A General Framework for Average-Case Performance Analysis of Shared Resources

Sahar Foroutan[1], Benny Akesson[2], Kees Goossens[2] and Frederic Petrot[1]
[1]TIMA Laboratory, Grenoble, France
[2]Eindhoven University of Technology, Eindhoven, the Netherlands

ABSTRACT. *Contemporary embedded systems are based on complex heterogeneous multi-core platforms to cater to the increasing number of applications, some of which have (soft) real-time requirements. To reduce cost, resources are shared using diverse arbitration mechanisms, such as Time-Division Multiplexing (TDM), Static-Priority (SP), and Round-Robin (RR), depending on application and resource requirements. However, resource sharing results in interference between sharing applications making it difficult to estimate if the average latency is sufficient to satisfy their real-time requirements. Existing work proposes isolated models that either fail to address the diversity of arbitration mechanisms or cannot capture the dynamic arrival and service processes of applications and resources in multi-core platforms.*
This paper addresses this problem by proposing a general framework for average-case performance analysis of shared resources in multi-core platforms. The two main contributions are: 1) a general model for resource sharing based on queuing theory that can be used with different arbiters and that captures architectural features of the shared resource, such as pipelining and arbitration delay, and 2) three arbiter models for TDM, SP, and RR, respectively that assume general distributions (G/G/1) and fits within the framework.

I. INTRODUCTION

Embedded systems are growing in complexity as more and more applications (functionality) are integrated into a single system. In consumer electronics, this is driven by the convergence of application domains, where applications from previously independent products, e.g. media players, navigators, personal digital assistants, handheld video games, and telephones, come together in a single device, such as a smart phone. To deliver on the computational requirements of these applications in a cost-effective and power-conscious manner, systems are based on heterogeneous multi-processor platforms with shared resources, such as interconnect and memories [1-3]. Access to these shared resources is provided by different resource arbiters, such as Time-Division Multiplexing (TDM), Static-Priority Arbitration (SP), and Round-Robin (RR), depending on resource characteristics and application requirements.

Some applications have *real-time requirements* that must be satisfied for the application to execute correctly. This work considers *soft* real-time requirements that are common in the multimedia domain. This type of requirement must typically be satisfied, but can occasionally be violated at cost of a slight decrease in quality of the output, such as audible or visual artifacts in an audio or video stream. To design these systems cost-efficiently, performance analysis of soft real-time applications typically consider the average execution time of the application and hence requires estimates on the average time required to access shared resources, which can be derived using probabilistic models, for example based on queuing theory. However, existing queuing models either assume that arrival and service processes follow an exponential distribution, which does not hold for dynamic applications, or address only a single arbitration mechanism in isolation, thus failing to recognize the diversity of arbiters in complex systems.

This paper addresses this problem by proposing a general framework for average-case performance analysis of shared resources in multi-core platforms. The two main contributions of the work are: 1) a general high-level model for resource sharing based on queuing theory that can be used with different arbitration mechanisms and that captures architectural features of the shared resource, such as pipelining and arbitration delay, 2) three arbiter models for TDM, SP, and RR, respectively, that assume general distributions of request inter-arrival and service times (G/G/1). We experimentally evaluate the proposed framework by comparing the models to results obtained by simulation of a shared SRAM memory controller using both synthetic traffic and traces from real applications in the multimedia domain. We show that the models have average deviations of 4.1% for TDM, 12.9% for SP, and 14.2% for RR with synthetic traffic and that although larger deviations are observed for realistic applications, they significantly outperform models assuming exponentially distributed traffic.

The rest of this paper is organized as follows. Section II discusses related work. Section III then presents our resource model and a short introduction to queuing theory. The proposed general framework is introduced in Section IV before Section V presents three arbiter models for TDM, SP, and RR, respectively, that fits within the framework. The framework and its models are experimentally evaluated in Section VI before conclusions are drawn in Section VII.

II. RELATED WORK

Platform resources are shared with a diversity of arbiters to address the heterogeneous performance requirements of their applications. Several isolated performance models of such arbiters have been proposed [4-13], mostly based on queuing theory [5-12]. However, they typically assume that request inter-arrival times and resource service times follow Poisson (exponential) distributions [4-9, 11, 13], which do not cover the dynamic

and bursty nature of applications executing through a cache or the behavior of resources with complex timing behavior, such as SDRAMs [14]. In [6], the authors propose a multiple-queue round-robin model for routers in best-effort networks-on-chips (NoCs). However, this model assumes data arrives according to a Poisson process, which is a common assumption in performance analysis of both off-chip and on-chip networks [7, 9-11, 13]. For SP, there exist models in literature [12, 15] that consider traffic arrivals following a general distribution. Based on the SP model of [15], which assumes infinite arbiter queues, [12] proposes a model for arbiters with limited buffering space in routers of QoS NoCs.

A problem with the aforementioned works is that they model single arbiters and do not consider the diversity of arbiters in contemporary platforms, restricting performance analysis to different tools and models for every shared resource. To this end, we propose a general framework that captures several arbiters in a unified manner, simplifying implementation in performance analysis tools. It also captures architectural features of the shared resource, such as pipelining and arbitration delay. Furthermore, we present three arbiter models for RR, SP, and TDM, respectively, that fit within this framework. All three arbiter models consider generally distributed traffic arrival times and resource service times (G/G/1) to increase their suitability for real applications executing on multi-core platforms.

