Conservative Application-Level Performance Analysis through Simulation of MPSoCs

Andrew Nelson
Technische Universiteit Delft
a.t.nelson@tudelft.nl

Andreas Hansson
Universiteit Twente
a.hansson@utwente.nl

Henk Corporaal
and Kees Goossens
Technische Universiteit Eindhoven
h.corporaal@tue.nl
k.g.w.goossens@tue.nl

***Abstract*—Applications, often with real-time requirements, are mapped onto Multiprocessor Systems on Chip (MPSoCs). Hard real-time applications require no deadline misses, and a formal modelling approach must be used to analyse the worst-case performance, which is complicated and time consuming. Such models are restricted to specific application behaviours and not generally applicable. Soft real-time applications such as video decoders often do not fit these models while having less strict requirements. An infrequent frame drop is barely noticeable, and a worst-case analysis is too pessimistic. For such applications it suffices to meet deadlines for a given set of traces.**

In this work we propose conservative simulation as an alternative approach to formal modelling for soft real-time applications. We introduce a hybrid simulation method which enables performance guarantees on a per-trace basis, without any modelling effort. Furthermore we evaluate an implementation of the described technique and compare it with an actual MPSoC instance implemented on an FPGA. Our results show that the simulation technique is conservative, with less than a 10% difference in timing compared to the actual implementation, for a software JPEG decoder.

I. Introduction

Real-time applications have temporal constraints to adhere to. The rigidity of this adherence can be defined subjectively as a hard or soft constraint [1]. Performance analysis of hard real-time applications uses formal modelling techniques such as dataflow analysis [2] or real-time calculus [3]. Formal models are restrictive of application behaviours [2] and are not generally applicable. Even if it is possible to modify the application to fit the model, considerable time and effort must be spent creating an application model and ensuring that it accurately captures potentially complex behaviours. Soft real-time applications such as video decoders often do not fit these models while having less strict temporal requirements. It is nevertheless necessary to perform application-level performance analysis on soft real-time applications in order to ensure that the temporal requirements of the application are met for a representative set of inputs [4].

Simulation provides the ability to generate per-trace timings without the need for an application model. Unfortunately, simulation does not take into account worst-case behaviour of the platform and thereby only produces timing estimates. The increased computational complexity of simulating a hardware platform also makes it relatively slow when compared with more abstract techniques. Combining the two techniques of formal modelling and simulation into a hybrid model presents the opportunity to produce per-trace conservative application-level performance analysis without the need to formally model the application.

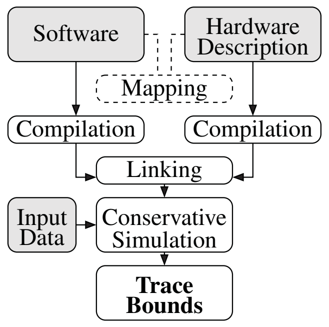

Figure 1. Conservative simulation based SoC co-design flow.

We propose the use of a hybrid technique called *conservative simulation* as an alternative to formal analysis to generate *per-trace temporal bounds*, without requiring any modelling of the application. We achieve this by combining already existent, cycle accurate processor Instruction Set Simulators (ISS) with a conservative NoC communication model of our own design. We further contribute a conservative simulation based hardware/software co-design method, as illustrated in Figure 1. Software code needs only to be mapped onto the hardware platform before being compiled and linked into a simulation executable program. The mapping process involves assigning portions of the application (tasks) to individual processing cores and ensuring that communication is carried out using explicit synchronisation. The resultant code only has to be compiled and linked in order to perform conservative simulation. The simulation processes the input data producing a trace bound. A more thorough application-level performance analysis can be carried out by inputting a representative dataset [4] to the simulated application, e.g. a H264 decoder application simulation, processing a set of videos of varied complexity. When we apply our technique to a JPEG decoder application, that is mapped to a three processor system, it produces conservative, per-trace timing results with less than 10% difference compared to the actual implementation.

The rest of this paper is structured as follows. In the following section we discuss the work of this paper in the context of related work and declare the contributions that we make. In section II we explain the rationality behind the conservative simulation technique for conservative application-level performance analysis of real-time MPSoCs. In section IV we present a hardware platform to which the conservative simulation technique has been applied, focussing on the necessities that make our approach applicable. We follow this in section V with a detailed description of our hybrid modelling approach to the hardware platform. In section VI we describe how to achieve application level conservativeness in a multi-core system using our NoC model. We provide a case study analysis of our technique by applying it to a JPEG decoder application, in Section VII. We end this paper with conclusions in section VIII.

II. Related Work

Many techniques for modelling hardware platforms have been proposed in the past. In this section we focus on techniques that overlap with our work in some way. Multiprocessor simulation methods such as those in [5], [6] enable the production of timing results through simulation of applications running on a cycle accurately modelled platform. Simulation techniques such as these do not take into account the worst case response times of individual components by construction. As a result any timing analysis will be an estimate, making it unsuitable to draw firm conclusions relating to real-time deadlines. In contrast to these simulation techniques, our method provides per-trace temporal guarantees.

Formal methods such as dataflow analysis [2], [4], [7] and Real-Time Calculus (RTC) [3] enable abstraction away from platform and application details by generalising functional and temporal behaviours. Using these methods, it is possible to analytically derive temporally conservative timings at the transaction level. In [8] it is shown that the Latency Rate (LR) server [9] abstraction technique can provide conservative abstraction for certain arbitration schemes and that they can be incorporated into dataflow graphs. The work in [10] describes a method to conservatively model a predictable MPSoC platform and an application as a combined Cyclo-Static Dataflow (CSDF) graph [2]. The technique is only applicable to applications that can be modelled as CSDF graphs, with time and effort needing to be spent in order to create CSDF models for each application that the platform will run. Our technique provides the ability to perform per-trace conservative application level performance analysis on software without the need to create application models. The formally modelled hardware in [10] could also be applied to an application trace to enable the analytical derivation of per-trace guarantees. In order to analytically bound a trace from an application the trace information must first be obtained. This trace can be obtained through simulation of the platform. Our proposed solution uses simulation to not only generate the trace but also perform the temporal analysis of the trace, in a single pass.

