Resource-Efficient Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration

Firew Siyoum, Benny Akesson, Sander Stuijk, Kees Goossens, Henk Corporaal
Eindhoven University of Technology, Department of Electrical Engineering,
email:{f.m.siyoum, k.b.akesson, s.stuijk, k.g.w.goossens, h.corporaal}@tue.nl

***Abstract*—A present-day System-on-Chip (SoC) runs a wide range of applications with diverse real-time requirements. Resources, such as processors, interconnects and memories, are shared between these applications to reduce cost. Resource sharing causes temporal interference, which must be bounded by a suitable resource arbiter. System-level analysis techniques use the service guarantee of the arbiter to ensure that real-time requirements of these applications are satisfied. A service guarantee that underestimates the minimum service provided by an arbiter results in more allocation of resources than needed to satisfy latency and throughput requirements. For instance, a linear service guarantee cannot accurately capture bursty service provision by many priority-based schedulers, such as Credit-Controlled Static Priority (CCSP) and Priority-Budget Scheduling (PBS). As a result, the timing analysis of these arbiters becomes too pessimistic. This leads to unnecessary cost penalties since some SoC resources, such as SDRAM bandwidth, are scarce and expensive.**

This paper addresses this problem for the CCSP arbiter. The two main contributions are: (1) a piecewise linear service guarantee that accurately captures bursty service provisioning, and (2) an equivalent dataflow model of the new service guarantee, which is an essential component to integrate the arbiter with dataflow-based system-level design techniques that analyze the worst-case latency and throughput of real-time applications. The new service guarantee enables efficient resource utilization under CCSP arbitration. Experimental results of an H.263 video decoder application show that memory bandwidth savings from 26% up to 67% can be achieved by using the new service guarantee as compared to the existing linear service guarantee.

***Index Terms*—real-time; SoC; CCSP; service guarantee; over allocation; arbitration; dataflow; latency-rate server**

I. Introduction

Convergence of application domains in consumer electronics requires a system to run a wide range of applications with diverse real-time requirements. To satisfy the computational requirements of these applications at low power consumption, sophisticated Multi-Processor Systems-on-Chip (MPSoC) with heterogeneous processing elements are used [1]. To reduce cost, system resources, such as processors, interconnects and memories, are shared between multiple users, which we refer to as *requestors*. However, resource sharing causes temporal interference between requestors, which must be bounded by a resource arbiter to guarantee a certain amount of service to each requestor. The minimum service an arbiter guarantees a requestor in an interval is referred to as its *service guarantee*. This guarantee bounds the worst-case response times (WCRT) of requestors, which is used by system-level analysis techniques to determine if real-time requirements, such as latency and throughput, of applications are satisfied.

A service guarantee that underestimates the minimum service provided by an arbiter results in more allocation of resources than needed, which we refer to as *over-allocation*. For instance, priority-based schedulers, such as Credit-Controlled Static Priority (CCSP) [2] and Priority-Budget Scheduling (PBS) [3] allow low latency to *bursty requestors*, which are users whose request rate fluctuate significantly over time. A linear service guarantee based on the latency-rate ($\mathcal{LR}$) server model [4] cannot capture the bursty service provision by these arbiters. Therefore, it provides a pessimistic WCRT that leads to exaggerated latency and diminished throughput. As a consequence, a designer has to over-allocate resources to satisfy latency and throughput requirements. This may overshadow the benefits of these arbiters for SoC resource sharing. CCSP, in particular, has three key properties that make it suitable for scheduling access to shared MPSoC resources [2], [5]: (1) it bounds temporal interference and guarantees each requestor a minimum allocated service in a time interval, (2) it has an efficient allocation mechanism that enables high resource utilization, and (3) it has a small area hardware implementation that is also fast enough to keep up with the speed of SoC resources and make a scheduling decision every clock cycle. As shown in Figure 1, the current service guarantee of CCSP, referred to as a *latency-rate service guarantee*, is a linear lower bound on the provided service. However, like many priority-based arbiters, a requestor under CCSP can have bursty provided service intervals, as illustrated in the figure. As a result, the latency-rate service guarantee gives a pessimistic WCRT for CCSP arbitration. System-level timing analysis based on this service guarantee, consequently, results in over-allocation of SoC resources.

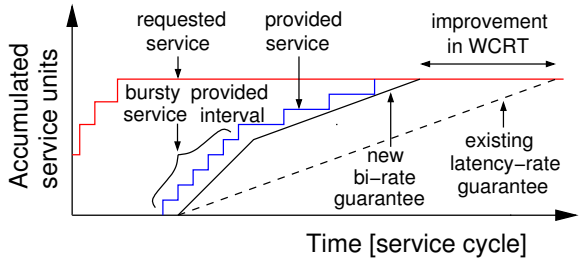


Fig. 1. New piecewise linear vs. existing linear service guarantee.

In this work, we show how to address the over-allocation problem through a piecewise linear service guarantee. Piecewise linear bounds have been previously used for traffic-shaping [6]–[8], as well as computing response time of arbiters [9]. However, these works focus on regulating unbounded *requested services* to make service guarantee analysis feasible. On the other hand, our model is a piecewise linear lower bound on the worst-case *provided service*. The model is an improved service guarantee that accurately characterizes the provided service. Most importantly, this work shows how a piecewise linear service guarantee can be represented with an equivalent dataflow model. This brings the service guarantee from individual access requests to the application level. Ultimately, this allows us to compute the worst-case latency

and throughput of applications that run on MPSoCs using dataflow-based system-level design techniques [10], [11].

This paper has two main contributions: (1) a piecewise linear service guarantee for CCSP, which we refer to as *bi-rate service guarantee*, as shown in Figure 1, and (2) an equivalent dataflow model of the bi-rate service guarantee. As a result of the new bi-rate model, a given SoC resource under CCSP arbitration can support more requestors, or a given set of requestors can be accommodated with less resource capacity. Experiments on traced traffic of an H.263 video decoder show that we can save from 27% up to 67% of memory bandwidth, compared to the existing latency-rate service guarantee.

The remaining part of this paper is organized as follows. In Section II, a review of related work is presented. Section III recaps the CCSP arbiter and presents the formal model used in this paper. The new bi-rate service guarantee and the associated dataflow model, which are the two major contributions of this paper, are presented in Sections IV and V, respectively. Section VI covers the experimental setup and obtained results. Finally, the paper concludes in Section VII, summarizing the highlights of this work.

II. Related work

Simple and starvation-free schedulers are natural candidates to arbitrate resource accesses in real-time MPSoCs. This is because they provide upper bounds on temporal interference and have small and fast hardware implementations. Several of such arbiters have been proposed, including TDMA [12], and extensions of Round-Robin arbitration, such as Weighted Round-Robin [13], and Deficit Round-Robin [14]. These arbiters offer requestors a fixed allocated budget (rate), which is replenished within a fixed frame size (period). These frame-based arbiters guarantee each requestor a minimum service proportional to its allocated rate. However, they fail to efficiently support diverse latency requirements without over-allocating resources. For instance, they cannot provide low latency to a requestor with low bandwidth requirement without allocating more slots in a frame. This problem is referred to as coupling between latency and rate [15]. Over-allocation in these types of arbiters may also result from a similar coupling between allocation granularity and latency. For instance, a single slot in a TDMA table with four entries corresponds to 25% of the bandwidth. Over-allocation occurs if the requestor requires any less. The over-allocation can be reduced by increasing the number of entries, although, this increases the WCRT of all requestors sharing the resource [2], [15]. Priority Budget Scheduling (PBS) [3] addresses the coupling between latency and rate using priorities. However, it only improves the latency of a single bursty requestor with high priority and still couples allocation granularity and latency. On the other hand, efforts to decouple the allocation granularity from latency have been made in [16]–[18], although the resulting arbiters couple latency and rate, and are unable to efficiently distinguish diverse latency requirements.

