ASCOM: Affordable Sequence-aware COntention Modeling in Crossbar-based MPSoCs

Jeremy Giesen*†, Enrico Mezzetti†, Jaume Abella†, Francisco J. Cazorla†

* Universitat Politècnica de Catalunya, Spain † Barcelona Supercomputing Center, Spain

***Abstract*—Multicore interference that arises when several accesses contend for the same shared hardware resources poses a challenge to the already demanding consolidated verification and validation practice. The Sequence-Aware Pairing (SeAP) model approach exploits the parallelism granted by crossbars to derive tighter contention bounds. We show that SeAP suffers from scalability issues that hinders its applicability to more complex contention scenarios. We address SeAP limitations in terms of scalability by identifying two complementary techniques to reduce SeAP execution time requirements. We assess the proposed approaches to show how they effectively enable the application of SeAP to large sequences of accesses to the crossbar with limited impact on tightness, and scaling gracefully with the number of co-running cores.**

***Index Terms*—Crossbar, multicore, contention, software timing analysis**

I. Introduction and Motivation

Deriving high-quality WCET [10] estimates is a challenging task especially when facing the complexity of Multicore Processor Systems on Chip (MPSoC). Massive hardware resource sharing in MPSoCs is one of the main challenges to be tackled before MPSoCs are fully embraced in high-integrity systems [1], [2], [8], [9].

When analyzing the impact of contention, *sequences of requests* from each core to the target *Hardware Shared Resources* (HSRs) have been considered to model the parallelism enabled by the crossbar when requests are sent to different devices. The specific order in which requests are sent to the devices allows to rule out infeasible contention scenarios. The Sequence-Aware Pairing (SeAP) [3] approach builds on precedence relation between accesses, which is a weaker concept than timestamps, to rule out from contention modeling those scenarios where requests from multiple cores to the same HSR cannot happen in parallel, despite happening in the same time window.

However, SeAP inherits the computational complexity of the pattern-matching approaches it builds on, which is in the worst-case exponential on the number and size of the considered sequences of requests [3], [4], [6]. Owing to the inherent complexity, the use of SeAP on realistic multicore scenarios (i.e., with more than 2 cores) can be challenging already with sequences above 10.000 elements, while sequences from real applications are expected to be in the order of 50.000 to 100.000 thousands elements for 10ms applications.

Contribution. We present *Affordable Sequence-aware COntention Modeling* (ASCOM) to overcome SeAP inherent complexity limitations so that it can be applied to sufficiently large sequences. ASCOM enforces linearity on the contention latencies as a way to enable a conservative pair-wise, compositional application of SeAP (COMP) and further reduces computational complexity by segmenting the input sequences in fixed size chunks (SEGM). We show how ASCOM bounds the computational complexity to be quadratic on the input size (instead of exponential) and delivers accurate results, incurring only 9.68% average overestimation compared to SeAP. Hence, ASCOM enables sequence-aware contention modeling even with large core counts and HSR access sequences with more than 100.000 elements.

II. Introduction to SeAP

We consider SeAP as the reference solution for contention modeling in crossbar-based systems since, interestingly, models based on request access counts only are outperformed by those building on request ordering. As illustrative example let's assume a dual-core processor where cores (c_0 and c_1) are connected to three HSR ($HSR\ A$, $HSR\ B$, and $HSR\ C$) through a crossbar. For the sake of clarity, we assume each HSR only accepts one type of request, respectively referred to as `A`, `B`, and `C`. Requests targeting the same HSR can suffer a contention delay of 2, 5, and 7 cycles respectively. We further assume the two cores are executing two applications simultaneously in the same time window, triggering two distinct sequences of requests through the crossbar: $q_0 = \{\texttt{BBBCCBAACA}\}$ and $q_1 = \{\texttt{AACCBBCBBC}\}$. The sequences sum up to $\{3{\cdot}\texttt{A}, 4{\cdot}\texttt{B}, 3{\cdot}\texttt{C}\}$ for q_0 in c_0 and $\{2 \cdot \texttt{A}, 4 \cdot \texttt{B}, 4 \cdot \texttt{C}\}$ for q_1 in c_1. Approaches building on access counts only derive contention bounds by cumulatively considering the maximum contention each request can suffer as:

$$\Delta = \sum_{h \in \mathcal{HSR}} min(n^h_{c0}, n^h_{c1}) \times l_h = \underbrace{2 \times 2}_{\texttt{A}} + \underbrace{4 \times 5}_{\texttt{B}} + \underbrace{3 \times 7}_{\texttt{C}} = 45$$

where $\mathcal{HSR}$ is the set of all resources, $n^h_{c_i}$ is the number of requests core c_i performs to HSR h, and l_h is the maximum contention delay incurred by any request to that resource. In this case, we ***pair*** 2 requests to $HSR\ A$, 4 requests to $HSR\ B$, and 3 requests to $HSR\ C$, resulting in 45 cycles of contention.