Stochastic methods such as those described in [1] target Soft Real-Time applications by enabling the derivation of the probability of an application missing a deadline. The stochastic methods described in [1] require mathematical analysis of the application and hardware of a similar degree of complexity [4] as the previously described formal methods, while not producing a definitive guarantee. Our method requires no temporal profiling of the application to produce definitive conservative per-trace guarantees. For Soft Real-Time applications the conservative simulation method is used to ensure that a representative set of input stimuli [4] meet their deadlines so that other subsequent stimuli are also likely to meet their deadlines.

An increasingly popular approach is to combine simulation and formal methods to create a hybrid analysis technique. In [11], [12] a hybrid approach is taken that combines real-time calculus with simulation in order to produce trace based timing values. The approach in [11], [12] requires the formal modelling of chosen application tasks, thus requiring much time and effort. In [13] the abstraction of specific application tasks is avoided by abstracting their platform's Real-Time Operating System (RTOS). This is beneficial as a new abstract model does not need to be created for every application that is to be run on the platform. The technique in [13], in contrast to our method, does not produce conservative temporal analysis. In [14] a multi-stage approach is taken starting with high-level abstract models for functional validation and eventually refining the model into a transaction-accurate model. In contrast to the approach in [14] our method produces conservative temporal results after a single simulation pass.

Our simulation technique enables the compilation and simulation of application code that has been mapped to the target platform. Only a single iteration of the simulation is required to produce conservative application timings. We contribute a conservative hybrid simulation approach that combines formal modelling with cycle accurate simulation, which enables the hardware/software co-design flow illustrated in Figure 1. To achieve this we further contribute a description of how to model an Æthereal NoC [15] for conservative simulation. We subsequently contribute an automated toolflow for the derivation of the parameter values required for this model. We finally contribute a case-study analysis of our technique in which we compare simulation results with those from an actual FPGA implementation of the platform, for an actual soft real-time application in the form of a JPEG decoder.

III. Conservative Simulation

In this paper we focus on the conservative performance analysis of real-time MPSoCs. A real-time MPSoC is a combination of hardware and software elements. For our purposes, at the hardware-level a MPSoC consists of multiple processing cores, memories and other peripherals all connected together in some configuration by an interconnect, e.g. the MPSoC illustrated in figure 2. Shared hardware resources require arbitration schemes to decide which of the sharing resources gets to use it at any given time. Arbitration schemes such as Time Division Multiplexing (TDM), Round Robin (RR) and Credit-Controlled Static-Priority scheduling (CCSP) [16] allow the starvation-free sharing of resources facilitating the calculation of a temporal worst case upper-bound for each arbitration. The actual duration of an arbitration depends on

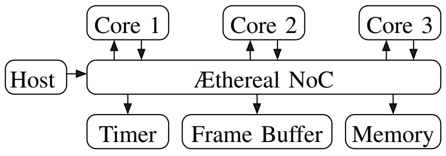

Figure 2. Hardware platform instance.

the arbiter's *settings*, e.g. a TDM table's length and utilisation, and the arbiter's *phase* at the start of the arbitration process, i.e. where a TDM arbiter is in its table.

At the software or application-level, applications execute processing input data. The execution of the application may be input dependent meaning that more or fewer cycles may be needed to execute an application depending on the input data. The performance at the application-level is also dependent on the hardware-level. Application-level performance analysis of an MPSoC is therefore dependent on the *input data*, the *arbitration settings* and the *arbitration phases*.

Formal modelling of real-time system designs enable analytical validation of worst case temporal behaviours for all input data, arbitration settings and phases. Formally modelling a real-time system is not trivial. All formal modelling techniques impose restrictions on behaviours in order to enable the analytical calculation of worst case temporal bounds. Formal modelling methods such as Dataflow [2], [4], [7] and Real-Time Calculus [3] use graph theory in order to analytically bound temporal behaviours. The behaviours of both the hardware and the software must be encapsulated within the restrictive modelling technique in order to guarantee Worst Case temporal bounds. Each application that is to be run on a platform must be modelled individually. If the application behaviours do not fit the model then a choice must be made to either invest time to find a new formal modelling technique that fits, if it exists, or spend time modifying the application to fit the model, if possible. As such a balance must be struck between the severity of the consequences of the real-time system missing a deadline and the complexity of carrying out a conservative application-level performance analysis. With vast amounts of available code that has not been written with formal modelling techniques in mind the process of verification adds complexity to the porting process.

Embedded system designers are already familiar with using simulation as part of the code porting process. Simulation is used to test the functionality of applications before they are run on a physical implementation of the platform. Cycle accurate simulation may be used to provide temporal indications of a design's performance. Cycle accuracy in this case refers to the modelling of the hardware components. These temporal indications can be provided without the restriction on system behaviours needed to perform a formal analysis. Unlike for formal analysis each cycle accurate simulation is only valid for the particular input data, arbiter settings and phase that were used for the simulation run. Even though the timings produced are not useful to provide any meaningful level of real-time guarantee, the lack of application restrictions in combination with the straight forward cycle accurate modelling of the hardware make this technique relatively easy to implement.

Table I
REAL-TIME GUARANTEES GIVEN BY EACH TECHNIQUE.

	$\forall$ inputs	$\forall$ arb. settings	$\forall$ arb. phases
Formal Modelling in general	Yes	Yes	Yes
Formal Modelling technique in [10]	Yes	Optional	Yes
Stochastic Modelling techniques in [1]	No	No	No
Conservative Simulation (per-trace)	No	Optional	Yes
Cycle Accurate Simulation	No	No	No

We contribute a hybrid method for conservative application-level performance analysis that combines formal modelling techniques with simulation to provide per-trace temporal guarantees. Our *conservative simulation* technique generates per-trace conservative application-level performance guarantees striking a balance between the ease of use of simulation and the assurances provided by formal modelling. For conservative simulation a per-trace guarantee means that the produced timing is the worst case timing for the input data and arbitration settings used for that simulation, but independent of arbitration phases. As such our technique is proposed as an alternative to formal methods for the conservative performance analysis of soft Real Time embedded systems where missing an occasional deadline may be tolerated. By conservatively simulating a representative set of data inputs [4] our technique guarantees that the deadline miss rates of the same set of traces on the implemented MPSoC could only be lower. Table I shows an overview of the capabilities of some performance analysis techniques to provide real-time guarantees.

