Power-performance assessment of different DVFS control policies in NoCs

Power-Performance Assessment of Different DVFS Control Policies in NoCs

Mario R. Casu[1], Paolo Giaccone[1]

Department of Electronics and Telecommunications, Politecnico di Torino, Italy

Abstract

We analyze the power-delay trade-off in a Network-on-Chip (NoC) under three Dynamic Voltage and Frequency Scaling (DVFS) policies. The first *rate-based* policy sets frequency and voltage of the NoC to the minimum value that allows to sustain the injection rate without reaching saturation. The second *queue-based* policy uses a feedback-loop approach to throttle the NoC frequency and voltage such that the average backlog of the injection queues tracks a target value. The third *delay-based* policy uses a closed-loop strategy that targets a given NoC end-to-end average delay. We first show that, despite the different mechanism and implementation, both rate-based and queue-based policies obtain very similar results in terms of power and delay, and we propose a theoretical interpretation of this similarity. Then we show that delay-based policy generally offers a better power-delay trade-off. We obtained our results with an extensive set of experiments on synthetic traffic, as well as multimedia, communications and PARSEC benchmarks. For all the experiments we report both cycle-accurate simulation results for the analysis of NoC delay, and accurate power results obtained targeting a standard-cell library in an advanced 28-nm FDSOI CMOS technology.

Preprint submitted to Journal of Parallel and Distributed Computing *June 14, 2017*

1. Introduction

Large-scale on-chip systems increasingly require high performance networks for efficiently connecting cores, caches, and hardware accelerators. As a result, the network-on-chip (NoC) is one of the leading contributors to the chip total power consumption [1][2][3][4]. To reduce the NoC power, researchers propose to apply Dynamic Voltage and Frequency Scaling (DVFS) with two different approaches, which can be termed as *local* and *global* DVFS. In the former approach, each NoC router belongs to a separate voltage/frequency (V/F) domain and has its own DVFS controller [5][6][7]. In the latter approach, the entire NoC belongs to a single V/F domain separated from the V/F domains of the processing elements, and is managed by a single DVFS controller [4][8].

Although local DVFS has potential for greater power savings, especially in NoC traffic scenarios where the load is not uniformly distributed, it also entails severe cost and performance overhead. We focus therefore on a global approach, which removes the cost of locally replicated DVFS controllers and DC-DC converters, and minimizes the extra latency caused by the resynchronization required at each clock-domain crossing. We assume, however, that the processing elements can have separate DVFS controllers.

Fig. 1 exemplifies this approach, by showing an NoC with a power manager (PM) that controls the NoC V/F domain, and the nodes connected to the NoC that belong to different V/F domains. Packets pay the latency penalty for clock-domain crossing only twice, when they enter and when they exit the NoC passing through the Network-Interface (NI). For reasons of scalability, for NoC mesh sizes greater than 8×8, we envision that multiple instances of the NoC V/F domain in Fig. 1 can be arranged, each with its own controller.

We consider that such a coarse-grain approach to multiple V/F domains is affordable. In this paper, though, we consider one instance of size up to 8×8.

In this context, we focus on the implementation of an NoC global DVFS controller and analyze the power-performance trade-off under three different power management policies. We term the first policy as *Rate-based Max Slow Down (RMSD),* because it runs the NoC at the slowest frequency—and so at the least voltage—compatible with the injected traffic rate. The second policy, which we term as *Queue-based Max Slow Down (QMSD),* slows down the NoC frequency in such a way that the injection queues located in the NIs remain loaded at a given target occupation. This result is obtained with a feedback-loop Proportional-Integral (PI) controller that tends to minimize the error between the average queue occupation and the target occupation. A similar closed-loop PI controller is used in the third policy, termed *Delay-based Max Slow Down (DMSD),* which tunes the NoC frequency to minimize the

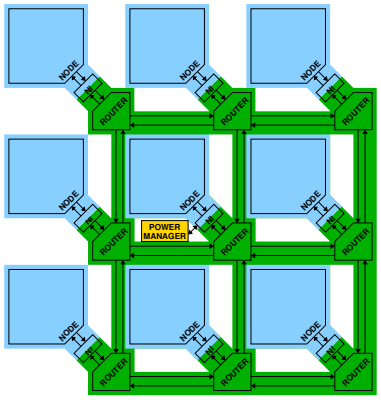

Figure 1: An NoC has its own voltage/frequency domain (in green) separated from the domains of the nodes connected to it (blue). Network-Interfaces sit at the crossing between two domains. A power manager is located in the central element (yellow).

error between the average end-to-end packet delay and a target delay.

Despite the apparent differences between RMSD and QMSD, we show that they behave in a very similar way, obtaining analogous results in terms of delay and power. We prove with a simplified model, based on a single-queue, that a sound theoretical reason holds for this equivalence.

We implemented the three power management policies in our modified version of Booksim2 [9], the Stanford's NoC cycle accurate simulator [10]. Moreover, targeting a 28-nm standard-cell technology, we ran logic synthesis and transistor-level simulations of the virtual-channel router used in network-level simulations to accurately characterize its power as a function of voltage, frequency, injection rate, and activity in the links, buffers, and crossbar.

Our main aim is to illustrate the trade-offs between performance and power saving that different DVFS policies obtain at the NoC level. We observe, though, that the DVFS policies analyzed share, for the most part, the underlying hardware structures, hence switching between policies at run-time to match the characteristics of the architecture and the workload is possible.

In summary, we provide the following novel contributions:

- We show that the QMSD and RMSD policies behave very similarly and typically pay a very high cost in terms of NoC delay compared to the DMSD policy. This is confirmed by experimental results obtained both with synthetic traffic and traffic generated by NoC benchmarks.

- Although RMSD and QMSD save more power, their delay degradation is often superior to the power advantage, when the NoC traffic is high. We conclude that a better trade-off between power and delay is typically

obtained with the DMSD policy. Instead, for the scenarios that generate a very low NoC traffic, for which delay degradation and power advantage are comparable, QMSD and RMSD appear to achieve the best tradeoff.

- We test our policies using different realistic benchmarks. In particular, we also simulate the PARSEC benchmarks in order to preserve the original packet dependencies and to achieve a similar accuracy than system-level simulations. The results show that all the policies behave as expected also under such realistic traffic patterns.

- We present a microarchitectural implementation of the DVFS controller that can be reconfigured to support all the DVFS policies.

This is the paper organization. Upon presenting the related work in Sec. 2, we focus on the policies implementation in Sec. 3: their performance is initially evaluated under simple traffic scenarios and then explained theoretically with an analytical queuing model. In Sec. 4 we outline the methodology through which we extend our performance analysis under realistic scenarios and power model. The results on performance and power are in Sec. 5 where we also discuss the implementation costs and evaluate a hardware implementation of the PM. We conclude in Sec. 6.

2. Related Work

Researchers have mostly focused on a fine-grain approach to DVFS, in which routers and links have separate DVFS controllers [5][6][11][12][13]. In these previous works, the overhead of replicated voltage regulators and frequency synthesizers, as well as the penalty for crossing many frequency

domains, is either not considered or assumed to be tolerable. Under a practical perspective, however, it is more realistic to assume that multiple domains with full-fledged voltage regulators and PLLs are reserved to the elements connected to the NoC, whereas the NoC forms a single separate domain. This *global* approach was proposed by Intel in the SCC chip [4] and in more recent works [8][14][15]. To make the local approach affordable, Yadav et al. in [7] propose a simplified DVFS with two voltages and discrete frequencies.

Admittedly, a fine-grain approach, if implementation cost and performance penalty were tolerable, can save more power than a global approach can. To close this gap, researchers use multiple NoC planes controlled by separate DVFS PMs [16]. We argue that a coarse-grain approach, with multiple DVFS domains each covering around 64 nodes, is affordable.

A DVFS controller that uses the occupancy of queues to automatically set voltage and frequency was first proposed in processors, in which the content of the queues represents the pending workload [17]. The application to NoCs consists in monitoring the flit queues located at the crossing between different voltage-frequency islands [18][7]. Although this queue-based approach to DVFS in NoCs is not new, here we discuss for the first time the drawbacks of this method in terms of degradation in the delay performance.

A similar performance problem affects the method suggested by Liang and Jantsch in [19] in the context of a global approach to NoC DVFS. Their method consists in slowing down the NoC frequency and forcing it to operate around its saturation point. We consider Liang and Jantsch's technique as an instance of the Rate-based Max Slow Down (RMSD) policy.

A delay-based approach to global DVFS using a PI controller has been

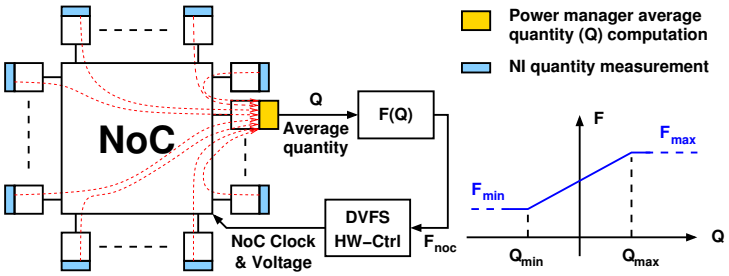

Figure 2: The power manager receives measurements from NoC nodes, computes the global average quantity Q, and computes the NoC clock frequency.

proposed in the context of a Chip Multi-Processor (CMP) [8][14], in which DVFS is applied simultaneously to the NoC and to the last-level cache. A different approach to this problem is proposed in [15], in which the DVFS controller is based on an artificial neural network trained with the help of a PI controller. Although we propose a similar delay-based approach, differently from these works, we propose a comparative analysis of the delay-based policy, the queue-based one, and the rate-based one. In addition, we do not focus on a specific architecture, like the CMP one in previous works, and put the accent instead on the network, with the aim of obtaining more general results.

3. DVFS Policies

In this section we describe a general framework that is common to the three policies. We introduce the required notation and discuss our assumptions. Then we describe each policy in a separate subsection. Finally, the last subsection introduces a theoretical framework based on an M/D/1 queueing model, which helps to clarify some of the findings related to the three policies.