III. BACKGROUND

This section presents relevant background information, required to understand the discussions in this work. First, we describe the resource model used in this paper, followed by an introduction to queuing theory.

III.1. Resource Model

We consider an abstract resource model, shown in Figure 1, with a single resource and multiple queues belonging to separate *requestors* competing for access to the shared resource. This single resource can be a memory, processor, or an output channel of a NoC router to which several input ports share access. We assume that the resource has separate queues per requestor and that they are sufficiently large to decouple production and consumption behavior. This is consistent with most other work on queuing theory, which assumes sufficiently large buffers either at the traffic source or at the resource. Although this assumption is not practical to implement in the context of embedded systems, it enables us to focus on modeling different arbitration mechanisms and to be comparable to related work. Improving on this assumption is left as future work.

Depending on the latency requirements of the requestors, different arbitration mechanisms can be used to schedule requests, such as TDM, RR, or SP. An arbitration mechanism can be classified as either *work-conserving* or *non-work-conserving*. A work-conserving arbiter is never idle while there a requestor with waiting requests. Conversely, a non-work-conserving arbiter does not schedule a waiting request until it is *eligible*, i.e. when it is its turn in the TDM schedule or when the requestor has sufficient budget in budget-based schedulers. The main advantage of being work-conserving is that it improves resource utilization and reduces the average latency by using slack capacity, i.e. unallocated resource capacity or capacity allocated to other requestors that is not used [16]. In contrast, a non-work-conserving arbiter may be idle even though there are waiting requests. This approach is useful to keep traffic from different requestors independent from each other and is beneficial to reduce burstiness in networks of resources [17]. In this work, we use the convention that RR and SP are work-conserving arbiters and TDM non-work-conserving.

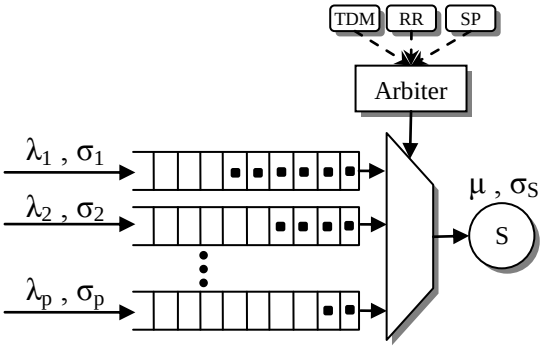

Figure 1. A general shared resource with separate queues per requestor supporting multiple arbitration mechanisms.

III.2. Queuing Theory

The real-time analysis in this work builds on queuing theory and the fundamentals of this framework are presented in this section. Although we consider a resource with multiple queues, we start by introducing the single-queue model, which is the traditional model used in queuing theory. This is later generalized to a multiple-queue model in Section IV.

In queuing theory, a queuing model ***A/S/N*** represents a single-queue system, where ***A*** is the arrival process of requests to the queue, ***S*** represents the service process of requests in the resource, and ***N*** represents the number of resources (here N=1) [18, 19]. In a single-queue system, requests are served in the same order they arrive to the queue, i.e. in first-in first-out (FIFO) order. The average latency of an incoming request depends on the number of requests in the queue that, in turn, depends on the arrival and service processes. To capture the dynamic behavior of real applications, this work considers general arrival and service distributions (G/G/1) instead of the commonly used exponential distributions (i.e. Poisson process). This is required to better cover the irregular behavior and bursty arrival patterns of realistic applications executing through a cache, as well as the variable service times caused by different request sizes or resources with complex service time behavior, such as SDRAMs. We motivate this by showing two histograms of request inter-arrival times from a JPEG decoder and an H.263 decoder executing through a cache in Figure 2 (the exact setup that generated these histograms are later described in Section VI). The figures

clearly show that the inter-arrival times of these applications do not follow an exponential distribution, or any other simple distribution for that matter, requiring more general models.

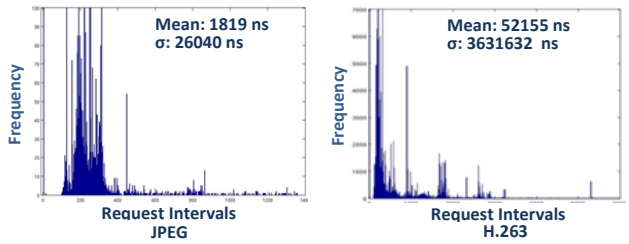

Figure 2. Histograms of request intervals of JPEG and H.263 decoders

In a G/G/1 model, both request inter-arrival times and service times are independent and identically distributed with a general distribution. The inter-arrival times and service times are furthermore independent from each other. The arrival process is characterized by the mean and standard deviation (or variance) of request inter-arrival times, T_A and σ_A (or σ^2_A), respectively. Similarly, the service time is characterized by the mean and standard deviation (or variance) of service times, T_S and σ_S (or σ^2_S). The mean arrival rate and mean service rate are determined by $\lambda=1/T_A$ and $\mu=1/T_S$, respectively, as indicated in Figure 3. The average waiting time a request spends in the queue, referred to as the *queuing delay*, is computed according to Equation 1, where ***n*** is the average number of requests already waiting in the queue and ***R*** is the *residual time*, explained later.

