s for LR. The service offered by a processing element is 226.57 macroblocks/ms.

A JPEG image or a macroblock is treated as a packet, and we derive arrival curves for the three packet streams:

$$\text{MJPEG stream } f_1\text{: } \alpha_1(t) = 0.218t + 3.0 \quad (18a)$$

$$\text{PiP HR stream } f_2\text{: } \alpha_2(t) = 0.175t + 13.109 \quad (18b)$$

$$\text{PiP LR stream } f_3\text{: } \alpha_3(t) = 0.086t + 4.37 \quad (18c)$$

And the baseline service curve is shown in (19) for a generic wormhole NoC router that can process one packet per cycle with a total pipeline length of five cycles.

$$\text{Baseline router: } \beta(t) = [1.0 \times t - 5]^+ \quad (19)$$

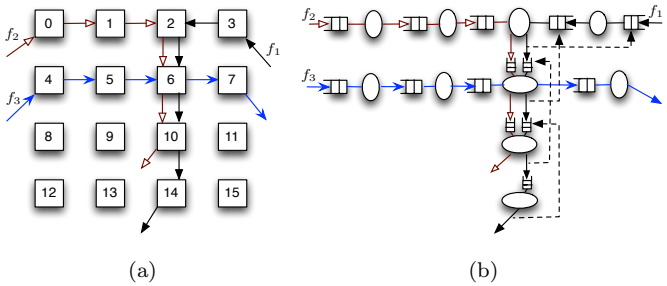
As a case study, we consider that the three application streams are mapped in a 4×4 mesh network shown in Fig. 4a with deterministic routing. Fig. 4b shows the resource sharing, including feedback loops in stream f_1 as an example. A detailed network configuration is shown in Table 2.

4.2 Experimental Results

We use the methodology in Section II to analyze the worst-case latency in our case study, and results are shown in Fig. 5.

At the same time, we run simulation to get the maximum packet latency for individual streams. A comparison between the calculated worst-case delay bound (solid lines) and the simulated maximum packet latency (dashed lines) when varying virtual channel buffer size B is shown in Fig. 6a. We can see that the calculated delay bounds are fairly tight.

Apart from the case study with three applications streams, we duplicate each of the three sample streams to generate more streams and map them on the NoC platform to form different traffic scenarios. The whole process is conducted randomly.



(a) (b)

Figure 4: Case study: Three video streams

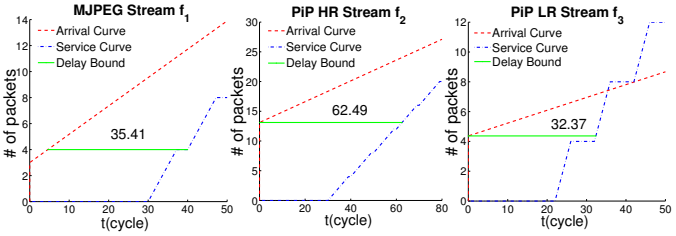
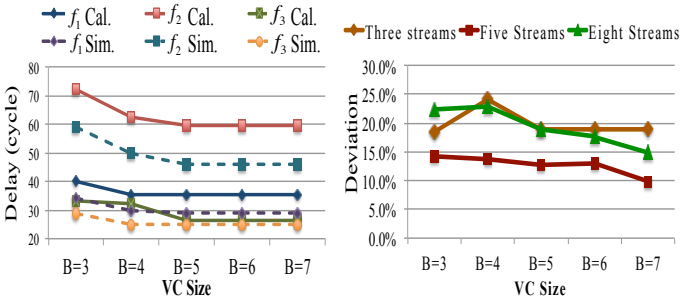