Fig. 2 describes our framework for DVFS control. In each NI that connects

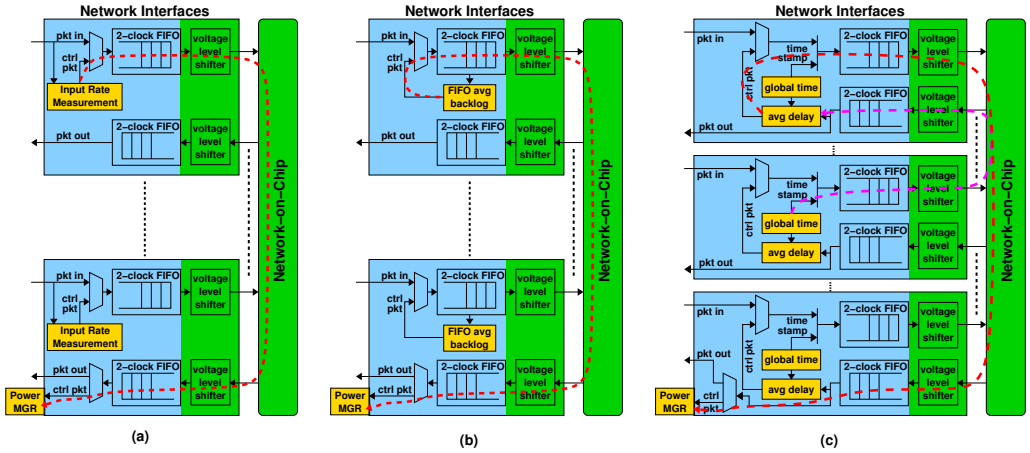

Figure 3: Network Interfaces send periodic control packets to the PM (in red dashed paths): (a) in RMSD the average input rate is measured; (b) in QMSD the average backlog is measured; (c) in DMSD the average delay is measured (the purple dashed lines shows the path of the data packet from its source NI to its destination).

node i to the NoC, a quantity is measured: In the RMSD policy it is the average injection rate at node i; in the QMSD policy, it is the average backlog of the injection queue located in the ith NI; in the DMSD one, it is the average end-to-end latency of all the packets received by the ith NI.

In each of the three policies the *locally* measured quantities are sent periodically to the *global* PM with period T_{ctrl}, as sketched in Fig. 3. In particular, Fig. 3(a) shows that in the RMSD policy each NI measures the rate of the flits injected by each node, and sends periodically to the PM a control packet with the average measured rate as payload. Fig. 3(b) shows that in the QMSD policy the control packet contains a locally measured average FIFO backlog. Fig. 3(c) shows that in the DMSD policy delays are measured at the receiver side, rather than at the transmitter side as

it happens in the other two policies, and control packets with the average delay are sent to the PM; to compute the delays of received data packets, a timestamp is added at the ingress NI. Note that control packets overhead is negligible, because one control packet per node made of two flits is sent every T_{ctrl}, which is on the order of on the order of 10 µs [8][1].

The PM uses the received measurements to compute a global quantity Q; then it uses a quantity-to-frequency mapping function to obtain the *network clock* of frequency F_{noc}. This mapping function is represented by the piecewise linear model in Fig. 2, which can be expressed as follows:

$$F_{noc} = \begin{cases} F_{\min} & \text{if } Q < Q_{\min} \\ F_0 + \dfrac{(F_{\max} - F_{\min})}{(Q_{\max} - Q_{\min})} Q & \text{if } Q_{\min} \leq Q \leq Q_{\max} \\ F_{\max} & \text{if } Q > Q_{\max} \end{cases} \quad (1)$$

where F_0 is the extrapolated value for $Q = 0$. Here $Q_{\min}$ and $Q_{\max}$ are introduced to linearize the controller behavior between $F_{\min}$ and $F_{\max}$; all the frequency values outside such range are clipped.

Depending on the frequency F_{noc}, the NoC voltage V_{noc} is also chosen in range $[V_{\min}, V_{\max}]$. The role of the *DVFS hardware controller (HW-ctrl)* in Fig. 2 is to tune a PLL and an on-chip DC-DC converter to ultimately set, respectively, the NoC clock to F_{noc} and the NoC voltage to V_{noc}. We assume that the DVFS HW-Ctrl is capable of nanosecond-scale voltage switching [20][21] and sub-nanosecond frequency switching [4].

[1]Although we did not perform an extensive sensitivity analysis, we found out, in accordance to [8], that this value is enough to collect traffic information and short enough to capture application phase changes.

Frequency and voltage are interdependent, because for a given circuit there is a minimum voltage value that guarantees correct operation at a given frequency. In Sec. 4 we discuss in detail this interdependence in the NoC routers implemented in our target technology. For now let us simply assume that once frequency F is chosen, voltage V is immediately obtained as a function of frequency $V(F) : [F_{\min}, F_{\max}] \rightarrow [V_{\min}, V_{\max}]$. This function is encoded in the DVFS hardware controller, which receives the information about the frequency selected by the DVFS policy and consequently sends a proper command to the DC-DC regulator for setting the appropriate voltage.

We assume that a *node clock* of frequency F_{node} is available at each injection node. For simplicity, we let F_{node} be fixed and equal for all the nodes: We prefer to keep the exposition simpler and more intuitive, but a more general treatment with different and variable node frequencies is possible. Without loss of generality, let the value of F_{node} be fixed to the maximum value $F_{\max}$.

Aim of each DVFS policy is to slow down the network clock with respect to the node clock (i.e., $F_{noc} < F_{\max}$) to reduce the power consumption while sustaining the traffic and keeping the throughput unaffected. The three policies achieve this goal in a different way, as we explain in Secs. 3.1-3.3.

In those sections, we introduce the policies and analyze their performance in two basic scenarios that represent two opposite cases: in the *uniform* case, the traffic matrix is such that all the nodes generate the same amount of traffic, destined uniformly to all other nodes; in the *hotspot* case, all the nodes generate the same amount of traffic destined to the same hotspot node. We do not introduce yet the power analysis, which we report instead later in Sec. 5, after the introduction of the detailed power model discussed in Sec. 4.

Table 1: Main NoC parameters: baseline case and cases considered for sensitivity analysis.

NoC parameter	Baseline	Sensitivity analysis	NoC parameter	Baseline	Sensitivity analysis
Mesh size	4×4	5×5, 8×8	Flits per packet	20	10, 15, 20
Routing	XY	XY	Flit width (bit)	64	64
Virt. Channels (VCs)	8	2, 4, 8	allocation	iSlip	iSlip
Buffers per VC	4	4, 8, 16	$[F_{\min}, F_{\max}]$ (GHz)	$[0.333, 1]$	$[0.333, 1]$

In Sec. 5, we extend our comparison under realistic traffic scenarios.

We implemented our policies in a micro-architectural cycle-accurate NoC simulator based on Stanford's Booksim2 [10], which we modified in various ways, but primarily by decoupling the NoC and the nodes frequencies. We considered the NoC baseline parameters in Tab. 1, but we also performed a sensitivity analysis, whose results are in Sec. 5. We obtained $F_{\min}$ and $F_{\max}$ in Tab. 1 with the circuit-level analysis presented in Sec. 4. We also set F_{node} equal to 1 GHz. Packets are generated according to a Bernoulli process.

We estimate the average *packet latency* L_{noc}, measured in terms of network clock cycles. Since the network clock may be slowed down with respect to the node clock, it is interesting to see the actual absolute delay D_{noc}, which is the ratio L_{noc}/F_{noc}. This delay will be denoted as *packet delay* and measured in nanoseconds. In addition, we evaluate the average backlog and report also the adopted NoC frequency. All these metrics will be shown as a function of the average injection rate, measured in terms of flits per node cycle.

As term of comparison, we report also the results for the case in which DVFS is not used (denoted as "No-DVFS") and F_{noc} is fixed at 1 GHz.

3.1. RMSD Policy

In the RMSD policy, Q is the average number of flits injected by each node in a node clock cycle, $Q = \lambda_{node}$. To obtain λ_{node}, each NI measures the rate of the flits injected by each node, $Q_i = \lambda_{node,i}$, and, as shown in Fig. 3(a), it sends periodically to the PM a control packet with $\lambda_{node,i}$ as payload. Since the NoC and the nodes belong to different Voltage/Frequency (V/F) domains, the NI is where the V/F crossing occurs. Therefore, the packet injection and ejection queues located in the NIs serve also as dual-clock resynchronizing FIFOs. To adapt the two voltage levels, level shifters are required.

Upon receiving all the $\lambda_{node,i}$ values, the PM computes the global average λ_{node}. To obtain the frequency-scaling law of (1), we first note that the average injection rate in *flits per second* at each node is simply obtained as the product of λ_{node} and the node clock frequency, $R_{node} = \lambda_{node} F_{node}$. Equivalently, the rate in flits per second seen by the NoC is $R_{noc} = \lambda_{noc} F_{noc}$, where λ_{noc} is the rate in flits per network clock cycle seen by the NoC.

The sustain the traffic, we must set $R_{node} = R_{noc}$ and we get:

$$\lambda_{noc} = \lambda_{node} \frac{F_{node}}{F_{noc}}. \tag{2}$$

Thus, when DVFS is applied and $F_{noc} < F_{node}$, the network sees more flits injected per network clock cycle and operates closer to its saturation point.

The main idea of RMSD is to minimize the network power consumption by slowing down F_{noc} as much as possible while preserving the maximum throughput. For a given traffic scenario, this is obtained by having the network work at an injection rate near but still below its saturation point.

We define this target rate as $\lambda_{\max}$. By setting $\lambda_{noc} = \lambda_{\max}$ in (2), we obtain:

$$F_{noc} = F_{node}\frac{\lambda_{node}}{\lambda_{\max}}. \tag{3}$$

For a given value of λ_{node}, by choosing F_{noc} as in (3), the network injection rate will be constant and equal to $\lambda_{\max}$. As a result, the average network latency will also be constant.

The frequency-scaling law in (3) is valid whenever F_{noc} is in range $[F_{\min}, F_{\max}]$. This frequency range corresponds, through (3), to a range of node injection rates $[\lambda_{\min}, \lambda_{\max}]$. The network injection rate λ_{noc} is equal to $\lambda_{\max}$, and the latency L_{noc} is constant, as long as the node injection rate λ_{node} stays within this range.

Under the assumption that $F_{node} = F_{\max}$, we can easily determine this range of node injection rates. In particular, from (3) we obtain $F_{noc} = F_{\max}$ when $\lambda_{node} = \lambda_{\max}$, and $F_{noc} = F_{\min}$ when $\lambda_{node} = \lambda_{\max}F_{\min}/F_{\max}$. We define this lower node injection rate as $\lambda_{\min}$. When λ_{node} is outside the range $[\lambda_{\min}, \lambda_{\max}]$, the network clock frequency is clipped to either $F_{\min}$ or $F_{\max}$.

In summary, the frequency-scaling law of (1) becomes

$$F_{noc} = \begin{cases} F_{\min} & \text{if } \lambda_{node} < \lambda_{\min} \\ \dfrac{F_{\max}}{\lambda_{\max}}\lambda_{node} & \text{if } \lambda_{\min} \le \lambda_{node} \le \lambda_{\max} \\ F_{\max} & \text{if } \lambda_{node} > \lambda_{\max} \end{cases} \tag{4}$$

where the extrapolated value for $Q = \lambda_{node} = 0$ is $F_0 = 0$ and the slope of the line is $F_{\max}/\lambda_{\max}$.