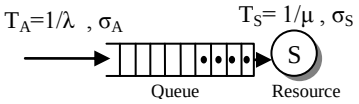

Figure 3. G/G/1 Single-queue, single-resource model

Equation 1: Queuing delay in a single-queue system.

$$W = nT_s + R$$

According to Little's law [15], shown in Equation 2, the long-term average number of requests in a queue, n, is equal to the long-term average arrival rate, λ, multiplied by the average time a request spends in the queue, W. Little's law is a general result holding for arbitrary distributions of inter-arrival times and service times, including G/G/1 models [15]. Based on Little's law, Equation 1 can be rewritten according to Equation 3.

Equation 2: Little's law [15].

$$n = \lambda W$$

Equation 3: Queuing delay as a function of λ and μ.

$$W = \frac{R}{1 - \lambda T_s} = \frac{R}{1 - \lambda/\mu}$$

In queuing models, the utilization of the resource is defined as $\rho = \lambda / \mu$, where $\rho \leq 1$ to prevent overload. Equation 3 can hence be rewritten according to Equation 4.

Equation 4: Single queue main equation.

$$W = \frac{R}{1 - \rho}$$

The residual time, *R*, is the mean remaining service time of a busy resource and is given by Equation 5.

Equation 5: Residual time [15]:

$$R = P(busy\ resource) \cdot t_s + P(idle\ resource) \cdot 0$$

When a request arrives, the request already in service needs $\boldsymbol{t_s}$ time units to be finished. This quantity ($\boldsymbol{t_s}$) is called the *remaining service time* and is approximated by Equation 6 for G/G/1 models [15], where C_A and C_S are coefficients of variation (CV) of the request inter-arrival distribution and service-time distribution, respectively. Here, we remind that the relationship between the CV of a random variable X and its mean (m_x) and variance (σ_X^2) is expressed as $CV_X^2 = \sigma_X^2 / m_X^2$.

Equation 6: Remaining Service time for G/G/1 model.

$$t_s \approx \frac{(C_A^2 + C_S^2)}{2\mu}$$

In a single-queue model, the probability *P(busy resource)* in Equation 5 is the server utilization ρ. Therefore, using ρ and $\boldsymbol{t_s}$ in Equation 5, we arrive at Equation 7. Combining this with Equation 4 results in Equation 8, which is the well-known Allen-Cunneen approximation formula for G/G/1 queues [15]. Note that there is no exact queuing solution for G/G/1 queues and only approximations have been proposed in the literature, among which the Allen-Cunneen approximation is one of the most commonly used.

Equation 7: Residual time for G/G/1 model.

$$R_{G/G/1} \approx \frac{\rho}{2\mu}(C_A^2 + C_S^2)$$

Equation 8: Allen-Cunneen approximation formula for G/G/1 queues.

$$W \approx \frac{\rho(C_A^2 + C_S^2)}{2\mu(1-\rho)}$$

IV. General Framework

Having presented our resource model and the fundamentals of queuing theory, we proceed by presenting our general framework for resource sharing with multiple queues. This framework extends on the basic queuing theory presented in Section III.2 by considering different arbiters, as well as architectural features of the shared resource, such as pipelining and arbitration delay.

For each requestor, we are interested in determining the average total time the requests of that requestor spend in the corresponding queue. This is referred to as the *scheduling latency*, expressed in Equation 9, and is defined as the (average) time between the moment a request arrives and the moment it is scheduled.

Equation 9: Scheduling latency.

$$\text{SchedLat} = \text{ArchDly} + \text{ArbDly} + \text{W}$$

The scheduling latency comprises three components. The first component is the *architecture delay* (ArchDly), which is a constant delay affecting all requests, even when

the queues are empty. This captures details of the arbiter architecture, such as pipelining. Pipelining typically only accounts for one to a few clock cycles, but this delay is still significant to model for resources that are shared at a very fine granularity, such as SRAM memories, where the service time may be as low as a single cycle for word-sized accesses. It is important to add this constant to the computed scheduling latency of requests and not to the mean service time, T_S, since pipelining affects latency of requests, but does not reduce the throughput of the resource.

The second component of the scheduling latency is the *arbitration delay* (ArbDly), which corresponds to the average time between two consecutive arbitration decisions. This models that arbitration in an idle resource does not always happen as soon as a request arrives, but may be restricted to predetermined, e.g. periodic, timeslots to create isolation between applications. For arbiters in which scheduling can take place every clock cycle, the arbitration delay is equal to zero.

The final component of the scheduling latency is the queuing delay (W), previously introduced in Section III.2, which considers the average waiting time in the queue. This depends on the average number of requests that are waiting to be served and depends on how the particular arbiter schedules requests from the different queues. Our goal is to have a general framework that covers multiple arbiters in a unified way. To achieve this, we introduce the general expression in Equation 10 to compute the queuing delay.

