Memory Interference Delay Estimation for Multicore Systems

Rodolfo Pellizzoni*, Andreas Schranzhofer†, Jian-Jia Chen†, Marco Caccamo* and Lothar Thiele†
*University of Illinois at Urbana-Champaign, Urbana, IL, USA, {rpelliz2, mcaccamo}@cs.uiuc.edu
†Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, {schranzhofer, jchen, thiele}@tik.ee.ethz.ch

Abstract—**Employing COTS components in real-time embedded systems leads to timing challenges. When multiple CPU cores and DMA peripherals run simultaneously, contention for access to main memory can greatly increase a task's WCET. In this paper, we introduce an analysis methodology that computes upper bounds to task delay due to memory contention. First, an arrival curve is derived for each core representing the maximum memory traffic produced by all tasks executed on it. Arrival curves are then combined with a representation of the cache behavior for the task under analysis to generate a delay bound. Based on the computed delay, we show how tasks can be feasibly scheduled according to assigned time slots on each core.**

I. Introduction

Real-time embedded systems are increasingly using Commercial-Off-The-Shelf (COTS) components in an effort to raise performance and lower production costs. In particular, fast multicore CPUs and high-performance DMA peripherals are required to service demanding applications such as video processing that are becoming more and more popular in markets such as avionics. Unfortunately, COTS components are not designed with timing predictability in mind, which makes it challenging to integrate them in real-time systems. In particular, most COTS architectures feature a single-port main memory that is shared among all CPU cores and peripherals. When a task suffers a cache miss, contention for access to main memory can significantly delay cache line fetch and greatly increase the worst case execution time (WCET) of the task. We performed an experiment on a standard Intel dual core platform to understand the severity of this issue (details are provided in Section V). We engineered a task that continuously suffers cache misses and measured its WCET while running it in isolation. We then added to the experiment a second copy of the task running on the other core and a PCI-E [5] peripheral using DMA to saturate main memory with write requests, and measured a WCET increase of 2.96 times for the task.

Solutions to this problem have been presented in the literature. Several works (see [2], [9] for example) have proposed modifications to either memory arbitration or cache behavior to improve predictability, typically enforcing some TDMA scheme; however, such modifications are incompatible with COTS reuse. A different approach has been used in other works, such as [11], [6], [8], [10]: they focus on estimating the maximum delay that a task can suffer in a COTS system due to memory interference. In particular, in [6], [8] an analytical technique was developed to compute an upper delay bound given a representation of peripheral traffic. Unfortunately, such technique is only applicable to monoprocessor systems with a single peripheral bus. The analysis employed in [10] can compute bounds for multicore systems; however, it assumes that cache misses can occur anywhere in the task period. As we will show in Section IV, this can lead to overestimation.

In this paper, we introduce a novel WCET analysis framework that can compute memory delay bounds for systems comprising any number of cores and any number of peripheral buses sharing a single main memory. In particular, we provide two main contributions: **(1)** we introduce the key idea of computing a memory traffic arrival curve for each core, given a set of executed tasks. The arrival curve provides an upper bound to the amount of memory traffic generated by the core in any interval of time. **(2)** We describe an innovative algorithm that computes a delay bound for a task given traffic curves for all other cores and peripheral buses in the system. The algorithm is able to distinguish the behavior of DMA peripherals, whose traffic is buffered, from the behavior of CPU cores, which stall on cache misses.

II. System Model

We consider a COTS system comprised of multiples *processing cores* implemented on CPU dies. Each processing core PE_i can employ one or more cache levels, but caches are private and not shared with other cores. Cache misses in the last cache level generate requests to the shared main memory, which must arbitrate among simultaneous requests by different cores. We assume that a specification for the arbitration scheme is available, which is usually the case for COTS components used in embedded platforms (see [3] for example). Different types of interconnection between CPU die and the rest of the system are availables: alternatives include direct connection through a memory controller implemented on the CPU die, dedicated buses (Front Side Bus) implemented as a system chip (also known as *northbridge*) located on the motherboard, and switching architectures such as AMD HyperConnect. In general, we are not interested in covering each and every interconnection type: in almost the totality of systems, main memory is implemented as dynamic, single-port RAM, and it is the main data bottleneck. Hence, in this paper we focus our attention on the arbitration for access to main memory, ignoring the details of the system interconnection. We shall make vary general assumptions, namely, arbitration can follow either a Round-Robin (RR), First-Come-First-Serve (FCFS), or Fixed Priority (FP) scheme.

Processing cores execute periodic tasks. The set of tasks executed on each core is static and tasks are not allowed to migrate among cores (partitioned scheduling). We do not assume any synchronization among schedules: each core can run asynchronously with respect to other cores. Scheduling follows a *restrictive preemption model*: the control flow graph for each task τ_i is divided into a series of S_i sequential *superblocks* $\{s_{i,1}, \ldots, s_{i,S_i}\}$. Each superblock can include branches and loops, but superblocks must be executed in sequence. Multiple tasks executed on the same processing core are scheduled according to fixed time slots, with a given set of superblocks assigned to each slot. This model allows us to bound the effect of preemptions on cache content: a preempting task could eject τ_i's instructions and data from cache, thus increasing the number of cache misses suffered by τ_i. We initially assume that the set of superblocks assigned to each time slot is known; since preemption can only happen between time slots, we can determine the worst case number of cache misses in each superblock by simply assuming that the cache is invalidated before the start of each time slot. In Section IV-A we will show how our delay analysis can be employed to assign superblocks to slots.

Based on these assumptions, each task τ_i is characterized by a *cache profile* $c_i^{prof} = \{exec_{i,j}^L,\ exec_{i,j}^U, \mu_{i,j}^{\min}, \mu_{i,j}^{\max}\}$. $exec_{i,j}^L$ and $exec_{i,j}^U$ are lower and upper bounds on computation times for superblock $s_{i,j}$ assuming that memory operations take zero time, e.g. they are the times required to execute the instructions in the superblock. $\mu_{i,j}^{\min}, \mu_{i,j}^{\max}$ are the minimum and maximum number of access requests to main memory in superblock $s_{i,j}$. The way $\mu_{i,j}^{\min}, \mu_{i,j}^{\max}$ are computed depends on cache architecture. In a write-back cache, whenever a dirty cache line must be replaced, the cache controller generates two requests to main memory: a write for the replaced cache line, and a read for the fetched cache line. Both requests must be accounted for in $\mu_{i,j}^{\min}, \mu_{i,j}^{\max}$. However, many modern architectures employ a *write buffer* for the last cache level: in this case, write requests are not immediately executed, but rather temporary stored in the write buffer. The write buffer then takes care of writing the dirty cache line to main memory when there is time available, e.g. the write buffer produces requests with lower priority than cache fetches. We provide more details on write buffer modeling in Section IV-E.

In this paper, we do not detail how to derive the cache profile for a task. In [6], it is shown that maximum and minimum superblock time and number of memory requests can be obtained by either experimental measures or static analysis. Note that in both cases, multiple execution traces could be derived: for example, one trace could have shorter execution time but larger number of memory requests and another one longer execution time but less requests. To reduce all traces to a single cache profile, we consider the maximum execution time among all traces and the maximum number of memory requests among all traces, independently from each other (the same holds for minima). Note that our model implicitly assumes that if the CPU fetch unit is delayed Δ time units in a superblock, its computation time increases by at most Δ. Modern CPU architectures can exhibit timing anomalies, in the sense that it is possible to produce a trace where the worst case computation time is produced when a specific memory access results in a cache miss rather than a hit; this is because the state of the pipeline depends on the time required for each memory access. Therefore, we assume a CPU architecture where execution time and communication time can be decoupled [13]. If that is not possible, then the (pessimistic) bounds computed by timing analysis must capture all effects of timing anomalies. In particular, the worst case execution time for a trace must be computed considering the uncertainties in the pipeline state due the fact that each memory access can result in either a cache hit or miss. Taking this uncertainty into consideration ensures that given a bound $exec^U$ on the execution time for a trace, if any memory access suffers a delay Δ, then $exec^U + \Delta$ is a valid (pessimistic) bound on the computation time for the trace.

Finally, we assume that peripheral traffic can be injected into main memory. In a typical COTS system, peripherals are connected through a dedicated interconnection such as the Peripheral Component Interconnect (PCI) and related standards (PCI-X and PCI Express). We assume that each peripheral in the system is characterized by an upper arrival curve $\alpha^*(t)$: for each interval of time t, $\alpha^*(t)$ represent the maximum amount of time required by the peripheral in main memory to perform DMA operations. Note that before reaching main memory, peripheral requests are buffered in interconnection elements such as bridges (PCI/PCI-X) and switches (PCI Express): as such, all peripheral requests coming from the same interconnection must be aggregated into a single *buffered flow* $\alpha_i^*(t)$ representing the cumulative requests produced by a given interconnection on main memory. An analysis to aggregate peripheral traffic on the PCI bus is presented in [4].

Each processing core PE_i is characterized by a parameter C_i, which is the length in time (assumed to be fixed) needed to service a memory request. Furthermore, for each buffered flow and processing core / unbuffered flow we define an arbitration parameter L_i, which is the maximum length of an atomic operation in main memory. Typically, C_i is a integer multiple of L_i. For example, if the size of a cache line and associated fetch request is 128 bytes and memory arbitration is based on 32 bytes operations, then every memory request consists of 4 atomic operations.

Delay Analysis: In [7], [6], an analysis was introduced to provide a bound on the delay due to cache interference caused to a task by a single peripheral flow. In our model, more

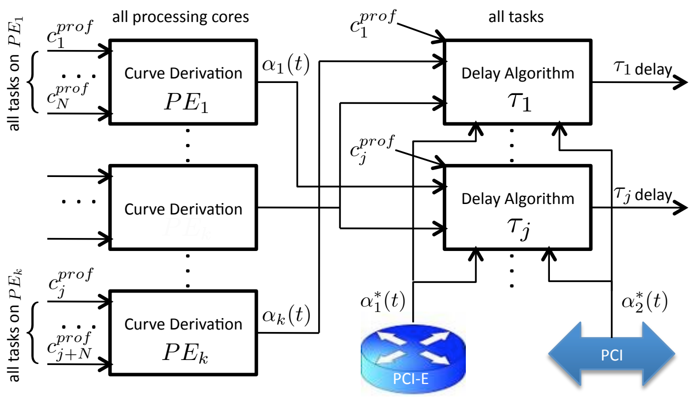

Fig. 1. Analysis Methodology.