3.1.1. RMSD Performance

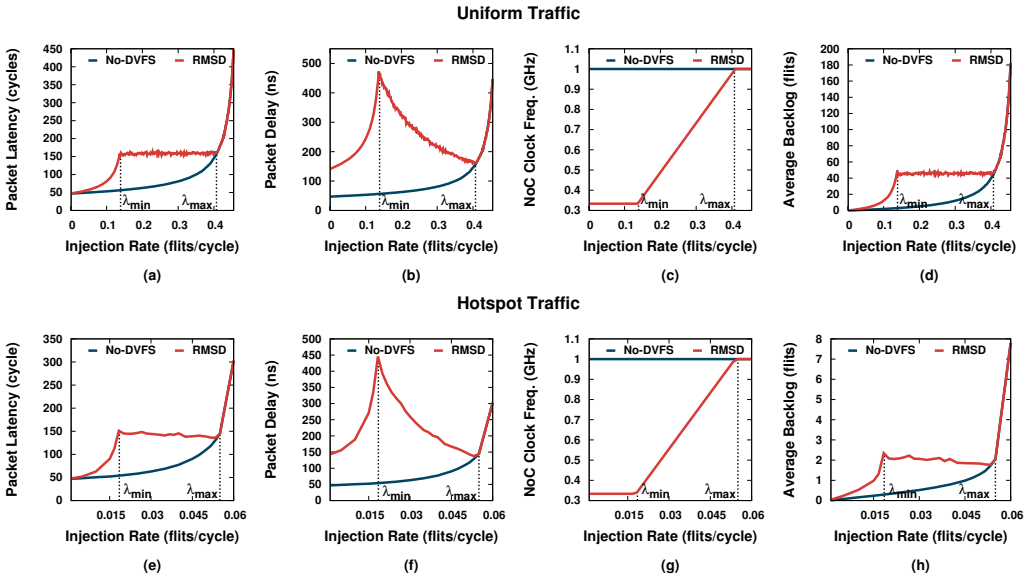

Figure 4: Performance figures without DVFS and with a RMSD policy in a 4×4 NoC.

For a given traffic scenario, RMSD requires to first set the value of $\lambda_{\max}$. This is obtained at design-time by slowing down F_{noc} till the NoC enters saturation. The value of λ_{node} at the onset of saturation is recorded and $\lambda_{\max}$ is set to a value 10% lower than this value. As an example, we report in Fig. 4 our simulation results for the uniform and the hotspot traffic scenarios. The saturation rate is 0.45 flits injected per clock cycle by each node for the uniform case and 0.06 for the hotspot case. At run-time, therefore, different values of $\lambda_{\max}$ will be used depending on the traffic scenario. In addition, if an application exhibits traffic phases with significantly different traffic patterns, we profile them at design-time and associate to them different $\lambda_{\max}$ values[2].

[2]Phase switching, of course, requires cooperation between the DVFS hardware controller and the OS, whose description is out of the scope of this paper.

As expected from (4), F_{noc} varies linearly between $F_{\min} = 333$ MHz and $F_{\max} = 1$ GHz, as shown in Figs. 4(c),(g).

Figs. 4(a),(e) show that the latency L_{noc}, as expected, is rather constant as long as λ_{node} is within the range $[\lambda_{\min}, \lambda_{\max}]$. What may be apparently surprising, is the behavior of the delay $D_{noc} = L_{noc}/F_{noc}$, reported in Figs. 4(b),(f). In the range $[\lambda_{\min}, \lambda_{\max}]$ the delay decreases because L_{noc} is constant and F_{noc} increases. In $[0, \lambda_{\min})$, the delay increases because F_{noc} is fixed to $F_{\min}$ and L_{noc} increases due to the increasing load. As a result, the delay is non-monotonic with very high values around the peak.

We have been the first to observe this non-monotonic behavior in a preliminary version of this paper [22] and in a previous work on DVFS policies for controlling the power of queue-based systems with a single server model [23]. The theoretical analysis of [23], albeit limited to a single queue, can be useful to understand what happens in a complex system of queues like the NoC. Therefore, we report in Sec. 3.4 a short summary of that analysis.

3.2. QMSD Policy

Both QMSD and DMSD policies use the Proportional-Integral (PI) loop controller of Fig. 5. In the QMSD case, the average backlog in the NI injection queues is compared with a reference backlog B_T, the *target* in Fig. 5. At each control update period n, the error signal E_n is computed as the difference between the sensed quantity (average backlog in QMSD) and the target value. E_n and the previous value of the error E_{n-1} are linearly combined—using proportional and integral gains K_I and K_P—with the previous value of the actuation function U_{n-1} so as to obtain the new value U_n.

As shown in Fig. 5, the role of the generic quantity Q in (1) is played

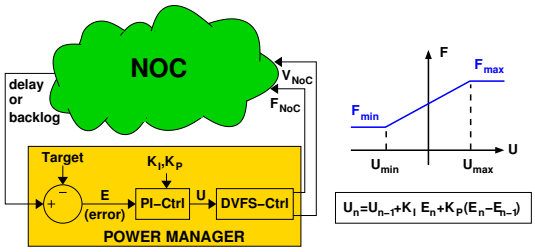

Figure 5: DVFS based on a Proportional-Integral control loop.

here by U, which is used to determine the NoC clock frequency F_{noc} within the range $[F_{\min}, F_{\max}]$ that corresponds to the range $[U_{\min}, U_{\max}]$. Since there is no reason not to keep the range of U symmetric around zero, we set $U_{\min} = -U_{\max}$. Therefore the frequency scaling law for QMSD becomes

$$F_{noc} = \begin{cases} F_{\min} & \text{if } U < -U_{\max} \\ F_0 + \dfrac{(F_{\max} - F_{\min})}{2U_{\max}} U & \text{if } -U_{\max} \leq U \leq U_{\max} \\ F_{\max} & \text{if } U > U_{\max} \end{cases} \tag{5}$$

where F_0 is the average of the two extreme frequencies, $F_0 = (F_{\max} + F_{\min})/2$.

To compute the average backlog of the various injection queues, the PM waits until it receives a control packet from all the NIs containing the locally measured backlog, as depicted by the red path in Fig. 3(b). Like in the RMSD case, the queues are dual-clock FIFOs and voltage-level shifters are required.

The control packet is sent periodically to the PM, at the update rate of the PI controller. Its payload is a time-averaged value of the FIFO occupation. In more detail, during the tth tick of the NI local clock, the "FIFO avg backlog" block in Fig. 3(b) computes the average number of flits in the ith NI queue, $B_{ave,i}[t]$, by means of a Cumulative Moving Average (CMA) as follows:

$$B_{ave,i}[t] = \frac{N-1}{N} B_{ave,i}[t-1] + \frac{1}{N} B_i[t] \tag{6}$$

where $B_i[t]$ is the instantaneous backlog of the ith queue, and N is a CMA parameter describing the averaging period. N cannot be set independently from the PI controller gains: We follow the procedure outlined in [24] to set these values such that the roots of the system characteristic equation are within the unit circle in the z-plane, which guarantees stability. We obtain a good compromise between response time and reduction of overshoot and oscillations with $K_I = 0.8$, $K_P = 0.4$ and $N = 8192$, for $T_{\text{ctrl}} = 10\,\mu\text{s}$.

The most critical parameter to determine is, however, the target average queue B_T. A too small value will cause the PI controller to almost always set the NoC frequency to $F_{\max}$ and the voltage to $V_{\max}$, hence reducing the potential advantage of the power management. Vice versa, a too large value will force the queues to fill even when the load is low or moderate, hence leading to very poor delay performance.

To choose the correct value, we propose the following method. For a given traffic scenario, we run an initial training at design-time to determine the *open-loop* average queue backlog, which is obtained when the PI controller is inactive and the NoC always runs at $F_{\max}$ and $V_{\max}$. Similarly to the RMSD tuning procedure, by progressively decreasing F_{noc}, we first empirically determine the saturation point. Then, by setting F_{noc} 10% greater than the value at saturation, we determine the operating point, which corresponds to an NoC injection rate 90% of the saturation rate. After a warm-up period, we measure the average queue content in that condition. That value is used at run-time as a reference B_T for that scenario: the entire procedure is repeated for possibly different traffic scenarios. In the case of applications with traffic phases, the procedure is repeated for each phase.

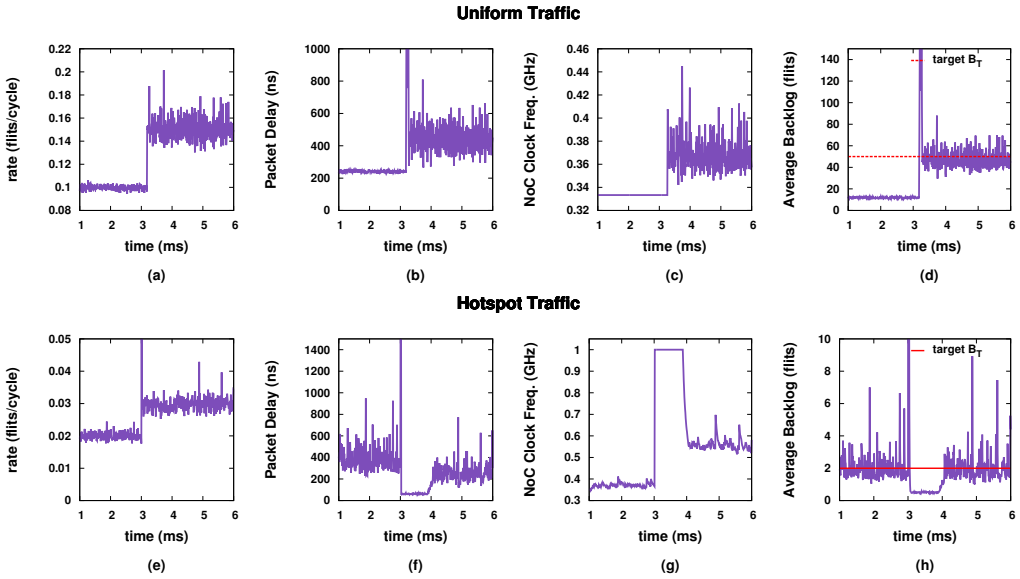

Figure 6: QMSD policy: rate step response in a 4×4 NoC.

When the PI controller works in *closed-loop*, by setting F_{noc} such that the average backlog tracks B_T, we obtain that, when the injection is between 90% and 100% of the saturation rate, the average queue tends to be larger than B_T and F_{noc} is set to $F_{\max}$. When instead the injection is less than 90% of the saturation rate, F_{noc} varies between $F_{\min}$ and $F_{\max}$. In conclusion, imposing a 90% saturation point allows to exploit DVFS for a wide rate range.