CCSP circumvents the shortcomings of existing resource arbiters, since (1) it efficiently supports diverse latency requirements, and (2) offers a small hardware implementation that runs sufficiently fast and keep up with resources like SDRAM memory controllers [19]. It has furthermore been shown in [2] that CCSP belongs to the class of $\mathcal{LR}$ servers, and hence offers a minimum guaranteed service to its requestors in an interval. This service guarantee is used at system-level to determine if latency and throughput requirements of applications are satisfied. However, the latency-rate service guarantee is a linear lower bound that cannot capture the bursty service provided by many priority-based schedulers. As a result, it yields a pessimistic WCRT for requestors under CCSP. This ultimately leads to unnecessarily excess resource allocations to attain the latency and throughput requirements.

This paper presents a piecewise linear service guarantee for CCSP that captures the bursty provided service by modeling two distinct modes in the arbiter, namely when a requestor has enough budget for a bursty service, and when it does not. Our analysis is based on *service curves*, commonly used in analysis of timing bounds and buffer requirements in the field of communication networks [20]–[23]. A common similarity between these works is their use of traffic-shaping to enforce a certain request arrival curve. Piecewise linear bounds have also been employed for traffic-shaping/policing in [6]–[8], as well as for computing response time of real-time arbiters by bounding resource access requests [9]. However, these traffic shapers are applied to the *requested service*, whereas our model is a piecewise lower-bound on the *provided service*.

Dataflow graphs allow analysis of the worst-case latency and throughput of real-time systems [24], [25]. State-of-the-art dataflow-based system-level analysis techniques [10], [11] require various aspects of the system, such as computation, buffers and arbitration, to be modeled with dataflow components. Modeling resource sharing with dataflow components is not trivial, as equivalence in temporal behavior with the service guarantees of the arbiters should be proved using rigorous algebraic steps [26], [27]. In [28], a general dataflow model for arbiters in the class of $\mathcal{LR}$ servers is presented that also covers CCSP. However, the basis of this dataflow model is the current linear $\mathcal{LR}$ service guarantee, and hence suffers from the associated over-allocation problem for priority-based arbiters. In [29], a three-actor dataflow model is presented for the PBS arbiter. However, this model only targets shared memory access and the latency improvement applies only to the single high-priority requestor. In this paper, we present a new dataflow model for CCSP based on our new piecewise linear service guarantee that applies to any MPSoC resource as well as to requestors with any priority level.

III. Background

The CCSP arbiter was originally proposed in [2]. A small and fast hardware implementation of the arbiter was presented in [5]. In this section, we iterate the basic operation of the CCSP arbiter and key definitions that are essential for a complete presentation of this work. In Section III-A, the formal model used in this paper is discussed. In Section III-B and III-C, the basic operation of the CCSP arbiter and the current service guarantee are explained using this formal model.

A. Formal Model

The analysis of the CCSP arbiter uses *service curves* [20] to model the interaction between the requestors and the resource. We use an abstract resource view where a *service unit* is the access granularity of the resource. Time is discrete and a time unit, referred to as a *service cycle*, is defined as the time required to serve such a service unit. We use closed discrete time intervals and hence $[\tau, t]$ includes all cycles in the sequence $\langle \tau, \tau + 1, ..., t - 1, t \rangle$. A service curve is a cumulative function of service units. For any service curve

ξ, $\xi(t)$ denotes its value at service cycle t. We furthermore use $\xi(\tau, t) = \xi(t+1) - \xi(\tau)$ to denote the difference in values between the endpoints of the closed interval $[\tau, t]$. The set of requestors is denoted as R. The k^{th} request from requestor $r \in R$ is ω_r^k and its size in service units is $s(\omega_r^k) \in \mathbb{N}^+$. Arriving requests from each requestor to the resource are placed in separate buffers in front of the resource. The arriving requests from a requestor are captured by the *requested service curve*, w, as defined in Definition 1.

Definition 1 (Requested service curve): The requested service curve of a requestor $r \in R$ is denoted $w_r(t) : \mathbb{N} \rightarrow \mathbb{N}$, where $w_r(0) = 0$ and

$$w_r(t+1) = \begin{cases} w_r(t) + s(\omega_r^k) & \omega_r^k \text{ arrived at } t+1 \\ w_r(t) & \text{no request arrived at } t+1 \end{cases}$$

The scheduled requestor at time t is denoted $\gamma(t) : \mathbb{N} \rightarrow R \cup \{\varnothing\}$ and receives service of one service unit. The service provided by the resource to a requestor is captured by the *provided service curve*, w', as given in Definition 2.

Definition 2 (Provided service curve): The provided service curve of a requestor $r \in R$ is denoted $w'_r(t) : \mathbb{N} \rightarrow \mathbb{N}$, where $w'_r(0) = 0$ and

$$w'_r(t+1) = \begin{cases} w'_r(t) + 1 & \gamma(t) = r \\ w'_r(t) & \gamma(t) \neq r \end{cases}$$

At any time t the difference between the requested service curve and the provided service curve gives us the number of service units waiting in the buffer for service. This is called the *backlog* of the requestor, defined in Definition 3. The concepts of requested service curve, provided service curve and backlog are illustrated in Figure 2.

Definition 3 (Backlog): The backlog of a requestor $r \in R$ at any time t is denoted $q_r(t) : \mathbb{N} \rightarrow \mathbb{N}$, and is defined as $q_r(t) = w_r(t) - w'_r(t)$.

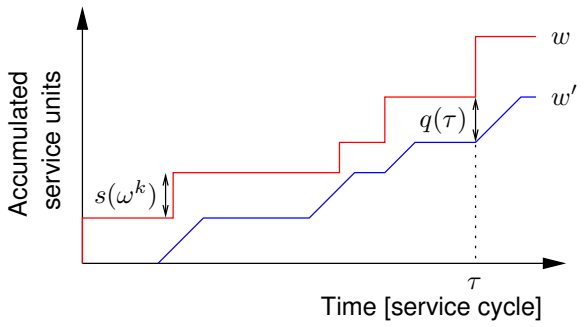


Fig. 2. Requested and provided service curves and related concepts.

B. CCSP Arbiter

A CCSP arbiter that arbitrates a set of requestors R sharing a common system resource consists of a *rate regulator* and a *scheduler* [2]. The scheduler uses a static-priority scheduling scheme. Each requestor is assigned a unique priority level and the set of requestors that have higher priority than a requestor $r \in R$ is denoted R_r^+. In addition, each requestor has an allocated service that consists of two parameters: the *allocated burstiness* (σ'_r) and the *allocated rate* (ρ'_r), as given in Definition 4. These two parameters, in combination, determine the fraction of resource capacity that the requestor is allocated.

Definition 4 (Allocated service): The service allocation of a requestor $r \in R$ is the pair $(\sigma'_r, \rho'_r) \in \mathbb{R}^+ \times \mathbb{R}^+$. For a valid allocation, it should hold that $\sum_{\forall r \in R} \rho'_r \leq 1$ and $\forall r \in R : \sigma'_r \geq 1$.

CCSP uses a continuous budget replenishment policy, which is based on the concept of *active periods*, which is defined in Definition 5. Intuitively, the active period of a requestor is the maximum interval of time where (1) it is backlogged and/or (2) it is *live*, as defined in Definition 6. A requestor is said to be live at time t if the cumulative of the requested service units is not less than the total service the requestor would get if it were continuously getting service at its allocated rate. A requestor in its active period interval is said to be active and R_t^a denotes the set of active requestors at time t.

Definition 5 (Active period): An active period of a requestor $r \in R$ is defined as the maximum time interval $[\tau_1, \tau_2]$, such that $\forall t \in [\tau_1, \tau_2] : q_r(t) > 0 \vee w_r(\tau_1 - 1, t-1) \geq \rho'_r \cdot (t - \tau_1 + 1)$.

Definition 6 (Live requestor): A requestor $r \in R$ is defined as live at a time t during an active period $[\tau_1, \tau_2]$ if $w_r(\tau_1 - 1, t-1) \geq \rho'_r \cdot (t - \tau_1 + 1)$.

Potential refers to the amount of budget a requestor has and it is defined in Definition 7. The allocated burstiness, σ', determines the initial potential of a requestor at the start of an active period, while the allocated rate, ρ', corresponds to the speed with which the budget is replenished during every service cycle in an active period. Together these two parameters determine the upper bound on the provided service of a requestor, $\hat{w}'$, as illustrated in Figure 3. A high allocated burstiness entitles a requestor to more service before exhausting its potential, forcing it to surrender the resource to lower priority requestors. For every service unit a requestor is provided, the potential is decremented by one. Figure 3 illustrates the relationships between allocated service (ρ', σ'), potential $(\pi(t))$ and active period ($[\tau_1, \tau_2]$ and $[\tau_3, \tau_4]$).