Our technique works for predictable hardware platforms such as that described in [17] that are temporally monotonic. The monotonic property ensures that if all the events in the multiprocessor platform are viewed as a single time-line the later occurrence of an event does not enable the earlier occurrence of subsequent events and similarly the earlier occurrence of an event does not enable the later occurrence of subsequent events. At the hardware level it is therefore possible to mix conservative modelling with cycle accuracy without losing overall conservativeness. This conservation of conservativeness does not automatically confer to the application level and is dealt with separately in section VI.

It is the monotonic property of the hardware that we exploit by using formal modelling techniques to conservatively model only the run time arbitrated components of the hardware platform. By incorporating these models into an otherwise cycle accurate simulation of the hardware platform and tuning them using the arbitration settings we are able to provide per-trace conservative guarantees that are valid for all arbitration phases with few restrictions on the application. The same results that are achieved using a single run of our conservative simulation technique can be achieved by using multiple runs of a cycle accurate simulation to cover all the arbitration phases and all the arbitration settings. The practicality of doing this is questionable for all but the simplest of designs.

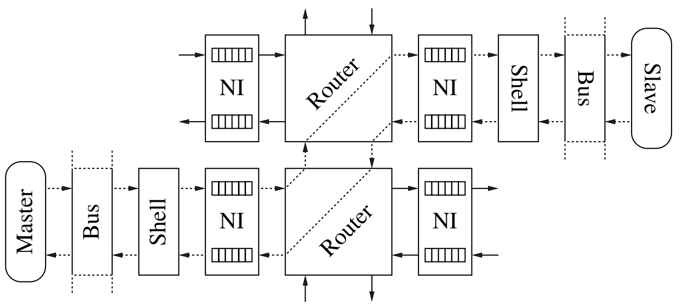

Figure 3. Example of an Æthereal NoC connection from a Master IP to a Slave IP.

We contribute a tested real-time SoC design flow that uses conservative simulation to verify on a per-trace basis if the system would meet it's tolerated deadline miss rate, as illustrated in figure 1. Hardware and software elements of the SoC are described separately before being compiled and linked into a conservative simulator of the system. Input data is loaded into the simulation at run-time, facilitating the production of multiple traces without the need to recompile the conservative simulator. In the following section we provide the description of a hardware MPSoC platform to which our technique has been successfully applied. In section VII we combine software case studies with the hardware platform and using our design flow we present their conservative application-level performance analyses.

IV. Hardware Platform

The hardware platform follows the predictable hardware template from [17] that combines components at the IP-level. Predictability of the platform is achieved through the deterministic execution-ordering and duration of processor instructions, facilitated by access to memories with deterministic access times via an interconnect that can provide guaranteed temporal upper-bounds on transaction latency and throughput. The work in this paper applies to the hardware template, not to a specific hardware instantiation, and the general theory should be applicable to other predictable hardware architectures. An example of a suitable hardware platform is a simple multi-core system with some distributed shared memory, connected by a bus that uses a TDM arbitration scheme.

The hardware platform to which we apply this technique is a more complicated multi-core system using multiple VLIW processing cores with local instruction and data memories, connected by an Æthereal NoC [15] to a distributed shared memory, as illustrated in Figure 2. A framebuffer and timer peripheral are also included in the design to facilitate visual output and measure run-times respectively. The scheduling of instructions for the VLIW cores is taken care of at compile time and the VLIW cores do not have any hardware-level optimisations that modify this ordering. The compiled code is used for both simulation and on the actual implementation of the platform facilitating direct comparison of timings. We have implemented this technique using an already existent cycle accurate Instruction Set Simulator (ISS) for the processing cores. The required monotonicity of the hardware platform for the conservative simulation method permits the use of conservative methods other than the use of cycle accurate ISS's for the processing cores.

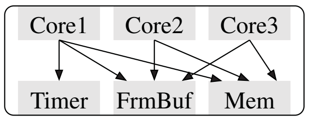

Figure 4. Example of an Æthereal NoC connection use case. Arrows indicate the connection orientation from master to slave.

The VLIW cores put MMIO transaction requests directly onto the Æthereal NoC. The Æthereal NoC connects the components together providing contention free routing, achieved using pipelined Time Division Multiplexed (TDM) circuit switching. Logical connections are set up from initiator (master) ports to target (slave) ports following a TDM schedule that is calculated off-line. An example of an Æthereal NoC use case can be seen in figure 4. Each arrow in figure 4 represents a single logical connection across the Æthereal NoC from a master port to a slave port. The TDM scheduling enables a logical connection's throughput and latency to be guaranteed independent of other connections as it cannot be affected by the network utilisation of other connections. The guarantees on throughput and latency for each logical connection allow each connection to be modelled individually.

We will now explain the physical transaction path across the NoC from a master port to a slave port. A transaction can be either a read or a write. The Æthereal NoC uses posted writes meaning that the IP on the master port may continue operation as soon as the transaction is accepted by the NoC without having to wait for an acknowledgement from the slave IP. By construction, the NoC in our implementation operates on a fixed word sized granularity. Figure 3 illustrates the request and response channels of a transaction path from master to slave and back again.

Taking a read request sent from a processing core (master) to a shared memory location (slave) as an example we will explain the hardware processes required to carry out this request. The processing core initiates the read transaction on the local bus where the address of the desired read location is decoded. The bus may have multiple outgoing logical connections, each represented by their own shell. Following the address decoding the appropriate shell is selected for the logical connection between the processor and the memory. The shell serialises the request, encapsulating the bus-level transaction. A read is encapsulated as two word sized headers and a write is encapsulated as two word sized headers plus one word of data payload. The serialised transaction is then passed to the Network Interface (NI) where it is buffered until it is scheduled for packetisation and injection onto the network.