3.2.1. QMSD Performance

Fig. 6 shows the system response to a rate step occurring at 3 ms in case of uniform and hotspot traffic scenarios. We observe the abrupt frequency change in Figs. 6(c),(g) and the corresponding variation of the backlog in Figs. 6(d),(h) and the packet delay in Figs. 6(b),(f). Notice that after the rate step the average backlog tends to reach the target B_T, which indeed depends

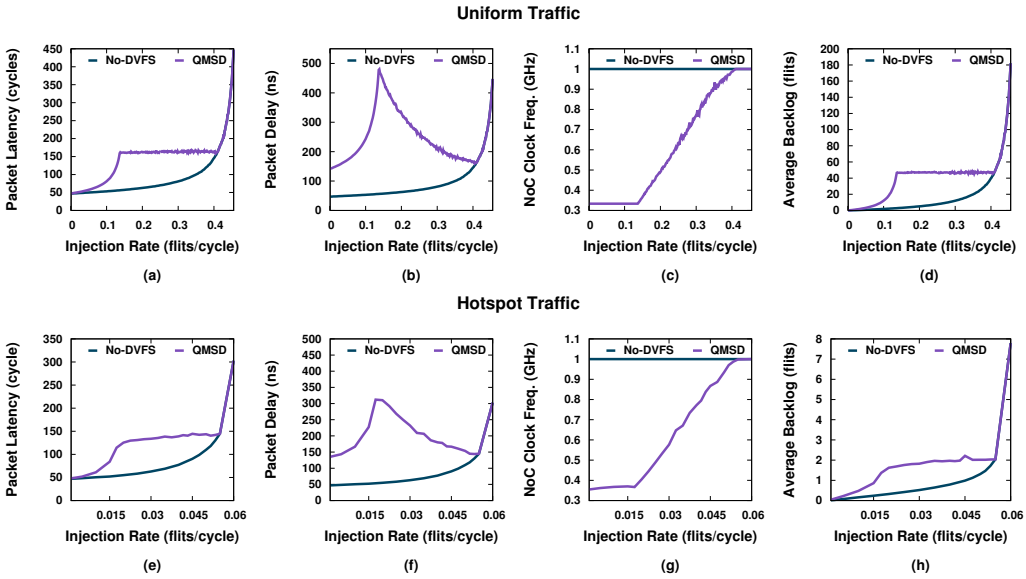

Figure 7: Performance figures without DVFS and with a QMSD policy in a 4×4 NoC.

on the traffic scenario: according to the previously outlined procedure, we obtain 50 and 2 flits for the two scenarios, respectively. Notice also that in the uniform scenario, the low rate before the step sets the frequency to $F_{\min}$ and the corresponding backlog settles to a value lower than B_T. Notice also that in the hotspot case a frequency overcorrection (capped at 1 GHz) is necessary to counteract the sudden backlog increase at 3 ms.

In Fig. 7 we report the simulation results obtained at steady-state (i.e. when all transients are expired). Under both traffic cases, QMSD keeps the backlog close to the target in a wide range of injection rates (Figs. 7(d),(h)). This range corresponds to the frequency range between $F_{\min}$ and $F_{\max}$ (Figs. 7(c),(g)). As expected, at around 90% of the saturation rate the average backlog in the QMSD case equals that of the No-DVFS case, and F_{noc} reaches $F_{\max}$.

By keeping the average occupancy constant, the QMSD policy obtains that

the end-to-end latency L_{noc} in terms of clock cycles remains constant in the range $[F_{\min}, F_{\max}]$, as shown in Figs. 7(a),(e). Since L_{noc} is constant and F_{noc} increases in that range, the end-to-end delay in nanoseconds, $D_{\text{noc}} = L_{\text{noc}}/F_{\text{noc}}$, decreases as we approach 90% of the saturation rate, as shown in Figs. 7(b),(f). Past this point, since L_{noc} increases and F_{noc} is constant and equal to $F_{\max}$, the delay increases, hence showing a minimum point.

The delay curve exhibits a peak in correspondence of the injection rate at which $F_{noc} = F_{\min}$. This happens because for lower rates F_{noc} is fixed to $F_{\min}$ and L_{noc} decreases, hence causing the delay to decrease; for higher rates, L_{noc} is constant and F_{noc} increases, hence also causing the delay to decrease. Such non-monotonic behavior is similar to what we obtained in the RMSD case. An explanation of such similarity is presented later in Sec. 3.4.

By comparing the two traffic scenarios we can observe that the level of average occupancy is very different, because the maximum injection rates in the two traffic cases differ by more than one order of magnitude. As a result, the target average occupancy depends strongly on the traffic scenario.

3.3. DMSD Policy

In the DMSD policy, the target of the PI controller is a fixed end-to-end packet delay, denoted as D_T, instead of the fixed backlog of the packet injection queues used in the QMSD policy. The scheme of the PI controller is, however, the same as the one previously shown in Fig. 5.

As shown in Fig. 3(c), like in the RMSD and QMSD cases, the NI packet queues serve also as resynchronizing FIFOs, and voltage-level shifters are needed. One main difference between the previously described policies and DMSD is that delays are measured at the receiver side, rather than at the

transmitter side: To measure packet delays, a timestamp is inserted in the header flit of each transmitted packet according to a global time reference and is extracted by the receiver NI (purple path in Fig. 3(c)).

To implement the NI global time references we use counters clocked at the same reference frequency from which the PM's PLL synthesizes the NoC clock frequency. A global reset synchronizes all the counters. The receiver NI extracts the timestamp and computes the packet delay as the difference (modulo 2^b, being b the bits of the counter and the timestamp) between the values of its counter and the timestamp. The number of bits b depends on the expected maximum delay. In all our experiments we measured end-to-end latencies less than 16,000 cycles, hence we used 14 bits. There are typically sufficient unused bits in the header flit for a 14-bit timestamp [8].

At every update period of the PI controller, which happens every 10 µs like in the QMSD case, each NI sends a control packet to the PM with the measured average delay and the number of received packets (red path in Fig. 3(c)). After collecting the control packets from all the nodes, the PM computes a weighted average delay (by weighting the delays with the corresponding number of received packets). Finally, the target delay D_T is subtracted from the average delay to obtain the error signal E_n of the PI controller in Fig. 5. The best values of the controller gains that we obtain in our simulations are K_I =0.025 and K_P=0.0125, which are obtained like in the QMSD controller in a way that guarantees stability [24].

Like in QMSD, the choice of the target is crucial. To determine the most suitable D_T value we propose a method similar to the one adopted in QMSD to determine B_T. With the PI controller disabled, we slow down F_{noc} until we

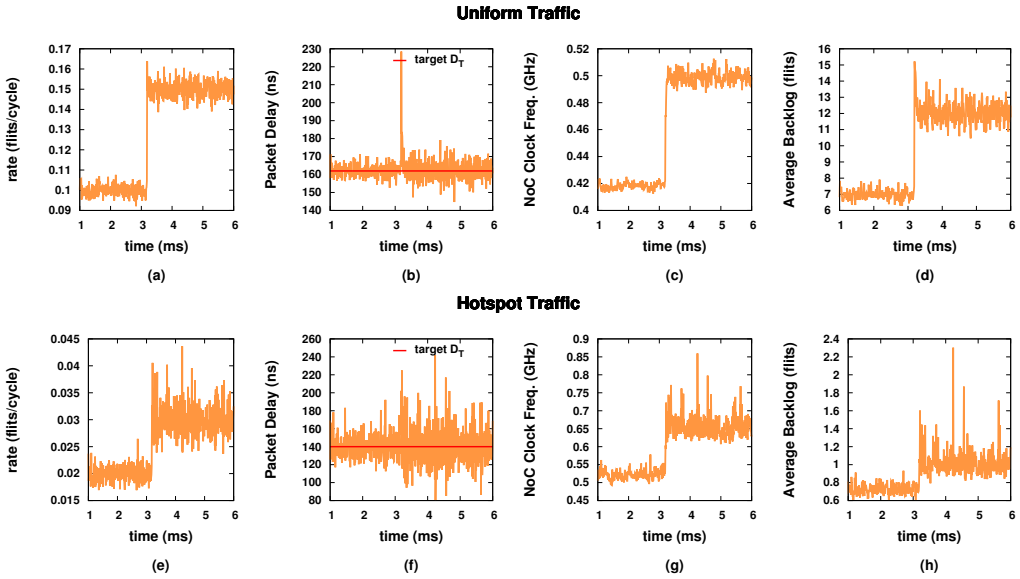

Figure 8: DMSD policy: rate step response in a 4×4 NoC.

reach saturation and we record the average delay at an injection equal to 90% of the saturation rate; finally we set this value as the target delay D_T. In this way, when the PI controller operates in *closed-loop* and the injection is less than 90% of the saturation rate, the controller will tune F_{noc} between $F_{\min}$ and $F_{\max}$ such that the average packet delay tracks D_T. When the injection is between 90% and 100% of the saturation rate, F_{noc} will be set to $F_{\max}$.

3.3.1. DMSD Performance

For the uniform and hotspot traffic, we have found the most suitable target packet delays are $D_T = 160$ ns and 140 ns, respectively. The delay target is different in the two scenarios, but not as much as the backlog target was in the QMSD case. Like for the QMSD policy, we report in Fig. 8 the DMSD response to a rate step. We observe that the frequency increases after

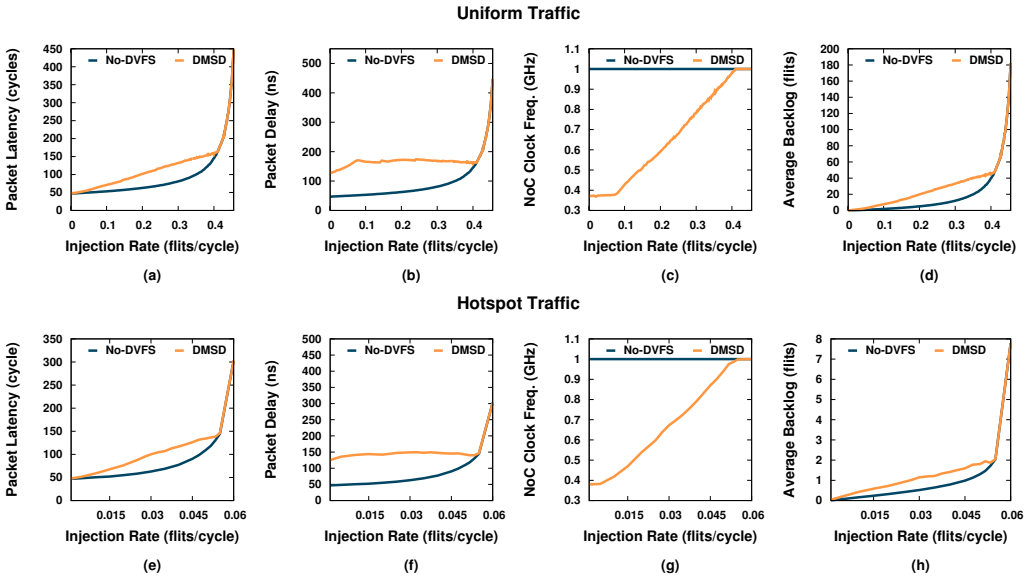

Figure 9: Performance figures without DVFS and with a DMSD policy in a 4×4 NoC.

the rate step such that the delay correctly tracks the target D_T, both for the uniform and the hotspot scenarios.