Equation 10: Unified model for queuing delay with multiple queues.

$$W_i = \overbrace{n_i T_{S_i}}^{I} + \overbrace{\sum_{j=1,\ j\neq i}^{p} T_{S_j} \,.\, A(n_i, n_j)}^{II} + \overbrace{R}^{III}$$

The objective of this model is to approximate the average latency of a request arriving to any queue of a multiple-queue resource with p queues, as previously shown in Figure 1. Requests of each queue are served in a FIFO fashion with respect to each other (i.e. there is no reordering between requests in the same queue). $\boldsymbol{W_i}$ is the average waiting time (queuing delay) of queue $\boldsymbol{i}$. T_{S_j} is the mean service time of any other queue $\boldsymbol{j}$. The first part ***(I)*** of the equation represents the waiting time due to requests in the same queue (queue i), which is similar to the single-queue waiting time from Section III.2. The second part of the equation (***II***) relates to the interference from other queues. At the arrival time of the considered request to queue $\boldsymbol{i}$, there are a number of requests ($\boldsymbol{n_j}$) in any other queue $\boldsymbol{j}$ that depending on the scheduling policy may be served earlier. For example, with a static-priority arbiter, only those requests that belong to queues with a higher priority than queue $\boldsymbol{i}$ are served before the considered request. The main novelty of Equation 10 is function $A(n_i,n_j)$ that indicates the interference of other queues on queue $\boldsymbol{i}$ and thus models the arbitration. We call this the *arbitration indicator* and we explain how it is determined for TDM, SP, and RR arbiters, respectively, in Section V. The third part of Equation 10 represents the residual time in multiple G/G/1 queue models. It is approximated according to Equation 11, which expresses the average residual times of all queues and is obtained similarly to Equation 7 (i.e. by multiplying t_{si} by the resource utilization, ρ_i, for each queue i) [15].

Equation 11: Residual time for G/G/1 multiple-queue model.

$$R_{G/G/1} \approx \sum_{i=1}^{p} \frac{\rho_i}{2\mu_i}\left(C_{A_i}^2 + C_{S_i}^2\right)$$

V. ARBITER MODELS

After presenting our general framework for resource sharing with multiple queues, we introduce three arbiter models for TDM, SP, and RR, respectively, that fits within this framework.

V.1. Time-Division Multiplexing

TDM arbitration is based on a periodically repeating frame (table) with fixed period, F^{TDM}. Service is allocated to requestors by assigning slots within the frame, each corresponding to service of one request. TDM is usually non-work-conserving, and the arbiter thus idles if the requestor assigned to a particular slot does not have waiting requests. This makes the timing behavior of the queue completely independent from the other queues, as expressed by the arbitration indicator in Equation 12. This enables Equation 10 to be rewritten according to Equation 13. Furthermore, since queues are independent, the residual delay (R) of each requestor depends only on mean and variance of request inter-arrival times to the same queue. Therefore R in Equation 13 is obtained from Equation 7 (and not Equation 11). These insights convert the TDM delay model from a multiple-queue model to an equivalent single-queue model with $T_s = F^{TDM}$ that can be approximated by Equation 8.

Equation 12: Arbitration indicator for TDM.

$$A\left(n_i, n_j\right)^{TDM} = n_i \quad for\ 1 \le j \le p$$

Equation 13: TDM queuing delay for any requestor $\boldsymbol{i}$.

$$W_i^{TDM} = n_i \sum_{j=1}^{p} T_{S_p} + R_i$$

Note that Equation 13 depicts the most straight-forward (and the most commonly used) version of TDM in which each requestor is assigned a single timeslot. In the general case, each requestor j can be allocated one or more timeslots, each equal to the service time of one request coming from j (i.e. T_{S_j}). There may furthermore be unallocated slots in the frame. Although, Equation 13 can be easily converted to capture the general case, we leave this out of the paper for simplicity.

V.2. **Static-Priority**

Static-priority arbitration is a commonly used arbiter in systems with diverse latency requirements. A unique priority level is assigned to each requestor at design time and the mechanism simply works by always scheduling the requestor with highest priority that has waiting requests. The queuing delay of a requestor hence depends only on requestors with higher priority, which is shown in the arbitration indicator in Equation 14.

Equation 14: Arbitration indicator for SP.

$$A(n_i, n_j)^{SP} = \begin{cases} n_j, & \text{if } \boldsymbol{j} \text{ has a higher priority than } \boldsymbol{i} \\ 0, & otherwise \end{cases}$$

A problem with queues no longer being independent is that the queuing delay depends on the number of requests in the queues of other requestors, which must first be determined. For SP, this is done by first solving the equation for the highest priority requestor, which is independent of others, as a single-queue model to obtain the average number of requests in the queue (n_1). This is then used to compute the queuing delay for queues with lower priorities, one after the other in descending order of priority, using the multiple-queue model. The computed average number of requests in each queue is used for the queues with next lower priority, resolving the dependency. This means that Equation 10 turns into Equation 16 for the case of SP, where R is obtained from Equation 11. We arrive at the case of the highest priority requestor (i=1) by substituting the arbitration indicator in Equation 14 into Equation 10, reducing it to only the first and last term, which is equal to Equation 1, and then substituting W using Equation 4. For requestors of lower priorities ($2 \leq i \leq p$) substituting the arbitration indicator in Equation 14 into Equation 10 gives:

Equation 15: SP queuing delay for requestor with lower priorities

$$W_i = n_i T_{S_i} + \sum_{j=1}^{i-1} n_j . T_{S_j} + R$$

Since the middle term of Equation 15 is previously solved (for requestors with higher priorities), it can be grouped with R (that together are considered as a constant for this step of computation because they are already solved independently). Then by substituting n_i from Little's law (Equation 2) in Equation 15, we arrive at the case of lower priority requestors ($2 \leq i \leq p$):

Equation 16: SP queuing delay for any requestor ***i***.

$$W_i^{SP} = \begin{cases} R/(1-\rho_i), & i = 1 \\ (R + \sum_{j=1}^{i-1} n_j \cdot T_{S_j})/(1-\rho_i), & 2 \leq i \leq p \end{cases}$$

Models similar to the one in Equation 16 have been previously proposed in literature, e.g. in [15] and [12]. The model presented here is strongly related, but has been slightly adapted to be consistent with our general framework.

V.3. **Round-Robin**

A round-robin arbiter operates by cycling through the different queues, serving one request from each before starting over. It is hence similar to the special case of TDM where all requestors have a single assigned slot each, but with the difference that RR is typically work-conserving. This makes the round robin arbiter the most complex arbiter considered in this paper with respect to dependencies between the different queues. Queues in the TDM arbiter are completely independent and can be transformed into single-queue equivalent models. On the other hand, queues under static-priority arbitration only depend on queues with higher priority, enabling dependencies to be resolved by solving equations starting from the highest priority requestor and continue in descending order of priority. In contrast, the number of requests in each queue under round robin arbitration depends on the number of requests in *all* other queues, since arbitration is cyclic and work-conserving. This cyclic dependency between the equations of all queues must be solved either by a fixed-point iterative technique (e.g. in [4], [20] and [21]), or be formulated and solved as a matrix problem (e.g. in [6]). In [6], the authors proposed a closed-form expression for the average number of packets at each buffer of a NoC router with RR arbitration. In their model, $A(n_i,n_j)=n_j$ for any i and j. This implies that the considered request (arriving to queue i) waits for all other requests of any queue j, which is more similar to FIFO arbitration. In contrast, our proposed arbitration indicator, shown in Equation 17, captures the cyclic behavior of RR arbitration by considering that there can be maximally n_i interfering request and even less if there are less than n_i waiting requests in queue j.

Equation 17: Arbitration indicator for RR.

$$A(n_i, n_j)^{RR} = Min\,(n_i, n_j) \quad \text{for all } \boldsymbol{j}$$

VI. EXPERIMENTAL RESULTS

This section experimentally evaluates the proposed models and compares their accuracy to results obtained by simulation. We start by introducing the experimental setup, followed by two experiments. The first experiment considers synthetic traffic and the second traces from real applications in the multimedia domain.

VI.1. **Experimental Setup**

The experiments of this work are conducted on a cycle-accurate SystemC implementation of a real-time memory controller, supporting a variety of memories and arbiters [14]. For simplicity, these experiments consider a 32-bit zero-bus-turnaround SRAM by the controller using the three arbiters considered in this paper. The memory runs at a frequency of 500 MHz (the clock period is 2 ns) and it has uniform access latency of a single cycle, resulting in a peak bandwidth of 2000 MB/s. Traffic is injected into the controller by traffic generators issuing either synthetically generated traffic or elastically replaying application traces.

The experiments consider a setup of four requestors sharing the bandwidth of the memory. The request size in all experiments is set to 64 B (16 words), resulting in a uniform service time of 16 cycles for both reads and writes. This memory controller has 4 pipeline stages, resulting in an architecture delay of 4 clock cycles. The (average) arbitration delay is 8 cycles for work-conserving arbiters (RR and SP), corresponding to half a timeslot of 16 cycles. For our non-work-conserving TDM arbiter, it is half the TDM period, 32 cycles, since it can be seen as a single-queue arbiter that makes new scheduling decisions once every TDM period (a requestor cannot be scheduled before its allocated slot). Requests from each requestor arrive in separate queues in the memory controller, as shown in Figure 1. These queues are dimensioned to be sufficiently large to decouple production and consumption behavior, which is a typical setup when evaluating average latency in multi-core platforms [22-24].

VI.2. **Results with Synthetic Traffic**

We start by evaluating the accuracy of our models using synthetically generated traffic. The intervals between requests are determined according to a normal distribution $N(T_A,\sigma_A)$, where T_A is the mean and σ_A the standard deviation of the request inter-arrival distribution (for ease of presentation we use σ instead of σ_A). Traffic is generated with different mean bandwidths, controlled with T_A (straight-forwardly derived from the bandwidth and the fixed request size), and different burstiness (controlled by changing the standard deviation, σ). The settings are identical for all four requestors and the mean bandwidth is varied in a range from 0 to 500 MB/s per requestor in steps of 10 MB/s, while the standard deviation is set to 30, 60, 90, and 120 ns, respectively. This results in reasonably regular traffic with some jitter, corresponding to the behavior of some hardware accelerators in the multimedia domain.