However, by looking at the sequence of requests generated *independently* by each core, we can conclude the assumed contention scenario is actually unfeasible. For instance, if 4 requests to $HSR\ B$ from both cores are conflicting in the crossbar, no request to $HSR\ A$ and at most 2 requests to

http://dx.doi.org/10.1145/3555776.3577793

$HSR\ C$ can ever collide. In fact, when a request from a sequence is assumed to collide with a request in the other core, that request acts as an 'anchor' and affects how the remaining requests in the sequence can collide. The sequence-aware cumulative contention (Δ_{SeAP}) is representing the actual worst-case scenario if we consider precedence relations among accesses within each core. In this case, the worst-case contention scenario, happens when 3 requests to $HSR\,C$ and 1 request to $HSR\,B$ are assumed to be colliding. The contention impact then sums up to $\Delta_{SeAP} = 1 \times 5 + 3 \times 7 = 26$, which is 42% tighter than the solution obtained by considering access counts only.

III. ASCOM

SeAP builds on *dynamic programming* to perform an exhaustive search over all possible overlapping among sequences of requests. The baseline implementation is inspired by state-of-the-art solutions for the Heaviest Common Subsequence (HCS) problem [7] that exploit a k-dimensional matrix, with k being the number of sequences considered, to guide and optimize the search. For illustrative purposes, we consider SeAP in its simplest instantiation and focus on a two-dimensional scenario, considering only two sequences of requests from the task under analysis and one contender task. In this scenario, the SeAP algorithm closely resembles the HCS solution as any pairing of requests is a full pairing (involving all contenders) which is not a necessary condition in SeAP. Also, with two sequences only, both SeAP and HCS are symmetric in the sense that the computed worst-case interference is the same for both tasks. The solution, adapted from [7], is shown in Algorithm 1.

Algorithm 1: SEAP with 2 sequences

Input: Two sequences of requests over the crossbar
Output: The worst-case contention delay caused by any feasible pairing.

1 **for** $i \leftarrow 1$ ***to*** $\text{LEN}(x) + 1$ **do**
2 **for** $j \leftarrow 1$ ***to*** $\text{LEN}(y) + 1$ **do**
3 $M[i][j] = \max\big(M[i\text{-}1][j], M[i][j\text{-}1], M[i\text{-}1][j\text{-}1] + W(x[i], y[j])\big)$
4 **return** M

SeAP receives as input two sequences of symbols x, y representing the ordered sequence of requests to shared resources through the crossbar performed by two tasks executing in parallel. A weight function W is also defined that returns the worst-case contention delay incurred by a given request to a target in the crossbar when colliding with another request (possibly of different type) to the same target. As anticipated, in this case, SeAP exploits a bi-dimensional data structure M which is used to hold temporary cumulative information on request pairing and on which the subsequence leading to the worst-case contention delay can be derived. The size of M, whose entries are initialized to 0, is determined by the length of x and y incremented by 1, as one additional row and column are used to represent the initial null position when parsing the sequences. Algorithm 1 iterates over the elements of the input sequences following a nested-loop. For each pair of elements from x and y, the cumulative worst-case contention delay is updated using the information from previous steps in the dynamic programming schema which can be retrieved from specific relative positions in M [7]. As a result of the algorithm, M holds both the worst-case contention delay ($M[LEN(x)+1, LEN(y)+1]$) and the necessary information to reconstruct an example (witness) of the subsequence of requests (pairing) that determined such maximum delay. SeAP can be generalized to an arbitrary number of sequences and lengths [3]. However, such generalization cannot escape SeAP scalability concerns, inherited from the HCS problem [4], [7]: its complexity remains in the worst-case exponential on the number and size of the considered sequences of requests [3], [4]. ASCOM enables a scalable, sequence-aware contention modeling in crossbar-based systems by leveraging two main practical approaches that build on SeAP peculiarities to work around its computational complexity limits.