Symbol	Description
PE_i	i-th processing core
C_i	max length of memory request in sec
L_i	max length of atomic operation in sec
τ_i	i-th task
p_i	period of i-th task
c_i^{prof}	cache profile for task τ_i
S_i	Number of superblocks of task τ_i
$s_{i,j}$	j-th superblock of task τ_i
$exec_{i,j}^U, exec_{i,j}^L$	upper and lower exec. time of $s_{i,j}$
$\mu_{i,j}^{\max}, \mu_{i,j}^{\min}$	max and min num. of memory requests
$\tilde{\mu}_{i,j}^{\max}, \tilde{\mu}_{i,j}^{\min}$	max and min num of replacements
$\alpha_i(t)$	unbuffered arrival curve for i-th flow
$\alpha_i^*(t)$	buffered arrival curve for i-th flow
$\bar{\alpha}_i(t)$	traffic delay curve for i-th flow
b_i	max backlog for i-th buffered flow
$*\tau_i'$	sequence set for two instances of τ_i
$t'_{m,d}$	superblock set $\{s_{i,m}, \dots s_{i,m+d}\}$
$\gamma^{\max}, \gamma^{\min}$	max and min accesses in time window
${\Delta^{\max}}^L, {\Delta^{\min}}^L$	length of time window with max, min accesses and lower execution time
$D_{j,k}$	max delay for superblocks $\{s_j, \dots, s_k\}$
$D_{j,k}^i$	max delay caused by i-th flow
$Ub_{j,k}$	computed upper bound on $D_{j,k}$
$Ub_{j,k}^i$	computed upper bound on $D_{j,k}^i$
$Ub_{j,k}^{(i)}$	upper bound on delay for $\{s_j, \dots, s_k\}$ caused by all flows **except** i-th flow
$u_k^{i,j}$	delay term for s_k, i-th flow computed based on $\{s_j, \dots, s_k\}$

Fig. 2. Paper Notation.

than two agents can contend for access to main memory. Furthermore, since processing cores are not strictly synchronized, while a task τ_i is running on core PE_j we do not know which tasks are running on the other cores. To solve the problem, our analysis follow two successive steps; in particular, we compute a delay bound for each task running the analysis once for every *task under analysis* on every processing core.

1) For each processing core PE_i, we derive an upper arrival curve to requests made by all tasks running on PE_i. In this way, each processing core is substituted by an *unbuffered flow* with arrival curve $\alpha_i(t)$: in any interval of time t, $\alpha_i(t)$ represents the maximum amount of time required by tasks running on PE_i to perform operations in main memory.
2) The analysis of Section IV computes an upper delay bound for the task under analysis given a set F of interfering flows, e.g. all peripheral buffered flows and the unbuffered flows for all processing cores except the one where the task under analysis is running.

A clarifying example is shown in Figure 1 for a system with two peripheral interconnections.

The rest of the paper is organized as follows. In Section III we show how to derive arrival curves for each unbuffered flow based on the cache profiles of the tasks running on the corresponding processing core. In Section IV we introduce our analysis and prove its correctness. In Section V we detail our experiments and simulation results. Finally, in Section VI we provide concluding remarks and future work. Notation used throughout the paper is summarized in Figure 2.

III. Computing Arrival Curves

A task τ_i causes interference to other tasks in the system by accessing main memory according to the pattern defined by c_i^{prof}. We specify this interference as an arrival curve [12] by considering the superblocks of task τ_i and their respective bounds on execution time and accesses to the shared resource. Tasks execute periodically on a processing element PE_i and therefore we can derive an arrival curve that represents a tasks behavior for any time window. We initially consider each task in isolation, assuming that no other system component accesses main memory; in Section III-B, we then show how to derive an arrival curve for multiple tasks executed on the same core. The derived arrival curves will be later used in Section IV to compute a delay bound for the task under analysis. Note that when computing the arrival curve for a core, we do not consider the interference caused by other flows on it. As shown in Section IV, this is because each atomic operation of the task under analysis can be delayed for its worst case amount while the interfering cores themselves suffer no delay.

Deriving arrival curves involves **(1)** computing all possible sequences of subsequent superblocks, **(2)** computing the feasible time windows for each sequence, **(3)** computing the minimal and maximal number of cache misses for each time window and **(4)** constructing the arrival curves accordingly.

A. Single Task per Processing Element

In this section we introduce our approach to represent a tasks accesses to a shared resource as arrival curve, assuming a single periodic task per processing element.

1) Computing sequences of super-blocks: Based on the parameters c_i^{prof} of task τ_i, we can derive time windows for which the minimal and maximal number of accesses to a shared resource are known. The time windows can be computed from the set of all possible sequences of subsequent superblocks, i.e., the *sequence set*. As an example, let $\tau_1 = \{s_{1,1}, s_{1,2}\}$ then the sequence set is $*\tau_1 = \{\{s_{1,1}\}, \{s_{1,2}\}, \{s_{1,1}, s_{1,2}\}\}$.

In order to account for the transition phase between succeeding periods of a task, we consider two subsequent instances of a task for the arrival curve derivation. Therefore we specify $\tau_i' = \{\tau_i\ \tau_i\}$, such that $\tau_i' = \{s_{i,1} \dots s_{i,S_i}, s_{i,1} \dots s_{i,S_i}\}$ and $*\tau_i'$ is the corresponding *sequence set*. Element $t'_{m,d} \in *\tau'$ is described by the offset m, representing the index of its first superblock from τ' and d, representing the number of superblocks considered, such that:

$$t'_{m,d} = \{s_{i,m}, \dots, s_{i,m+d}\}\ \forall d \in [0 \dots S_i{-}1], \forall m \in [1 \dots S_i]. \quad (1)$$

2) Computing time windows: Each element $t'_{m,d} \in *\tau_i'$ results in four different time windows, considering all combinations of minimal and maximal execution times and accesses to the shared resource respectively. Two of these time windows represent the worst case, i.e., they represent the maximal number of accesses for the shortest time windows. The time windows Δ and their corresponding number of accesses to the shared resource γ are represented as tuples $\hat{t} =<\gamma, \Delta>$ and we show how to compute them in Section III-A3. Accesses to the shared resource can happen at any time during a superblocks execution. In other words, for the first and last super-block in a sequence $t'_{m,d}$, the accesses to the shared resource happen at the end and at the beginning respectively. As a result, the first and last super-blocks' execution times $exec_{i,j}$ are not considered for the representative time windows but their accesses to the shared resource are considered.

Consider Fig. 3 for an example how to compute time windows. In the first example, denoted `1 super-block`, the time window computes as zero, meaning that accesses to the shared resource occur concurrently at one instant of time. For example `2 super-blocks`, the time window Δ computes as the time required to process the first super-blocks accesses, while the number of accesses γ computes as the sum of both superblocks' accesses. Execution times are not considered for the time window, since in the worst case the actual computation is performed before and after the first and second superblock respectively. The first super-blocks accesses to the shared resource need to be processed before the second superblock can be activated, specifying the time window. In other words, we arrange the accesses to the shared resource in subsequent superblocks such that the resulting time window is minimized, conclusively maximizing the interference onto other tasks.

Computing the time window for elements $t'_{m,d}$, whose superblock sequence spans over the period, needs to consider the gap g between the last superblock of a task and its period. Example `4 super-blocks` in Fig. 3 illustrates such a case. Minimizing the gap, and deductively the time window, is done by assuming the maximal execution time $exec_{i,j}^U$ and number of accesses $\mu_{i,j}^{max}$ for superblocks not included in $t'_{m,d}$.

$$\begin{aligned} g(e,r) &= p_i - \sum_{\forall s_{i,j} \in \tau_i \setminus (\tau_i \cap t'_{m,d})} exec_{i,j}^U + \mu_{i,j}^{max} \cdot C_i \quad (2) \\ &\quad - \sum_{\forall s_{i,j} \in \tau_i \cap t'_{m,d}} exec_{i,j}^e + \mu_{i,j}^r \cdot C_i, \end{aligned}$$

where e and r denote the actual values for execution time and accesses to the shared resource, e.g., $e = U$ and $r = min$.

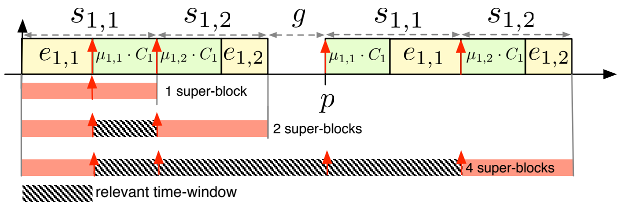

Fig. 3. Computing time windows for sequences of 1, 2 and 4 super-blocks including the gap between periods.

3) Computing the cache misses for each time window: Time windows Δ and the corresponding number of accesses to the shared resource γ for an element $t'_{m,d}$ are computed in Equations 3 and 4. Based on these values, the tuples for element $t'_{m,d} \in *\tau'$ are computed in Equations 5 to 6.

$$\gamma^{min} = \sum_{j=m}^{m+d} \mu_{i,j}^{min} \quad (3)$$

$$\Delta^{min^L} = \sum_{j=m+1}^{m+d-1} exec_{i,j}^L + \sum_{j=m}^{m+d-1} \mu_{i,j}^{min} \cdot C_i \quad (4)$$

$$\hat{t}_{m,d}^{min^L} =< \gamma^{min}\ ;\ \Delta^{min^L} + g(L, min) > \quad (5)$$

$$\hat{t}_{m,d}^{max^L} =< \gamma^{max}\ ;\ \Delta^{max^L} + g(L, max) > \quad (6)$$

Equation 5 can be transformed into the tuple computed with Equation 6 by simply increasing $\mu_{i,j}^{min}$ to $\mu_{i,j}^{max}$. In other words, they show a linear relation, since the time required to process an access is constant and intermediate tuples can be computed by linear approximation. For any number of accesses to the shared resource within the range of the tuples computed in Equations 5 and 6, we can therefore compute a safe upper bound to the number of accesses performed in the corresponding time window by linear approximation, see Fig. 4.

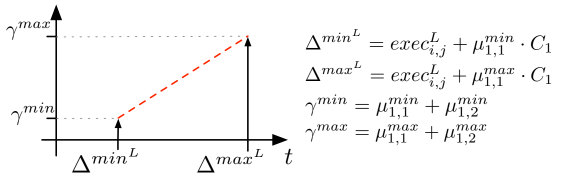

Fig. 4. Linear approximation between minimum and maximum number of accesses to the shared resource for a single super-block

4) Deriving arrival curves: Retrieving the minimal and maximal number of accesses to the shared resource for every time interval $\Delta = \{0 \dots 2p_i\}$ from the computed tuples and linear approximations allows to compute the arrival curve. Consider the function $\delta(\hat{t})$ to return the length of the time window and $\nu(\hat{t})$ to return the number of cache misses for each tuple, then the upper arrival curve $\tilde{\alpha}_i$ can be obtained as:

$$\tilde{\alpha}_i(\Delta) = \underset{\forall \hat{t}_{m,d};\delta(\hat{t}_{m,d})=\Delta}{\operatorname{argmax}} \nu(\hat{t}_{m,d}). \tag{7}$$

We construct the infinite curves $\hat{\alpha}_i$ as an initial aperiodic part, that is represented by $\widetilde{\alpha}_i$ and a periodic part which is repeated k-times for $k \in \mathbb{N}$.

$$\hat{\alpha}_i(\Delta) = \begin{cases} \tilde{\alpha}_i(\Delta) & 0 \leq \Delta \leq p \\ \max\left\{\widetilde{\alpha}_i(\Delta), \tilde{\alpha}_i(\Delta - p_i) + \sum\limits_{\forall j} (\mu_{i,j}^{max})\right\} & p_i \leq \Delta \leq 2p \\ \widetilde{\alpha}_i(\Delta - k \cdot p_i) + k \sum\limits_{\forall j} (\mu_{i,j}^{max}) & \text{otherwise} \end{cases} \tag{8}$$

The computational complexity to obtain the overall arrival curves is $O(S_i^2)$. Following the previous computation we derive Lemma 1.