Definition 7 (Potential): The potential of a requestor $r \in R$ is denoted $\pi_r(t) : \mathbb{N} \rightarrow \mathbb{R}$, where $\pi_r(0) = \sigma'_r$ and

$$\pi_r(t+1) = \begin{cases} \pi_r(t) + \rho'_r - 1 & r \in R_t^a \wedge \gamma(t) = r \\ \pi_r(t) + \rho'_r & r \in R_t^a \wedge \gamma(t) \neq r \\ \sigma'_r & r \notin R_t^a \wedge \gamma(t) \neq r \end{cases}$$

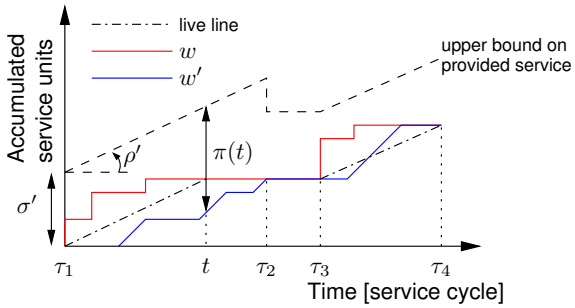


Fig. 3. Allocated service, potential and active period.

A requestor is said to be *eligible* for scheduling: (1) if it is backlogged i.e. $q_r(t) > 0$ and (2) if it has enough potential for at least one service unit i.e. $\pi_r(t) \geq 1 - \rho'_r$ (since the potential of an active requestor increments by ρ'_r every service cycle). The set of eligible requestors at time t is denoted R^e_t. The scheduler of CCSP is non-work-conserving and schedules the highest priority eligible requestor every service cycle.

C. Latency-rate Service Guarantee

Previous work [2] on the service guarantee of the CCSP arbiter shows that a requestor is guaranteed service according to its allocated rate, ρ', after a maximum latency, Θ. This linear service guarantee defines a lower bound, $\check{w}'$, on the provided service curve during an active period (cf. Figure 4). This lower bound, which we refer to as the *latency-rate service guarantee*, shows that CCSP belongs to the class of $\mathcal{LR}$ servers [4] and it is used to derive the WCRT of requests. Based on this latency-rate service guarantee, an active requestor $r \in R$ is guaranteed a minimum service during an active period $[\tau_1, \tau_2]$ according to the following relation: $\forall t \in [\tau_1, \tau_2] : \ \check{w}'_r(\tau_1, t) = \max(0, \rho'_r \cdot (t - \tau_1 + 1 - \Theta_r))$, where

$$\Theta_r = \frac{\sum\limits_{\forall s \in R^+_r} \sigma'_s}{1 - \sum\limits_{\forall s \in R^+_r} \rho'_s} \tag{1}$$

IV. Arbiter Analysis for Bi-rate Service Guarantee

In this section, we present a piecewise linear service guarantee for the CCSP arbiter. The service guarantee of an arbiter is the minimal service it can provide to a requestor, irrespective of the situation of other requestors that are sharing the resource. This minimal service is normally computed by considering the worst-case scenario that leads to the WCRT. For a requestor under the CCSP arbiter, this worst-case scenario happens when it experiences the maximum interference from higher priority requestors. This is stated in Lemma 1 and proven in [2].

Lemma 1 (Maximum interference): The maximum interference experienced by a requestor $r \in R$ during an interval $[\tau_1, \tau_2]$ occurs when all higher priority requestors start an active period at τ_1 and remain active $\forall t \in [\tau_1, \tau_2]$, and equals

$$\hat{i}_r(\tau_1, \tau_2) = \sum_{\forall s \in R^+_r} \sigma'_s + \rho'_s \cdot (\tau_2 - \tau_1 + 1) \tag{2}$$

As discussed in Section III-C, the existing service guarantee of CCSP guarantees a requestor $r \in R$ its allocated rate, ρ'_r, after a maximum latency, Θ_r. However, in CCSP a requestor can be temporarily served at a rate higher than its allocated rate after its maximum latency. Intuitively, this can be explained as follows. At the end of the maximum latency, a requestor can have an accumulated potential from two sources, according to Definition 7: (1) from its allocated burstiness i.e. if $\sigma'_r > 1$ and (2) from potential accumulated while being blocked by higher priority requestors, i.e. during the maximum latency, Θ_r. As a result, at the end of the maximum latency, a requestor has a potential that is equal to the sum of these two.

When a requestor is at the end of its maximum latency, it implies that higher priority requestors have utilized their accumulated potential. Thus, they have to accumulate potential at their respective allocated rate during multiple service cycles, before they are eligible to access the resource again. Consequently, the requestor can use the resource whenever it is not used by higher priority requestors. This means, the requestor can get service at a higher rate $\rho^*_r \geq \rho'_r$, where

$$\rho^*_r = 1 - \sum_{\forall s \in R^+_r} \rho'_s. \tag{3}$$

The requestor receives service at this higher rate as long as its potential does not go below $1 - \rho'_r$, which is the minimum potential a requestor needs to have to be eligible for scheduling. The service cycle at which the potential drops below $1 - \rho'_r$ is referred to as the *boundary cycle*. After the boundary cycle the requestor has to wait multiple service cycles and accumulate potential at its allocated rate to be eligible. Therefore, it receives service at its allocated rate, ρ'_r. If the potential does not drop below $1 - \rho'_r$ throughout the active period, the boundary cycle is considered to be the last service cycle of the active period. In this case, the requestor receives all its service units in the active period with the higher service rate, ρ^*_r.

Definition 8 (Boundary Cycle): The boundary cycle of a requestor $r \in R$ is defined as the maximum service cycle τ^b_r during an active period $[\tau_1, \tau_2]$ of the worst-case scenario, such that

$$\forall t \in [\tau_1, \tau^b_r] : \pi_r(t) > 1 - \rho'_r. \tag{4}$$

The boundary cycle can be determined based on Definition 8, as shown in Lemma 2. Note that in this paper, we only consider requestors for which $\rho^*_r > \rho'_r$, since otherwise the higher service rate is equal to the allocated rate. The case where $\rho^*_r = \rho'_r$ can occur only to the lowest priority requestor in a fully loaded resource i.e. $\sum_{\forall r \in R} \rho'_r = 1$.

Lemma 2: For a requestor $r \in R$ with active period $[\tau_1, \tau_2]$ of the worst-case scenario, the boundary cycle $\tau^b_r = min(\tau_2, \lfloor t_x \rfloor)$ where

$$t_x = \tau_1 + \frac{\sigma'_r - 1 + \rho'_r + \sum\limits_{\forall s \in R^+_r} \sigma'_s}{\rho^*_r - \rho'_r} \tag{5}$$

Proof: The potential of a requestor r at time $t \in [\tau_1, \tau_2]$ is given in Equation (6). The term $\sigma'_r + \rho'_r \cdot (t - \tau_1)$ is the potential the requestor gets and $w'_r(\tau_1, t-1)$ is the potential it spends for the service units it receives in the interval $[\tau_1, t-1]$, according to Definition 7.

$$\pi_r(t) = \sigma'_r + \rho'_r \cdot (t - \tau_1) - w'_r(\tau_1, t-1). \tag{6}$$

The service provided to requestor r during the worst-case scenario is given by Equation (7), where $\hat{i}_r$ is the maximum interference, given by Lemma 1. It is computed by subtracting the maximum interference from higher priority requestors from the total service units available in the interval.

$$w'_r(\tau_1, t-1) = t - \tau_1 - \hat{i}_r(\tau_1, t-1) \tag{7}$$

Putting Equation (7) into Equation (6), gives us the complete relation for $\pi_r(t)$. Solving for t_x from the relation $\pi_r(t_x) = 1 - \rho'_r$ results in Equation (5). Then the boundary cycle τ_r^b is the largest integer less than t_x, since time is discrete, and does not exceed τ_2, which is the end of the active period. This gives us the relation $\tau_r^b = min(\tau_2, \lfloor t_x \rfloor)$. □

Since the higher service rate of CCSP is not captured in the latency-rate service guarantee, the timing analysis results in resource over-allocation to satisfy real-time requirements of requestors. To solve this problem, we propose a piecewise linear service guarantee, which we refer to as *bi-rate service guarantee*. The bi-rate service guarantee takes the higher service rate into account and improves the WCRT of requests, as shown in Figure 4. Δ in the figure illustrates the improvement by using a bi-rate service guarantee.

