Trace-Driven Performance Estimation of Multi-core Platforms

Kyoungwon Kim
Center for Embedded Computer Systems
University of California Irvine
Irvine, CA, 92697
kyoungk1@uci.edu

Daniel D. Gajski
Center for Embedded Computer Systems
University of California Irvine
Irvine, CA, 92697
gajski@uci.edu

ABSTRACT

In hardware/software co-design for a multi-core platform, a crucial role is played by fast and accurate early performance estimation. One type of such estimation that is as fast as native simulation, cycle-approximate and applicable to both software and custom hardware is Transaction Level Model (TLM) Estimation that depends on TLM simulation. For every platform selection and mapping, however, the entire platform must be simulated. The simulation overhead is reduced by Trace-driven estimation but such estimation is not applicable to custom hardware and often requires Cycle Accurate Models (CAMs), which may not available for the whole platform.

In this paper, we present Trace-Driven Performance Estimation (TDPE) of multi-core designs. TDPE is a trace-driven estimation but applicable to both software and hardware and requires no CAM. Since TDPE includes a mere single functional simulation, it is orders of magnitude faster than TLM Estimation. TLM Estimation is cycle-approximate by considering the data path of each Processing Element (PE), memory hierarchy, RTOS scheduling and overheads, bus arbitration and overhead. TDPE takes all of them into account so is as accurate as TLM Estimation.

Categories and Subject Descriptors

[**Hardware/software co-design**]

1. INTRODUCTION

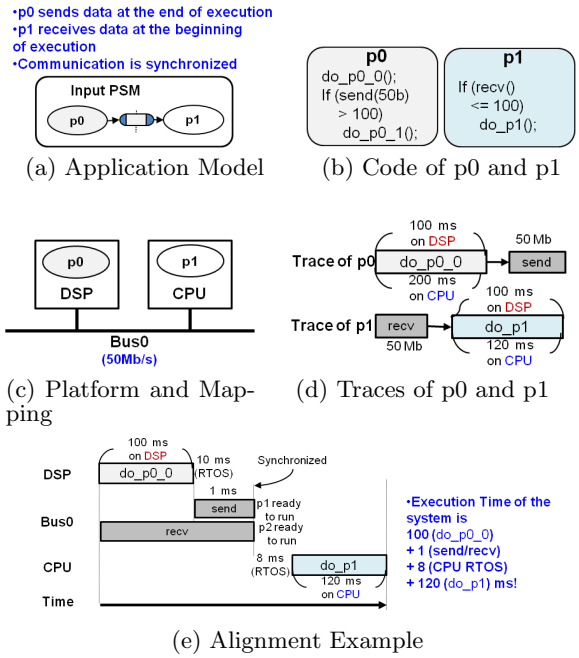
To tackle the ever-increasing design complexity and performance demands of contemporary and/or future applications, more and more embedded system designs involve heterogeneous multi-core designs. In the hardware/software co-design of such embedded systems, it is crucial that designers select optimal platforms and mapping of the given application. Such decisions are made in the early design stages, making it mandatory to have, for the given design choices, fast and accurate early estimation of performance.

An estimation technique can be evaluated in terms of speed, accuracy, and generality. Such techniques include Cycle Accurate Model (CAM) estimation, trace-driven estimations, and TLM estimations. CAM estimations use execution of CAMs that are highly accurate. CAMs, however, may not be available for the whole platform, so it is less general, and the estimation speed is insufficient for efficient design-space exploration. Trace-driven estimation is both fast and accurate but often requires CAMs and is limited in generality because it is not applicable to custom hardware. TLM Estimation estimates the design decisions by using TLM simulation. TLM Estimation is cycle-approximate, general, and as fast as native simulation.

However, room for improvement is available regarding speed. For every platform selection and mapping, the unnecessary task of timing annotation and simulation must be conducted. Note that there are many applications in which the execution path of each task is not affected by platform selection and mapping. Typical examples include multimedia applications captured in Synchronous Data Flow (SDF) models.