Fig. 9 shows the DMSD performance obtained in steady-state, which can be compared with Fig. 4 and Fig. 7 for the RMSD and QMSD policies, respectively. The PI controller effectively keeps the delay constant for a wide range of injection rates, as shown in Figs. 7(b),(f). As expected, when the injection reaches 90% of the saturation rate, F_{noc} reaches $F_{\max}$ and the DMSD delay becomes equal to the No-DVFS delay. To keep the delay constant, the average backlog changes and increases almost linearly with the increase of the injection rate. This behavior can be explained with Little's law[3].

[3]Little's law is a well-known property [25] holding under remarkably general network scenarios (individual queues, networks) and *very generic* arrival and service processes, given that the system reaches a stationary regime. The property states that the average total

By comparing the results in the uniform and hotspot cases, we observe that the target delay D_T has been set equal to about 160 ns and 140 ns, respectively, according to the previously described tuning procedure for D_T. This is a striking difference with the previous policies, in which target rate and target backlog depend significantly on the traffic scenario. A target almost independent on the traffic makes the DMSD policy more appealing from a practical viewpoint. We also do not observe any apparent difference in the efficiency in keeping the delay close to the target in the two scenarios.

The comparison between end-to-end delays obtained in the RMSD and QMSD cases, Figs. 7(b),(f), and those obtained in the DMSD one, Figs. 9(b),(f), clearly shows a large difference in terms of delay. In Sec. 5, we will show that this performance gap is typically not equally compensated in terms of power.

3.4. Theoretical ground for the three policies

It is possible to obtain results very similar to those obtained with NoC simulations under the three policies by using a simple M/D/1 queueing model, in which the service time depends on the frequency chosen by the DVFS control. Indeed, we can associate this M/D/1 model to the queuing occurring to the flits before being injected through the NoC.

Let us assume that λ is the number of arrived flits per node clock cycle. For simplicity, we assume that the delay experienced by a flit transmitted through the NoC depends only on the transmission time of the flit at the ingress of the NoC, measured in terms of node clock cycles, equal to F_{node}/F_{noc}.

---

backlog in the system equals the product of the average delay and the overall arrival rate into the system. Using the notation of the M/D/1 model adopted in Sec. 3.4: $B = \lambda D$.

Thus, we can formally define the inverse of the service time of the queue as $\mu = F_{noc}/F_{node}$, with $\mu \leq 1$. Notably, the service time in the M/D/1 model is fixed, whereas F_{noc} varies over time due to the control loop. Thus, we assume that μ is actually computed based on the average NoC frequency $E[F_{noc}]$ observed on a time window: $\mu = E[F_{noc}]/F_{node}$. This simplification leads to an approximated model, nevertheless quite accurate as shown below. By defining the classical utilization factor as

$$\rho = \lambda/\mu, \tag{7}$$

we can use the standard M/D/1 formula to compute the average delay D as

$$D = \frac{2-\rho}{2(1-\rho)} \cdot \frac{1}{\mu}. \tag{8}$$

By exploiting Little's law in (8), we can also obtain the average backlog B

$$B = \frac{2-\rho}{2(1-\rho)} \cdot \rho \tag{9}$$

We now observe that DMSD and QMSD policies set the delay and the backlog equal to a target, i.e. $D = D_T$ and $B = B_T$, respectively. As for RMSD, setting the NoC injection rate equal to $\lambda_{\max}$ is equivalent to setting the utilization factor equal to a given target, i.e. $\rho = \rho_T$, with $\rho_T \in (0, 1)$. Thus, by setting some target value in (7)-(9), we can analytically derive the required μ. In particular, by setting the utilization factor ρ_T, we obtain that the RMDS policy in (4) corresponds exactly to the following one:

$$\mu = \max\left\{\min\left\{\frac{\lambda}{\rho_T}, 1\right\}, \frac{F_{\min}}{F_{\max}}\right\}. \tag{10}$$

By setting instead $B = B_T$ in (9) we obtain the QMSD policy:

$$\mu = \max\left\{\min\left\{\frac{\lambda}{2B_T} \cdot \left(B_T + 1 + \sqrt{B_T^2 + 1}\right), 1\right\}, \frac{F_{\min}}{F_{\max}}\right\}. \tag{11}$$

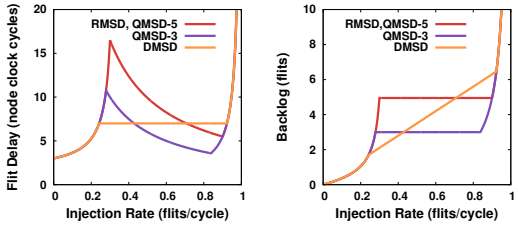

Figure 10: Average delay (left) and backlog (right) obtained through the M/D/1 model.

Finally, if we set a target delay D_T in (8) we obtain the DMSD policy:

$$\mu = \max\left\{ \min\left\{ \frac{\lambda D_T + 1 + \sqrt{\lambda^2 D_T^2 + 1}}{2D_T}, 1 \right\}, \frac{F_{\min}}{F_{\max}} \right\}. \tag{12}$$

In all the three different expressions, μ is chosen to keep the quantity in question equal to the target whenever μ is in the interval $[F_{\min}/F_{\max}, 1]$; outside such interval μ is clipped within the same interval.

The left graph in Fig. 10 shows the theoretical delay obtained by combining (8) with the specific policies expressed by (10)-(12), in the case $F_{\min} =$ 333 MHz, $F_{\max} = 1$ GHz and the target utilization in RMSD is $\rho_T = 0.9$. The right graph shows instead the average backlog. For the QMSD case, in particular, we show the results for two target backlog values: $B_T = 3$ flit (denoted as "QMSD-3") and $B_T = 5$ flit (denoted as "QMSD-5"). The DMSD curves are instead obtained by setting $D_T = 7$ timeslots.

The curves obtained with the M/D/1 model and reported in Fig. 10 lead to a number of observations.

First, by comparing Fig. 4(b) and Fig. 7(b) with the theoretical RMSD and QMSD delays in Fig. 10, we note that the simple M/D/1 model accurately captures the non-monotonic behavior. For the QMSD case, the fact that the delay decreases as the load increases has a rather intuitive explanation: thanks

to Little's law λD is constant, therefore if λ increases, D must decrease.

Second, we notice that the behavior of QMSD-5 is exactly the same as RMSD with $\rho_T = 0.9$. This is due to the *equivalence* between QMSD and RMSD policies, for an appropriate choice of the design parameters, i.e. the target utilization ρ_T in RMSD and the target backlog B_T in QMSD. Indeed, from (10)-(11) we obtain that the two policies are equivalent if $\rho_T = B_T + 1 - \sqrt{B_T^2 + 1}$. This property was already proven formally in [23] for the same M/D/1 model adopted here. This property remarkably holds also in the NoC scenario, as shown by comparing the behavior of RMSD in Figs. 4(d),(h) with the behavior of QMSD in Figs. 7(d),(h).

Finally, we observe that the DMSD average delay is kept mostly constant, whereas the backlog grows linearly with the load, as expected by Little's law. If we compare the M/D/1 delay and backlog curves with the delays in Figs. 9(b),(f) and the backlog in Figs. 9(d),(h), we notice that our M/D/1 model remarkably and accurately mimics the performance of the NoC.

4. Accurate Power Evaluation for DVFS

To evaluate the NoC power consumption, we first determined $F(V)$, i.e. the maximum clock frequency for a given voltage, which we hinted at in Sec. 3. We used the flow illustrated in the left part of Fig. 11.

We first synthesized with Synopsys Design Compiler (DC), an RTL version of the virtual-channel router used in simulations, targeting a low-power standard-cell library in a 28-nm FDSOI CMOS technology. After converting the gate-level netlist into a transistor-level netlist, we extracted the netlist portion related to the critical path, which we simulated with Mentor Graph-

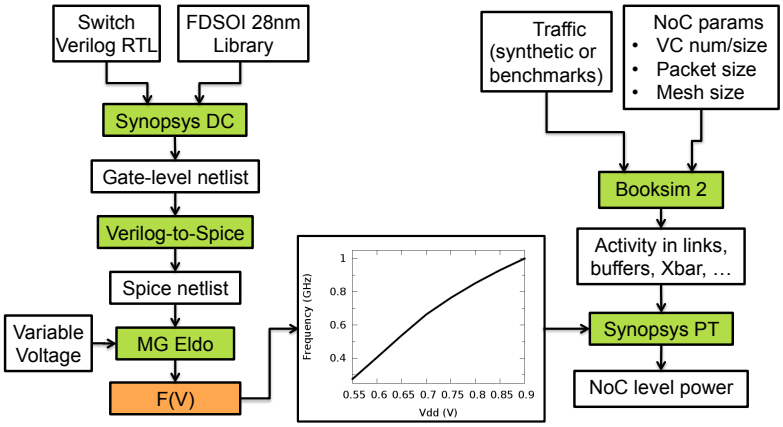

Figure 11: Methodology for the power evaluation.

ics' Eldo transistor-level simulator. For each given supply voltage V_{dd}, we simulated the critical path with proper input patterns aimed to obtain the maximum clock frequency that resulted in no timing errors. The result of this analysis is the $F(V)$ curve that we also report in Fig. 11. A reasonable range for the clock frequency goes from $F_{\min} = 333$ MHz to $F_{\max} = 1$ GHz, which corresponds to a voltage range from 0.56 V to 0.9 V.

The so-obtained $F(V)$ curve is used in our power estimation flow, as shown in the right part of Fig. 11. We fed Booksim cycle accurate simulator with all the required NoC parameters and with the traffic scenario. Thanks to the capabilities of Booksim, not only we obtained cycle-accurate performance measurements, but we could also save information of activity in the links, buffers, crossbar, etc. Therefore we imported this information in Synopsys Prime Time and obtained an accurate power estimation for any input rate, any router in the NoC, and any frequency-voltage pair.