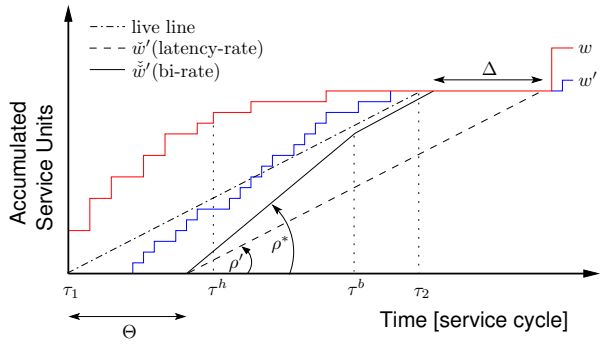


Fig. 4. Improving WCRT by piecewise linear model.

Under the bi-rate service guarantee, a requestor r is guaranteed a minimum service at two different rates, ρ_r^* and ρ'_r, after a maximum latency Θ_r. These two rates correspond to the cases when a requestor has enough potential to access the resource at a high rate, and when it does not. The maximum latency, Θ_r, is the same as the one in the latency-rate service guarantee, which is given in Equation (1). The bi-rate guarantee is defined based on two linear equations: higher-rate guarantee $\check{w}'^h_r$ and allocated-rate guarantee $\check{w}'^a_r$, which are given in Equations (8) and (9), respectively.

$$\check{w}'^h_r(\tau_1, t) = \rho_r^* \cdot (t - \tau_1 + 1 - \Theta_r) \tag{8}$$

$$\check{w}'^a_r(\tau_1, t) = \rho'_r \cdot (t - \tau_1 + 1 - \Gamma_r) \tag{9}$$

where

$$\Gamma_r = -\frac{\sigma'_r + \rho_r^* - 1}{\rho'_r}$$

The new bi-rate service guarantee is, then, the minimum of the two at any time. Hence, during an active period $[\tau_1, \tau_2]$, a requestor $r \in R$ is guaranteed a minimum service, $\check{\check{w}}'_r(\tau_1, t)$, as given in Equation (10).

$$\check{\check{w}}'_r(\tau_1, t) = max(0, min(\check{w}'^h_r(\tau_1, t), \check{w}'_{r_a}(\tau_1, t))) \tag{10}$$

Γ_r is computed such that the intersection of the two linear equations, $\check{w}'^h_r(\tau_1, t)$ and $\check{w}'^a_r(\tau_1, t)$, is at t_x, given in Equation (5). Hence, solving the two linear equations simultaneously shows that their intersection point is, indeed, at t_x. If τ_2, which is the end of the active period, is greater than this intersection point t_x, then the boundary cycle $\tau_r^b = \lfloor t_x \rfloor$ according to Lemma 2. Therefore, as illustrated in Figure 5, for $t \leq \tau_r^b$, $\check{w}'^h_r(\tau_1, t)$ is the minimum of the two equations and the service guarantee is determined solely by the higher rate linear equation, i.e. $\check{\check{w}}'_r(\tau_1, t) = \check{w}'^h_r(\tau_1, t)$. In contrast, for $t > \tau_r^b$, $\check{w}'^a_r(\tau_1, t)$ is the minimum and the service guarantee is determined by the allocated rate, i.e. $\check{\check{w}}'_r(\tau_1, t) = \check{w}'^a_r(\tau_1, t)$. If τ_2, i.e. the end of the active period, is less than t_x, then the requestor gets service at its higher rate throughout the active period.

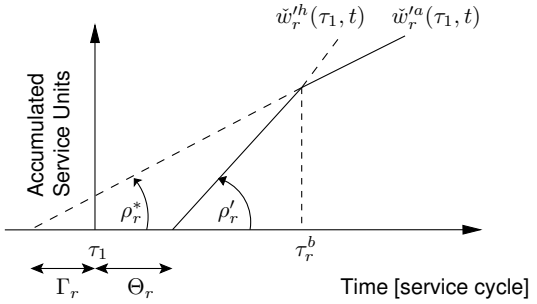


Fig. 5. Higher-rate and allocated-rate guarantee.

To guarantee a provided service at the higher rate, ρ^*, the requestor has to ask for service at this rate. Thus, the new service guarantee applies only to requestors that have a minimum service request rate of ρ^* at least up to time point τ^h (cf. Figure 4), given in Equation (11). We refer to such requestors as *high-rate requestors*, defined in Definition 9.

$$\tau_r^h = \tau_r^b - \Theta_r. \tag{11}$$

Definition 9: (High-rate requestor) A requestor $r \in R$ is a high-rate requestor during an active period $[\tau_1, \tau_2]$ if $\forall t \in [\tau_1, \tau_2]$:

$$w_r(\tau_1, t) \geq \begin{cases} \rho_r^* \cdot (t - \tau_1 + 1) & t \leq \tau_r^h \\ w_r(\tau_1, \tau_r^h) + \rho'_r \cdot (t - \tau_r^h) & \text{Otherwise} \end{cases}$$

Next, we present the mathematical proof of this new service guarantee, but first two important lemmas are provided to simplify this proof. Equation (6) shows an interesting relationship between the provided service and potential. By rearranging, we arrive at Lemma 3.

Lemma 3: For a requestor $r \in R$ during an active period $[\tau_1, \tau_2]$, it holds that $\forall t \in [\tau_1, \tau_2]$

$$\pi_r(t) \leq \sigma'_r - \rho'_r \iff w'_r(\tau_1, t-1) \geq \rho'_r \cdot (t - \tau_1 + 1) \tag{12}$$

From Lemma 3, it has also been proved in [2] that the eligibility of a requestor can be determined by looking at its potential, as given in Lemma 4.

Lemma 4: For a requestor $r \in R$ during an active period $[\tau_1, \tau_2]$, it holds that $\forall t \in [\tau_1, \tau_2]$

$$\pi_r(t) > \sigma'_r - \rho'_r \implies r \in R_t^e. \tag{13}$$

Theorem 1 defines the bi-rate service guarantee, making use of the concepts and results presented so far.

Theorem 1: (Bi-rate service guarantee) During any active period $[\tau_1, \tau_2]$ and $\forall t \in [\tau_1, \tau_2]$, a high-rate requestor $r \in R$ is guaranteed a minimum service according to Equation (10).

Proof: The strategy of the proof is showing that the service guarantee holds for four different types of time intervals $[\tau_i, \tau_j]$ that cover all possible scenarios during the active period.

Case 1: $\forall t \in [\tau_i, \tau_j] \wedge \tau_j \leq \tau_r^b : q_r(t) > 0$. This case covers time intervals before the boundary cycle where the requestor is backlogged. For this case, we want to show that a minimum service is guaranteed at the higher rate of the requestor; i.e. $\check{\check{w}}'_r(\tau_1, t) = \check{w}'^h_r(\tau_1, t)$. Since we are before the boundary cycle, we know that $\forall t \in [\tau_i, \tau_j], \pi_r(t) > 1 - \rho'_r$, according to Definition 8. This means the requestor is eligible, since the definition of this case states that it is also backlogged. The resource provides $(\tau_j - \tau_i + 1)$ service units in the interval. Taking into account the maximum interference possible from higher priority requestors (cf. Lemma 1), the minimum service available to r is given as:

$$\check{\check{w}}'_r(\tau_i, \tau_j) = \tau_j - \tau_i + 1 - \hat{i}_r(\tau_i, \tau_j). \tag{14}$$

Expanding and rearranging the right side of Equation (14) gives $\check{\check{w}}'_r(\tau_i, \tau_j) = \rho_r^* \cdot (\tau_j - \tau_i + 1 - \Theta_r)$, proving a minimum service at the higher rate for all timing intervals of this type.

