Analysis of Memory Latencies in Multi-Processor Systems

J. Staschulat, S. Schliecker M. Ivers, R. Ernst
Technical University of Braunschweig
Hans Sommer Str. 66, D-38106 Braunschweig, Germany
{staschulat|schliecker|ivers|ernst}@ida.ing.tu-bs.de

Abstract

Predicting timing behavior is key to efficient embedded real-time system design and verification. Current approaches to determine end-to-end latencies in parallel heterogeneous architectures focus on performance analysis either on task or system level. Especially memory accesses, basic operations of embedded application, cannot be accurately captured on a single level alone: While task level methods simplify system behavior, system level methods simplify task behavior. Both perspectives lead to overly pessimistic estimations.

To tackle these complex interactions we integrate task and system level analysis. Each analysis level is provided with the necessary data to allow precise computations, while adequate abstraction prevents high time complexity.

1 Introduction

Memory is a critical bottleneck in embedded systems, as the gap between processor speed and memory access time is increasing. Current chip designs use hierarchical memory architectures with caches, multi-threading and co-processors to reduce memory latency time. Embedded applications often require real time constraints, but performance verification of complex systems is a challenge.

State-of-the-art in industrial practice is using functional test and simulation. Simulation times are often too long for a complete code coverage, which would need exponential time. Therefore, only critical paths are tested, and safe timing bounds cannot be given.

Formal analysis is an alternative. With simplified assumptions the analysis complexity is reduced and safe upper and lower performance bounds are calculated. One such assumption is a constant memory latency, even though it is influenced by the memory hierarchy, bus arbitration, and buffer sizes as well as the background memory latency. Static analysis approaches for single tasks assume constant memory access time, such as [5]. The behavior of caches has been studied as well [5] [3] [8], but these approaches assume a constant cache miss penalty. A small overestimation of memory access time will lead to a high overestimation for worst case execution time of single tasks.

This overestimated task execution time is used for higher level system analysis for resource scheduling or for throughput estimation of heterogeneous multi-processor architectures. Such analyses have been proposed by [7] [6]. Compositional performance analysis methodologies combine local techniques on the system level by connecting their input and output event streams according to the overall application and communication structure [4] [1].

Crowley and Baer propose in [2] an analysis for multi-threaded processors. They identify parallelism in the execution of individual threads on a processor and use special nodes with a negative execution time to model the gain of multithreading.

While task level methods simplify system behavior, system level methods simplify task behavior. Both perspectives lead to overly pessimistic estimations, one reason for the marginal influence of formal methods in industrial system design. To tackle these complex interactions task level and system level analysis are integrated. Each analysis level is provided with the necessary data to allow precise computations, while adequate abstraction prevents state space.

The paper is structured as follows. Section 2 describes the problem statement and Section 3 reviews previous work. Section 4 describes the integrated task and system level analysis approach. Finally, we conclude in Section 6.

2 Problem Statement

A simple multi-processor architecture is given in Fig 1. Two processors are connected by a shared bus with memory and a co-processor. For example, suppose that on CPU_1 runs a heat control application. A sensor on CPU_2 periodically transmits temperature values, which are saved in the shared memory. The application on CPU_1 checks this value but also loads its instructions and other data from this memory device. The heat control is managed by the co-processor, which is triggered by the application on CPU_1.

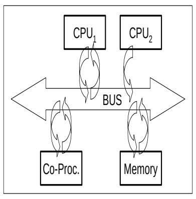

Figure 1. Multi-processor design example.

This simple setup already shows the analysis complexities: Memory traffic by the CPU_1 and CPU_2 uses a shared bus. The memory access time depends on the system state, including bus state, buffers and memory state. We will call a memory or co-processor request latency of a resource as *transaction latency* in this paper. This denotes the end-to-end latency, including the time for requesting the data, transmission over the bus, processing on the remote component and transmission to its source.

The objective is to compute the worst case execution time (WCET) of a task, e.g. the heat control on CPU_1, while analyzing transaction times on system level.