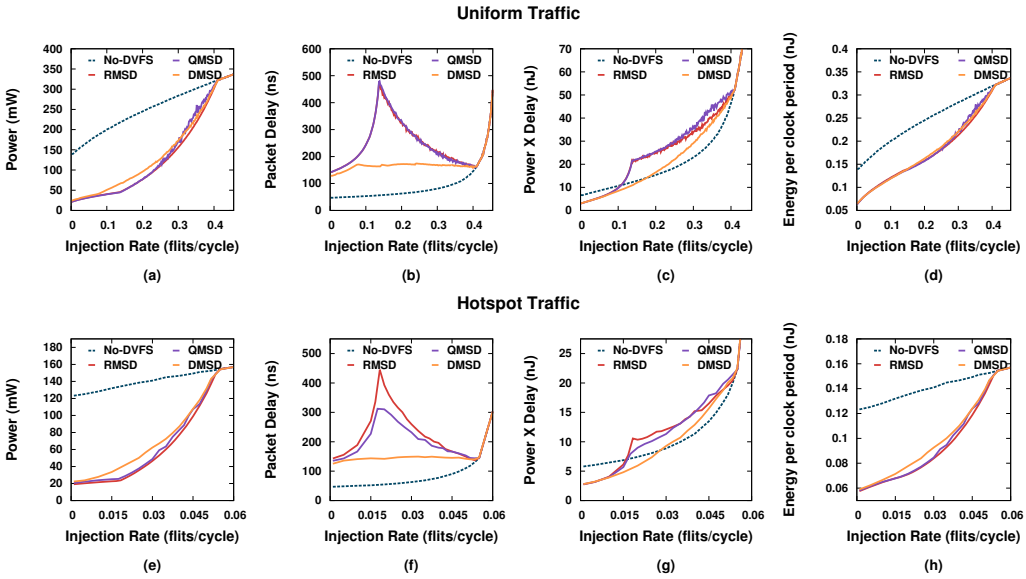

Figure 12: Power and performance metrics for the baseline case in Tab. 1.

5. Experimental Results

We report three kinds of experimental results, each in a separate subsection. We present power and performance results obtained under synthetic and realistic traffic in Sec. 5.1 and Sec. 5.2-5.3, respectively. The results have been obtained through the methodology for the power evaluation described in Sec. 4. We also evaluate a hardware implementation of the PM in Sec. 5.4.

5.1. Synthetic Traffic

For the baseline NoC parameters in Tab. 1, under the same uniform and hotspot synthetic traffic scenarios of Sec. 3, we report power consumption in Figs. 12(a),(e), packet delay in Figs. 12(b),(f), product of delay and power in Figs. 12(c),(g), and energy spent per network clock period in Figs. 12(d),(h).

We observe that all the three policies significantly reduce power compared

to the No-DVFS case. RMSD and QMSD show the best power saving, but at the same time they exhibit the worst delay increase. DMSD consumes more power, but such power increase is compensated by a larger decrease of delay. This is apparent in the curves reporting the power-delay product: under this metric, DMSD is the winner among the three policies[4]. The energy per clock period is similar, only slightly larger for DMSD in the hotspot scenario.

For the case of uniform traffic pattern, we performed a sensitivity analysis by varying the NoC parameters as shown in Tab. 1. The results in Fig. 13 confirm that the delay-power trade-off always tips in favor of DMSD.

5.2. NoC Communication and Multimedia Benchmarks

We selected nine NoC benchmarks from the fields of signal and image processing for communications and multimedia [26][27][28][29][30], whose main features are summarized in Tab. 2. This selection offers sufficiently diverse characteristics—number of nodes and row/columns arrangement in an NoC mesh, injection rates, distribution of the load across the NoC—to enable an in-depth exploration and validation of the three DVFS policies.

The maximum throughput per node is 8 GB/s at $F_{\max} = 1$ GHz and with a 64-bit flit width. By comparing the injection rates (per node) in Tab. 2 with the maximum throughput, we can classify the benchmarks in two groups: ERICSSON, AV, MPEG4, VOPD, and H264 have a *high injection rate*; EQUALIZER, PIP, MWD, and VCE have a *low injection rate*.

The maps in Fig. 14 illustrate with a colored shading the injection rates in

[4]Clearly, the results depend on the metric used. The power-delay product gives equal importance to power and delay.

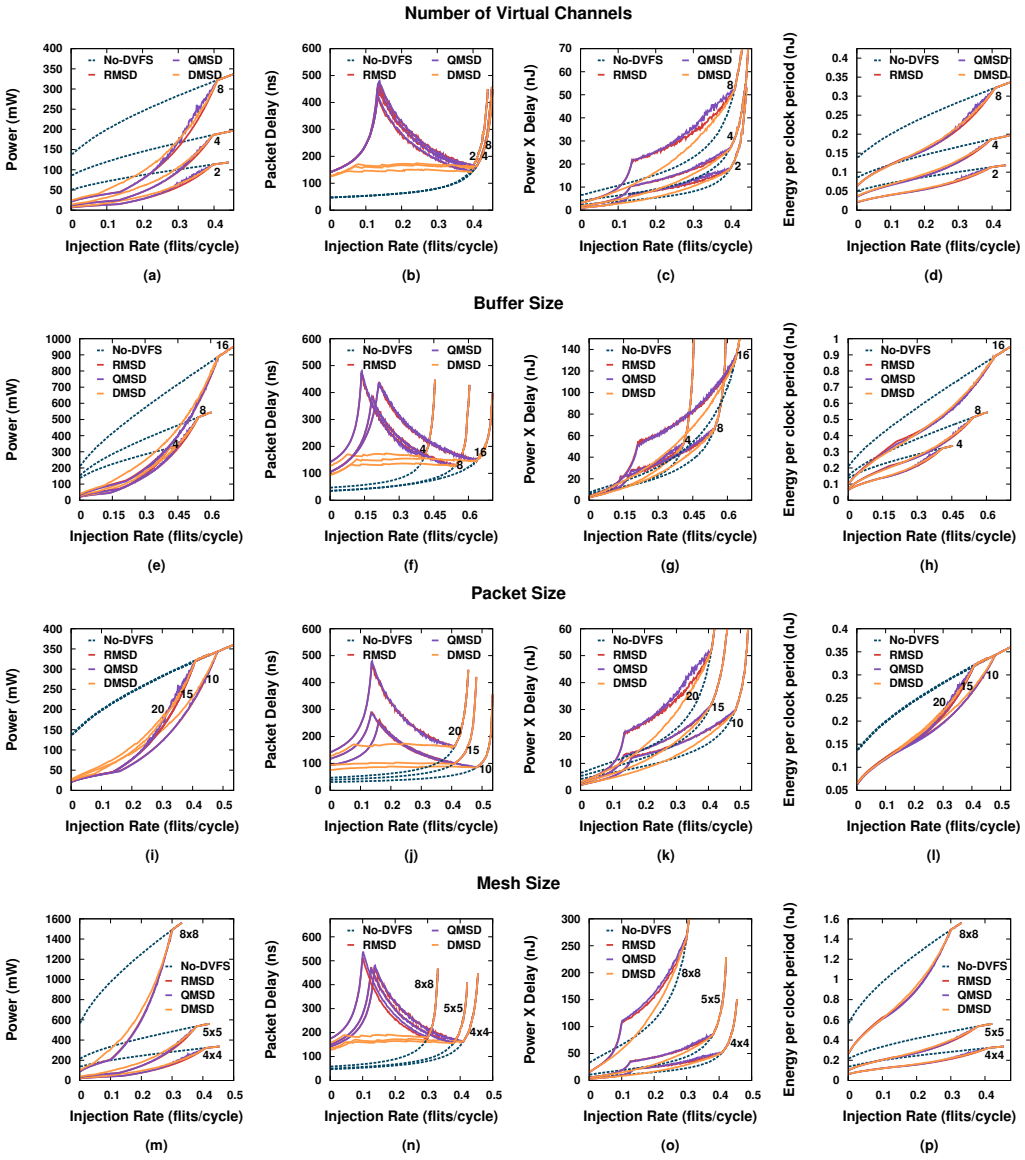

Figure 13: Sensitivity analysis under uniform traffic as a function of: number of virtual channels (a)-(d), buffer size in flits (e)-(h), packet size in flits (i)-(l), and mesh size (m)-(p).

the nine selected benchmarks, clearly showing that these can be partitioned in two groups according to the above criteria. We can also easily notice that

Table 2: Main features of nine selected NoC benchmarks.

Benchmark	Nodes	Type	Min/Max/Ave Rate (GB/s)	Min/Max/Ave Rate (flits/cycle)
ERICSSON [26]	16 (4x4)	high-rate	0.0/3.0/0.27	0.0/0.395/0.036
AV [28]	16 (4x4)	high-rate	0.0/1.83/0.38	0.0/0.24/0.05
MPEG4 [29]	12 (4x3)	high-rate	0.0/1.75/0.42	0.0/0.23/0.056
VOPD [29]	12 (4x3)	high-rate	0.0/0.58/0.213	0.0/0.076/0.028
H264 [30]	12 (4x3)	high-rate	0.0/0.55/0.278	0.0/0.072/0.037
EQUALIZER [27]	15 (5x3)	low-rate	0.0/3.4E-3/7.4E-4	0.0/4.3E-5/4.9E-6
PIP [29]	8 (4x2)	low-rate	0.0/0.19/0.035	0.0/0.025/4.6E-3
MWD [29]	12 (4x3)	low-rate	0.0/0.19/0.068	0.0/0.025/9.0E-3
VCE [30]	25 (5x5)	low-rate	0.0/0.088/0.022	0.0/0.012/2.9E-3

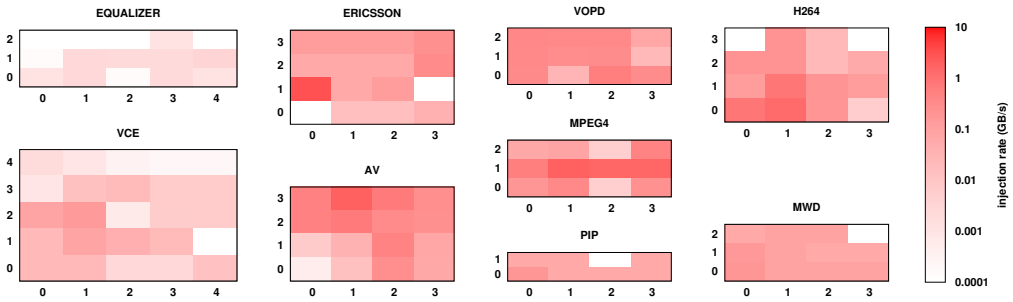

Figure 14: Injection rate in GByte/s for each node in the selected NoC benchmarks.

in various cases the injected traffic is far from uniform, and usually one node injects much more traffic, hence likely leading to hot-spot situations.