Each logical connection is buffered separately to avoid interference with traffic from other connections. The buffers maintain a FIFO ordering of requests. Once a request reaches the head of the queue a TDM arbitration scheme in combination with credit based flow control decides when to inject the request onto the router mesh. Arbitration is carried out on a

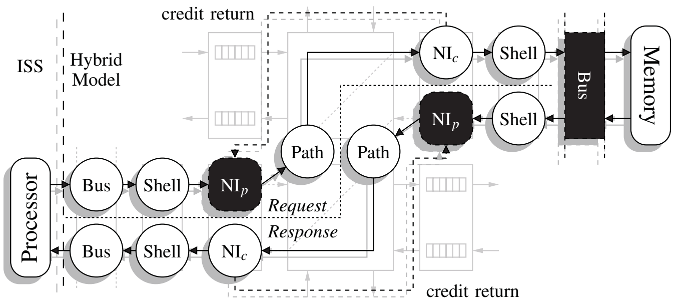

Figure 5. Representation of a virtual connection using a combination of formally modelled components (Black) and cycle accurate components (White).

word sized granularity meaning that a single read transaction requires two arbitrations and single write transaction requires three arbitrations, in the producing NI. The TDM scheme is configured to ensure that the request will reach its destination without being blocked. The credit based flow control system ensures that buffer overflow is avoided. To facilitate this a credit tally is kept at the producing NI that represents space in the connection's corresponding buffer in the consuming NI. When data is sent across the network the tally is deducted. Once there are no more credits, data cannot be transmitted until credits are returned from the receiving NI. The combination of TDM scheduling and credit based flow control ensures that a guaranteed bound on latency and throughput can be given per logical connection, from producing NI to consuming NI.

Once the request is injected onto the router mesh the request crosses the network making a fixed amount of hops for any given connection before being de-packetised and buffered again in the receiving NI. The request is buffered until it can be passed to the shell that is allocated to logical connection from the processor to the memory. The shell de-serialises the request and passes it to the local bus of the memory. The bus may have multiple incoming shells making an arbitration scheme necessary for access to the slave. In our implementation a Round Robin (RR) arbitration scheme is used to provide access to the slave, permitting access only if space is available in the logical connections buffer in the producing NI of the response channel. The RR arbitration scheme ensures that a guaranteed latency and throughput is maintained for access to the memory. Once scheduled for access to the memory the read request is serviced, retrieving the requested data.

The response path of the logical connection operates in the same manner as the request path. The response is buffered in the producing NI of the response channel where TDM scheduling and flow control are used again to ensure a guaranteed bound on latency and throughput per logical connection, from producing NI to consuming NI. Upon arrival at the consuming NI the response is buffered until it is de-serialised in the shell. Once de-serialised the retrieved data from the memory is passed back to the processor, completing the transaction. Due to the construction of the NoC and the arbitration schemes used, guaranteed upper bounds on latency and throughput can be given per logical connection, for the servicing of transactions.

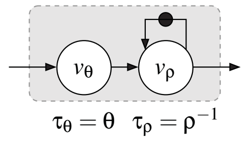

Figure 6. Dataflow representation of a Latency Rate (LR) server.

The Æthereal NoC also exhibits the required temporally monotonic behaviour for the application of our conservative simulation technique. In section V we show how the monotonic property of the NoC can be used in conjunction with the ability to model logical connections individually to create a hybrid model that can be used for transaction-level conservative performance analysis.

V. NoC Communication Model

In this section we explain how the individual NoC connections can be modelled conservatively using a hybrid of formal modelling techniques in combination with cycle accurate simulation.

The work in [10] describes how the Cyclo Static Dataflow (CSDF) [2] formal modelling technique can be applied to real-time MPSoCs, that use the Æthereal NoC, in order to provide conservative application-level performance analysis. In contrast to our method, in [10] it is required that applications are also modelled using the CSDF formal modelling technique so that conservative analytical bounds may be derived. A method for dataflow analysis [2] of the Æthereal NoC is described in [10] that models each connection individually as a request and a response channel. In [10] the entire request and response channels are modelled conservatively using a dataflow representation of Latency Rate (LR) servers [8], [9]. For our conservative simulation technique we use formally modelled LR servers in combination with cycle accurately simulated hardware components. By only modelling the runtime arbitration schemes using dataflow modelling we achieve a finer grained conservative abstraction level for each connection. This facilitates the production of tighter conservative guarantees.

An Æthereal connection, as illustrated in figure 3 may be modelled for use with conservative simulation as illustrated in figure 5. Many of the hardware components have response times that are unaffected by when they are used. These components are modelled cycle accurately and are represented in white in figure 5. Run-time arbitrated components have response times that are related to when they are used, e.g. the response time of a TDM arbitration depends on where in the table the scheduler is at the start of the arbitration. These components are formally modelled and are represented in black in figure 5. This model can be used in simulation to provide transaction-level guarantees that are valid for all arbitration phases and for all arbitration settings depending on how it is configured.

The TDM and RR arbitration schemes used in the Æthereal NoC enable upper bounds to be derived for their worst case temporal performance. While these arbitrations schemes may

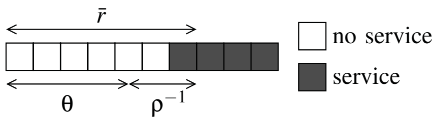

Figure 7. Example TDM arbitration table.

be conservatively modelled as a static delay of their worst case performance, LR servers enable *less pessimistic*, *temporally conservative* abstraction. This is achieved by characterising the temporal behaviour of the arbitration scheme as two values *Latency* and *Rate*, represented by θ and ρ respectively. Detailed rationalisation of the workings of LR servers is beyond the scope of this paper, and we direct you to [8], [9] for more details. LR servers may be represented conservatively as a dataflow graph [8], as illustrated in Figure 6. Dataflow actors [2] V_θ and V_ρ represent the *Latency* and *Rate* components of the LR server, respectively. The response time of the *Latency* and *Rate* dataflow actors are represented by τ_θ and τ_ρ respectively.