3 Previous Work

3.1 Single Task Analysis

The timing analysis of tasks is separated in two stages: program path analysis and micro-architecuture modeling. Program path analysis is used to determine the path which is executed in the worst case. To derive all the possible program paths the program is transformed into a control-flow graph. Based on this control-flow graph the worst case path which starts at the beginning node and ends at an terminating node of the control-flow graph is determined.

Micro-architectural modeling is understood as the timing analysis of sequences of instructions. Li and Malik [5] integrate program path analysis and micro-architecture modeling to analyze the worst case execution time of tasks under the influence of instruction caches. They use an Integer Linear Programm (ILP) to avoid an explicit enumeration of all program paths. The micro-architectural model is used to derive execution-time of individual basic-blocks and the ILP is used to find the path through the control-flow graph which maximizes the execution time spent. In [8] means are proposed for identifying infeasible paths in the program and the timing analysis is based on single-feasible-paths instead of basic-blocks.

In this paper we use this analysis framework, called SymTA/P, to transform a program into its control flow graph and to compute the WCET by solving the ILP. We assume that the execution time of single-feasible paths are given including micro-architectural influences. SymTA/P assumes that a conservative memory access time, either as cache miss penalty or general memory access time, must be specified a priori. A more precise estimation of the worst case memory access time should rather be done at system level, where timing properties of all other components are available.

3.2 System Level Analysis

In SymTA/S [4] a compositional performance analysis methodology is proposed which integrates different local scheduling analysis techniques through event stream propagation. The local techniques are composed on the system level by connecting their input and output event streams according to the application and communication structure. Instead of considering each event individually, as simulation does, the formal scheduling analysis abstracts from individual events to event streams. The analysis requires only a few simple characteristics of event streams, such as an event period, a maximum jitter and minimum distance. From these parameters, the analysis systematically derives worst case scheduling scenarios.

One way to extend the compositional analysis to memory accesses would be to model each access as an event. However, this would require splitting the task into many smaller atomic tasks and it would lead to a complex task description.

4 Integrated Analysis Approach

Our approach integrates the single task analysis and global system analysis. Fig. 2 shows the workflow of the integrated analysis. The global system analysis consists of tasks and event streams. A task i consumes tokens from an input event stream, E_i^{in} and produces tokens for the output event stream E_i^{out}. The event streams are connected to tasks according to the overall application.

Memory access are modeled with additional communication between task and system level. The task requests a number of memory transactions, T_{mem} from the system level (dashed arrow). The memory request is propagated via a system path. Based on the event model of the remaining tasks, the end-to-end latency of this request $L(T_{mem})$ is computed by the system level analysis. This latency is used by the task level analysis to compute the worst case response time (WCRT), which finally determines its output event model E_i^{out}. All tasks of the system which share event streams on this path, or which are directly affected by the memory access, have to be re-analyzed in the next iteration loop. The analysis will stop when a fixed point is found.

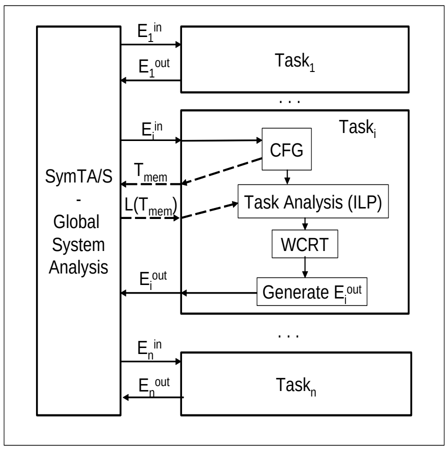

Figure 2. Workflow of integrated analysis.

This abstract modeling allows several compositional implementations. From task level only the memory requests need to be specified. This means any task level analysis which can determine the memory access patterns can be used here. In the next section we describe in detail how we extend our single task analysis. In Section 4.2 we present an extension to our system level analysis, which implements this general framework.