For such application domains, we propose a new Trace-Driven Performance Estimation (TDPE), one that is orders of magnitude faster than TLM Estimation, is applicable to both software and custom hardware, and verges on being cycle-approximate. Trace generation is conducted at the Transaction Level instead of the Cycle Accurate Level. Each event in the traces is annotated with delay estimates based on existing cycle-approximate techniques [1] [2] that are applicable even to custom hardware. The events in the traces are aligned instead of being simulated. Alignment is to place each event in the traces at the right location in the global time line. The required information is sufficiently given ahead of mapping and platform selection. During alignment, abstract models at the Transaction Level—such as RTOS, memory hierarchy, bus protocol, and Processing Element (PE) models—are emulated so accuracy approaches that of TLM Estimation. Experiments show our TDPE to be as accurate as TLM Estimation while orders of magnitude faster.

The rest of this paper is organized as follows. Section 2 reviews previous works. Section 3 explains TDPE and its limitations. Section 4 presents a case study with an MP3 decoder and 4 different platforms. Multiple different configurations are given to each platform as well. Finally, Section 5 offers our conclusions.



(a) Application Model (b) Code of p0 and p1

(c) Platform and Mapping (d) Traces of p0 and p1

(e) Alignment Example

Figure 1: Trace-Driven Performance Estimation

2. PREVIOUS WORK

Over the past few decades, multiple studies have explored the early performance estimation of multi-core designs. Each study should be evaluated in terms of speed, accuracy and generality. Estimation based on static analysis, proposed by Russell and Jacome [3], is fast but not at the cycle level. CAM Estimations using CAMs such as Instruction Set Simulation (ISS) models [4] [5] are highly accurate but slow and, since CAMs may not be available for the whole platform, not sufficiently general. [1] [2] [6] presented TLM Estimation. Once a platform is selected and mapping has been given, each basic block of the application source code is annotated with a single delay estimate. TLM is generated, native compiled and simulated to estimate the design decisions. Abstract models such as RTOS, memory hierarchy, Processing Elements (PEs) and bus protocol models are used to improve the accuracy. This estimation is as fast as native simulation, cycle-approximate, re-targetable, and, since the estimation does not require CAMs, general. The estimation speed, however, is slowed down because for every single combination of mapping and platform, annotation and simulation are mandatory. Such annotation and simulation overhead can be reduced with trace-driven estimations, such as in [7] [8], but these are not sufficiently general, as indeed, trace generation often, for accuracy, requires CAMs. Moreover, these techniques are inapplicable to custom hardware.

The proposed approach is trace-driven and thus minimizes the annotation and simulation overhead that comes with TLM Estimation. Moreover, our alignment places the events in the traces without simulation. Abstract models in TLM are emulated in the same way during the alignment. Thus, compared to TLM Estimation, the proposed approach is orders of magnitude faster and nearly as accurate. And compared to existing trace-driven estimations, the proposed approach is applicable to both software and hardware and does not require CAMs.

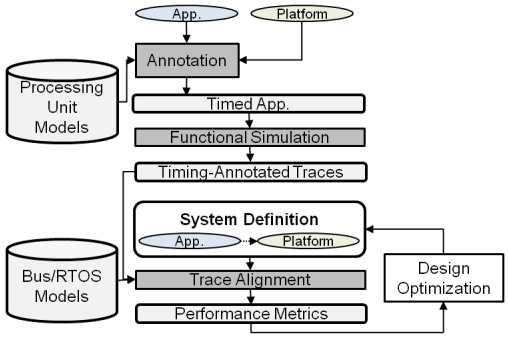


Figure 2: Design Flow with Trace-Driven Performance Estimation

3. TRACE-DRIVEN PERFORMANCE ESTIMATION

3.1 Assumptions

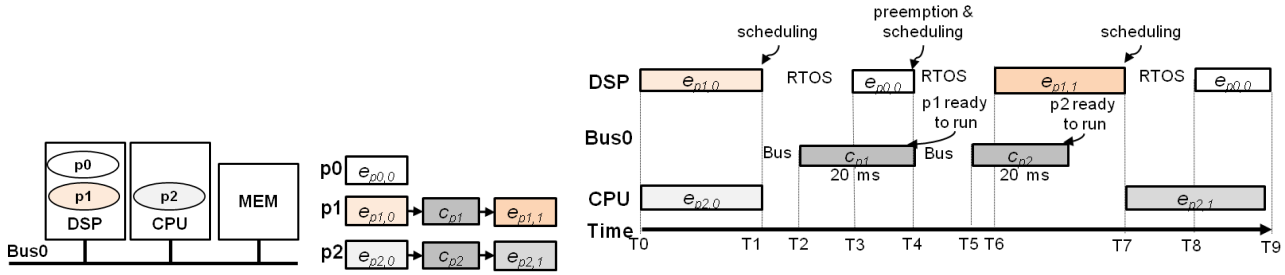
We assume that inter task communications are conducted by calling communication APIs. We assume that the order of communication API calls of each task depends on the data only. These assumptions are valid for multiple applications such as multimedia applications captured in SDF.