The dataflow actors can fire (start executing) as soon as tokens become available on all of their incoming edges. In figure 6 actor V_θ can fire as many times in parallel as there are tokens on its incoming edge, producing a token on its outgoing edge when its firing is complete. Actor V_ρ has a self edge with a single token meaning that only one firing of the actor may take place at any time. Once the token on the self edge has been consumed the actor V_ρ cannot fire again until it has finished firing and produced a token on its self edge. In figure 5 it is shown that the formal model used for the producing NI's (NI$_p$) is dependent on the credit return of the flow control just as the actual scheduling is dependent on the availability of credits. If no credits are available, indicating that there is no space in the buffer in the consuming NI (NI$_c$), then the formal model of the arbitration in the NI$_p$ will not start operating on any new data arriving in the buffer until credits are returned. This behaviour is in keeping with the physical implementation of the arbitration in the producing NI's.

In order to conservatively model run-time arbitrated components using LR server abstraction they must first be characterised as *Latency* and *Rate* values. In Figure 8 we contribute an automated toolflow for LR value derivation. TDM arbitration tables for use in the NIs are generated from the desired individual NoC connection parameters, such as bandwidth. Æthereal NoC resource allocation for individual NoC connection parameters is described in detail in [18]. The Latency and Rate values for individual connections are subsequently derived from the TDM tables. An example of a TDM table is illustrated in figure 7. The table in figure 7 is made up of slots of service and non-service. The table continuously cycles meaning that an element arriving for arbitration may have to wait until a slot where it can be serviced comes round again. The worst case response time $\bar{r}$ of the TDM table is the longest time that an element requiring would have to wait before it is scheduled. In the worst case a Round Robin arbitrations scheme also acts as a TDM table. From these TDM tables the Latency θ and Rate ρ values are calculated from the Worst

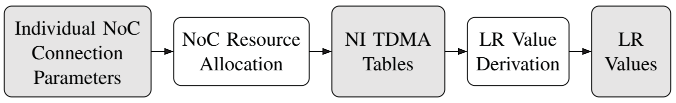

Figure 8. Tool flow for Latency Rate (LR) value derivation.

Case response time $\bar{r}$ of the table as follows.

$$\bar{r} = \theta + \rho^{-1}$$

The derivation of this equation is beyond the scope of this paper, for more information we refer you to [8]. The LR values θ and ρ can be computed prior to the simulation and stored in an XML document, for example, to be loaded by the simulation. This allows the simulation to be easily reconfigured to conservatively bound other arbitration settings. It is also possible to generate LR values that bound the worst case arbitration setting and hence the per-trace temporal bound produced by our simulation technique will be conservative for all arbitration settings and arbitration phases, e.g. generate the LR values for a maximum length TDM table that can be instantiated with the lowest possible utilisation. This is shown in table I for the formal modelling technique in [10] and conservative simulation as an optional property.

The LR values are used in the simulation to calculate the length of time data would have to wait for scheduling by the arbitration scheme in the context of data that was previously scheduled, as per the LR server model illustrated in Figure 6. FIFO buffers in the NIs are modelled so that the scheduling time of the previously scheduled element, by the LR server, is taken into account when scheduling the subsequent element. The absolute response time of an element E_0 entering a FIFO buffer to be scheduled can be calculated using E_0's start time in the FIFO, the previously scheduled element E_{-1}'s absolute response time and the LR values θ and ρ respectively.

$$responseE_0 = MAX\left(startE_0 + \theta, responseE_{-1}\right) + \rho^{-1}$$

Using the hybrid Æthereal NoC model in combination with cycle accurate processing core ISS's, we can now generate per-trace timing results that are *conservative for single processor systems*. The model is conservative at the hardware-level but the conservativeness does not automatically confer to the application-level for multiprocessor systems. In the following section we demonstrate that application-level conservativeness is not guaranteed because of hardware-level conservativeness and explain how we can give conservative application-level bounds in this case.

VI. Application-Level Conservativeness

In the previous sections we take the general idea of Conservative Simulation, show a hardware platform to which it is applicable and describe how our technique can be applied to this platform. The monotonic behaviour of the hardware platform is utilised to permit conservative abstractions of hardware components while preserving overall conservativeness. In

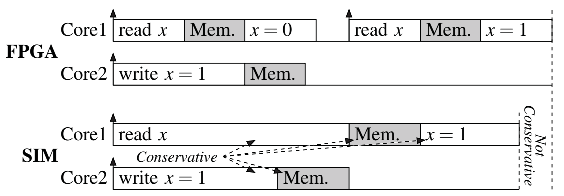

Figure 9. Timing analysis of inter-IP synchronisation modelled using conservative transaction times.

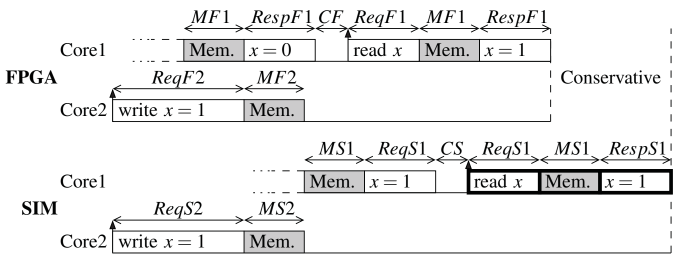

Figure 10. Timing analysis of inter-IP synchronisation modelled using our extra read technique.

order to provide conservative application-Level performance analysis of a MPSoC it must be possible to also conservatively bound the application-level behaviours of the system. In this section we show that conservatively modelled hardware is not enough to guarantee application-level conservativeness for the application-level behaviour of synchronisation. We subsequently explain how conservative application-level performance analysis can still be performed in this case.

In section V we describe how to model the Æthereal NoC for use with conservative simulation. Our description covers how to conservatively model individual NoC connections, such as those illustrated in figure 4, by incorporating formal modelling techniques to bound the temporal behaviour of run-time arbitrated components. This is sufficient to provide hardware-level or transaction-level temporal bounds. Consequently, individual transactions across the NoC can be simulated to generate conservative timings. However, the application-level act of synchronisation is not necessarily conservative. As we show in Figure 9 application-level conservativeness in simulation is not automatically conferred from transaction-level conservativeness in simulation.