Case 2: $\forall t \in [\tau_i, \tau_j] \wedge \tau_j \leq \tau_r^b : q_r(t) = 0$. This case covers time intervals before the boundary cycle where the requestor is not backlogged. For time intervals of this type, we want to prove that a minimum service can be guaranteed at the higher rate. By Definition 3, the backlog of a requestor is the difference between the requested service and the provided service. If there is no backlog, this means the requested and provided services are equal i.e. $w_r(t) = w'_r(t)$. Since r is a high-rate requestor, it holds that the requested service $w_r(\tau_1, t) \geq \rho_r^* \cdot (t - \tau_1 + 1)$ for the interval $[\tau_1, \tau_r^h]$. Therefore, for this interval, $w'_r(\tau_1, t) \geq \rho_r^* \cdot (t - \tau_1 + 1) \geq \check{w}'^h_r(\tau_1, t)$. Since $w_r(t)$ is a non-decreasing function, it also holds for the interval $[\tau_r^h, \tau_r^b]$ that $w'_r(\tau_1, t) \geq \check{w}'^h_r(\tau_1, t)$. Therefore, $\check{w}'^h_r(\tau_1, t)$ is the guaranteed service.

Case 3: $\forall t \in [\tau_i, \tau_j] \wedge \tau_i > \tau_r^b : \pi_r(t) > \sigma'_r - \rho'_r$. This case considers intervals after the boundary cycle where the requestor has more potential than its allocated burstiness. After the boundary cycle, the requestor is no more in the higher rate service interval of the active period. Therefore, in this interval we want to prove that a minimum service can be guaranteed at its allocated rate, i.e. $\check{\check{w}}'_r(\tau_1, t) = \check{w}'^a_r(\tau_1, t)$. Since $\pi_r(t) > \sigma'_r - \rho'_r$, the requestor is eligible according to Lemma 4. Similar arguments as in Case 1 of this proof regarding the available service units and the maximum interference give the guaranteed service as $\rho_r^* \cdot (\tau_j - \tau_i + 1 - \Theta_r)$, which equals $\check{w}'^h_r(\tau_i, \tau_j)$. For $t > \tau_r^b$, $\check{w}'^a_r(\tau_i, \tau_j)$ is less than $\check{w}'^h_r(\tau_i, \tau_j)$. Therefore, $\check{w}'^a_r(\tau_i, \tau_j)$ can be taken as the guaranteed service.

Case 4: $\forall t \in [\tau_i, \tau_j] \wedge \tau_i > \tau_r^b : \pi_r(t) \leq \sigma'_r - \rho'_r$. This case considers time intervals after the boundary cycle where the potential of the requestor is less than its allocated burstiness. The strategy of the proof is first to show that $w'_r(\tau_1, t-1) \geq \check{w}'^a_r(\tau_1, t)$. Then, based on Definition 2 of provided service curve, it follows that $w'_r(\tau_1, t) \geq w'_r(\tau_1, t-1) \geq \check{w}'^a_r(\tau_1, t)$.

1) The provided service $w'_r(\tau_1, t-1)$ can be expanded as shown in Equation (15), since for any service curve ξ, $\xi(\tau, t) = \xi(t+1) - \xi(\tau)$.

$$w'_r(\tau_1, t-1) = w'_r(\tau_1, \tau_r^b - 1) + w'_r(\tau_r^b, \tau_r^b) + w'_r(\tau_r^b + 1, t-1). \tag{15}$$

We proceed by bounding the three terms in Equation (15) individually.

2) At τ_r^b, requestor r is scheduled, because its potential drops, which happens only when a requestor gets service, according to Definition 7. This implies that $w'_r(\tau_r^b + 1) = w'_r(\tau_r^b) + 1$. Using this, $w'_r(\tau_r^b, \tau_r^b)$ can be formulated as follows.

$$w'_r(\tau_r^b, \tau_r^b) = w'_r(\tau_r^b + 1) - w'_r(\tau_r^b) = 1$$

3) From Case 1 and 2 of this proof, we already proved that

$$w'_r(\tau_1, \tau_r^b - 1) \geq \rho_r^* \cdot (\tau_r^b - \tau_1 - \Theta_r)$$

In addition, computing the intersection of the two linear equations $\rho_r^* \cdot (\tau_r^b - \tau_1 - \Theta_r)$ and $\rho'_r \cdot (\tau_r^b - \tau_1) + \sigma'_r - 1$ shows that for $\forall t > \tau_r^b$ it holds that

$$\begin{aligned} w'_r(\tau_1, \tau_r^b - 1) &\geq \rho_r^* \cdot (\tau_r^b - \tau_1 - \Theta_r) \\ &\geq \rho'_r \cdot (\tau_r^b - \tau_1) + \sigma'_r - 1. \end{aligned}$$

4) Since $\pi_r(t) \leq \sigma'_r - \rho'_r$ we have from Lemma 3 that

$$w'_r(\tau_r^b + 1, t-1) \geq \rho'_r \cdot (t - \tau_r^b)$$

5) Now putting, the results from steps 2, 3 and 4 into Equation (15), we get:

$$\begin{aligned} w'_r(\tau_1, t-1) &\geq 1 + \rho'_r \cdot (\tau_r^b - \tau_1) + \\ &\quad \sigma'_r - 1 + \rho'_r \cdot (t - \tau_r^b) \\ &= 1 + \rho'_r \cdot (t - \tau_1) + \sigma'_r - 1 \\ &\geq \rho'_r + \rho_r^* + \rho'_r \cdot (t - \tau_1) + \\ &\quad \sigma'_r - 1 \text{ (Since } 1 > \rho'_r + \rho_r^*) \\ &= \rho'_r \cdot (t - \tau_1 + 1) + \rho_r^* + \sigma'_r - 1 \\ &= \rho'_r \cdot (t - \tau_1 + 1 - \Gamma_r) \end{aligned}$$

Hence, we can guarantee a minimal service $\check{\check{w}}'_r(\tau_1, t) = \rho'_r \cdot (t - \tau_1 + 1 - \Gamma_r) = \check{w}'^a_r(\tau_1, t)$.

□

In summary, the bursty service a requestor can receive depends mainly on its allocated burstiness and the allocated burstiness of its higher priority requestors. Increasing its allocated burstiness increases its potential at the start of the active period. Increasing the allocated burstiness of higher priority requestors enables the requestor to accumulate more potential at its allocated rate while being blocked by them. High accumulated potential means a requestor can receive service at a high rate, as it does not have to wait to accumulate potential to be eligible. The bi-rate service guarantee integrates this fact through the boundary cycle parameter, τ^b, and guarantees each requestor high-rate service until its boundary cycle for every active period. This results in an improvement in the WCRT of requests as compared to latency-rate service guarantee that does not consider this bursty provided service. The reduction in the WCRTs of individual requests lead to improvements in latency and throughput of applications. Since designers are concerned in satisfying these real-time requirements, a system-level representation of the service gurantee is needed. Dataflow-based system-level design technique is one efficient approach to achieve this goal. In the next section, we present a dataflow model of the bi-rate service guarantee.

V. Dataflow Model

A dataflow graph (DFG) is a directed graph that allows high-level modeling and analysis of real-time applications. These graphs have efficient analysis techniques to compute throughput and buffer requirements of applications that are running on MPSoCs. They can capture cyclic data dependencies between processes of an application, which exist in many real-life systems. Because of these reasons, DFGs play an indispensable role in today's MPSoC design flows that carry out application binding and resource scheduling [12], [30]. In addition, dataflow design tools enable fast design-space-exploration (DSE) of MPSoCs to find arbiter configurations and other system parameters that satisfy latency and throughput requirements of applications [10], [11], [27].

State-of-the-art dataflow-based system-level analysis techniques require that the various aspects of the system, such as computation, storage and arbitration, to be modeled with dataflow components. Dataflow modeling of arbitration is not trivial, as equivalence in temporal behavior with the service guarantees of the arbiters should be proved using rigorous algebraic steps [26]–[29]. In this section, we present a dataflow model for CCSP arbitration, based on the bi-rate service guarantee previously derived in Section IV. First in Section V-A, we briefly introduce how DFGs can be used for system-level design and analysis of MPSoCs. Section V-B then presents the dataflow model for the CCSP arbiter, along with the intuition behind its execution. Due to space reasons, the complete mathematical proof of the dataflow model is not presented in this paper, but it is provided in [31].