Figure 5: Delay bounds for three flows



(a) Delay comparison for three streams

(b) Average difference for three, five and eight streams

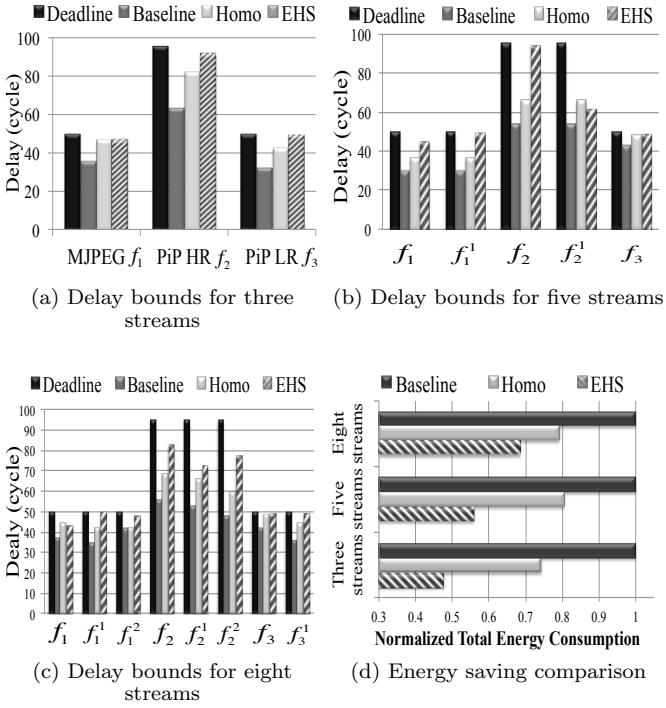
Figure 6: Calculated (*Cal.*) vs simulated (*Sim.*) delay bound

Here we consider running another five streams (Two MJPEG, two PiP HR, one PiP LR) and eight streams (Three MJPEG, three PiP HR, two PiP LR). Similarly, we do worst-case delay analysis to derive the delay bound and run simulation to get the maximum packet latency. Due to space limitation, we only show the differences between calculated delay bounds and simulated results as the ratio with respect to simulated results, when buffer size varies from 3 to 7, as shown in Fig. 6b.

We find that the average difference is 17.2% while the ratio is generally decreasing as VC size increases. This is because with a small VC size, the effect of back-pressure is more salient and the results from our analytical model is more pessimistic. With VC size increased, the effect of back-pressure is smoothed out, resulting in a tighter delay estimate that converges with the simulation results.

Furthermore, we perform the proposed EHS algorithm to reduce the energy of routers, assuming that three discrete frequencies are available (1.0, 1.5 and 2.0 GHz), and the minimum required voltage is 0.8, 1.2 and 1.5 volts in 45nm CMOS, respectively. As a comparison, we also evaluate homogeneous scaling (*Homo*) in which case the frequency-voltage level of the routers in the whole network is scaled together. Results of the worst-case delay bounds and the normalized total network energy consumption are shown in Fig. 7, where we refer to the

jth duplicate of the ith sample stream as f_i^j.



(a) Delay bounds for three streams

(b) Delay bounds for five streams

(c) Delay bounds for eight streams

(d) Energy saving comparison

Figure 7: Worst-case delay bound after frequency scaling and energy saving comparison for three, five and eight streams

As we can see from Figs. 7a to 7c, there is no deadline miss using any of these scaling mechanisms. We define *slack utilization* as the amount of slack scavenged by the algorithm to save energy, divided by the amount of initial slack under baseline configuration. Our proposed EHS algorithm has effectively exploited individual stream slack, making the completion time (delay) close to the deadline with a slack utilization of 80.7% on average. In contrast, $Homo$ only has a slack utilization of 53.9% on average.

As for energy optimization, shown by Fig. 7d, our proposed EHS mechanism significantly outperforms $Homo$. On average, EHS achieves 42.7% energy reduction, while $Homo$ saves 22.0%. These results confirm the effectiveness of our energy optimization algorithm. EHS can efficiently utilize the available slack of individual application streams for energy optimization, while for $Homo$ case, it cannot exploit slack in fine granularity and therefore leads to relatively poor energy savings.