Figure 9 illustrates a timing analysis of a two cores synchronising in a MPSoC, using a flag value (x) in shared memory. In this scenario *Core 1* reads the flag value in shared memory that *Core2* sets equal to 1. If the flag has not been set, *Core1* repeatedly reads the flag until it reads that the flag value has been set equal to 1. In the implementation of the system (**FPGA**) *Core 1* reads the flag just before *Core 2* sets the flag value equal to 1, forcing *Core 1* to read the flag again to receive the flag value that *Core 2* set. In simulation (**SIM**), even though all the transactions are conservative in comparison to the implementation, the overall act of synchronisation at the application-level is not conservatively bound. This may occur, as illustrated in Figure 9, when *Core 2* gains access to the memory first and sets the flag value equal to 1. *Core 1* subsequently reads the set flag value and does not perform another read unlike in the implementation version. The application-level act of inter-IP synchronisation is therefore not conservatively bounded. This is not a quirk of our particular implementation but a general observation that *conservative transaction-level modelling does not automatically confer conservativeness to the application-level.*

It is possible to bound synchronisations conservatively at the application level, by simply making *Core1* in the simulated version of the application read the flag one more time, upon reading that the flag value (x) set by *Core2* is now equal to 1, as illustrated in Figure 10. Since transactions are conservatively modelled, the absolute response time of the write from *Core2*, to set the flag value, is never less than the implementation. The worst case response time for the act of synchronisation occurs whenever *Core1* just misses reading the flag value that *Core2* sets and has to read again, as is illustrated in Fig 10 for the **FPGA**. The best case response time for the act of synchronisation occurs whenever *Core1* reads the flag value immediately after it has been set by *Core2*. Since individual transactions are conservatively bounded, it is sufficient for *Core1* in the simulation to read the flag again upon reading the set value in order to conservatively bound the act of synchronisation. This bounds the scenario that the implementation experiences the worst case response time, while the simulation experiences the best case response time, as is illustrated in Fig 10 for the **SIM**. It is necessary that there is only a fixed duration of computation between reads during synchronisation, otherwise the temporal behaviour of the synchronisation cannot be bounded using this technique.

Theorem 1: Conservatively modelled NoC transactions in simulation do not guarantee application level conservativeness.

Assumption 1: All timing values exist in the $\mathbb{R}_{\geq 0}$ domain.

Assumption 2: Simulated transaction times are never shorter than implementation transaction times. This is also true of the timed elements that the transactions are composed of.

Assumption 3: A simulation write will never start earlier than the same write in implementation.

Assumption 4: Simulation time between reads is never shorter than implementation time between reads.

Assumptions 1–3 are obvious observations of the properties of a conservative simulation. Assumption 4 is not inherently true in a conservative simulation. One method of ensuring that this is the case is by stipulating that only a fixed number of fixed duration instructions may be executed between reads for the specific case of synchronisation.

In order to maintain conservativeness;

$$SIM\ Sync\ Time \geq FPGA\ Sync\ Time$$

To show that reading an extra time in simulation, upon reading the desired value, maintains conservativeness this must be shown to be true in all cases. In order to bound all cases the worst case timing of the implementation must be bound in simulation.

As is illustrated in figure 10, the worst case for maintaining conservativeness of a synchronisation occurs when the core that is reading the synchronisation flag value (x), in the FPGA implementation, reads the flag value just before it is written requiring that the flag must be read again to receive the set value. In simulation (SIM) the core that is reading the flag value reads the flag value just after it has been set, and therefore the synchronisation is complete. Even though the transaction lengths are conservatively bounded the act of synchronisation has a shorter duration in simulation (SIM) than in the FPGA implementation. Reading one extra time in SIM, even though it is functionally unnecessary, conservatively bounds the synchronisation of the FPGA implementation. This produces the following synchronisation timings, using the naming from Figure 10:

$$\begin{aligned} SIM\ Sync\ Time = \\ ReqS2 + MS2 + MS1 + RespS1 + CS + ReqS1 + MS1 \\ + RespS1 \end{aligned}$$

$$\begin{aligned} FPGA\ Sync\ Time = \\ ReqF2 + RespF1 + CF + ReqF1 + MF1 + RespF1 \end{aligned}$$

Conservativeness of a simulation is lost if the timings in simulation are less than the timings in the implementation. The worst case for maintaining conservativeness therefore occurs when SIM transactions and FPGA transactions are equal in duration. Therefore cancelling out equivalent terms:

$$\begin{aligned} \cancel{ReqS2} + MS2 + MS1 + \cancel{RespS1} + \cancel{CS} + \cancel{ReqS1} + \cancel{MS1} \\ + \cancel{RespS1} \geq \cancel{ReqF2} \\ + \cancel{RespF1} + \cancel{CF} + \cancel{ReqF1} + \cancel{MF1} + \cancel{RespF1} \end{aligned}$$

Leaving

$$MS2 + MS1 \geq 0$$

This is always true due to Assumption 1; memory access times cannot be negative. Therefore reading an extra time in simulation, after reading the desired synchronisation flag value, will always cause the simulation synchronisation to conservatively bound the implementation synchronisation. This method is not tightly conservative and will bound the synchronisation plus the simulated memory access times for Core 1 ($MS1$) and Core 2 ($MS2$).

To ensure that synchronisation is performed in this manner, we suggest incorporating the extra complexity in a communication API and stipulate that all inter-IP synchronisation should be carried out using it. Rectifying the inter-IP synchronisation conservativeness at the transaction-level in the simulation model adds the complication of being able to distinguish between synchronisation reads and communication reads. The abstract classification of the reads as synchronisation reads originates at the application-level. We propose an application-level solution using an application-level communication protocol such as C-HEAP [19], in order to avoid the necessity for such a classification.

The C-HEAP Application-Level Communication API works well for streaming applications, i.e. signal processing applications, and may be modified to incorporate our extra read technique to maintain application-level conservativeness. The C-HEAP API uses explicit synchronisation followed by communication via circular buffers to communicate data between distributed IP. The synchronisation stage involves reading a pair of counters that represent buffer occupancy. The C-HEAP API may be made suitable for use with our extra read technique by simply modifying the synchronisation stage to read the counters an extra time upon reading that the counter values have changed. The communication stage can be performed as normal because the explicit synchronisation stage ensures that only one core accesses a particular area of the circular buffer at a time.