Figure 4, Figure 5, and Figure 6 show the analytical average scheduling latencies obtained by applying the proposed models for TDM, RR, and SP arbiters, respectively, and compare them to results obtained by simulation. We start by making some general observations about the relation between requested bandwidth and latency for different burstiness (standard deviation) and scheduling policies. In Figure 4 and Figure 5, we see that higher burstiness leads to higher average latency, which is an expected result. This is because burstier traffic quickly increases queue fillings, resulting in high transient queuing delays and thus a significant growth in average latency. However, the increase of average latency towards infinity is due to a small mean request interval, i.e. high requested bandwidth, and not because of burstiness. When the arriving traffic persists indefinitely with mean request intervals close to mean service time, the number of queued requests grows towards infinity. In this case, the only impact of burstiness is how fast the latencies go towards infinity at the saturation threshold.

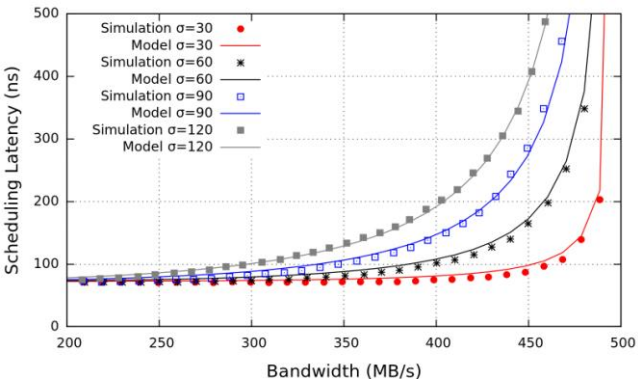

Figure 4. Average scheduling latency for TDM, 4 requestors, and for standard deviations σ=30, 60, 90, and 120 ns

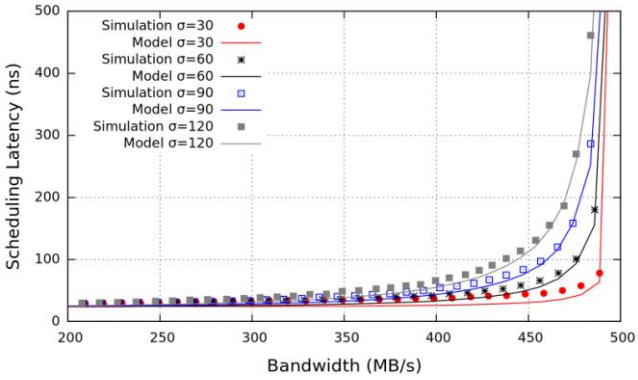

Figure 5. Average scheduling latency for RR, 4 requestors, and for standard deviations σ=30, 60, 90, and 120 ns

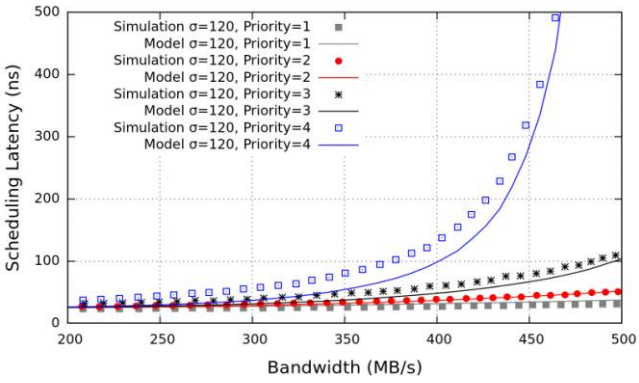

Figure 6. Average scheduling latency for SP, 4 requestors with 4 priority levels, and standard deviation σ =120 ns

Figure 6 shows the impact of static priorities on average latency when all requestors have the same stochastic characteristics. In this case, experiments were conducted with a standard deviation of 120 ns and results are shown for all four requestors. We see that only the queue with the lowest priority saturates, while the other three queues do not saturate even at high bandwidths. This is because requestors under static-priority arbitration are independent of requests generated by lower priority requestors, which means that the interfering load is decreasing with increasing priority. For this reason, the accuracy of the proposed model is better for higher priority requestors, since there is less room for variation in interference from other requestors. We also see that the requestors with the

two highest priorities have average scheduling latencies close to ArchDly plus ArbDly (see Equation 9). This shows that request arriving to these queues hardly experience any queuing delay and are served immediately.

Results from the three different arbiters are summarized in Figure 7 for σ =120 ns. For SP, only the queue with the lowest priority is shown. We see that TDM scheduling leads to the worst average latency, even compared to the lowest priority requestor in SP scheduling. This is due to the non-work-conserving nature of the arbiter, which significantly increases average latencies by not exploiting idle slots.