In addition, for sensitivity evaluation, we apply the proposed EHS algorithm with different slack ratios available. Slack ratio is the amount of slack over baseline worst-case latency. As shown in Fig. 8, Our EHS algorithm works well under different slack ratios, saving energy by 23.1%, 31.1%, 38.4%, 52.0%, and 59.7%, respectively.

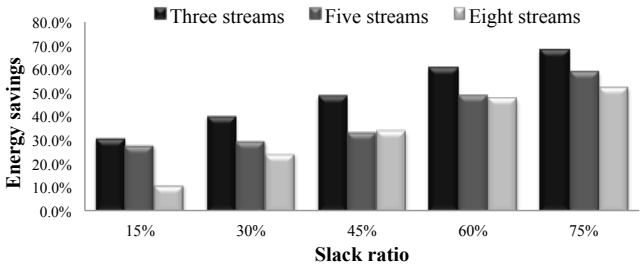


Figure 8: Energy savings when slack varies

5 Conclusion

In this paper, a formal analysis based on network calculus is adopted to obtain the worst-case slacks of packets in the NoC for hard real-time embedded systems, and used to trade slacks for energy savings by applying different voltages and frequencies to individual routers. Experimental results show that our worst-case delay analysis can derive a upper bound for packet latency, and our energy-aware heuristic search algorithm can effectively find the frequency-voltage assignment that can reduce network energy significantly under variable slack ratios.

6 References

[1] L. Benini and G. D. Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–78, 2002.

[2] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *DATE*, pages 1226–1231, 2005.

[3] J.-Y. L. Boudec and P. Thiran, editors. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, 2001.

[4] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, pages 190–195, Munich, Germany, 2003.

[5] C.-S. Chang, editor. *Performance Guarantees in Communication networks*, pages 78–83. Springer-Verlag, London, UK, 2000.

[6] W. Dally and S. Tell. The even/odd synchronizer: A fast, all-digital, periodic synchronizer. In *ASYNC*, pages 75–84, 2010.

[7] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *DAC*, pages 684–689, 2001.

[8] W. J. Dally and B. Towles, editors. *Principles and Pracitices of Interconnection Networks*, pages 245–247. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[9] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Aergia: exploiting packet latency slack in on-chip networks. In *ISCA*, pages 106–116, 2010.

[10] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: concepts, architectures, and implementations. *Design & Test of Computers, IEEE*, 22(5):414–421, 2005.

[11] K. Goossens and A. Hansson. The Æthereal network on chip after ten years: Goals, evolution, lessons, and future. In *DAC*, pages 306–311, 2010.

[12] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally. *BookSim 2.0 User's Guide*. Standford University, 2010.

[13] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A power-area simulator for interconnection networks. *IEEE Trans. on VLSI*, pages 191–196, 2012.

[14] D. Lackey, P. Zuchowski, T. Bednar, D. Stout, S. Gould, and J. Cohn. Managing power and performance for system-on-chip designs using voltage islands. In *ICCAD*, pages 195–202, 2002.

[15] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das. A case for dynamic frequency tuning in on-chip networks. In *MICRO*, pages 292–303, 2009.

[16] K. Niyogi and D. Marculescu. Speed and voltage selection for GALS systems based on voltage/frequency islands. In *ASPDAC*, pages 292 – 297, 2005.

[17] U. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung. Design and management of voltage-frequency island partitioned networks-on-chip. *IEEE Trans. on VLSI*, 17(3):330 –341, 2009.

[18] Y. Qian, Z. Lu, and W. Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *NOCS*, pages 44–53, 2009.

[19] L. Shang, L.-S. Peh, and N. K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *HPCA*, pages 91–102, 2003.

[20] A. Sharifi, H. Zhao, and M. Kandemir. Feedback control for providing QoS in NoC based multicores. In *DATE*, pages 1384–1389, 2010.

[21] P. Zhou, J. Yin, A. Zhai, and S. S. Sapatnekar. NoC frequency scaling with flexible-pipeline routers. In *ISLPED*, pages 403–408, 2011.