VII. Case Studies

In this section we will describe how our conservatively modelled simulation platform is applied in practice and evaluate our technique using case study analysis. Our case studies will consist of a synthetic application and an actual soft real-time application in the form of a JPEG decoder.

For the purposes of our experiments we use the multiprocessor simulator from the Silicon Hive HiveCC Software Development Kit [20]. The Silicon Hive multiprocessor simulator facilitates the use of our conservative simulation based design flow, illustrated in figure 1. The hardware platform from section IV is implemented in FPGA and modelled using cycle accurate Instruction Set Simulators (ISS) for the processing cores and the conservative model described in section V for the connections of the Æthereal NoC. The latency rate values required for the NoC connection models are derived from the arbitration settings used in the FPGA implementation following the flow illustrated in figure 8. The hardware needs only to be described and compiled once for use with multiple applications.

Our case study analysis of our conservative simulation technique is carried out for different applications that are mapped to the example platform. The applications are mapped onto the platform manually ensuring that the conditions required for conservative application level performance analysis are met, as described in section VI. The applications are compiled and linked with the hardware model creating a conservative simulation model of the entire real-time MPSoC. By running the same applications on both the FPGA and in conservative simulation, and by using the same input data sets it is possible to perform a direct comparison of the temporal results of the simulation with the actual implementation. We compare the results of the FPGA implementation with two different hybrid simulation models. One of the simulation models represents the runtime arbitration using a static Worst Case (WC) bound. The other simulation model uses the Latency Rate (LR) dataflow model as illustrated in figure 6, to bound the arbitration scheme.

Our first case study demonstration makes use of a synthetic application that enables the timing of an 8 kbyte array data transfer to and from distributed shared memory. The array is

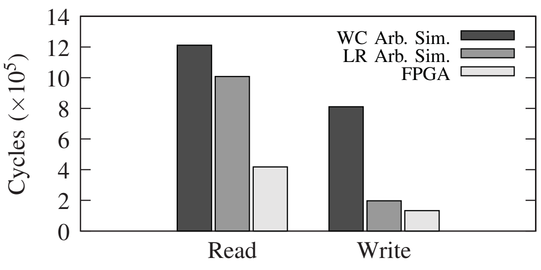

Figure 11. Array transfer case study results.

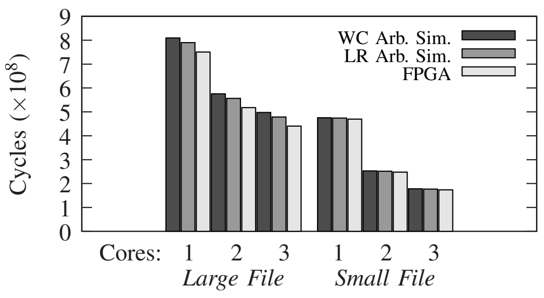

Figure 12. JPEG decoder case study results for a 3 core MPSoC.

written to the remote memory location one word (32 bits) at a time, and read back in the same manner. We do this because our NoC by construction works on a word size granularity. The results illustrated in Figure 11 compare the application timing results for both simulation models and execution on an actual implementation of the system. When comparing the simulation times of the two simulation models, the simulation that uses the LR dataflow model for the arbitration gives a tighter conservative bound than the static WC arbitration. We can also see that the LR simulation time for writing the array in figure 11, shows a higher accuracy compared to reading the array, in comparison to the timings on the FPGA implementation. This is to be expected, as the NoC's ability to perform posted write transactions enable the processor to continue processing as soon as the data to be transmitted is put on the NoC's local Bus interface.

Since the processor can continue processing, it may send multiple writes one after the other onto the NoC. The Latency Rate (LR) servers that are used to model the run-time arbitration provide tighter conservative bounds if kept busy. The Latency component (v_θ) delay of the LR server model illustrated in figure 6, allows multiple transactions to be delayed at the same time. The pipelining of the Latency delay allows for tighter conservative bounds for streaming transactions. The static WC arbitration simulation for writing the array can be seen to be much more conservative than the LR arbitration simulation in figure 11 for this reason.

Read transactions can only be carried out one at a time since the processor has to wait for the data to be returned before it can continue processing. A series of read transactions are therefore not able to take advantage of the tighter conservative bounds of the LR server like write transactions can. The difference between the WC arbitration simulation and the LR arbitration simulation is caused by how a read transaction is physically represented. In the Æthereal NoC, as illustrated in figure 3, the shell encapsulates the read transaction as two word sized headers. The arbitration in the Network Interface works on a word sized granularity meaning that both words for the same transaction get scheduled separately. In the LR arbitration simulation this means that the second word of the read transaction can take advantage of the pipelining of the Latency component of the LR server. The WC arbitration simulation bounds both words of the read transaction using the worst case temporal bound, causing this simulation to be less tightly conservative than the LR arbitration simulation. Most importantly for conservative application-level analysis, from figure 11 it can be seen that regardless of accuracy all of the simulation models temporally bound the FPGA implementation.

We will now subsequently demonstrate the applicability of this technique to an actual soft real-time application in the form of a JPEG decoder. The JPEG decoding algorithm has much in common with video decoding algorithms such as MPEG. For this case study we start with the code of an already existent unparallelised JPEG decoder. The JPEG application is parallelised in a data-partitioned fashion, on a Minimum Coded Unit (MCU) granularity [21]. In keeping with our design flow, illustrated in Fig 1, the JPEG application needs only to be mapped to the hardware in order to perform conservative simulation. The data partitioning of the JPEG application means that the entire JPEG decoding algorithm is mapped onto each processor. The mapped JPEG application is able to be simulated conservatively without the need to formally model the application. The JPEG decoding algorithm has an input data dependent execution. The experiment is repeated for two different input images using one, two and three cores to decode the images. The two images both have the dimensions 1024×768 but have different file sizes after JPEG encoding. The JPEG algorithm's Variable Length Decoding (VLD) step is not easily parallelised and as such each core carries out this step in its entirety. MCU's are assigned for decoding to each of the cores following a modulo counting scheme.