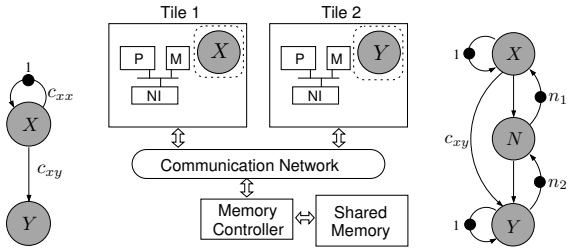
A. Dataflow-based System-level Analysis

A DFG [32] is a directed graph that consists of *actors* that are connected through *channels*. A channel represents a FIFO buffer through which actors communicate by sending *tokens*. A token represents an abstract data unit. The connection point between an actor and a channel is referred to as a *port*. An example DFG consisting of two actors X, Y and two channels c_{xx}, c_{xy} is shown in Figure 6(a). Channels that have the same source and destination actor are referred to as *self-edges* (e.g. channel c_{xx}). A number next to a black dot over a channel represents the number of *initial tokens* that are available in the channel at the beginning of the execution of the graph.

A variety of DFGs exist today with different levels of analyzability, expressiveness and implementation efficiency [33]. Homogeneous Synchronous Dataflow graph (HSDFG) is one of them. In HSDFG, when there exists at least one token on every input port of an actor, the actor *fires*. At the end of the firing, the actor produces one token on each of its output ports. The execution duration of a single firing of an actor Y is denoted $\chi(Y) \in \mathbb{R}^+$. Depending on the system aspect a given actor models, the implication of the execution duration may vary. For instance, for an actor that models a software process, the execution duration may represent its worst-case execution time (WCET) for a given processor type.

Figure 6(b) shows an example binding of actors X and Y onto an example MPSoC platform. The platform is a multi-tile architecture, where each tile comprises a processing unit (P), a local memory (M) and a network interface (NI).

By introducing dataflow components that model various sorts of architectural aspects, such as computation, storage, arbitration and communication, into the DFG, an architecture-aware DFG can be created. A simple example of an architecture-aware HSDFG is shown in Figure 6(c). In this graph the initial tokens n_1 and n_2 model the allocated buffer sizes for the channel c_{xy} on Tile 1 and Tile 2, respectively. The inter-tile communication latency between actors X and Y is modeled by the execution time of actor N, $\chi(N)$.



(a) DFG (b) Binding of actors (c) Architecture-aware DFG

Fig. 6. Use of DFGs for system-level design and analysis of MPSoCs

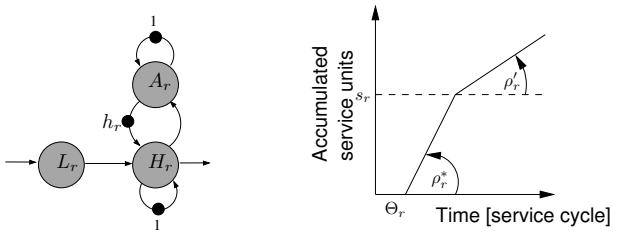
Timing analysis and scheduling of this system can be carried out on the architecture-aware DFG. For example, the worst-case throughput of the application is the inverse of the *maximum cycle mean (MCM)* of the graph, which is the largest average execution duration among all cycles in the graph [24]. The approach presented in this section can also be applied to complex graphs that use more expressive DFGs. Detailed discussions on dataflow-based system-level design and analysis techniques can be found in [10], [11] and [33].

B. Dataflow Model for CCSP arbitration

We present a HSDFG model, which is shown in Figure 7, for service provision using CCSP arbitration for a requestor $r \in R$. The dataflow model consists of three actors: *latency actor (L_r)*, *higher-rate actor (H_r)* and *allocated-rate actor (A_r)*. The number of initial tokens h_r and the execution durations of the three actors are determined by the requestor's allocated service and the allocated services of its higher-priority requestors, R_r^+. The number of initial tokens h_r is given in Equation (16), where s_r, provided in Equation (17), refers to the number of service units that can be served at the higher rate during every active period.

$$h_r = \left\lfloor s_r - \frac{(s_r - 2) \cdot \rho'_r}{\rho^*_r} \right\rfloor \tag{16}$$

$$s_r = \left\lfloor \frac{\Theta_r - \Gamma_r}{\frac{1}{\rho_r} - \frac{1}{\rho^*_r}} \right\rfloor \tag{17}$$



(a) CCSP dataflow model (b) Rate of provided service

Fig. 7. Dataflow model for CCSP arbitration.

The execution durations of the three actors for a requestor $r \in R$ are given in Equations (18), (19) and (20). The case where $h_r = 1$ is unique from all other cases ($h_r > 1$), since this implies that actors H_r and A_r cannot be executed in parallel. This is because there can exist a maximum of

only one token at a time over their cyclic dependency. The execution duration of actor A_r, $\chi(A_r)$ for $h_r = 1$ is hence different from the other cases, as shown in Equation (20).

$$\chi(L_r) = \Theta_r \tag{18}$$

$$\chi(H_r) = \frac{1}{\rho_r^*} \tag{19}$$

$$\chi(A_r) = \begin{cases} \frac{1}{\rho'_r} & \text{if } h_r > 1 \\ \frac{1}{\rho'_r} - \frac{1}{\rho_r^*} & \text{if } h_r = 1 \end{cases} \tag{20}$$

Figure 8 demonstrates the use of the CCSP dataflow model in the context of system-level design by extending Figure 6(c) to model CCSP arbitrated shared memory communication between actors X and Y of Figure 6(a). Actor N of Figure 6(c) is replaced by two CCSP dataflow models for the write and read latencies of actors X and Y. Actors N_x and N_y model the latencies over the communication network.

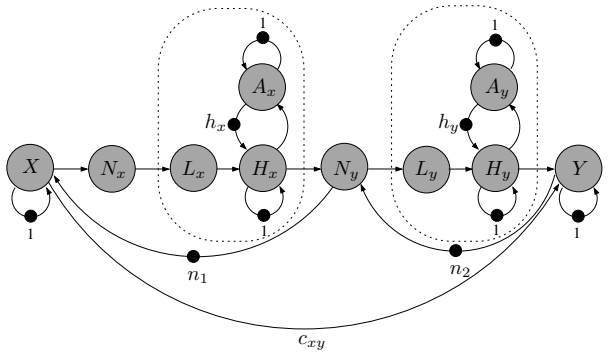


Fig. 8. Architecture-aware DFG with shared memory communication.

The rate of service provision by the CCSP dataflow model, shown in Figure 7(b), is equivalent to the bi-rate service guarantee, previously given in Equation (10). In the remaining part of this section, the intuition behind the rate of service provision of the dataflow model is explained.

Arriving requests at the input of the CCSP arbiter can be multiple service units long, i.e. for the k^{th} request $s(\omega^k) \geq 1$. Each service unit is represented by a single token in our dataflow model. The arrival of a request at the input of the arbiter is analogous to the arrival of multiple tokens at the input of actor L_r. This means the arrival time of the j^{th} token at actor L_r, denoted $\mathcal{E}(j)$, marks the arrival time of the j^{th} service unit at the input of the arbiter. Actor L_r produces a single token for every arriving token after an execution duration of Θ_r service cycles, which models the maximum waiting time every service unit encounters. The production time of the j^{th} token by the latency actor equals $\mathcal{E}(j) + \Theta_r$.

It is important to notice that actor L_r does not have a self-edge with initial tokens. The number of initial tokens on a self-edge determines the *auto-concurrency* of the actor, which is the number of possible simultaneous firings of the actor. This implies that multiple tokens can wait in parallel, which accurately reflects what actually happens at the input of the arbiter. On the other hand, actors H_r and A_r have self-edges, each with a single initial token that implies only one service unit can be served at a time by the resource.