Lemma 1: Deriving alpha curve $\hat{\alpha}_i(\Delta)$ by Equation 8 is the upper bound of accesses to a shared resource by task τ_i for any time window Δ.
Arrival curve $\alpha_i(t)$ represents the maximum amount of time a task τ_i requires to perform its accesses to the shared resource in a time window of length t and is obtained as $\alpha_i(t) = \hat{\alpha}_i(t) \cdot C_i$.

B. Multiple Tasks per Processing Element

In this section we show how to extend the previously shown approach to multiple tasks executing on each processing element. Consider a set of periodic tasks $T = \{\tau_1 \dots \tau_N\}$

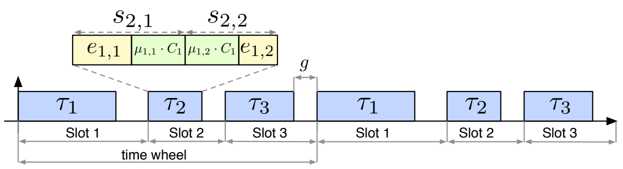

Fig. 5. Example of 3 periodic tasks executing in a static time wheel

scheduled statically, e.g., a static time slot assignment as in Fig. 5. Then we can compute a sequence of all superblocks that constitute tasks in T as:

$$\sigma = \{s_{1,1} \dots s_{1,S_1}, s_{2,1} \dots s_{2,S_2} \dots s_{N,1} \dots s_{N,S_N}\} \tag{9}$$

Based on σ the *sequence set* $*\sigma$ is derived, as shown in Section III-A1. We define $\sigma' = \{\sigma \ \sigma\}$ and compute the sequence set $*\sigma'$ such that $t'_{m,d} \in *\sigma'$, resulting in two periods of the static time wheel being considered.

Time windows are computed from the sequence set $*\sigma'$ following the concept presented for a single task per processing element. We compute the maximal and minimal time windows for any possible sequence of subsequent superblocks in $*\sigma'$ and count the corresponding minimal and maximal number of accesses to the shared resource respectively. Elements $t'_{m,d} \in *\sigma'$ contain superblocks from different statically scheduled tasks and therefore the gap between each tasks' last superblock and its period has to be considered. Similarly to the gap g for the single task approach, minimizing the gap results in the worst case.

Equation 2 can be rewritten to compute the gap for a sequence of superblocks, by maximizing all the superblocks that are not considered by element $t'_{m,d}$:

$$\begin{aligned} g(e, r) \quad = \quad & \sum_{\forall \tau_i \in t'_{m,d}} p_i - \sum_{\forall s_{i,j} \in \sigma \setminus (\sigma \cap t'_{m,d})} exec_{i,j}^{U} + \mu_{i,j}^{max} \cdot C_i \\ & - \sum_{\forall s_{i,j} \in \sigma \cap t'_{m,d}} exec_{i,j}^{e} + \mu_{i,j}^{r} \cdot C_i. \end{aligned} \tag{10}$$

The tuples can now be computed as shown in Equations 5 to 6 and based on them the arrival curve can be derived as shown in Sections III-A3 and III-A4

IV. Delay Analysis

The goal of the analysis is the derivation of worst case delay that the task under analysis can suffer due to memory contention given a set F of interfering flows. In this section, we assume that the task under analysis runs in isolation on its assigned processing core. In Section IV-A, we will cover how superblocks can be assigned to fixed timeslices. To simplify notation, we drop the subscript denoting the number of the task under analysis and use c^{prof} as its cache profile, $\{s_1, \dots, s_S\}$ as its superblocks and C, L as the parameters for its core. We also use $i \in F$ in place of $\alpha_i \in F$ or $\alpha_i^* \in F$. Let $D_{j,k}$ be the maximum delay suffered by the task in superblocks $\{s_j, \dots, s_k\}$; the overall task delay is equal to $D_{1,S}$. The analysis is based on the following main idea: we first compute an upper bound $Ub_{j,k}$ to the maximum delay $D_{j,k}$ for all $j, k : 1 \leq j \leq k \leq S$, meaning $D_{j,k} \leq Ub_{j,k}$. We then progressively decrease the bound by taking the intersection of multiple such constraints.

Since multiple flows contend with the task under analysis for access to main memory, we divide the delay contribution among all interfering flows. To be more precise, assume that an atomic memory operation by the task under analysis is requested at time t' and first serviced at time t''. Since we only consider work-conserving memory arbitration schemes, it follows that one or more interfering flows must be serviced in interval $[t', t'']$. Suppose that a flow α_i is serviced for Δ time units in $[t', t'']$; then we say that α_i has delayed the task under analysis Δ time units for that memory operation. Based on this definition, we use $D_{j,k}^i$ to denote the maximum total delay caused by flow α_i (or α_i^*) on all operations in superblocks $\{s_j, \dots, s_k\}$ and $Ub_{j,k}^i$ for its upper bound. We can then obtain $Ub_{j,k}$ as follows:

$$Ub_{j,k} = \sum_{i \in F} Ub_{j,k}^i. \tag{11}$$

Delay bound derivation depends on the memory arbitration scheme. In details, we assume that arbitration among the task under analysis and other unbuffered flows follows either a RR or FCFS policy, while arbitration among those and buffered flows follows either RR, FCFS or FP with buffered flows being assigned lowest priority[1]. Arbitration effects are captured by the following lemma.

Lemma 2: Under RR arbitration (or FP for unbuffered flows with lowest priority), each atomic memory operation of the task under analysis can be delayed by flow α_i (or α_i^*) for at most L_i. Under FCFS arbitration, an unbuffered flow can delay each atomic operation for at most C_i, while a buffered flow can delay it for at most b_i, where b_i is the maximum time required to service the backlog (buffered data) of the flow.

Proof: Let t' be the time at which an atomic operation of the task under analysis is requested, and t'' be the time at which it is first serviced. Under RR arbitration, the worst case interference is produced when an atomic operation of α_i (α_i^*) is serviced in $[t', t'']$, resulting in a delay of L_i time units. Note that no more than one atomic operation of α_i (α_i^*) can finish in $[t', t'']$, since the priority of the flow immediately becomes lower than the priority of the task under analysis upon finishing an atomic operation.

Consider FP arbitration, with buffered flow α_i^* having lower priority than the task under analysis. The worst case interference is caused when an atomic operation of α_i^* is first serviced at $t' - \epsilon$, with $\epsilon > 0$, resulting in a delay of $L_i - \epsilon$. Note that no operation of α_i^* can be first serviced in $[t', t'']$, since the flow has lower priority.

Now consider first-come-first-served arbitration for an unbuffered flow α_i. The worst case interference is produced when the task/tasks generating α_i perform a memory request, consisting of C_i/L_i atomic operations, before t' and all C_i/L_i operations are serviced in $[t', t'']$ in C_i time units. Note that since the task/tasks stall while waiting for the last atomic operation to complete, no more than C_i/L_i outstanding operations can be serviced in $[t', t'']$. Finally, for a buffered flow α_i^* the worst case interference is generated when the entire backlog is requested before t' and serviced in $[t', t'']$; since by definition b_i is an upper bound to the time required to service the backlog, the lemma follows. ■

Note that in a superblock s_j, the number of atomic memory operations performed by the task under analysis is at most $\mu_j^{\max}\frac{C}{L}$. We can capture the arbitration property expressed by Lemma 2 introducing a *blocking function*[2] B_j^i.

$$B_j^i \equiv \begin{cases} \mu_j^{\max}\frac{C}{L}L_i & \text{for RR and FP} \\ \mu_j^{\max}\frac{C}{L}C_i & \text{for FCFS, unbuffered flow} \\ \mu_j^{\max}\frac{C}{L}b_i & \text{for FCFS, buffered flow} \end{cases} \qquad (12)$$

[1]This is a common optimization in system controllers for embedded systems, see [3].

[2]Note that the blocking term is much larger for FCFS arbitration than RR. This shows that the fairness added by FCFS is counterproductive in the determination of worst-case guarantees.

We can then express a first upper delay bound as follows:

Lemma 3: Blocking Delay Bound: For each flow and superblocks $\{s_j, \ldots, s_k\}$:

$$D_{j,k}^i \leq \sum_{p=j}^{k} B_p^i \qquad (13)$$

Proof: Consider RR arbitration. According to Lemma 2, the maximum delay caused by flow α_i (α_i^*) on an atomic operation of the task under analysis is L_i. The maximum number of atomic operations in superblocks $\{s_j, \ldots, s_k\}$ is $\sum_{p=j}^{k}\mu_p^{\max}\frac{C}{L}$. Therefore, the maximum delay $D_{j,k}^i$ caused by α_i (α_i^*) in superblocks $\{s_j, \ldots, s_k\}$ can not be greater than $(\sum_{p=j}^{k}\mu_p^{\max}\frac{C}{L})L_i = \sum_{p=j}^{k} B_p^i$.

The proof for the other arbitration cases is similar. ■

The bound expressed by Lemma 3 is not tight, because each flow might not present enough traffic to cause maximum delay to the task under analysis. We can refine the bound by expressing a condition on the amount of service time required by each flow in superblocks $\{s_j, \ldots, s_k\}$. For simplicity, let $\Delta_{j,k}^{\max^U} \equiv \sum_{p=j}^{k}(exec_p^U + \mu_p^{\max}C)$, e.g. $\Delta_{j,k}^{\max^U}$ is the maximum time required to executed superblocks $\{s_j, \ldots, s_k\}$ with no flow interference.