A. Leveraging compositionality (COMP)

In a compositional platform with blocking requests, the maximum delay suffered by a request is linearly determined by the number of contending requests. In our reference crossbar scenario with k co-runners, the contention incurred by a request `A` from the task under analysis to $HSR\ A$ is the value of the weight function W on a pairing tuple of at most k requests of the same type. Assuming `A` is contending with r requests, then the delay incurred by `A` can be linearly computed as a function of the delay incurred upon a single contending requests, that is:

$$\Delta_{\texttt{A}} = W(\overbrace{\texttt{A} \cdots \texttt{A}}^{r}) = \sum_{i=1}^{r-1} W(\texttt{AA}) \tag{1}$$

Compositionality of timing interference is a platform requirement that manifests itself as an *additive* weight function W, supporting linear composition over the input size. Under compositionality, timing interference can be independently computed among k contending input sequences by a pairwise application of SeAP to the sequence of the task under analysis and one of the remaining sequences at a time. The obtained contention figures are eventually merged to provide the cumulative contention bounds.

An additive weight function, in fact, allows to drastically reduce the complexity of applying SeAP by a *divide and conquer* approach that simplifies the scenario to $k-1$ invocations of SeAP over 2 sequences. The compositional application of SeAP is illustrated in Algorithm 2. Given $\mathcal{Q}$ as the set of sequences that need to be considered for contention analysis, with q_1 being the sequence from the core under analysis, the algorithm simply accumulates in Δ_{COMP} the sum of the worst-case pairing independently computed between the target sequence and each of the other sequences independently. The computational complexity of the overall process drops from exponential, $\mathcal{O}(n^k)$, to quadratic $\mathcal{O}(kn^2)$. While interference at platform level is seldom compositional, we can still conservatively adjust the weight function W to model contention impact additively, without incurring excessive over-approximation. A sub-additive variant of W can be defined as:

$$W^{+}(\overbrace{\mathtt{A}\cdots\mathtt{A}}^{p}) = (p-1) \times \max_{p \leq r \leq k} \left(\left\lceil \frac{W(\overbrace{\mathtt{A}\cdots\mathtt{A}}^{r})}{r-1} \right\rceil \right) \quad (2)$$

Algorithm 2: COMP

Input: $\mathcal{Q}$ as a set of k sequences of requests
$q_1 \in \mathcal{Q}$ as the sequence of task under analysis
Output: The worst-case contention delay caused by any feasible pairing.

1 **for** $q_i \in \mathcal{Q} \setminus \{q_1\}$ **do**
2 $\quad \Delta_{COMP}$ += $SeAP(q_1, q_i)$
3 **return** Δ_{COMP}

with k representing the maximum input size for the original weight (i.e., requests in parallel) and $2 \leq p \leq r \leq k$ representing the input size of the weight functions. On real scenario, it is often the case that $W^{+} = W$ for pairing involving a non-negligible amount of conflicting requests. For COMP we are mainly interested to W^{+} for 2 sequences, where most conservative fixes to W occur.

B. Segmenting sequences (SEGM)

With SEGM, we divide SeAP input sequences into fixed size segments. In order to apply SeAP at the granularity of segments, all sequences from the core under analysis and from the contenders, are split into an ordered list of fixed size segments (lines 1-2), with minor differences determined by different size among sequences. Then segments are fed in input to SeAP following the original order (lines 3-4). The sum of the bounds computed for each segment set is returned as a cumulative contention bound.

Algorithm 3: SEGM

Input: $\mathcal{Q}$ as a set of k sequences of requests
$q_1 \in \mathcal{Q}$ as the sequence of task under analysis
c as the size of sequence segments
Output: The cumulative worst-case contention delay for all sets of segments.

1 **for** $q_i \in \mathcal{Q}$ **do**
2 $\quad S_i \leftarrow$ ordered list of c-sized segments of q_i
3 **for** $i \in \text{LEN}(S_1)$ **do**
4 $\quad \Delta_{SEGM}$ += $SeAP(S_1[i], S_2[i], \ldots, S_k[i])$
5 **return** Δ_{SEGM}

Splitting the sequences affects the contention results as we are possibly excluding some elements from the set of feasible pairings. However, while the impact of dropped pairing scenarios happens to be potentially huge for extreme cases, it is expected to be considerably small in practice. In fact, with realistically long sequences of access (e.g., with more than 5K elements) and with a non negligible number of HSRs that are uniformly accessed, the worst-case pairing of requests is involving the larger number of requests both from the task under analysis and its contenders.