3.2 Definitions: Execution, Communication Events and Traces

A *communication event* of a task is defined as the interval between calling and returning from a communication API call. Each communication event is coupled with the number of bytes transferred. The length of a communication event is computed during alignment according to the bus protocol model. An *execution event* of a task refers to the execution of the task between any two of the following: the start of the task, the end of the task and communication events. An execution event is coupled with (1) all possible optimistic scheduling delays [2] for the event, (2) the number of branches, (3) and the total amount of local memory accesses. Optimistic scheduling delay of an execution event is the delay estimate where there is no branch prediction failure and no cache miss. The impact by branch prediction failures and cache misses will be added in the alignment step. The number of cache misses is computed by multiplying the total amount of local memory accesses and the probability of cache miss. The penalty of a branch prediction failure and a cache miss is given as a part of the PE configuration. The *trace* of a task is an ordered list of the communication and execution events of the task. The order depends on the execution path of the task, which is affected not by the platform or mapping but by the data.

3.3 Trace-Driven Performance Estimation

Figure 1 shows the basic idea of TDPE. Figures from 1a through 1c show the application, the code of each task, the platform and mapping. In Figure 1b, do_p0_1 is not called and do_p1 is called. Thus, the trace of each task is as it appears in Figure 1d. The alignment engine must place each



(a) Platform and Mapping (b) Traces (c) Alignment

Figure 3: Alignment Example w/ RTOS Model and Bus Protocol Model

event in the traces on the global time line. A trace defines the order of its events. Moreover, inter task communications defines the partial order of events in different traces. In this example, p1 executes **recv** first and yields CPU. CPU is now in idle state. Since the communication is assumed to be synchronized, **send** and **recv** must be end at the same time. **Send** is called after completion of the execution event do_p0_0. Thus, the order of the execution events and communication events must be as in Figure 1e. Between do_p1 and **recv**, there may be RTOS overhead.

Figure 2 shows the design flow with TDPE. Each basic block of the application source code is annotated with all possible delay estimates. During this annotation, small codes are instrumented to profile local memory accesses and branch operations. Our trace generation reuses the source code annotation described in [2]. Each basic block goes through the abstract data path of each PE. Following that, the optimistic scheduling delay estimate of each basic block is given. Performed with this timed application is a single functional simulation.Inside the communication APIs are recorded the traces and their performance metrics. After system definition, which is basically platform selection and mapping, the generated traces are aligned and the performance metrics given. Note that there is no simulation. Even if either the platform or mapping changes by design optimization, trace alignment provides performance metrics, making simulation unnecessary.

3.4 Abstract RTOS, Memory Hierarchy, Processing Element and Bus Protocol Models

As Hwang et al [2] did, we compute the impact of cache based on statistics. Functional simulation profiles the amount of memory accesses. TDPE multiplies this by the cache miss rate. A PE can have such configurations as cache sizes and frequency. The alignment engine takes all of these into account when computing the length of each execution event.

Figure 3 describes RTOS models and bus protocol models. Figures 3a and 3b show the platform, mapping, and traces. Note that c_{p1} and c_{p2} are read/write operations to the global memory MEM. The core of abstract RTOS models of TLM is the abstract RTOS operations such as scheduling with the proper RTOS overheads [1]. Likewise, the core of the abstract bus models in TLM [6] is abstract bus operations and their overhead estimates. In our TDPE, those are emulated during alignment. In this example, the RTOS on DSP is preemptive and priority based. p1 is assumed to have the highest priority. Thus, at T0, the first event of the task p1 is selected. Once p1 yields DSP at T1 for communication, p0 is selected by the scheduler and thus the first event of p0 $e_{p0,0}$ is placed on the global time line following $e_{p1,0}$. Note that a proper RTOS overhead is prepended to $e_{p0,0}$ so $e_{p0,0}$ is placed at T3 instead of at T2. The alignment engine can also take into account preemption. At T4, the communication event c_{p1}— which is a communication API call by p1—is finished. However, $e_{p0,0}$ is not completed yet. Thus, the alignment engine shortens $e_{p0,0}$ and preempts. The first of the remaining events of p1 is $e_{p1,1}$ and the event is placed at T6. The rest of $e_{p0,0}$ is placed at T8 after $e_{p1,1}$ is finished.