Lemma 4: Consider a flow α_i, and let $Ub_{j,k}$ be an upper bound to the total delay suffered by the task under analysis in superblocks $\{s_j, \ldots, s_k\}$. Then:

$$D_{j,k}^i \leq \alpha_i\big(\Delta_{j,k}^{\max^U} - C + Ub_{j,k}\big) \qquad (14)$$

Proof: Let t_j be the start time of superblock s_j and t_{k+1} be the end time of superblock s_k, modified by contention for access to main memory. Furthermore, let t' be the time the first memory operation is requested and t'' be the time the last memory request is serviced in superblocks $\{s_j, \ldots, s_k\}$. Since $Ub_{j,k}$ is an upper bound to the delay suffered by the task in $\{s_j, \ldots, s_k\}$, it holds: $t_{k+1} - t_j \leq \Delta_{j,k}^{\max^U} + Ub_{j,k}$. Furthermore, note that $t' \geq t_j$ and $t'' \leq t_{k+1} - C$ since the last request takes C time to complete and must be finished before the end of s_k. It follows: $t'' - t' \leq \Delta_{j,k}^{\max^U} - C + Ub_{j,k}$. By definition, $D_{j,k}^i$ can not be greater than the total amount of traffic generated by α_i in interval $[t', t'']$, hence the lemma follows. ■

Lemma 4 holds for unbuffered flows because $\alpha_i(t'' - t')$ represents the maximum amount of service received in interval $[t', t'']$. However, the same assumption is not true for a buffered flow, since at time t' there can be additional buffered data, hence the total amount of service time required in $[t', t'']$ can be greater than $\alpha_i(t'' - t')$. In [8], it is shown that the problem can be solved by replacing each buffered flow arrival curve $\alpha_i^*(t)$ with a new arrival curve $\alpha_i(t) = \alpha_i^*(t) + b_i$, where b_i is the maximum time required to service the backlog as previously defined. Lemma 4, as well as the remaining theorems in this section, can then be applied to both unbuffered and buffered flows using arrival curve α_i. The computation of b_i is discussed in Section IV-D.

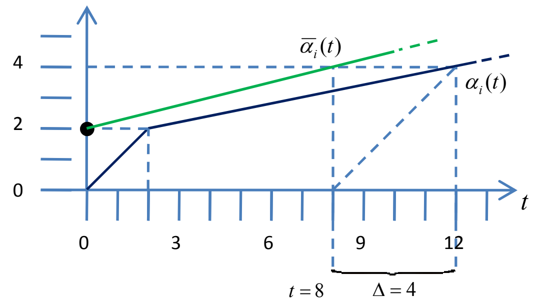

Fig. 6. Delay Curve $\bar{\alpha}_i$ derivation.

There is a remaining issue. Based on Equation 11, the $Ub_{j,k}$ term in Lemma 4 depends on the delay bound $Ub^i_{j,k}$ for flow α_i, but in turn we would like to compute $Ub^i_{j,k}$ based on the delay bound for $D^i_{j,k}$ provided by Lemma 4. To solve this mutual dependency problem, we introduce a new *traffic delay curve* $\bar{\alpha}_i(t)$:

$$\bar{\alpha}_i(t) \equiv \max\{\Delta | \Delta = \alpha_i(t+\Delta)\}. \tag{15}$$

A graphical representation of the traffic delay curve is shown in Figure 6 for a simple arrival curve α_i. Intuitively, $\bar{\alpha}_i(t)$ is the ordinate of the intersection of $\alpha_i(x)$ with the line $x - t$ (where t is fixed and x varies). We can then obtain $Ub^i_{j,k}$ according to the following lemma.

Lemma 5: Traffic Delay Bound: Consider flow α_i, and let $Ub^{(i)}_{j,k} \equiv \sum_{p \in F, p \neq i} Ub^p_{j,k}$ be an upper bound to the total delay in superblocks $\{s_j, \ldots, s_k\}$ caused to the task under analysis by all flows except flow α_i. Then:

$$Ub^i_{j,k} = \bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k}\big) \tag{16}$$

is a valid upper bound to $D^i_{j,k}$.

Proof: Let $Ub^i_{j,k}$ be an upper bound to $D^i_{j,k}$. Then since $Ub^{(i)}_{j,k} + Ub^i_{j,k}$ is a valid upper bound to $D_{j,k}$, from Lemma 4 it immediately follows that $\sup\{\Delta | \Delta \leq \alpha_i(\Delta^{\max^U}_{j,k} - C + Ub_{j,k} + \Delta)\}$ is a valid value for $Ub^i_{j,k}$.

To prove the lemma it is then sufficient to show that $\forall t, \sup\{\Delta|\Delta = \alpha_i(t+\Delta)\} = \max\{\Delta|\Delta = \alpha_i(t+\Delta)\}$, which is trivially true if the maximum exists for all t. Note that this is the case because of the following properties for any valid arrival curve $\alpha_i(t)$: 1) $\lim_{t\to\infty} \alpha_i(t)/t < 1$ (otherwise the generating task/tasks would not execute any instruction, or the generating peripheral would continuously occupy the bus); 2) $\alpha_i(t)$ is non-decreasing; 3) $\alpha_i(t)$ is right-continous and has a finite number of discontinuities in any finite time interval. ■

A better bound $Ub_{j,k}$ can be obtained by combining Lemmas 3, 5. In particular, the following theorem trivially holds.

Theorem 6: $Ub^i_{j,k}$ is a valid upper bound to $D^i_{j,k}$, where:

$$Ub^i_{j,k} = \min\Big(\sum_{p=j}^{k} B^i_p, \bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k}\big)\Big) \tag{17}$$

While the $Ub_{j,k}$ bound computed according Theorem 6 is a valid upper bound, it is fairly pessimistic. Note that since $D_{j,k}$ is the maximum delay for superblocks $\{s_j, \ldots, s_k\}$, it must hold: $D_{j,k} \leq D_{j,q} + D_{q+1,k}$ for all $q : j \leq q < k$. We can therefore refine the bound on $D_{j,k}$ by considering the minimum between $Ub_{j,k}$ and $Ub_{j,q} + Ub_{q+1,k}$. As an example, consider a task with three superblocks, $exec^U_1 = 11, exec^U_2 = 29, exec^U_3 = 11, L = C = 1$, and a single flow with $\bar{\alpha}_1(t) = \frac{2}{5}t, B^1_1 = 9, B^1_2 = 2, B^1_3 = 9$; superblocks 1 and 3 are short and have many cache misses (example: cold misses due to calling new functions) while superblock 2 is longer and experiences few cache misses (example: a cycle). Then by computing $Ub_{1,1} + Ub_{2,2} + Ub_{3,3}$ we obtain a delay bound of $\bar{\alpha}_1(10) + B^1_2 + \bar{\alpha}_1(10) = 10$, while computing $Ub_{1,3}$ yields a bound of $B^1_1 + B^1_2 + B^1_3 = \bar{\alpha}_1(11+29+11-1) = 20$.

The example shows that to obtain a better bound on $D_{j,k}$, multiple subintervals might need to be analyzed, but the number of possible sets of subintervals is exponential in $k-j$. Luckily, as it was shown in [6] for a single flow α_i, there is no need to analyze an exponential number of subinterval: rather, it is possible to use an algorithm that only analyzes a quadratic number of subintervals. The main idea is to compute a delay term $u^{i,j}_k$ for each superblock s_k by iteratively checking all superblocks in $\{s_j, \ldots, s_k\}$. The algorithm iterates over j, k, obtaining at each iteration a bound $Ub^i_{j,k} = \sum_{p=j}^{k} u^{i,j}_p$, which uses the newly computed $u^{i,j}_k$ term. More in details, $u^{i,j}_k$ is computed based on the blocking term B^i_k, the delay terms $u^{i,j}_j, \ldots, u^{i,j}_{k-1}$ and the traffic delay bound of Lemma 5 for each subinterval $\{s_q, \ldots, s_k\}$ with $j \leq q \leq k$. By computing a delay term $u^{i,j}_k$ for each interfering flow i, the described main idea allows us to produce a tighter bound than Lemmas 3, 5. However, handling multiple flows has an added complexity: when at iteration j, k we compute the traffic delay bound for flow α_i in any subinterval $\{s_q, \ldots, s_k\}$ according to Lemma 5, we must know the maximum delay $Ub^{(i)}_{q,k}$ caused by other flows in that subinterval. The problem can be solved using a dynamic programming approach: instead of iterating over j, k, we first compute delay bounds $Ub_{j,j} = \sum_{i \in F} Ub^i_{j,j}$ for all $j : 1 \leq j \leq S$, then we compute a delay bound $Ub_{j,j+1}$ for all $j : 1 \leq j < S$, then $Ub_{j,j+2}$ and so on and so forth. As shown in Algorithm 1, this is done by iterating over variables d, j, obtaining at each step delay bounds $Ub^i_{j,j+d}$, with $k = j + d$.

Delay term $u^{i,j}_k$ is computed in Equation 18 as the minimum of three delay terms: **(1)** term B^i_k for the blocking delay bound of Lemma 3; **(2)** the minimum over all subinterval $\{s_q, \ldots, s_k\}$, with $j+1 \leq q \leq k$, for the traffic delay bound of Lemma 5 (the case of $q = j$ is covered in the third term). Assume that this term becomes minimum among all three terms for a specific choice of q. Then Equation 18 can be rewritten as:

$$\sum_{p=q}^{k-1} u^{i,j}_p + u^{i,j}_k = \sum_{p=q}^{k} u^{i,j}_p = \bar{\alpha}_i(\Delta^{\max^U}_{q,k} - C + Ub^{(i)}_{q,k}), \tag{19}$$

Algorithm 1 Compute $\{Ub_{j,k}\}$

```
1: $\forall i \in F, \forall j, 1 \leq j \leq S : Ub^i_{j,j-1} := 0$
2: $\forall i \in F, \forall j, 1 \leq j \leq S : Ub^{(i)}_{j,j-1} := 0$
3: for $d = 0 \ldots S-1$ do
4:     for $j = 1 \ldots S-c$ do
5:         $k := j + d$
6:         solve the following system of equations $\forall i \in F$:
               $u^{i,j}_k = \min\Big(B^i_k,$   (18)
               $\min_{q:j+1 \leq q \leq k} \big\{\bar{\alpha}_i\big(\Delta^{\max^U}_{q,k} - C + Ub^{(i)}_{q,k}\big) - \sum_{p=q}^{k-1} u^{i,j}_p\big\},$
               $\bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k\big) - \sum_{p=j}^{k-1} u^{i,j}_p\Big)$
7:         $u^j_k := \sum_{i \in F} u^{i,j}_k$
8:         $\forall i \in F : Ub^i_{j,k} := Ub^i_{j,k-1} + u^{i,j}_k$
9:         $\forall i \in F : Ub^{(i)}_{j,k} := Ub^{(i)}_{j,k-1} + u^j_k - u^{i,j}_k$
10:        $Ub_{j,k} := \sum_{i \in F} Ub^i_{j,k}$
11:    end for
12: end for
13: return $\{Ub_{j,k}\}$
```

where following Lemma 5, the rightmost part is an upper bound to $D^i_{q,k}$. Note that since $q \geq j+1$, it holds $k - q < d$, thus the $Ub^{(i)}_{q,k}$ values have already been computed by the algorithm. Also note that for $d = 0$, $j = k$ and there is no valid value for q, so the term is ignored altogether. **(3)** The traffic delay bound for superblocks $\{s_j, \ldots, s_k\}$. This is a special case of the second term for $q = j$: in the equation, $Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k = Ub^{(i)}_{j,k}$, but we can not directly use $Ub^{(i)}_{j,k}$ because the $u^{p,j}_k$ values need to be computed together with $u^{i,j}_k$. Hence, we actually need to solve a system of equations computing values $u^{i,j}_k$ for all $i \in F$ simultaneously.