IV. Experimental Evaluation

We explore different parameters and values. First, we focus on the *sequence size* focusing on sequences of 10K and 100K elements, and their sub-multiples (in case of SEGM). 10K already hits the complexity wall for SeAP over three sequences. In terms of *number of sequences*, we focus on 2 and 3 sequences scenarios as they allow to gather baseline reference values with SeAP. We cover different *sequence dictionaries* with a varying number of symbols (i.e. types of memory operations to devices accessed through the crossbar). We report on scenarios for 5 and 7 symbols, with the latter corresponding to read or write requests sent through the crossbar in the AURIX TC297 setup. We also considered different *sequence shapes and distributions* with different patterns of request to each device (i.e., grouping a stream of requests to the same device). We explored scenarios where requests are triggered in a variable size cluster (uniformly) in the ranges: [2-2] (fixed range), [2-4], [2-6] and [2-12]. Additionally, in consideration of the fact that smaller clusters are more frequent, we also explored sequences in the cluster ranges 2-6 and 2-12 but with a non-uniform distribution, biased towards smaller clusters ([2-6B] and [2-12B]). Sequences have been synthetically generated using a bespoke random sequence generator.

We model SeAP weight function based on access latencies observed in the Infineon AURIX TriCore TC297 [5], a crossbar-based multicore platform widely used in the automotive domain. The TC297 memory subsystem includes both core-local and shared memories. At core level, the TC297 features separated scratchpads and caches for instructions and data, whereas at platform level it equips a 32KiB shared SRAM, called Local Memory Unit (LMU), and multiple FLASH devices, 4 independent 2MB program flashes (PFlash) and 1 data flash (DFlash). Shared memory devices are accessed by cores (and other external devices) via dedicated slave interfaces through the Shared Resources Interconnnect (SRI) crossbar. The maximum observed contention slowdown incurred by memory requests over the SRI are reported in Table I (W values) where latencies are reported per device and operation type. Essentially, the table provides a simple representation of the SeAP weight function. In particular, latencies are relative to contention scenarios where a request is conflicting with one or two other requests from other cores, hence matching the 3 cores available in the TC297. Each sequence in the considered target has been generated from a dictionary with at most 7 distinct symbols: 2 symbols (read and write operations) for the LMU and 1 symbol each for the 4 PFlashes and the DFlash devices (only supporting read requests).

We run our experiments on a homogeneous computing cluster equipped with 40x nodes each with 2x Intel Xeon E5-2630L v2 @2.40GHz, each one equipped with 128GB of private memory. Experiment batches were automatically executed and the obtained results were automatically collected and processed off-line.

A. COMP

Execution time: Figure 4b compares the time required by SeAP on 3 sequences altogether or by leveraging compositionality with COMP, for a fixed sequence dictionary and shape, by only varying the sequence sizes from 1K to 100K

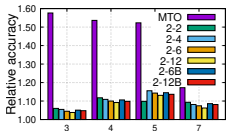
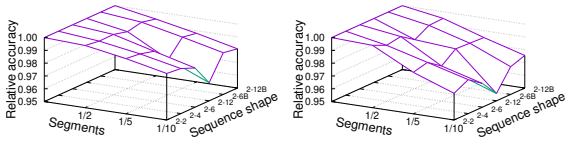
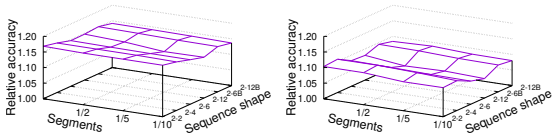


Fig. 1: COMP accuracy.



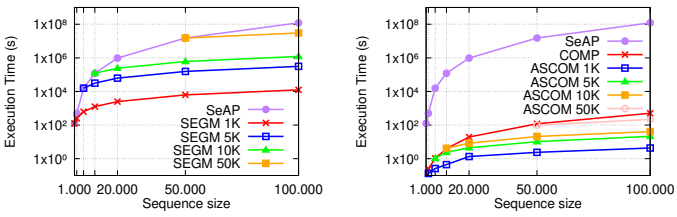
(a) 5 Symbols (b) 7 Symbols

Fig. 2: SEGM relative accuracy for three sequences of 10k elements.



(a) 5 Symbols (b) 7 Symbols

Fig. 3: ASCOM relative accuracy for three sequences of 10k elements.



(a) SeAP and SEGM (b) SeAP, COMP, and ASCOM

Fig. 4: Execut. time of SeAP, COMP, SEGM & ASCOM.

TABLE I: Real (W) and forced (W^+) linearity weight function.