Output tokens produced by actor H_r represent completed service units. This means that the production time of the j^{th} token by actor H_r marks the finishing time of the j^{th} service unit, denoted $\mathcal{F}(j)$. $\mathcal{F}(j)$ is determined by the arrival time of tokens from its three input channels, shown in Figure 7(a). These are: (1) the arrival of the j^{th} token from the output of actor L_r, $\mathcal{E}(j) + \Theta_r$, (2) the finishing time of the previous firing, $\mathcal{F}(j-1)$, which models that only one service unit can be served at a time and (3) the production time of the $(j - h_r)^{th}$ token from actor A_r. This is because there are already h_r initial tokens in this channel, representing the number of service units that can be served at the higher rate, ρ_r^*, before the potential is exhausted at the boundary cycle. For every service unit served at the higher rate, one token is consumed from these initial tokens. As long as there are tokens available on this channel, service units can be served at the higher rate. This is because in order to fire, actor H_r does not have to wait for the production of tokens by actor A_r. However, after all tokens are consumed, the rate of production of tokens by actor H_r is determined by the rate of production of tokens by the actor A_r. According to Equation (20), this implies that service units are served at the allocated rate of the requestor, ρ'_r. Notice that for $h_r = 1$, the rate of token production by actor H_r is the sum of the execution durations of both actors H_r and A_r. This is because the two actors cannot be fired simultaneously, since there can exist a maximum of only one token over the cyclic dependency at a time.

Bringing the above three limiting factors together, the finishing time of service units can be expressed with a max-plus equation [34] that bounds the completion time of service units, as shown in Equation (21). $\mathcal{G}(j)$ denotes the finishing time of the j^{th} token at the output of the allocated-rate actor.

$$\mathcal{F}(j) = \begin{cases} max(\mathcal{E}(j) + \Theta_r, \\ \qquad \mathcal{F}(j-1), \\ \qquad \mathcal{G}(j-h)) + \frac{1}{\rho_r^*} & j > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{21}$$

The mathematical proof of equivalence between Equation (21) and the bi-rate service guarantee, previously given in Equation (10), is provided in [31].

VI. Experiment Results

This section presents experimental results that demonstrate the implications of our new bi-rate service guarantee for a real-life application. First, in Section VI-A, the experimental setup is presented. Then, Section VI-B shows improvements in guaranteed throughput by applying the bi-rate service guarantee. Finally, Section VI-C demonstrates how the throughput improvement translates to resource savings when satisfying real-time requirements.

A. Experimental Setup

Our experimental setup emulates a MPSoC with a SRAM memory controller that uses the CCSP arbiter to arbitrate memory accesses from multiple requestors. We limit the number of requestors to five, r_1 to r_5, and they are assigned unique priority levels in decreasing order, giving r_1 the highest and r_5 the lowest priorities. Each requestor r_i has an associated allocated rate (ρ'_{r_i}) and allocated burstiness (σ'_{r_i}).

In real-time system design, the resource allocation of requestors should guarantee that each of them satisfies their timing requirements in the worst-case interference scenario.

The objective of this experiment is to demonstrate improvements in throughput and resource allocation by applying the bi-rate service guarantee as compared to the existing latency-rate service guarantee. For this purpose, a MATLAB module is written that models a simple SRAM memory controller with a 32-bit interface, both according to the latency-rate dataflow model of [28] and our new dataflow model, presented in Section V. Given a requested service curve, w_{r_i}, the module computes two lower bounds on the provided service curve, $\check{w}'_{r_i}$ and $\check{\check{w}}'_{r_i}$, based on the latency-rate and the bi-rate dataflow models, respectively. The experiment setup is illustrated in Figure 9, where the execution duration of actor S_{r_i} of the $\mathcal{LR}$ server model is given by $\chi(S_{r_i}) = 1/\rho'_{r_i}$.

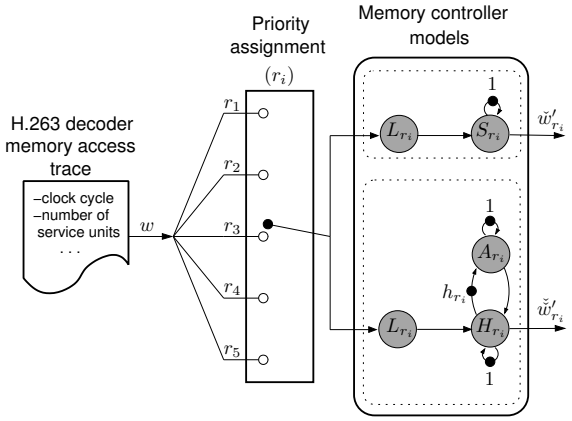


Fig. 9. The experimental setup.

A requested service curve w is generated by tracing memory transactions of a real-life application. The SimIt-ARM [35] cycle-accurate simulator is used to trace all external memory transactions (cache misses) from a StrongArm processor that runs an H.263 video decoder application (written by Telenor research, a popular public-domain implementation of H.263). The simulator is configured with a 16 Kbyte instruction cache with 32-byte blocks and a 16 Kbyte write-back data cache with 32-byte blocks. In addition, the memory reading and writing latencies of the simulator are set to zero while generating the trace. The service unit of the memory controller module is set to 32-bytes (the cache line) and the service cycle to 8 clock cycles. In the experiment, it is assumed that all memory read and write transactions stall the processor. The stalling of the processor is taken into account by modifying the requested service curve w in such a manner that the arrival time of every transaction (requested service unit) is delayed by the response time of the previous transaction. This is important as the trace is originally generated by setting the write and read latencies of the simulator to zero.

B. Throughput Improvement

In this experiment, we intend to demonstrate differences in guaranteed throughput according to the latency-rate and bi-rate service guarantees. Assuming the traced traffic represents the worst-case memory access pattern, we compute the throughput by taking the inverse of the total time required to decode one complete frame. This gives us the minimum throughput in frames per second (fps), since the worst-case memory access pattern yields the maximum total time duration to decode one frame. Figure 10 shows the service curves while the system decodes one complete frame where $\forall i\ \rho'_{r_i} = 0.15$, $\forall i\ \sigma'_{r_i} = 2$ and the application is assigned priority level 3 in the CCSP arbiter. We return to the experiment with these parameters later. The modified requested service curves (according to both service guarantees) are not shown in this figure, since they overlap with the two lower bounds ($\check{w}'_{r_3}$ and $\check{\check{w}}'_{r_3}$) at this graph scale. This graph shows the improvement in latency by using the bi-rate service guarantee, compared to the latency-rate rate service guarantee.

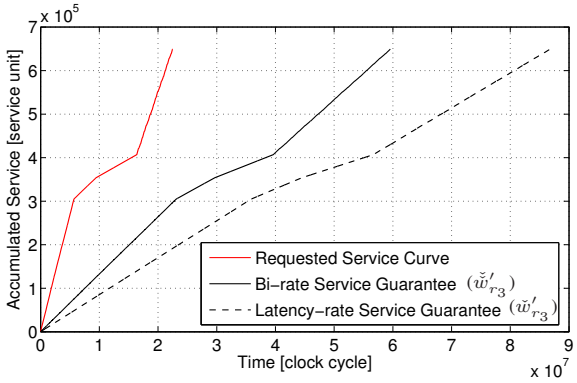


Fig. 10. Requested service (*generated with zero memory access latency*) and lower bounds for the provided service based on latency-rate and bi-rate service guarantees.

Table I shows the improvements in throughput as the assigned priority of the application is varied from 1 to 5. The allocated rate is fixed at 0.15 for all cases and the allocated burstiness of all requestors is varied together from 1 to 4. The result shows that requestors with average priority improve more than both high and low priority requestors. This is because high-priority requestors have smaller Θ, which implies that they accumulate less potential while being blocked and low-priority requestors have smaller ρ^* relative to high-priority requestors (cf. Equation (3)). Though it is not shown here, we are also able to observe from this experiment that the achievable throughput improvements for different allocated rates have similar trends as the results shown in Table I.