Finally, Lines 7-10 are used to compute all delay bounds in $O(\sharp F)$ at each step, where $\sharp F$ is the number of interfering flows. Note that the definitions in Lines 1-2 are required to make sure that the value of $Ub^i_{j,k-1}, Ub^{(i)}_{j,k-1}$ in Equation 18 and Lines 8-9 are zero when $d = 0$. Since updating all delay variables takes linear time in the number of interfering flows, the algorithm complexity is dominated by the complexity of solving the system of Equation 18, which must be done $O(S^2)$ times. A discussion of how the system can be solved and its complexity is provided in Section IV-B. We can now show that Algorithm 1 computes a valid upper bound $Ub_{j,k}$ to the delay $D_{j,k}$ in any superblock interval $\{s_j, \ldots, s_k\}$.

Theorem 7: Assume that the system of Equations 18 always admits solution. Then for each superblock interval $\{s_j, \ldots, s_k\}$, Algorithm 1 computes a valid upper bound $Ub_{j,k}$ to $D_{j,k}$.

Proof sketch.
We prove the theorem by induction on d. In particular, we show that $\forall i, \forall j, 1 \leq j \leq S - d : Ub^i_{j,j+d}$ is a valid upper bound to $D^i_{j,j+d}$, from which it follows that $Ub_{j,j+d} = \sum_{p \in F} Ub^p_{j,j+d}$ is an upper bound to $D_{j,j+d}$. The induction step is split into three cases, based on which term in Equation 18 is minimal. The correctness of term (1) is proven based on Lemma 3 and the correctness of terms (2) and (3) is based on Lemma 5. The complete proof is reported in Appendix. □

A. Multitasking

The analysis of Section IV can be easily extended to a multitasking scenario where tasks are assigned to fixed timeslices. In [7], an algorithm is introduced to compute the minimum number of time slots of fixed length T that must be assigned to the task under analysis. The algorithm works by iterating over superblock $s_k, 1 \leq k \leq S$: at each step, the algorithm tries to fit s_k in the current time slot. If there is not enough time left, s_k is assigned to a new time slot and the cache profile for s_k and subsequent superblocks is modified assuming that the cache is invalid at the start of s_k. The algorithm can be reused in our model by simply substituting the delay analysis used in [7] with our new delay analysis for multiple flows.

B. Solving the Delay System

In this section, we detail how to solve the system of Equation 18. Due to the third term in Equation 18, each $u^{i,j}_k$ value depends on all other $u^{p,j}_k$ values, which must thus be computed at the same step. We can obtain a solution for all $u^{i,j}_k$ terms using a recurrence: the idea is to start from a vector of values $\vec{u}^j_k(0) = (u^{1,j}_k(0), \ldots, u^{i,j}_k(0), \ldots)$, where $\forall i \in F, u^{i,j}_k(0) \geq u^{i,j}_k$. At each step of the recurrence we then compute a new vector $\vec{u}^j_k(c+1) = (u^{1,j}_k(c+1), \ldots, u^{i,j}_k(c+1), \ldots)$ based on the previous vector $\vec{u}^j_k(c) = (u^{1,j}_k(c), \ldots, u^{i,j}_k(c), \ldots)$ and we show that the series converges to a fixed point that is equal to $\vec{u}^j_k = (u^{1,j}_k, \ldots, u^{i,j}_k, \ldots)$, e.g. it is the only solution to Equation 18.

The initial element of the series is obtained as follows:

$$u^{i,j}_k(0) = \min\Big(B^i_k, \min_{q:j+1 \leq q \leq k} \big\{\bar{\alpha}_i\big(\Delta^{\max^U}_{q,k} - C + Ub^{(i)}_{q,k}\big) - \sum_{p=q}^{k-1} u^{i,j}_p\big\}\Big); \tag{20}$$

intuitively, we compute the minimum of terms (1) and (2) in Equation 18. Each successive element of the series is computed based on term (3):

$$u^{i,j}_k(c+1) = \min\Big(u^{i,j}_k(c), \bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k(c)\big) - \sum_{p=j}^{k-1} u^{i,j}_p\Big). \tag{21}$$

Note that the $Ub_{j,k-1}^{(i)}$ and $u_p^{i,j}$ values are not part of the recurrence because they have already been computed in Algorithm 1 at a previous step. The following theorem shows that the iteration is correct.

Theorem 8: The series $\vec{u}_k^j(c)$ defined by Equations 20, 21 converges to $\vec{u}_k^j = (u_k^{1,j}, \ldots, u_k^{i,j}, \ldots)$, which is the unique solution to Equation 18.

Proof sketch.
In what follows, given any two delay vectors $\vec{u}_k^{\prime j} = (u_k^{\prime 1,j}, \ldots, u_k^{\prime i,j}, \ldots), \vec{u}_k^{\prime\prime j} = (u_k^{\prime\prime 1,j}, \ldots, u_k^{\prime\prime i,j}, \ldots)$, we shall write $\vec{u}_k^{\prime j} \geq \vec{u}_k^{\prime\prime j}$ iff $\forall i \in F, u_k^{\prime i,j} \geq u_k^{\prime i,j}$ (the other comparison operators are similarly defined, in particular, $\vec{u}_k^{\prime j} > \vec{u}_k^{\prime\prime j}$ iff $\vec{u}_k^{\prime j} \geq \vec{u}_k^{\prime\prime j} \wedge \vec{u}_k^{\prime j} \neq \vec{u}_k^{\prime\prime j}$). In Appendix we obtain four results that are used to derive the theorem: **(1)** $\forall c \geq 0, \vec{u}_k^j(c) \geq \vec{u}_k^j(c+1)$ (Lemma 9); **(2)** the $\vec{u}_k^j(c)$ series can not diverge for $c \to \infty$ (Lemma 10). Results **(1)** and **(2)** imply that the series must converge; let $\vec{u}_k^j$ be its fixed point. We then show that: **(3)** if the series converges to a fixed point $\vec{u}_k^j$, then $\vec{u}_k^j$ satisfies Equation 18 (Lemma 11). This implies that Equation 18 admits at least one solution. Finally, we conclude the proof by showing that: **(4)** the system of Equation 18 can not admit more than one solution (Lemma 12). □

A final note is relative to computational complexity. Let K be the maximum number of iterations required by the series for convergence. Computing the value of any traffic delay curve $\bar{\alpha}_i(t)$ for a specific t has a maximum complexity of $O(S^2)$, the same as computing the curve. Computing the initial vector $(u_k^{1,j}(0), \ldots, u_k^{i,j}(0), \ldots)$ according to Equation 20 takes $O(\sharp F S^3)$; the traffic delay curve must be computed at most $O(\sharp F S)$ times, while $\Delta_{q,k}^{\max^U}, \sum_{p=q}^{k-1} u_p^{i,j}$ can be computed in constant time for each i, q by starting with $q = k$ and accumulating the values. Similarly, the whole iteration according to Equation 21 can be computed in $O(\sharp F K S^2)$; $\Delta_{j,k}^{\max^U}$ and $\sum_{p=j}^{k-1} u_p^{i,j}$ can be obtained from the corresponding terms computed in Equation 20 in constant time, and $\sum_{p \in F, p \neq i} u_k^{p,j}(c)$ can be computed in $O(\sharp F)$ at each step of the iteration using the same strategy as in Lines 7-10 of Algorithm 1. Since Algorithm 1 must solve Equation 18 $O(S^2)$ times, its overall computation complexity is $O\big(\sharp F S^4 \max(S, K)\big)$. Note that for general traffic delay curves, the series might converge in an infinite number of steps. In practice, as we show in Section V, the iteration typically converges quickly in a limited number of steps. Furthermore, following the proof of Theorem 8, it can be shown that $\forall c, i : u_k^{i,j}(c) \geq u_k^{i,j}$. Therefore, it is possible to halt the recurrence after a fixed number of steps and still obtain a valid upper delay bound.

C. Bound Tightness

It remains to discuss whether the $\{Ub_{j,k}\}$ bounds returned by Algorithm 1 are tight or not, e.g. if $Ub_{j,k} = D_{j,k}$ holds for all possible cache profiles and flows. For a system with a single flow α_i, in [6] we show that the obtained $Ub_{j,k}^i$ bounds are tight if $\alpha_i(t)$ is a concave function. Unfortunately, in a system with multiple flows, the bounds are not tight anymore. This is because each $Ub_{j,k}^i$ value is maximized independently of the delay bound $Ub_{q,k}^{(i)}$ for all other flows that is used in the second term of Equation 18. Assume that $u_k^{i,j}$ is equal to the second term of Equation 18 for a specific value of q. Then $Ub_{j,k}^i$ is computed based on the assumption that traffic from all other flows cause maximum interference in subinterval $\{s_q, \ldots, s_k\}$. However, in the pattern of cache interference that results in the actual worst case delay $D_{j,k}$, the interference of flows other than α_i on $\{s_q, \ldots, s_k\}$ could be less than $Ub_{q,k}^{(i)}$, in particular because they could cause more interference on $\{s_j, \ldots, s_{q-1}\}$. Therefore, the computed $Ub_{j,k}^i$ would be larger than the real worse case.

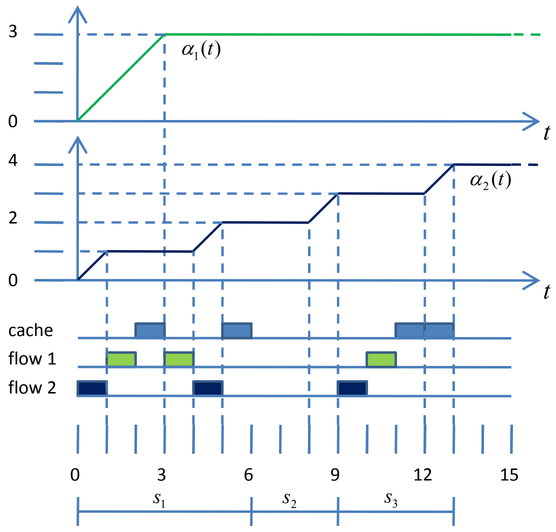

Fig. 7. Example of non-tight bound.