Slowdown	1 Request				2 Requests					
	R		W		R+R		R+W		W+W	
	W	W^+	W	W^+	W	W^+	W	W^+	W	W^+
LMU Read	1	**3**	3	**4**	4	4	6	6	8	8
LMU Write	1	**4**	3	**5**	5	5	7	7	9	9
PFlash Read	4	**6**	n/a	n/a	11	11	n/a	n/a	n/a	n/a
DFlash Read	34	**35**	n/a	n/a	69	69	n/a	n/a	n/a	n/a

elements. COMP timing requirements stays affordable (<10 minutes) even for larger sequences, where SeAP shows off its limitations already with average-size sequences (∼1 day for 10K elements).

Accuracy: Figure 1 shows the relative increase (overestimation) incurred by COMP with respect to SeAP. Each plot also reports the Maximum Theoretical Overestimation (MTO), representing an upper bound to the overestimation incurred by enforcing linearity in the specific weight function W in Table I. The observed overestimation ranges in between 4% and 17% in the worst case, with an average overestimation of 9.5%.

B. SEGM

Execution time: Figure 4a shows, in logarithmic scale, SeAP timing requirements for a 3-sequences scenario and variable sequence size and compares them against the application of SEGM with segment size from 1K up to 50K elements. The performance improvement is directly proportional to the segment ratio, so that splitting a sequence in half lead to halved execution time. What is an acceptable segment size depends on the impact on accuracy.

Accuracy: Given that we cannot resort to extrapolation, in order to have a reference SeAP result (over the full sequences), we are forced to focus on sequences with 10K elements. Surface plots in Figure 2 are relative to distinct dictionaries of symbols/devices, and report the relative underestimation incurred when splitting the original sequences in 1 (full sequence), 2, 5 and 10 segments, and for different sequence shapes/distributions. In general terms, results confirms that SEGM can deliver accurate results. We observed a 0.67% average under-approximations, with a peak of 4.42% in the 7 symbol (2-12, 1/10 segments) setup.

C. ASCOM

Execution time: Figure 4b shows for a 3-sequences scenario and variable sequence size the execution time incurred by ASCOM compared to SeAP. Combining SEGM+COMP provides further benefits over SEGM and COMP alone. This is confirmed also by SEGM results (Figure 4a). The performance improvement is significant over SeAP and it is proportional to the segment ratio.

Accuracy: For 3 sequences with 10K elements results are reported in Figure 3. The assessment is dominated by COMP: the values in the left wall of each plot correspond to the results observed for COMP alone, reported in Figure 1. The observed impact of segmenting on accuracy is pretty small and less dependent on the particular shape/distribution of the sequences. Moreover, the (optimistic) accuracy loss contributed by SEGM becomes less evident (∼1% less impact) when combined with COMP. In fact, by applying COMP we are reducing to a 2 sequence problem, where the effect of SEGM in removing feasible pairing scenarios becomes marginal.

V. Conclusions

We presented ASCOM as the combination of two complementary techniques to reduce SeAP time complexity by enabling the application of the technique either compositionally on a smaller number of sequences (COMP) or on smaller portions of sequences (SEGM). We assess the proposed approaches both individually and in conjunction, to show how they can effectively enable a notable reduction in complexity with a moderate increase in pessimism compared to SeAP. ASCOM allows to deal with large sequences comprising more than 100K requests in less than 10 minutes and just 9.68% of average overestimation and 16.85% of peak overestimation, and scaling gracefully with number of co-running cores.

VI. Acknowledgments

The research leading to these results has received funding from the European Research Council (ERC) under the Eu-

ropean Union's Horizon 2020 research and innovation programme (grant agreement No. 772773). This work has also been partially supported by Grant PID2019-107255GB-C21 funded by MCIN/AEI/ 10.13039/501100011033.

REFERENCES

[1] Jon Perez Cerrolaza et al. Multi-core devices for safety-critical systems: A survey. *ACM Comput. Surv.*, 2020.

[2] G. Fernandez et al. Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In *WCET*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

[3] Jeremy Giesen et al. Modeling contention interference in crossbar-based systems via sequence-aware pairing (SeAP). In *RTAS*. IEEE, 2020.

[4] Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 1975.

[5] Infineon. AURIX TriCore TC2xx. https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc2xx/, 2022.

[6] Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 1992.

[7] Rao Li. A linear space algorithm for the heaviest common subsequence problem. *Utilitas Mathematica*, 75, 03 2008.

[8] Hamid Tabani et al. Assessing the adherence of an industrial autonomous driving framework to iso 26262 software guidelines. In *ACM/IEEE DAC*, 2019.

[9] Reinhard Wilhelm. Mixed feelings about mixed criticality (invited paper). In *WCET*, OASICS. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[10] Wilhelm R. et al. The worst-case execution-time problem: overview of methods and survey of tools. *ACM Trans. on Embedded Comp. Systems*, 2008.