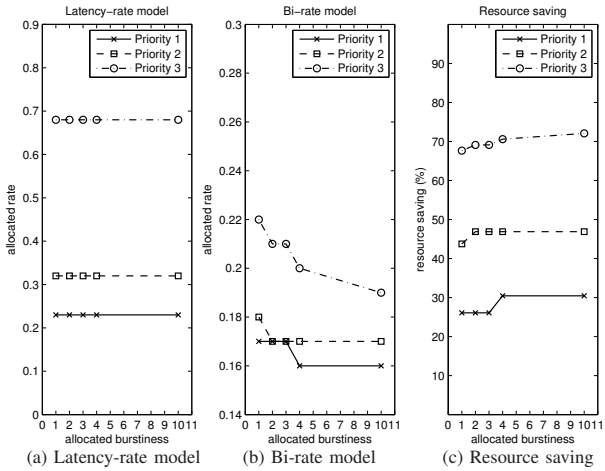
TABLE I
THROUGHPUT IMPROVEMENTS IN %.

Priority	$\forall i\ \sigma'_{r_i} = 1$	$\forall i\ \sigma'_{r_i} = 2$	$\forall i\ \sigma'_{r_i} = 3$	$\forall i\ \sigma'_{r_i} = 4$
1	27	30	30	31
2	36	48	54	53
3	44	45	36	30
4	41	28	21	17
5	24	15	11	9

It is important to note that these achieved improvements are neither at the cost of additional latency penalty of lower priority requestors nor by additional allocated resource capacity. It is a direct result of the tighter bi-rate service guarantee that accurately models the service provided by the CCSP arbiter.

C. Efficient Resource Utilization

The improvements in throughput can be traded for resource savings, since the throughput requirement of a real-time requestor can now be satisfied with less allocated resources. Assume that our application runs on a 266 MHz processor and has a throughput requirement of 15 fps. Figure 11 shows all combinations of service allocations and assigned priorities that satisfy this requirement, based on both the latency-rate and bi-rate guarantees.



(a) Latency-rate model (b) Bi-rate model (c) Resource saving

Fig. 11. Resource allocation for throughput-constrained application: a) shows latency-rate cannot capture burstiness and b) shows bi-rate can.

It can be seen that by using the bi-rate service guarantee and without allocating any additional burstiness (keeping $\sigma'_{r_i} = 1$), the resource allocation can be reduced by 67% (from $\rho'_r = 0.68$ to $\rho'_r = 0.22$) at priority level 3, by 43% at priority level 2 and by 26% at priority level 1. The resource savings further increase when increasing the allocated burstiness. For instance, at priority level 3 and $\sigma'_{r_3} = 10$, the resource allocation can be reduced by 72% (from 0.68 to 0.19). These improvements are achieved without imposing any additional latency on lower-priority requestors, since the arbiter configuration is the same in both lower-bound computations.

This experiment shows that the latency-rate service guarantee cannot capture bursty provided service and, hence, fails to completely decouple latency and rate. In this use case, the allocated rate must be increased at all priority levels to satisfy the throughput requirement of 15 fps. This results in substantial over-allocation of the resource. For instance, when the application is assigned priority level 3, 46% of the memory bandwidth is wasted due to the pessimism of the latency-rate model, which allocates 68% of the bandwidth. In contrast, the bi-rate service guarantee, presented in this paper, accurately models the behavior of the CCSP arbiter and accomplishes this at 22%, and consequently, enables efficient resource utilization by avoiding over-allocation of the resource.

VII. Conclusions

The Credit-Controlled Static-Priority (CCSP) arbiter is suitable for scheduling shared System-on-Chip resources, such as memories. The existing linear service guarantee of the arbiter fails to capture that service may be provided in a bursty manner, resulting in over-allocation of resources to satisfy real-time requirements. In this paper, we address this drawback through a new piecewise linear (bi-rate) service guarantee. In addition, for system-level analysis, we present a novel dataflow model of the arbiter, based on the new service guarantee. As a result of the new bi-rate guarantee, a given resource under CCSP arbitration can support more requestors, or a given set of requestors can be accommodated with less resource capacity. Experimental results on traced traffic of an H.263 video decoder application show that savings from 26% up to 67% in memory bandwidth can be achieved by using the new bi-rate guarantee compared to the existing linear guarantee.

References

[1] "International Technology Roadmap for Semiconductors (ITRS)," 2009.
[2] B. Akesson *et al.*, "Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration," in *Proc. RTCSA*, 2008.
[3] M. Steine *et. al*, "A priority-based budget scheduler with conservative dataflow model," in *DSD*, 2009.
[4] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans.*, 1998.
[5] B. Akesson *et al.*, "Efficient Service Allocation in Hardware Using Credit-Controlled Static-Priority Arbitration," in *Proc. RTCSA*, 2009.
[6] A. Francini and F. Chiussi, "Minimum-latency dual-leaky-bucket shapers for packet multiplexers: Theory and implementation," in *IWQoS*, 2000.
[7] H. J. Chao and J. S. Hong, "Design of an ATM shaping multiplexer with guaranteed output burstiness," *Computer Systems Science and Engineering*, 1996.
[8] F. Guillemin *et. al*, "Extremal traffic and bounds for the mean delay of multiplexed regulated traffic streams," in *INFOCOM*, 2002.
[9] S. Stein *et. al*, "A polynomial-time algorithm for computing response time bounds in static priority scheduling employing multi-linear workload bounds," in *ECRTS*, 2010.
[10] A. Bonfietti *et. al*, "An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms," in *DATE*, 2010.
[11] C. Lee *et. al*, "A systematic design space exploration of MPSoC based on synchronous data flow specification," *Journal of Signal Processing Systems, Springer*, 2010.
[12] A. Hansson *et. al*, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *ACM TODAES*, vol. 14, no. 1, 2009.
[13] M. Katevenis *et. al*, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 8, Oct. 1991.
[14] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. SIGCOMM*, 1995.
[15] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, no. 10, 1995.
[16] C. Kalmanek *et. al*, "Rate controlled servers for very high-speed networks," *Proc. GLOBECOM*, 1990.
[17] S. S. Kanhere and H. Sethu, "Fair, efficient and low-latency packet scheduling using nested deficit round robin," *High Performance Switching and Routing, 2001 IEEE Workshop on*, 2001.
[18] D. Saha *et al.*, "Carry-over round robin: a simple cell scheduling mechanism for ATM networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, 1998.
[19] B. Akesson *et. al*, "Predator: a predictable SDRAM memory controller," in *Proc. CODES+ISSS*, 2007.
[20] J.-Y. L. Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the Internet*. Springer-Verlag New York, Inc., 2001.
[21] R. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, 1991.
[22] G. Kesidis and T. Konstantopoulos, "Worst-case performance of a buffer with independent shaped arrival processes," *IEEE Communications Letters*, 2000.
[23] E. Wandeler *et. al*, "Performance analysis of greedy shapers in real-time systems," in *DATE*, 2006.
[24] A. Ghamarian *et. al*, "Throughput analysis of synchronous data flow graphs," in *ACSD*, 2006.
[25] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," in *CODES/ISSS*, 2010.
[26] M. Bekooij *et. al*, "Performance guarantees by simulation of process," in *SCOPES*, 2005.
[27] M. Bekooij *et. al*, "Dataflow analysis for real-time embedded multiprocessor system design," in *chapter 15. Dynamic and Robust Streaming Between Connected CE Devices*. Kluwer Academic Publishers, 2005.
[28] M. Wiggers *et. al*, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *SCOPES*, 2007.
[29] J. Staschulat and M. Bekooij, "Dataflow models for shared memory access latency analysis," in *EMSOFT*, 2009.
[30] A. Kumar *et. al*, "Multi-Processor System-Level Synthesis for Multiple Applications on Platform FPGA," in *FPL*, 2007.
[31] F. Siyoum *et. al*, "Dataflow Model for Credit-Controlled Static-Priority Arbitration," http://www.es.ele.tue.nl/esreports/, TU/Eindhoven, Tech. Rep., 2010.
[32] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, 1987.
[33] S. Stuijk, "Predictable Mapping of Streaming Applications on Multiprocessors," Ph.D. dissertation, Eindhoven University of Technology, 2007.
[34] B. Heidergott *et. al*, *Max Plus at Work: Modeling and Analysis of Synchronized Systems*. Princeton University Press, 2006.
[35] http://simit-arm.sourceforge.net/.