A clarifying example is shown in Figure 7 for a system with two flows α_1, α_2 and RR arbitration with $L_i = C_i = 1$. The task under analysis has three superblocks $\{s_1, s_2, s_3\}$ with $exec_1^U = exec_3^U = 2$, $\mu_1^{\max} = \mu_3^{\max} = 2$ and $exec_2^U = 3, mu_2^{\max} = 0$. Figure 7 shows the pattern of memory requests that results in the maximum delay $D_{1,3} = 6$ for the task under analysis. Note that in the figure, flow α_2 can not delay the task under analysis in $[12, 13]$ because $\alpha_2(13-9) = 1$, e.g. the flow does not have enough traffic to cause a delay of 2 time units in s_3. However, running Algorithm 1 returns a bound $Ub_{1,3} = 7$. This is because when computing $u_k^{2,j} = u_3^{2,1}$ in Equation 18 at step $j = 1, k = 3$, for $q = 3$ the algorithm uses a value $Ub_{q,k}^{(2)} = Ub_{3,3}^1 = 2$, while in Figure 7 flow α_1 delays the task under analysis for one time unit ($Ub_{3,3}^1$ is a bound on the delay for superblock s_3 only, hence flow α_1 has enough traffic to delay both memory requests of the task under analysis). As a consequence, in the algorithm superblock s_3 is delayed more than in the real worst case, allowing an additional request of

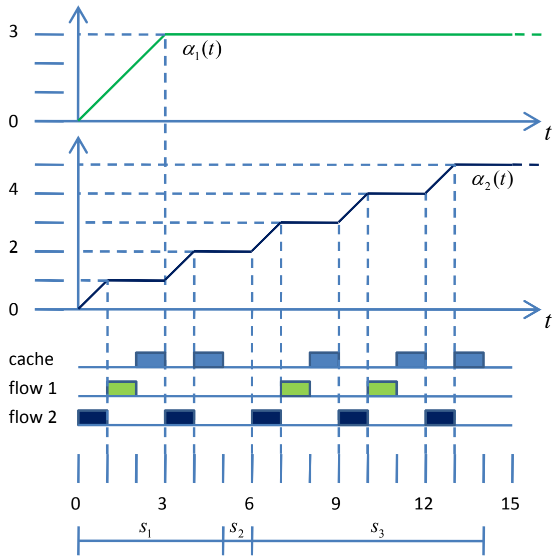

Fig. 8. Example complex delay pattern.

flow α_2 to delay the task under analysis.

Whether it is possible to obtain tight bounds, at least for concave arrival curves, in polynomial time in the number of superblocks of the task under analysis is left as an open question. Consider the scenario in Figure 8, where we use the same α_1 function as in Figure 7 but different α_2 and cache profile, with $exec_1^U = 0, \mu_1^{\max} = 2$, $exec_2^U = 1, \mu_2^{\max} = 0$, $exec_3^U = 0, \mu_3^{\max} = 3$. It is easy to see that to obtain the maximum delay $D_{1,3}$, flow α_1 must now delay superblock s_1 for one and s_3 for two time units; otherwise, flow α_2 could not generate the required three units of traffic in s_3. This example shows that it can be difficult to predict which subinterval should suffer maximum interference for each flow. In fact, we strongly suspect that if a tight bound can be obtained, then an exponential number of superblock intervals would need to be examined. We plan to study this case in more details as part of our future work.

D. Backlog computation for buffered flows

An upper bound on the backlog b_i for each buffered flow α_i^* can be easily obtained using the theory of Network Calculus [1]. Let β_i be a strict service curve for α_i^*, e.g. in any interval of length t in which the flow is backlogged, $\beta_i(t)$ is a lower bound to the amount of service provided by main memory to the flow. Then:

$$b_i \leq \sup_{t \geq 0}\{\alpha_i^*(t) - \beta_i(t)\}. \tag{22}$$

It remains to compute β_i. Given our assumptions, it can be shown that β_i is minimized when flow α_i^* has lowest static priority. Let $\alpha(t)$ be the arrival curve for the processing core where the task under analysis is executed. It follows:

$$\beta(t) \geq t - \sum_{\alpha_i \in F} \alpha_i(t) - \sum_{\alpha_i^* \in F} \alpha_i^*(t) - \alpha(t). \tag{23}$$

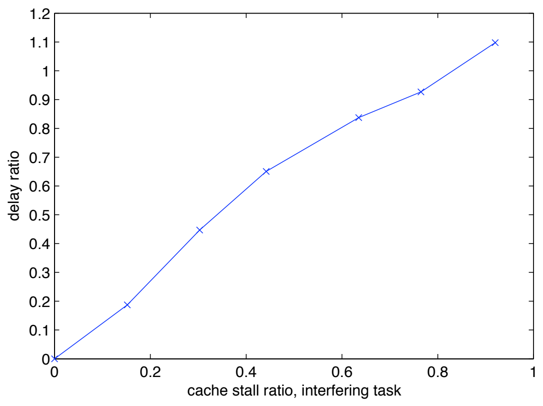

Fig. 9. Measured computation time increase with two cores.

E. Handling Write Buffers

Write buffers can be included in the analysis by modeling them as additional buffered flows. Since replacements of dirty cache lines do not generate immediate write-back, they should not be accounted for in $\mu_{i,j}^{\max}, \mu_{i,j}^{\min}$. Instead, we define new values $\tilde{\mu}_{i,j}^{\max}, \tilde{\mu}_{i,j}^{\min}$ to be the maximum and minimum number of replaced cache lines in superblock $s_{i,j}$ or equivalently, the number of cache lines pushed to the write buffer. A buffered arrival curve $\tilde{\alpha}_i^*$ can then be computed for the write buffer of processing core PE_i using the methodology described in Section III with the $\tilde{\mu}_{i,j}^{\max}, \tilde{\mu}_{i,j}^{\min}$ values. The analysis of Section IV is applied including buffered flows for all write buffers and assigning them lowest static priority. Note that the analysis must include a flow for the write buffer of the processing core where the task under analysis is executed, since it can interfere with cache fetches; however, since we know that the task under analysis is running, the arrival curve for the write buffer can be obtained assuming that no other task runs on the processing core.

V. Experimental Results and Simulations

We performed experiments on an Intel Core microarchitecture platform to understand the severity of the memory interference problem. Using a PC COTS platform allowed us to access PCI-E slots and easily measure task performance using Intel performance counters; however, to obtain meaningful measures we slowed down the Front Side Bus obtaining a speed of 1Ghz for each core and a theoretical bandwidth of 2.4GB/s, which is in line with typical speeds for embedded systems and closely mirrors the parameters used in [6].

We selected a CoreQuad CPU implementing four cores on two joined CPU dies. The two cores on each die share level 2 cache; hence, to mirror the model described in the paper we disabled cores PE_2 and PE_4 and used only one core on each CPU die. We engineered a software task that continuously suffers cache misses in order to measure an upper bound to the delay suffered due to memory interference. The task allocates a buffer with size double its level 2 cache, and then cyclically reads one word at a time from each cache line. This forces the task to suffer a cache miss for every read operation; we measured both the total duration of each task instance and the number of cache misses, and found a cache stall time (defined as the ratio $\frac{\mu^{\max}_{i,j} C}{exec^U_{i,j} + \mu^{\max}_{i,j} C}$, e.g. the percentage of time spent by the time stalling with no external interference) of 92%. We then modified the task by inserting a variable number of *nop* instructions between successive reads. In this way, we obtained versions of the task with stall time of around 15%, 30%, 45%, 60% and 75%. We ran one copy of the task with 92% stall time on core PE_1 and one copy of the task with variable stall time on PE_3, and measured the increase in computation time.

Results are reported in Figure 9 as the ratio between the maximum measured delay and the computation time of the task when run in isolation, obtained over 4 runs. Note that the delay ratio for the first task increases almost linearly with the cache stall time of the second task, peaking at a 1.1 when both tasks saturate memory with cache fetch requests. We believe that the increase in computation time is over two times due to additional arbitration overhead; in particular, our platform uses DDRAM for main memory, whose response time is highly dependent on data location. To build a safe upper bound, a safe estimate for the time C required to service each memory request must be used, but in practice, when the task is run in isolation each subsequent cache fetch can often be completed in less time, leading to a decreased computation time. In this sense, it is important to note that the results reported in Figure 9 are average case delays, since exactly synchronizing the tasks to obtain the worst case interference as described in Lemma 2 is impossible. Finally, we repeated the experiment after adding to our testbed a custom-designed PCI-E peripheral that continuously sends write requests to main memory. Our developed 8-lanes PCI-E peripheral has a theoretical throughput of 2GB/s and can therefore almost saturate the available memory bandwidth. When running both tasks at 92% stall time, we measured a delay ratio of 1.96, e.g. the computation time of the measured task increased 2.96 times.

Unfortunately, the obtained experimental results can not be directly compared to our analysis bounds because we do not know the exact L_i values nor the arbitration scheme for the employed platform. This is typical in the PC market, where vendors are wary of revealing full implementation details. However, as we discussed in Section II such information is usually available for embedded system platforms.

We also performed extensive simulations to understand how the delay bound varies as a function of task parameters and how fast the delay iteration of Section IV-B converges. In particular, we decided to simulate a N-core system with RR arbitration, $C = L$ and one task for each core with $S = 10$ superblocks. Tasks are synthetically generated according to three parameters σ, β and α. For each task and superblock,

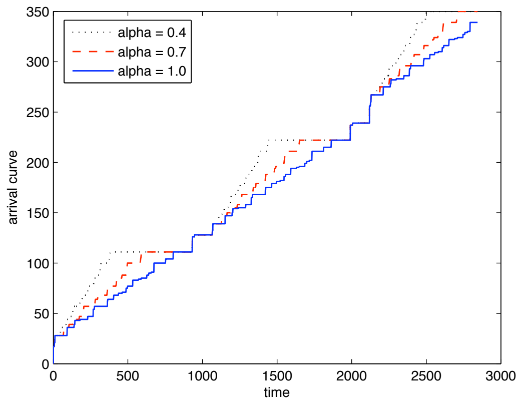

Fig. 10. Computed arrival curves for different α values.

we first generate $exec^U_{i,j}$ according to a uniform distribution with mean $100C$ and coefficient of variation σ. We then generate a cache stall ratio for the superblock according to a uniform distribution with mean β and coefficient of variation σ and compute $\mu^{\max}_{i,j}$ accordingly. Finally, we set $exec^L_{i,j} = \alpha\, exec^U_{i,j}, \mu^{\min}_{i,j} = \alpha\, \mu^{\max}_{i,j}$ and we set the period to $p_i = \Delta^{\max^U}_{1,S_i}$.

Figure 10 shows computed $\alpha_i(t)$ functions for a synthetic task τ_i with $\sigma = 0.2, \beta = 0.1$ over the interval $[0, 3p_i]$. Three different $\alpha_i(t)$ functions are obtained by setting α equal to $0.4, 0.7$ and 1.0 for the task. Note that for smaller values of α, the arrival curve becomes larger. This is because the time window computation in Section III assumes that the maximum amount of memory requests can happen together with the lowest execution time for each superblock; hence, the same amount of requests can arrive in a smaller window since $exec^L_{i,j}$ decreases with α. However, all arrival curves eventually assume the same value at the end of each period since the time window is constrained by the gap g between the last superblock and the next period.