4.1 Extension of Single Task Analysis

Worst case execution time analysis is based on the control-flow graph (CFG) of an application. To include compiler optimizations, the assembly code is parsed and the corresponding CFG is constructed. At this stage we assume that the core execution time has been computed. For each basic block (or single-feasible pathes) the number of memory accesses is extracted. For example, if a basic block i requests three 32 byte memory blocks, then the transaction request is $T_{mem}(b_i) = (32,32,32)$. For each basic block such a request is generated. In the second step all requests are collected and propagated to the system level analysis where the transaction latency $L(T_{mem}(b_i))$ considering the system state is computed. This is described in Section 4.2.

The worst case execution time is computed by solving the ILP, which is constructed from the control flow graph. The general ILP consists of an objective function and several structural and functional constrains. Structural constrains represent possible control flow and functional constraints describe loop bounds or denote infeasible pathes. The sum-of-basic blocks, as proposed by Li and Malik [5] is given by the following objective function: $\max \sum_{i=1}^{n} c_i \cdot x_i$ where c_i denotes the core execution time of basic block i, n the maximum number of basic blocks of the task and x_i the execution count of basic block i. We extend the objective function to $\max \sum_{i=1}^{n} (c_i + L(T_{mem}(b_i))) \cdot x_i$ to include the memory access time of $T_{mem}(b_i)$ memory access during the basic block. Further below we omit b_i and use only the term T_{mem} for simplicity reasons. The ILP is then solved to find the longest execution path in the program.

4.2 System-Level Analysis of Transaction Latency

The basic SymTA/S model of [4] is composed of tasks and event streams. In the following we show how to extend the SymtTA/S framework to compute $L(T_{mem})$. A transaction starts at some task τ_i and is transmitted via a chain of intermediate tasks and ends at the same task τ_i. The total latency of this transaction is denoted by $L(T_{mem})$.

A request T_{mem} has to be translated into some event stream, which SymTA/S supports. An event stream is a tuple $E = \{\mathcal{P}, \mathcal{J}, d_{min}\}$, where $\mathcal{P}$ denotes the period, $\mathcal{J}$ the jitter and d_{min} the minimum distance between two events. Given these parameters, the system level analysis calculates the worst case response time $R(E)$ for this event stream E along the chain considering every other event stream of the system, which relates to this chain. In this paper we assume that the transactions of other ressources are given and are independent of system behavior. From the response time $R(E)$ the response time of the transaction $L(T_{mem})$ is calculated. First, we give a translation for single transactions. In a second step we translate multiple transactions.

4.2.1 Single Transaction

The idea for a single transaction, such as $T_{mem} = \{32\}$, is to restrict an event stream to a single event by choosing a period greater than the jitter. If the period were smaller then the jitter the second event of the event stream could arrive together with the first one. We define the event stream E for a single transaction by $\mathcal{P} = \mathcal{J}'$, $\mathcal{J} = 0$ and $d_{min} = 0$. The minimum distance is zero because this parameter is not used. The jitter is set to zero, to exclude any future events. As period we choose a large number, e.g. $\mathcal{J}'$. If the jitter along the chain, $\mathcal{J}^{max}$, is larger than the initially assumed $\mathcal{J}'$, the period will be adjusted to $\mathcal{J}^{max} + 1$ and the system analysis is called for a second time.

4.2.2 Multiple Transaction

Multiple events, such as $T_{mem} = \{32,32,32\}$, are modeled as a burst, this means all requests are issued at the same time instant. As period $\mathcal{P}$ we start with a great value, $\mathcal{J}'$. The jitter is set to $\mathcal{P} \cdot (|T_{mem}| - 1)$ to guarantee that exactly $|T_{mem}|$ events arrive together. Future events will be excluded if the period is greater than the total jitter $\mathcal{J}^{max}$ on the chain. Unfortunately, $\mathcal{J}^{max}$ is only available after the first system level analysis iteration. So possibly a second iteration is necessary. We define E for multiple transactions by $\mathcal{P} = \mathcal{J}'$, $\mathcal{J} = \mathcal{P} \cdot (|T_{mem}| - 1)$ and $d_{min} = 1$.