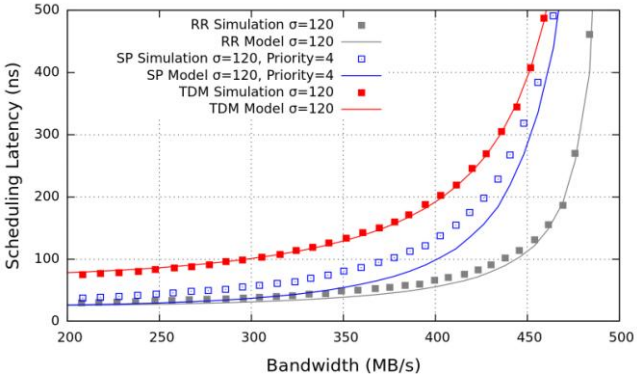

Figure 7. Average scheduling latency with standard deviation σ=120 ns, there are 4 requestors scheduled by TDM, RR, and SP arbiters. For SP, the latency of the queue with the lowest priority is reported.

We conclude this experiment by evaluating the accuracy of the proposed models compared to simulation results. The figures show that the proposed models provide reasonable accuracy compared to simulations with average deviations of 4.1% for TDM, 12.9% for SP, and 14.2% for RR. The reason the model for TDM is more accurate is because it is non-work-conserving, suggesting that this property implies a trade-off between average performance and accuracy of the model. Note the average deviation of for SP is based on all queues (with all priorities) i.e. from Figure 6, while Figure 7 only shows the requestor with the lowest priority, which has the least accurate results.

VI.3. **Results with Real Applications**

To evaluate the accuracy of the proposed analytical model, we run experiments by injecting requests according to traces from two multimedia applications, an H.263 decoder and JPEG decoder, respectively, taken from Mediabench benchmark suite [25]. For each application in the benchmark, a memory-trace file was generated by running it on a SimpleScalar 3.0 processor simulator [26]. The simulator was slightly modified to record the time and address of each L2 cache miss which results in a trace file containing all requests that go to the SDRAM. The traces are generated using the out-of-order execution engine (sim-outorder) with default settings except for the cache configuration. We use a unified 128KB L2 cache with 64 byte cache lines, 512 sets and an associativity of 4. We filtered out the L2 cache misses, and obtained a trace of the requests meant for the memory. The generated application traces assume a miss penalty of a single cycle, but are elastically replayed by the traffic generators to capture the actual latencies in the memory controller. The number of outstanding requests is set to 10, which means that after maximum 10 read requests without any returning responses, the traffic generator waits for at least one response, before sending the next request. However, this holds only for read requests since writes are posted and do not block the processor. There may hence be bursts of more than 10 write requests. The mean and standard deviation of request intervals are obtained from simulation and then used in the analytical model. The histograms of request inter-arrival times of these applications were previously shown in Figure 2.

Figure 8 shows analytical versus simulation results for the two applications for TDM, SP, and RR arbiters from left to right, respectively. For SP, the results of the requestor with the lowest priority, and hence the largest deviation from the simulated results, are shown. For each combination of arbiter and application, there is also a third bar on right of each set showing the analytical result obtained under the common assumption that traffic distributions in multi-processor embedded systems follow an exponential (Poisson) arrival distribution [6, 7, 9, 13].

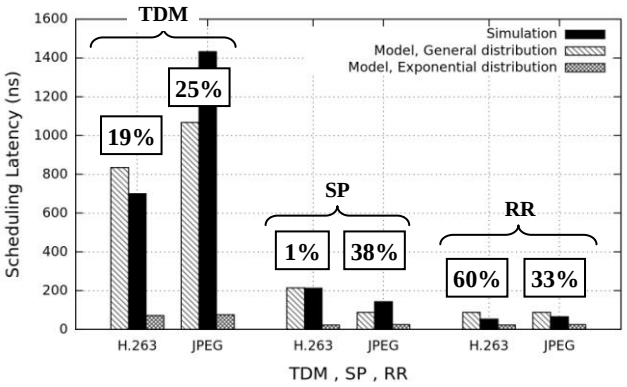

Figure 8: Experiment with an H.263 decoder and a JPEG decoder for TDM, SP, and RR arbiters

Similarly to the results with synthetic traffic, we observe that work-conserving arbitration, such as TDM, results in much higher average latency compared to non-work-conserving arbiters, such as RR and SP (even compared to the requestor with the lowest priority in SP). The errors of the proposed models compared to simulation are indicated beside the corresponding results in the figure. The errors are more significant with real traces compared to synthetic results, in particularly for the RR scheduler where an error of 60% is observed for the H.263 decoder. There are two main reasons for this inaccuracy. 1) The first reason that holds for all models presented in this paper is that limiting the number of outstanding requests has a similar effect as limiting buffering space in terms of controlling traffic injection. Both mechanisms impose a maximum burst length during which requests can be issued without any hardware restrictions. Then, traffic injection is stopped until the reception of a response (in case of limited number of outstanding requests) or until space is available in the destination buffer (in case of limited buffering

space). This implies a round-trip dependency between traffic consumption and traffic production that is not captured by the current state-of-the-art in queuing theory to the best of our knowledge, and is an important topic for future work to increase the applicability of queuing theory in the context of multi-core platforms. 2) The second reason that holds for work-conserving arbiters (e.g. SP and RR) is that the latency of each queue depends on arrival of traffic to other queues. This does not hold for non-work-conserving arbiters, such as TDM, where queues are independent. This is why the maximum error of the TDM model is smaller than the two others, just like in the case of synthetic traffic.