To evaluate performance and power savings under our DVFS policies, it has been necessary to first set proper references (i.e. $\lambda_{\max}$, B_T, and D_T). We propose a similar approach as the one devised for the synthetic benchmarks. We slow down F_{noc} by some factor $\phi > 1$ until we could bring the NoC into saturation. Then we determined the value of ϕ that corresponds to 90% of the saturation rate and recorded network injection rate, average backlog and

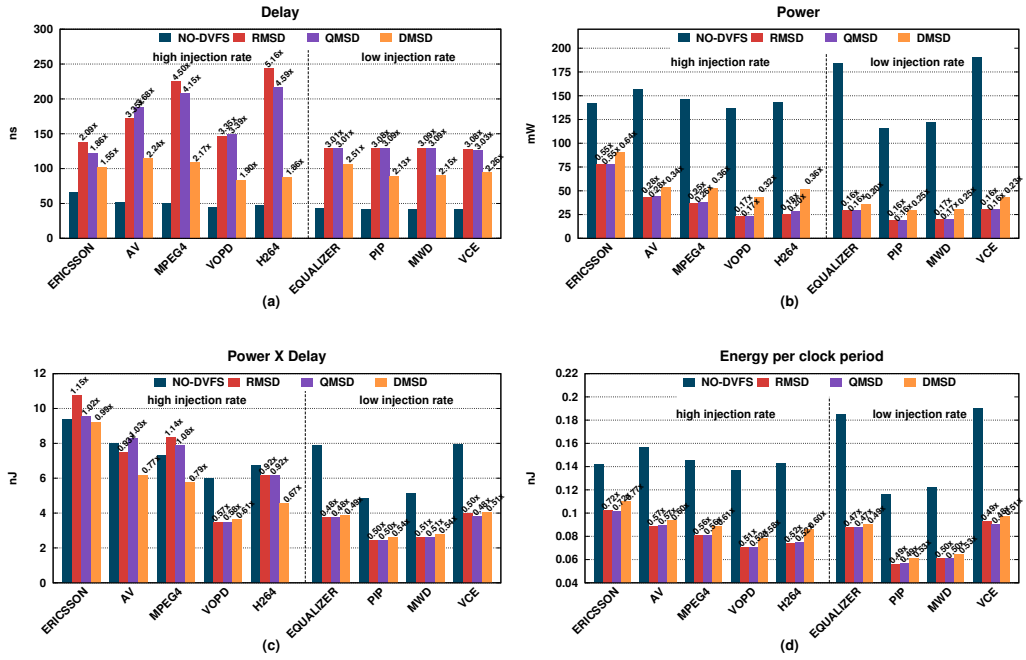

Figure 15: Power and performance metrics in the benchmark traffic scenarios.

average delay, which we set as references. Finally, we ran simulations with DVFS enabled and let the PM set autonomously F_{noc}, and we determined delay and power consumption with the methodology described in Sec. 4.

Results of delay, power, power-delay product, and energy per clock period for the nine benchmarks are reported in the histograms in Fig. 15. Notably, the actual injection rate is fixed and depends on the considered benchmark. The histogram in Fig. 15(a) shows that the packet delay degradation under QMSD and RMSD policies with respect to the No-DVFS case is very high, with both large average and deviation across the benchmarks. On the contrary, the delay under the DMSD policy increases much less on average and has a small deviation. Interestingly, the *absolute* delay under the DMSD policy (i.e. not in relative terms) remains more or less constant for all the benchmarks,

even if in theory the tracked reference value could be different for each of them. This means that, regardless of the benchmark, the delay at 90% of saturation remains more or less constant, and suggests that it could be possible to determine *a priori* the reference delay in DMSD.

Fig. 15(b) shows that all the policies effectively reduce power compared to the No-DVFS case. Coherently with what we observed under synthetic traffic, QMSD and RMSD outperform DMSD in terms of power reduction.

In terms of power-delay trade-off, we observe that in the group of *high-injection-rate* benchmarks the average power advantage of QMSD and RMSD over DMSD is compensated by a larger average delay degradation, which results in a low power-delay product, as shown in Fig. 15(c) (with the exception of VOPD benchmark, in which the power-delay product is similar in all policies). In the group of *low-injection-rate* benchmarks, instead, the power advantage of QMSD and RMSD over DMSD is larger or equal to the delay degradation. We conclude that the DMSD policy offers a better power-delay trade-off for high-rate applications, whereas for low-rate applications QMSD and RMSD appear to be preferable solutions.

As for RMSD and QMSD, we already noticed that they lead to very similar results. The choice between the two depends on practical implementation issues. Since QMSD and DMSD use a PI controller, the same PM can be reconfigured to use one or the other depending on specific requirements, e.g. privileging power reduction or power-delay trade-off.

5.3. NoC PARSEC Benchmarks

The network simulations of the previous sections allows to get important insights on the performance of the NoC based on different power control

Figure 16: Injection rate in GByte/s for each node in the PARSEC benchmarks.

algorithms. Nevertheless, the accuracy of the results obtained with system-level simulations cannot be achieved with such network simulations.

To improve the accuracy of our comparison, we adopted the Netrace methodology proposed by [31], since it was shown to provide an accuracy comparable with system-level simulations but with a simpler approach. Indeed, Netrace allows to inject the packets into the NoC based on realistic workload traces in a such a way that the temporal dependency among packets is preserved. We modified the specific development branch of Booksim2 that integrates the native support of Netrace (available in [32]), in order to support the DVFS model and our PM policies.

We ran simulations on 8 different PARSEC traces available on [33]. In Fig. 16 we show the maps of the injections rates for each NoC node. The traffic appears clearly unbalanced among the different nodes. The offered load in each node is very small, varying from 3.2 MB/s up to 268 MB/s, with an average equal to 58 MB/s, computed across all the traces and nodes.

The results of Fig. 17 are coherent with the ones obtained in Sec. 5.2,

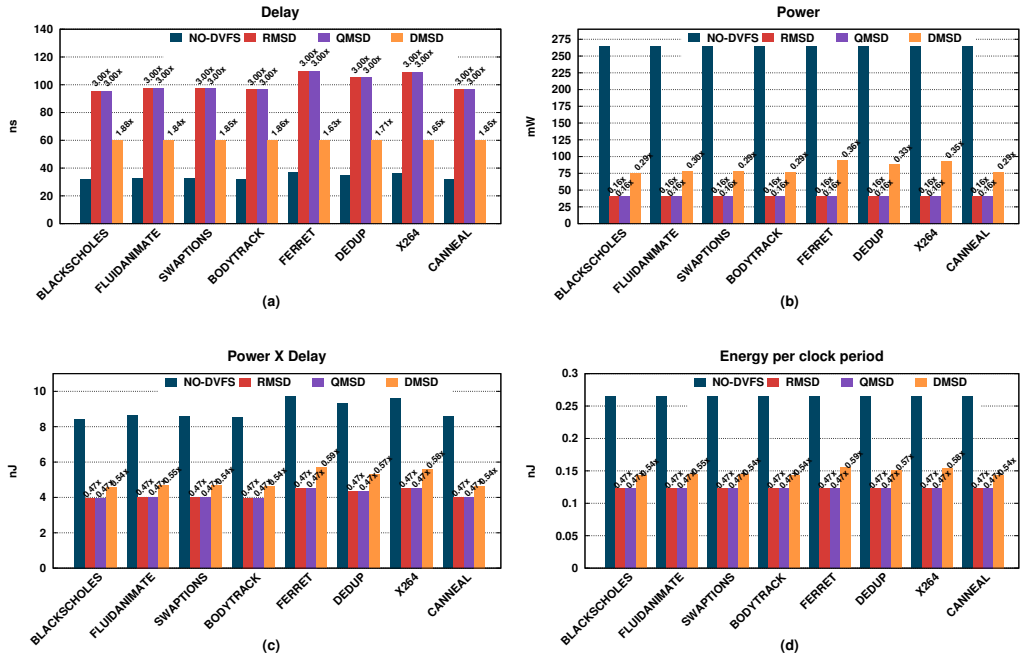

Figure 17: Power and performance metrics in the PARSEC benchmarks.

specifically with the benchmarks characterized by a low injection rate. Indeed, QMSD and RMSD behave identically, since the low injection rate permits reducing the voltage and the frequency to the minimum one for both policies. Interestingly, all the benchmarks show almost the same performance, despite their different traffic patterns, due to the experienced low injection rate.

By comparing the delays, DMSD is able to correctly stabilize the delay around the target $D_T = 60$ ns, which was set to be smaller than the one achieved by QMSD and RMSD. This delay reduction comes at a power cost, since DMSD is not as power-efficient as the other PM policies. In absolute terms, the power experienced for all the benchmarks is coherent with the actual reductions observed in Sec. 5.2 for the traces with low injection rates.

The power-delay product for DMSD is slightly worse than QMSD and

RSMD and this suggests that the best tradeoff is achieved for QMSD and RMSD in the scenarios when the injection rates are so small that the experienced delay is still acceptable.

All these results depend strongly on the specific system-level architecture considered in Netrace, according to which the original PARSEC traces were obtained. For different architectures, the injection rates could be higher, thus motivating a careful selection of the optimal PM policy.

5.4. Hardware Support for the DVFS Policies

We focus on the implementation of QMSD and DMSD, having noticed the equivalence between RMSD and QMSD.

Each NI in QMSD computes the average backlog according to (6). By choosing N in (6) as a power of 2, the multiplication by $N-1$ is replaced by a hard-wired left shift (by $\log_2 N$ positions) and a subtraction, and the division by N is a $\log_2 N$ hard-wired right-shift. Therefore, the additional NI complexity in QMSD (one subtracter and one adder: hard-wired shifts add negligible complexity) is negligible compared to the complexity of a regular NI dominated by the packet FIFOs. As for the PM, it computes first the final average backlog by summing all the partial quantities and dividing by the number of NIs. Then, it computes the U term through two multiplications by K_I and K_P, a subtraction to get the error E_n, and a final addition. A further division and addition are required by (5). As a result, the PM datapath requires an adder/subtracter, a multiplier, a divider, and some registers.

Each NI in DMSD has a counter for the timestamp, a subtracter to evaluate the delay, a counter of the received packets, and an accumulator for the accumulated delay. This additional complexity is negligible given the

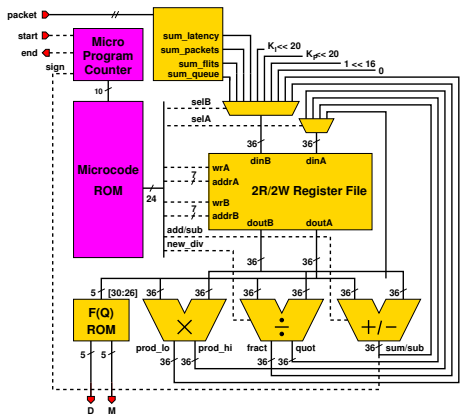

Figure 18: Microcoded DSP implementing the power manager.

complexity of a standard NI. As for the PM, it gets from each NI the number of received packets and the accumulated delay, which are used to evaluate the final average delay through addition and division. Since the PI control is the same of the QMSD, the PM datapath is similar.