Figure 12 illustrates the timing results for both WC and LR arbitration simulations, and execution on an actual system implementation on FPGA. In figure 12 the expected diminishing return of adding extra cores can be seen for both the large and small JPEG file. This result is in keeping with Amdahl's law. More importantly for our conservative simulation technique these results show an accuracy of the conservative simulations to within 10% of the actual runtime. These results also concur with our findings in the array transfer case study; that writes enable more accurate simulation results. The smaller file has a greater proportion of writes to reads than the larger file and subsequently demonstrates more accuracy in the simulation results. The results in figure 12 also show that the LR arbitration simulation is more tightly conservative than the WC arbitration simulation, which also concurs with our findings in the array transfer case study.

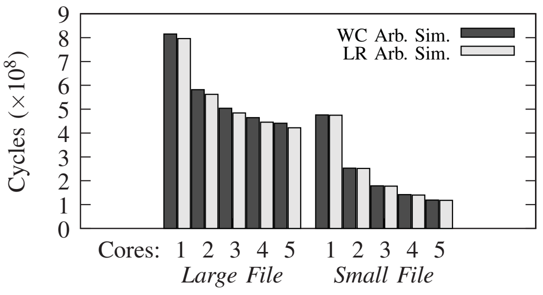

Figure 13. JPEG decoder case study results for a 5 core MPSoC hardware description.

The usefulness of the conservative simulation technique for soft real-time application-level performance analysis can also be seen from the results in Figure 12. The produced timing results from our technique are not only accurate but also conservative providing application-level, per-trace temporal bounds through simulation. Even though our case study only shows our conservative simulation method being applied to 3 processing cores, our method is not bounded to an upper limit of processing cores. By modifying the simulation hardware description to have more cores, it is possible to map the application to the modified simulation platform and produce conservative per-trace timings without actually implementing the platform. We demonstrate this possibility by mapping the JPEG application onto a hardware description of an MPSoC with 5 processing cores instead of 3. The results of these simulations are presented in figure 13. Even though we do not implement the 5 core hardware platform on the FPGA, by using our conservative simulation technique we are guaranteed that if the platform was implemented the timings produced by these traces would be less than our simulation timings.

Our case study analyses show that our hybrid modelling technique, that combines cycle accurate simulation with formal analysis in order to provide per-trace temporal bounds for all arbitration phases of a real-time MPSoC, can be applied practically. We show that our conservative simulation technique can provide conservative application-level performance analysis without the need to formally model the application. In contrast to regular simulation techniques, we are able to guarantee for soft real-time applications that the deadline miss-rates, for the set of conservatively simulated traces, can only be less when executed on the physical implementation.

VIII. Conclusion

Application-level performance analysis of real-time applications mapped onto a MPSoC are necessary to ensure that the applications meet their deadline miss rate. This can be achieved through a formal modelling approach, although this restricts the application to specific behaviours in order to fit the model, and requires significant effort. Soft real-time applications, such as a video decoder, may miss some deadlines without much of a detriment to their perceived performance. In these instances we propose a conservative simulation approach to Application-Level Performance Analysis on a per-trace basis.

We propose using a hybrid modelling approach to model a predictable hardware platform, enabling per-trace conservative application-level performance analysis through simulation. Our method requires very little effort to modify applications beyond mapping them to the platform. We demonstrate the technique's applicability to performance analysis of soft real-time applications by conservatively simulating a JPEG decoder. Analysing the results of the JPEG case study, we find that the simulation is conservative and accurate to within 10%. We conclude, based on the complexity of the technique and the case study results, that Conservative Simulation is a feasible alternative to Formal Analysis for Application-Level Performance Analysis of soft real-time applications.

References

[1] G. Buttazzo *et al.*, *Soft Real-Time Systems Predictability vs. Efficiency*. Springer US, 2005.
[2] G. Bilsen *et al.*, "Cyclo-static dataflow," *IEEE Trans. on Sig. Proc.*, vol. 44, no. 2, 1996.
[3] L. Thiele *et al.*, "Real-time calculus for scheduling hard real-time systems," in *Circ. and Sys., 2000.*, vol. 4, 2000.
[4] M. Bekooij *et al.*, "Dataflow analysis for real-time embedded multiprocessor system design," in *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*. Springer, 2006, ch. 4.
[5] L. Benini *et al.*, "MPARM: Exploring the multi-processor soc design space with systemc," *J. VLSI Signal Process. Syst.*, vol. 41, no. 2, 2005.
[6] J. Cong *et al.*, "MC-Sim: An efficient simulation tool for MPSoC designs," *ICCAD*, 2008.
[7] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, 1987.
[8] M. Wiggers *et al.*, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *SCOPES*, 2007.
[9] S. Dimitrios *et al.*, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, 1998.
[10] A. Hansson *et al.*, "Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis," *IET Comp. & Digital Techn.*, 2009.
[11] F. Ophelders *et al.*, "Intra- and inter-processor hybrid performance modeling for MPSoC architectures," in *CODES+ISSS*, 2008.
[12] S. Künzli *et al.*, "Combining simulation and formal methods for system-level performance analysis," in *DATE*, 2006.
[13] M. Krause *et al.*, "Combination of instruction set simulation and abstract RTOS model execution for fast and accurate target software evaluation," in *CODES+ISSS*, 2008.
[14] E. Moreno *et al.*, "Integrating abstract NoC models within MPSoC design," in *RSP '08*, 2008.
[15] K. Goossens *et al.*, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Des. and Test of Comp.*, vol. 22, no. 5, 2005.
[16] B. Akesson *et al.*, "Real-time scheduling using Credit-Controlled Static-Priority arbitration," in *RTCSA*, 2008.
[17] A. Hansson *et al.*, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, 2009.
[18] ——, "Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip," in *DATE*, 2007.
[19] A. Nieuwland *et al.*, "C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of the embedded signal processing systems," *Des. Auto. for Emb. Syst.*, vol. 7, no. 3, 2002.
[20] Silicon Hive, website, `http://www.siliconhive.com`.
[21] G. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, no. 4, 1991.