Figure 11 shows simulation results in terms of the delay ratio between the computed upper delay bound Ub_{1,S_i} and the WCET $\Delta^{\max^U}_{1,S_i}$ for the task executed on the first core in a system with $N = 4$ cores. In the figure, we fix $\alpha = 0.8, \sigma = 0.2$ and vary the cache stall parameter β in $[0, 0.4]$ for the task under analysis and in $[0, 0.2]$ for the other three interfering tasks. Each point in the graph is computed as the average over 100 runs; each run, which involves generating the tasks, computing arrival curves and applying Algorithm 1 took less than a second on a modern PC.

Note that the delay ratio increases almost linearly with the stall ratio for the interfering tasks, until it saturates at roughly three times the stall ratio for the task under analysis (for example, for $\beta = 0.4$ the graph saturates at a value slightly higher than 1.2). This is expected: at a certain point, the memory traffic generated by each interfering core becomes so

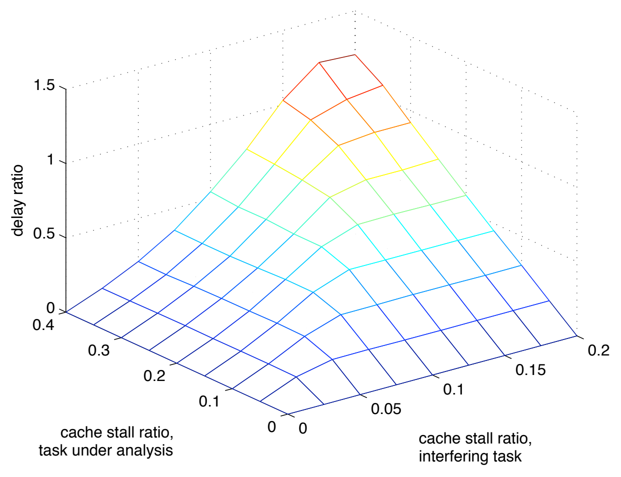

Fig. 11. Delay ratio, $\sigma = 0.2, \alpha = 0.8$, 4 cores.

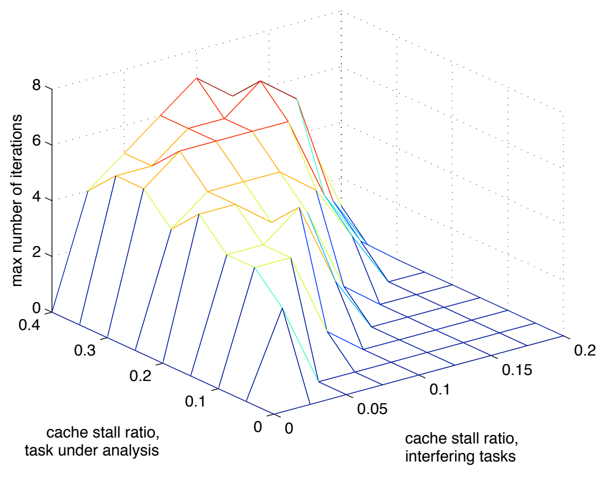

Fig. 13. Max number of iterations, $\sigma = 0.2, \alpha = 0.8$, 4 cores.

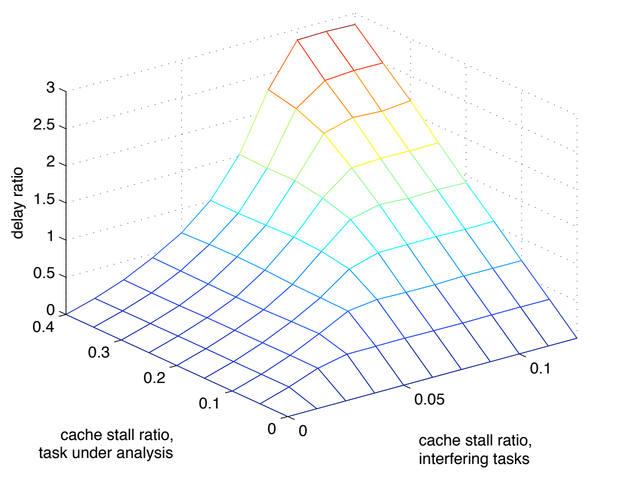

Fig. 12. Delay ratio, $\sigma = 0.2, \alpha = 0.8$, 8 cores.

high that the delay computed by Algorithm 1 is dominated by the blocking factor B_j^i, which does not depend on flow traffic. Also note that a stall ratio of 0.2 for the interfering tasks is enough to cause delay saturation for a task under analysis with stall ratio of 0.4. This is mainly because Algorithm 1 must take into account the mutual effect of multiple flows; the delay caused by one flow can "stretch" a superblock allowing more interfering traffic from other flows.

Figure 12 shows a similar simulation for a system with $N = 8$ cores and the same $\alpha = 0.8, \sigma = 0.2$ parameters. Note that the graph is very similar to the one in Figure 11, except that it saturates at roughly seven times the stall ratio for the task under analysis, and saturation is reached for a stall ratio of 0.1 for the intefering tasks. Finally, while we performed additional simulations by varying the values of α and σ, their results are very similar to the one showed in Figures 11, 12. This is mainly because as shown in Figure 10, independently of the value of β all arrival curves reach the same value at the end of a period. As a consequence, the graphs saturates at roughly the same stall ratio for the interfering tasks.

Finally, in Figure 13 we plot the maximum number of steps required for convergence by the series defined by Equations 20, 21, given the same scenario as in Figure 11. Note that the series converged in at most 7 steps in all simulations. This is because the delay functions $\bar{\alpha}_i(t)$ obtained from the arrival curves computed in Section III resemble step functions; when step functions are used in the series, at least one element $u_k^{i,j}(c+1)$ decreases by the size of one "step" at each iteration; this in turn causes quick convergence. As expected, the number of required steps is larger when the difference between the cache stall ratio for the task under analysis and the cache stall ratio for the interfering tasks is large, since in this case the computed delay bound tends to be dominated by traffic delay curves $\bar{\alpha}_i(t)$.

VI. Conclusions and Future Work

In a COTS system comprising multiple CPU cores and DMA peripherals, contention for access to main memory can significantly increase a task's WCET. In this paper, we have introduced a new analysis methodology that computes upper bounds to the contention delay suffered by each task. In particular, our analysis is able to abstract each interfering core into an arrival curve which can then be combined with peripheral traffic to yield a delay bound for the task under analysis. The methodology is applicable to a variety of COTS arbitration schemes and cache parameters.

As future work, we plan to extend the analysis to cover more general cache architectures, in particular cache levels shared among a subset of cores. Furthermore, we will investigate how to apply the analysis to dynamic real-time schedulers such as rate-monotonic and earliest-deadline-first.

References

[1] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, LNCS, 2001.

[2] S. Edwards and E.A. Lee. The case for the precision timed (pret) machine. Technical Report UCB/EECS-2006-149, EECS Department, University of California, Berkeley, November 17 2006.

[3] Marvell. *Discovery II PowerPC System Controller MV64360 Specifications*. available at http://www.marvell.com/.

[4] M.-Y. Nam, R. Pellizzoni, L. Sha, and R. M. Bradford. ASIIST: Application Specific I/O Integration Support Tool for Real-Time Bus Architecture Designs. In *Proc. of the 14th IEEE ICECCS*, June 2009.

[5] PCI SIG. *Conventional PCI 3.0, PCI-X 2.0 and PCI-E 2.0 Specifications*. http://www.pcisig.com.

[6] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha. Coscheduling of CPU and I/O transactions in COTS-based embedded systems. In *Proc. of the* 29th *IEEE Real-Time System Symposium*, Dec 2008.

[7] R. Pellizzoni and M. Caccamo. Towards the predictable integration of real-time COTS based systems. In *Proc. of the* 28th *IEEE Real-Time System Symposium*, Dec 2007.

[8] R. Pellizzoni and M. Caccamo. Impact of peripheral-processor interference on WCET analysis of real-time embedded systems. *IEEE Transactions on Computers*, 2009. To appear. Available at: http://netfiles.uiuc.edu/rpelliz2/www/.

[9] J. Rosen, P. Eles A. Andrei, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Proc. of the* 28th *IEEE RTSS*, Dec 2007.

[10] S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, and R. Ernst. Reliable performance analysis of a multicore multithreaded system-on-chip. In *Proceedings of the 6th CODES+ISSS*, Oct 2008.

[11] S. Schönberg. Impact of pci-bus load on applications in a pc architecture. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, Cancun, Mexico, Dec 2003.

[12] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS 2000*, volume 4, pages 101–104, Geneva, Switzerland, March 2000.

[13] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):966–978, Jul 2009.

Appendix

Theorem 7: Assume that the system of Equations 18 always admits solution. Then for each superblock interval $\{s_j, \ldots, s_k\}$, Algorithm 1 computes a valid upper bound $Ub_{j,k}$ to $D_{j,k}$.

Proof: We prove the theorem by induction on d. In particular, we show that $\forall i, \forall j, 1 \leq j \leq S-d : Ub^i_{j,j+d}$ is a valid upper bound to $D^i_{j,j+d}$, from which it follows that $Ub_{j,j+d} = \sum_{p \in F} Ub^p_{j,j+d}$ is an upper bound to $D_{j,j+d}$.

Induction Step: Assume that $\forall q, 0 \leq q < d, \forall i, \forall j, 1 \leq j \leq S-q : Ub^i_{j,j+q}$ is a valid upper bound to $D^i_{j,j+q}$. We have to prove that $\forall i, j : Ub^i_{j,k}$, with $k = j+d$, is an upper bound to $D^i_{j,k}$. By contradiction, assume that $\exists i, j : Ub^i_{j,k}$ is not an upper bound to $D^i_{j,k}$. Then since according to Line 8 $Ub^i_{j,k} = Ub^i_{j,k-1} + u^{i,j}_k$, it follows that at least one of the following two assertions is true for any pattern of memory operations and flow traffic that produces a delay greater than $Ub^i_{j,k}$: the delay suffered by superblocks $\{s_j, \ldots, s_{k-1}\}$ due to interference caused by flow α_i is strictly greater than $Ub^i_{j,k-1}$; or the delay suffered by s_k due to interference of α_i is strictly greater than $u^{i,j}_k$. However, since $k-1-j < d$, the first assertion is impossible due to the induction hypothesis. Hence, let the delay suffered by s_k due to interference of α_i be $u^{i,j}_k + \Delta$, with $\Delta > 0$. We consider three cases, based on which of the three terms in Equation 18 is minimum for $u^{i,j}_k$.