We conclude this experiment by discussing the effects of using the common assumption that applications in embedded systems follow an exponential arrival distribution. These results are derived by replacing the standard deviation in our equations with Poissonian characteristics, i.e. making it equal to the mean, which gives a coefficient of variation equal to one (C_A=1) in related equations. Figure 8 shows that models assuming exponential arrivals grossly underestimate the latency. The relative error under this assumption is 89% using TDM for the H.263 decoder and 94% for the JPEG decoder, respectively. The corresponding numbers for SP are 88% and 82%, respectively, and for RR 56% and 63%. The reason is that the considered applications have standard deviations that are much higher than their mean, 69 and 14.5 times higher for the H.263 decoder and JPEG decoder, respectively, which is ignored by a Poissonian traffic characterization. This clearly shows that using exponential distributions to model bursty memory traffic from applications executing through a cache to Poissonian traffic results in inaccurate results, highlighting the benefits of using queuing models based on more general distributions.

VII. CONCLUSIONS

This paper addresses the problem of average-case performance analysis in multi-core platforms with dynamic applications and resources shared by a diverse set of arbiters by proposing a general analysis framework based on queuing theory. The framework can be used with different arbiters and three models supporting general arrival and service processes are proposed for time-division-multiplexing, static-priority, and round-robin arbitration, respectively.

We experimentally evaluate the proposed framework by comparing the models to simulation results of a shared memory controller using both synthetic and traces from real applications. The results show that the models only have average deviations of 4.1% for TDM, 12.9% for SP, and 14.2% for RR with synthetic traffic and that although larger deviations are observed for realistic applications, they significantly outperform models assuming exponentially distributed traffic.

REFERENCES

[1] C. Van Berkel, "Multi-core for mobile phones," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 1260-1265.

[2] P. Kollig, *et al.*, "Heterogeneous multi-core platform for consumer multimedia applications," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, 2009, pp. 1254-1259.

[3] D. Melpignano, *et al.*, "Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1137-1142.

[4] S. Foroutan, *et al.*, "An Iterative Computational Technique for Performance Evaluation of Networks-on-Chip," *IEEE Transactions on Computers,* 2012.

[5] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis," in *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, 2007, pp. 1-6.

[6] U. Y. Ogras, *et al.*, "An analytical approach for network-on-chip performance analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 29, pp. 2001-2013, 2010.

[7] H. Sarbazi-Azad, *et al.*, "Analytical modeling of wormhole-routed k-ary n-cubes in the presence of hot-spot traffic," *IEEE Transactions on Computers,* pp. 623-634, 2001.

[8] H. Sarbazi-Azad, *et al.*, "Performance analysis of deterministic routing in wormhole k-ary n-cubes with virtual channels," *Journal of Interconnection Networks,* vol. 3, pp. 67-83, 2002.

[9] M. Ould-Khaoua, "A performance model for Duato's fully adaptive routing algorithm in k-ary n-cubes," *IEEE Transactions on Computers,* vol. 48, pp. 1297-1304, 2002.

[10] Z. Guz, *et al.*, "Network delays and link capacities in application-specific wormhole NoCs," *VLSI Design,* vol. 2007, 2007.

[11] W. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers,* pp. 775-785, 1990.

[12] A. E. Kiasari, *et al.*, "An Analytical Latency Model for Networks-on-Chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* pp. 1-11, 2011.

[13] S. Foroutan, *et al.*, "An analytical method for evaluating network-on-chip performance," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pp. 1629-1632.

[14] B. Akesson and K. Goossens, "Architectures and modeling of predictable memory controllers for improved system integration," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, 2011, pp. 1-6.

[15] G. Bolch, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*: Wiley-Blackwell, 2006.

[16] L. L. Peterson and B. S. Davie, *Computer networks: a systems approach*: Morgan Kaufmann, 2003.

[17] B. Akesson, *et al.*, "Real-time scheduling using credit-controlled static-priority arbitration," in *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on , vol., no., pp.3-14, 25-27 Aug.* , 2008.

[18] A. Willig, "A short introduction to queueing theory," *Technical University Berlin, Telecommunication Networks Group,* vol. 21, 1999.

[19] L. Kleinrock, "Queueing systems," ed: Wiley, New York, 1975.

[20] J. Kim and C. Das, "Hypercube communication delay with wormhole routing," *IEEE Transactions on Computers,* vol. 43, pp. 806-814, 2002.

[21] J. T. Draper and J. Ghosh, "A comprehensive analytical model for wormhole routing in multicomputer systems," *Journal of Parallel and Distributed Computing,* vol. 23, pp. 202-214, 1994.

[22] P. P. Pande, *et al.*, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Transactions on Computers,* vol. 54, pp. 1025-1040, 2005.

[23] E. Salminen, *et al.*, "On the credibility of load-latency measurement of network-on-chips," 2008, pp. 1-7.

[24] E. Salminen, *et al.*, "On network-on-chip comparison," *Euromicro DSD,* pp. 503–510, 2007.

[25] C. Lee, *et al.*, "MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems," *MICRO 30 Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture,* pp. 330-335, 1997.

[26] T. Austin, *et al.*, "SimpleScalar: An infrastructure for computer system modeling," *Computer,* vol. 35, pp. 59-67, 2002.