We assume a minimum distance of one instruction to be conservative. In the future we will analyze the minimum distance between memory accesses within basic blocks to increase analysis precision. We also restrict each event to request the same amount of data for simplicity reasons. For instruction caches this is not a limitation, since entire cache lines are requested.

Now the system analysis can compute the response time $R(E)$ for single and multiple events. In case of single events this is equal to the response time of the transaction $L(T_{mem}) = R(E)$. In the case of multiple events the $R(E)$ denotes the response time of that event, which possesses the worst case response time among all events of this event stream. This includes latencies of previous events. Because it is unknown which event caused the worst case response time, we have to assume that it is the last one. Therefore

$$L(T_{mem}) = (|T_{mem}| - 1) \cdot d_{min} + R(E) \qquad (1)$$

The first term denotes the time until the last event is activated. This completes the description of system chain analysis. All tasks belonging to this chain need to be reanalyzed, since their input event model might have changed due to the transaction.

5 Experiments

We applied the analysis to the architecture as shown in Fig. 1. For comparison, we use a simulation environment for a network processor and an isolated approach, where every transaction is assumed to take the maximum time observed during the simulation.

Eight applications are executed on CPU_1. For simplicity we provided some background traffic for CPU_2 and the the Co-processor. Figure 3 compares the worst case response times for constant memory access time (isolated), the WCRT of the analysis described in this paper (integrated) and the maximum WCRT observed in simulation.

The analysis provides a significantly reduced WCRT compared to constant memory access times. As simulation cannot provide the worst case scenario we cannot evaluate the accuracy using simulation.

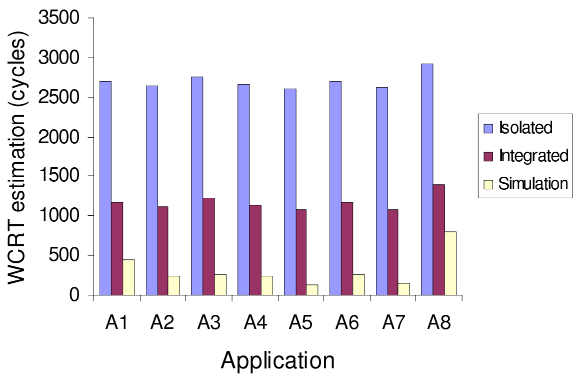

Figure 3. Experiment for WCRT analysis.

6 Conclusions and Future Work

In this paper we have integrated a static timing analysis on task level and a schedulability analysis on system level. Current approaches focus only on one level which assume a constant delay for each memory access. In this approach several memory transactions within a basic block are grouped together. In a second step, the total access time of these transactions is determined by system level schedulability analysis. The experiment shows that the analysis precision increases significantly compared to the isolated approach.

Future research includes to consider the type and size of a memory transaction as well as as a greater minimum distance of memory accesses beyond basic blocks.

References

[1] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Design, Automation, and Test in Europe*, Munich, Germany, March 2003.

[2] P. Crowley and J.-L. Baer. Worst-case execution time estimation for hardware-assisted multithreaded processors. In *The Second Workshop on Network Processors*, Anaheim, California, USA, Feb. 2003.

[3] C. Ferdinand and R. Wilhelm. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Systems*, 1999.

[4] M. Jersak, K. Richter, and R. Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC*, 2004.

[5] S. Malik and Y.-T. S. Li. *Performance Analysis of Real-Time Embedded Software*. Kluwer Academic Publishers, 1999.

[6] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *CODES*, Eates Park, Colorado, USA, May 2002.

[7] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time systems. *Journal of Real-Time Systems*, 6(2):133–152, March 1994.

[8] F. Wolf, J. Staschulat, and R. Ernst. Hybrid cache analysis in running time verification of embedded software. *Design Automation for Embedded Systems*, 7(3):271–295, Oct 2002.