1) $u^{i,j}_k = B^i_k$: then applying Lemma 3 to $D^i_{k,k}$ it follows $\Delta = 0$, a contradiction.
2) Let the second term of Equation 18 be minimal for q, e.g. $u^{i,j}_k = \bar{\alpha}_i\big(\Delta^{\max^U}_{q,k} - C + Ub^{(i)}_{q,k}\big) - \sum_{p=q}^{k-1} u^{i,j}_p$. Note that the computation of $Ub^{(i)}_{j,k}$ in Lines 7, 9 of the algorithm is equivalent to computing $Ub^{(i)}_{j,k} = \sum_{p \in F, p \neq i} Ub^p_{j,k}$. Since $q \geq j+1$, it holds $k-q < d$, hence by the induction hypothesis $Ub^{(i)}_{q,k}$ is a valid upper bound to the delay caused by all flows except α_i on superblocks $\{s_q, \ldots, s_k\}$. Therefore, according to Lemma 5, the delay caused by α_i in superblocks $\{s_q, \ldots, s_k\}$ can not be greater than $\bar{\alpha}_i\big(\Delta^{\max^U}_{q,k} - C + Ub^{(i)}_{q,k}\big)$. Now note that we can rewrite the second term of Equation 18 as $u^{i,j}_k + \sum_{p=q}^{k-1} u^{i,j}_p = \bar{\alpha}_i\big(\Delta^{\max^U}_{q,k} - C + Ub^{(i)}_{q,k}\big)$: since we assumed that the delay in s_k is $u^{i,j}_k + \Delta$, it follows that the delay in $\{s_q, \ldots, s_{k-1}\}$ is at most $\sum_{p=q}^{k-1} u^{i,j}_p - \Delta$. Following Line 8 in the algorithm, $Ub^i_{j,k-1} = Ub^i_{j,q-1} + \sum_{p=q}^{k-1} u^{i,j}_p$; due to the induction hypothesis, $Ub^i_{j,q-1}$ is an upper bound to the delay caused by α_i in $\{s_j, \ldots, s_{q-1}\}$, hence the total delay suffered in $\{s_j, \ldots, s_{k-1}\}$ is at most $Ub^i_{j,k-1} - \Delta$. This contradicts the hypothesis that $Ub^i_{j,k-1} + u^{i,j}_k$ is not an upper bound to $D^i_{j,k}$.
3) Finally, assume that $u^{i,j}_k + \sum_{p=j}^{k-1} u^{i,j}_p = \bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k\big)$. We prove that $\bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k\big)$ is an upper bound to the delay caused by α_i in superblocks $\{s_j, \ldots, s_k\}$, which contradicts the hypothesis that $Ub^i_{j,k} = u^{i,j}_k + \sum_{p=j}^{k-1} u^{i,j}_p$ is not an upper bound to $D^i_{j,k}$. This is possible applying Lemma 5 if we can show that $Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k = \sum_{p \in F, p \neq i}(Ub^p_{j,k-1} + u^{p,j}_k)$ is an upper bound to the delay caused by all flows except α_i in $\{s_j, \ldots, s_k\}$.
   Consider flow $\alpha_p, p \neq i$. There are two possible cases: a) $u^{p,j}_k$ is computed based on either term (1) or (2) in Equation 18; b) $u^{p,j}_k$ is computed based on the third term. For case a), we can apply the same reasoning as in the previous two points to show that $Ub^p_{j,k}$ is an upper bound to the delay $D^p_{j,k}$. For case b), let $\mathbb{F}$ be the set of all flows (including α_i) that are computed according to the third term. We note the following: 1) according to Equation 18, each delay term $u^{p,j}_k, p \in \mathbb{F}$ is monotone non-decreasing in each other delay term in $\mathbb{F}$; 2) due to the definition of $\bar{\alpha}_i(t)$, each $u^{p,j}_k$ term is computed as the maximum value for which the system of equations hold; 3) the system admits solution by hypothesis. This implies that $\forall p \in \mathbb{F} : Ub^p_{j,k} = Ub^p_{j,k-1} + u^{p,j}_k$ is an upper bound to the delay caused by α_p in $\{s_j, \ldots, s_k\}$, which concludes the induction step.

Base Case: For $d = 0$, we have to prove that $\forall i, \forall j, 1 \leq j \leq S : Ub^i_{j,j}$ is a valid upper bound to $D^i_{j,j}$. Note that according to the algorithm, the delay term for α_i at iteration j is computed as $u^{i,j}_j = \min\Big(B^i_j, \bar{\alpha}_i\big(\Delta^{\max^U}_{j,j} - C + \sum_{p \in F, p \neq i} u^{p,j}_j\big)\Big)$. The same arguments as in the induction step for terms (1) and (3) in Equation 18 can be used to prove that $Ub^i_{j,j} = u^i_{j,j}$ is a valid upper bound.

■

Lemma 9: In the iteration defined by Equations 20, 21, $\forall c \geq 0 : \vec{u}^j_k(c) \geq \vec{u}^j_k(c+1)$.

Proof: The proof trivially follows from Equation 21 since $u^{i,j}_k(c+1)$ is computed as the minimum of $u^{i,j}_k(c)$ and another term. ■

Lemma 10: Assume that in Algorithm 1: $\forall q, 0 \leq q < d, \forall i, \forall j, 1 \leq j \leq S - q : Ub^i_{j,j+q}$ is a valid upper bound to $D^i_{j,j+q}$. Then at step j, k of the algorithm, with $k = j + d$, in the iteration defined by Equations 20, 21 it holds: $\forall c \geq 0, \forall i : u^{i,j}_k(c) \geq 0$.

Proof: We prove the lemma by induction on c. The base case follows directly from the proof of Theorem 7, since the first two terms of Equation 18 are computed independently from the result of the iteration defined by Equations 20, 21.

For the induction step, assume that $\forall i : u^{i,j}_k(c) \geq 0$. We prove that $\forall i : u^{i,j}_k(c+1) \geq 0$. If according to Equation 21 , $u^{i,j}_k(c+1) = u^{i,j}_k(c)$, this is trivially true. Therefore, assume that $u^{i,j}_j(c+1) + \sum_{p=j}^{k-1} u^{i,j}_p = \bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k(c)\big)$. Note that by the assumption on the correctness of Algorithm 1 up to step q and from Lemma 5 it follows: $\sum_{p=j}^{k-1} u^{i,j}_p \leq \bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1}\big)$. Note that since $\alpha_i(t)$ is an arrival curve, both $\alpha_i(t)$ and $\bar{\alpha}_i(t)$ are monotonically non-decreasing. Since furthermore by the induction hypothesis $\sum_{p \in F, p \neq i} u^{p,j}_k(c) \geq 0$, it holds: $u^{i,j}_j(c+1) + \sum_{p=j}^{k-1} u^{i,j}_p \geq \sum_{p=j}^{k-1} u^{i,j}_p$, which concludes the proof. ■

Lemma 11: Assume that the series defined by Equations 20, 21 converges to a fixed point $\vec{u}^j_k$. Then each $u^{i,j}_k$ value satisfies Equation 18.

Proof: Since $\bar{\alpha}_i(t)$ is monotonically non-decreasing and $u^{i,j}_k(c+1) \leq u^{i,j}_k(c)$ by Lemma 9, from Equation 21 it follows: $\bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k(c+1)\big) - \sum_{p=j}^{k-1} u^{i,j}_p \leq \bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k(c)\big) - \sum_{p=j}^{k-1} u^{i,j}_p$. Therefore, each $u^{i,j}_k(c+1)$ term can also be computed as:

$$u^{i,j}_k(c+1) = \min\Big(u^{i,j}_k(0), \quad (24)$$
$$\bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k(c)\big) - \sum_{p=j}^{k-1} u^{i,j}_p\Big).$$

Since $\vec{u}^j_k$ is a fixed point, it then follows:

$$u^{i,j}_k = \min\Big(u^{i,j}_k(0), \quad (25)$$
$$\bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k\big) - \sum_{p=j}^{k-1} u^{i,j}_p\Big)$$
$$= \min\Big(B^i_k,$$
$$\min_{q: j+1 \leq q \leq k} \big\{\bar{\alpha}_i\big(\Delta^{\max^U}_{q,k} - C + Ub^{(i)}_{q,k}\big) - \sum_{p=q}^{k-1} u^{i,j}_p\big\},$$
$$\bar{\alpha}_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p \in F, p \neq i} u^{p,j}_k\big) - \sum_{p=j}^{k-1} u^{i,j}_p\Big)$$

which is equivalent to Equation 18. ■

Lemma 12: The system of Equation 18 admits at most one solution.

Proof: By contradiction, assume that both $\vec{u}'^j_k$ and $\vec{u}''^j_k$ are different solutions to the system of Equation 18. Let $\mathbb{F}$ be the set of delay terms that have different values in $\vec{u}'^j_k, \vec{u}''^j_k$, e.g. $i \in \mathbb{F}$ iff $u'^{i,j}_k \neq u''^{i,j}_k$. Each delay term in $\mathbb{F}$ must be computed according to the third term in Equation 18, since in terms (1) and (2) $u^{i,j}_k$ is computed independently from all others $u^{p,j}_k, p \neq i$.

There are two possibilities: a) $\vec{u}'^j_k > \vec{u}''^j_k$ (or $\vec{u}''^j_k > \vec{u}'^j_k$); b) neither $\vec{u}'^j_k > \vec{u}''^j_k$ nor $\vec{u}'^j_k < \vec{u}''^j_k$ holds. In case a), since by definition of $\bar{\alpha}_i(t)$, each $u^{i,j}_k$ term must be maximal, it follows that $\vec{u}''^j_k$ (respectively, $\vec{u}'^j_k$) is not a solution to Equation 18.

In case b), without loss of generality assume that $\sum_{p \in F} u'^{p,j}_k \geq \sum_{p \in F} u''^{p,j}_k$. Since $\vec{u}'^j_k > \vec{u}''^j_k$ does not hold and the two solutions are different, then it must exist an element $i \in \mathbb{F}$ such that $u'^{i,j}_k < u''^{i,j}_k$. By definition of $\bar{\alpha}_i(t)$ and since both $\vec{u}'^j_k$ and $\vec{u}''^j_k$ are valid solutions it follows:

$$u'^{i,j}_k = \alpha_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p=j}^{k-1} u^{i,j}_p + \quad (26)$$
$$+ \sum_{p \in F, p \neq i} u'^{p,j}_k + u'^{i,j}_k\big) - \sum_{p=j}^{k-1} u^{i,j}_p$$

and

$$u''^{i,j}_k = \alpha_i\big(\Delta^{\max^U}_{j,k} - C + Ub^{(i)}_{j,k-1} + \sum_{p=j}^{k-1} u^{i,j}_p + \quad (27)$$
$$+ \sum_{p \in F, p \neq i} u''^{p,j}_k + u''^{i,j}_k\big) - \sum_{p=j}^{k-1} u^{i,j}_p.$$

Note that $\alpha_i(t)$ is monotonically non-decreasing and furthermore the argument of $\alpha_i(t)$ in Equation 26 is not smaller than the argument of $\alpha_i(t)$ in Equation 27 since by assumption $\sum_{p \in F, p \neq i} u'^{p,j}_k + u'^{i,j}_k \geq \sum_{p \in F, p \neq i} u''^{p,j}_k + u''^{i,j}_k$. Hence, it follows $u'^{i,j}_k \geq u''^{i,j}_k$, a contradiction. ■