In summary, QMSD and DMSD need hardware of marginal added complexity for the NIs, and one unique PM. We propose for it the microcoded DSP in Fig. 18. One datapath (yellow) serves both policies; what changes is the content of the microcode ROM (purple). After receiving the NIs local quantities, the DSP executes in around 130 clock cycles. Its output is used to look-up the *F(Q) ROM*, whose outputs M and D feed a PLL that synthesizes clock frequency $F_{noc} = M/D \cdot F_{\text{ref}}$, where F_{ref} is the PLL input frequency.

Regarding scalability, note that the hardware cost of this implementation is independent on the number of nodes of the NoC. In addition, we did not observe a relevant performance impact in our experiments of up to 64 nodes. For larger size, however, we believe that using different V/F domains with separate controllers is a more effective design solution.

We first validated the design of the NIs and the PM DSP on a Field-Programmable Gate Array (FPGA) emulation platform based on a Xilinx Virtex-4 FPGA. Here we could successfully deploy a 3×3 NoC with frequency-scaling only (voltage scaling is not possible in a standard FPGA emulation platform) thanks to the Xilinx Digital Clock Managers that permit on-the-fly clock frequency reconfiguration [34]. In a second step, we evaluated the silicon area of the PM in the same CMOS 28-nm FDSOI technology that we used for power and performance evaluation of the NoC. We obtained that the area of the PM is $0.038\,\mathrm{mm}^2$, which corresponds, for example, to 27% of the area of an NoC switch in the baseline configuration of Tab. 1, which is $0.14\,\mathrm{mm}^2$. The average power consumption of the PM is only $7\,\mathrm{mW}$ at $1\,\mathrm{GHz}$.

6. Conclusions

We investigated three different policies to support DVFS in NoCs: one based on a target utilization factor given the arrival data rate (RMSD), one based on a target queue size (QMSD) and one based on a target packet delay (DMSD). Throughout extensive simulations, based on both synthetic and realistic benchmark traffic patterns, we analyzed the variety of tradeoffs between performance and power consumption in a target CMOS 28-nm technology. We also provided a sound theoretical insight of the policies' behavior through a simple queuing model, which highlighted the theoretical equivalence between RMSD and QMSD. Finally, we described in details the hardware implementation of a power manager (PM) able to support QMSD and DMSD policies. We validated the functionality of our design with an implementation on an FGPA emulation platform, and showed that the

additional complexity of the PM once implemented in the 28-nm technology is small.

As main result we show that for scenarios in which the injection rates are high, DMSD appears to achieve an efficient tradeoff between performance, power consumption and implementation complexity. Instead, for low injection rates, RMSD and QMSD tend to outperform DMSD since the achieved delays are small in absolute terms.

From our results, it is clear that the optimal selection of the policy to support DVFS requires to analyze carefully not only the requirements in terms of performance and power of the NoC, but also the actual traffic injection rates.

Bibliography

[1] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, S. Borkar, An 80-tile sub-100-W teraFLOPS processor in 65-nm CMOS, IEEE Journal of Solid-State Circuits 43 (1) (2008) 29–41. doi:10.1109/JSSC.2007.910957.

[2] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, S. Borkar, A 5-GHz mesh interconnect for a teraflops processor, IEEE Micro 27 (5) (2007) 51–61. doi:10.1109/MM.2007.4378783.

[3] J. S. Kim, M. B. Taylor, J. Miller, D. Wentzlaff, Energy characterization of a tiled architecture processor with on-chip networks, in: Proceedings of the 2003 International Symposium on Low Power Electronics and Design, ISLPED '03, ACM, New York, NY, USA, 2003, pp. 424–427. doi:10.1145/871506.871610.

[4] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, N. Borkar, A 2 Tb/s 6x4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS, IEEE Journal of Solid-State Circuits 46 (4) (2011) 757–766.

[5] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, C. R. Das, A case for dynamic frequency tuning in on-chip networks, in: Proc. 42nd Int. Symp. Microarchitecture (MICRO-42), 2009, pp. 292–303.

[6] L. Guang, E. Nigussie, L. Koskinen, H. Tenhunen, Autonomous DVFS on supply islands for energy-constrained NoC communication, in:

Arch. Comput. Sys. ARCS 2009, Vol. 5455 of Lect. Notes Comput. Sc., Elsevier, 2009, pp. 183–194.

[7] M. K. Yadav, M. R. Casu, M. Zamboni, LAURA-NoC: Local automatic rate adjustment in network-on-chips with a simple DVFS, IEEE Transactions on Circuits and Systems II: Express Briefs 60 (10) (2013) 647–651.

[8] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, U. Ogras, In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches, ACM Trans. on Design Automation of Electronic Systems 18 (4) (2013) 1–21. doi:10.1145/2504905.

[9] Booksim2.
URL `https://github.com/booksim/booksim2`

[10] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, W. J. Dally, A detailed and flexible cycle-accurate network-on-chip simulator, in: Proc. Int. Symp. Performance Analysis Systems and Software (ISPASS), 2013, pp. 86–96.

[11] L. Shang, L.-S. Peh, N. K. Jha, Dynamic voltage scaling with links for power optimization of interconnection networks, in: Proc. 9th Int. Symp. High-Perf. Comput. Arch. (HPCA), 2003, pp. 123–124.

[12] J. Zhan, N. Stoimenov, J. Ouyang, L. Thiele, V. Narayanan, Y. Xie, Optimizing the NoC slack through voltage and frequency scaling in hard real-time embedded systems, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 33 (11) (2014) 1632–1643. doi:10.1109/TCAD.2014.2347921.

[13] R. Hesse, N. E. Jerger, Improving DVFS in NoCs with coherence prediction, in: Proceedings of the 9th International Symposium on Networks-on-Chip, NOCS '15, ACM, New York, NY, USA, 2015, pp. 24:1–24:8.

[14] X. Chen, Z. Xu, H. Kim, P. V. Gratz, J. Hu, M. Kishinevsky, U. Ogras, R. Ayoub, Dynamic voltage and frequency scaling for shared resources in multicore processor designs, in: Proc. 50th Design Automation Conference (DAC), ACM Press, 2013, pp. 114:1–114:7.

[15] J.-Y. Won, X. Chen, P. Gratz, J. Hu, V. Soteriou, Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management, in: High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on, 2014, pp. 308–319. doi:10.1109/HPCA.2014.6835941.

[16] A. Bianco, P. Giaccone, M. R. Casu, N. Li, Exploiting space diversity and dynamic voltage frequency scaling in multiplane network-on-chips, in: 2012 IEEE Global Communications Conference (GLOBECOM), IEEE, 2012, pp. 3080–3085.

[17] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, D. W. Clark, Formal control techniques for power-performance management, IEEE Micro 25 (5) (2005) 52–62.

[18] U. Y. Ogras, R. Marculescu, D. Marculescu, Variation-adaptive feedback control for networks-on-chip with multiple clock domains, in: Proc. 45th Design Automation Conference (DAC), ACM Press, 2008, pp. 614–619.

[19] A. Jantsch, L. Guang, Adaptive power management for the on-chip communication network, in: Proc. 9th EUROMICRO Conf. on Digital System Design (DSD'06), IEEE, 2006, pp. 649–656. doi:10.1109/DSD.2006.21.

[20] W. Kim, D. M. Brooks, G.-Y. Wei, A fully-integrated 3-level DC/DC converter for nanosecond-scale DVS with fast shunt regulation, in: 2011 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), IEEE, 2011, pp. 268–270.

[21] T. M. Andersen, F. Krismer, J. W. Kolar, T. Toifl, C. Menolfi, L. Kull, T. Morf, M. Kossel, M. Brandli, P. Buchmann, et al., 4.7 a sub-ns response on-chip switched-capacitor DC-DC voltage regulator delivering 3.7 w/mm2 at 90% efficiency using deep-trench capacitors in 32nm SOI CMOS, in: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), IEEE, 2014, pp. 90–91.

[22] M. R. Casu, P. Giaccone, Rate-based vs delay-based control for DVFS in NoC, in: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, EDA Consortium, 2015, pp. 1096–1101.

[23] A. Bianco, M. R. Casu, P. Giaccone, M. Ricca, Joint delay and power control in single-server queueing systems, in: Proc. IEEE Online Conf. on Green Communications (GreenCom), 2013, pp. 50–55. doi:10.1109/OnlineGreenCom.2013.6731028.

[24] Q. Wu, P. Juang, M. Martonosi, D. W. Clark, Formal online methods for voltage/frequency control in multiple clock domain microprocessors, in: Proceedings of the 11th International Conference on Ar-

chitectural Support for Programming Languages and Operating Systems, ASPLOS XI, ACM, New York, NY, USA, 2004, pp. 248–259. doi:10.1145/1024393.1024423.
URL http://doi.acm.org/10.1145/1024393.1024423

[25] J. D. C. Little, Little's law as viewed on its 50th anniversary, Operations Research 59 (3) (2011) 536–549. doi:10.1287/opre.1110.0940.

[26] Z. Lu, A. Jantsch, TDM virtual-circuit configuration for network-on-chip, IEEE Trans. VLSI Syst. 16 (8) (2008) 1021–1034. doi:10.1109/TVLSI.2008.2000673.

[27] A. Moonen, M. Bekooij, R. van den Berg, J. van Meerbergen, Evaluation of the throughput computed with a dataflow model - a case study, Tech. Rep. ESR-2007-01, ES Reports – Eindhoven Univ. of Technology, Dept. Electrical Engineering, Electronic Systems (Mar 2007).

[28] J. Hu, U. Y. Ogras, R. Marculescu, System-level buffer allocation for application-specific networks-on-chip router design, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 25 (12) (2006) 2919–2933.

[29] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, G. De Micheli, NoC synthesis flow for customized domain specific multiprocessor systems-on-chip, IEEE Trans. Parallel Distrib. Syst. 16 (2) (2005) 113–129.

[30] K. Latif, Design space exploration for MPSoC architectures, Ph.D. thesis, Univ. of Turku, Turku Center for Computer Science (TUCS), Finland

(Dec. 2013).
URL http://www.doria.fi/handle/10024/93883

[31] J. Hestness, B. Grot, S. W. Keckler, Netrace: Dependency-driven trace-based network-on-chip simulation, in: Proceedings of the Third International Workshop on Network on Chip Architectures, NoCArc '10, ACM, New York, NY, USA, 2010, pp. 31–36.

[32] Booksim2 ("classes" branch).
URL https://github.com/booksim/booksim2/tree/classes

[33] Netrace: Dependency-tracking trace-based network-on-chip simulation.
URL http://www.cs.utexas.edu/∼netrace/

[34] Xilinx, Virtex-4 FPGA Configuration User Guide UG071 (v1.11) (June 2009).