The alignment engine emulates the abstract bus protocol models. In this example, p1 and p2 request, at the same time, the global memory access through Bus0. In the bus protocol model, these requests must be sequentialized according to the arbitration policy. Since DSP has the highest priority, p1 on CPU gets the bus first at T2. Note that there is an arbitration delay between T2 and T1. The length of each communication event is equal to 20 ms. The length is computed by the alignment engine according to the bus protocol model.

Table 1: Comparison in Estimation Time

Design	CAM Estimation	TLM Estimation	TDPE
		Annotation+ Simulation	Annotation+ Simulation
SW	15.93h	31.262s+0.004s	x+0.076s
SW+1	17.56h	49.986s+0.217s	x+0.077s
SW+2	17.71h	47.290s+0.254s	x+0.081s
SW+4	18.06h	71.131s+0.357s	x+0.084s

$x = \frac{93.56}{N} sec$ is the average estimation time for a platform selection&mapping if we need N platform selection&mapping to find the solution that meets all design constraints.

4. CASE STUDY

We compared TDPE to TLM estimation [1]. In this case study, we reused the application, platform, mapping and configurations used by the authors. The application is MP3 decoder. In the platform, there are 3 types of PEs: MicroBlaze processor, custom HW and IMDCT. SW, SW+1, SW+2 and SW+4 are the platform with 0, 1, 2 and 4 hard-

Table 2: Accuracy Results: Error % against TLM Estimation

I/D Cache Size	SW			SW+1			SW+2			SW+4		
	TLM Est.	TDPE	Error	TLM Est.	TDPE	Error	TLM Est.	TDPE	Error	TLM Est.	TDPE	Error
0k/0k	291.05ms	290.19ms	0.29%	308.88ms	308.44ms	0.14%	253.05ms	248.25ms	1.89%	225.92ms	221.55ms	1.94%
2k/2k	104.58ms	100.17ms	4.22%	93.07ms	92.75ms	0.33%	82.31ms	81.20ms	1.36%	83.16ms	81.48ms	2.02%
8k/4k	65.34ms	63.73ms	2.47%	59.47ms	59.28ms	0.31%	267.70ms	266.99ms	0.26%	52.53ms	51.91ms	1.18%
16k/16	58.64ms	57.39ms	2.14%	50.25ms	48.95ms	2.58%	51.45ms	50.95ms	0.97%	49.41ms	49.03ms	0.77%
32k/16k	58.46ms	56.36ms	3.60%	58.09ms	57.27ms	1.40%	52.92ms	51.67ms	2.34%	48.02ms	46.97ms	2.19%

The application is an MP3 decoder. MicroBlaze has a priority-based non preemptive RTOS.

ware components. Instruction/Data cache size of MicroBlaze processor was configurable. The experiments with CAM were taken from [2].

TABLE 1 compares, in terms of estimation time, TLM Estimation, estimation using CAM and TDPE. Estimation using CAMs is accurate but, compared to the rest, orders of magnitude slower. TLM Estimation time is the sum of annotation time and simulation time. Accordingly, TDPE time is the sum of annotation time and alignment time. As expected, alignment time was much shorter than simulation time. Moreover, if N-system definitions are required for an optimal design to be found, annotation time of TDPE is shortened inversely proportional to N since annotation is conducted once regardless of N. Therefore, TDPE time was 18.84% than TLM Estimation when N is 10 and 2.03% when N was 100.

We are grateful to the authors of [2], who allowed us access to the internals of their implementation. We implemented TDPE so that it emulated the authors' framework. During alignment, TDPE emulated the RTOS operations with their overhead values and bus operations with their overhead values. The operations and overhead values were almost the same as the framework of [2]. TDPE reused the authors' annotation engine to compute the optimistic scheduling delays. Once the platform was selected and mapping given, the right optimistic scheduling delay was chosen. Following that, TDPE computed the branch prediction and cache miss delay in the same way as [2] and added the computed delay to the optimistic scheduling delay. Therefore, as TABLE 2 shows, the accuracy of TDPE is almost the same as that of the TLM Estimation.

5. CONCLUSION

TDPE provides performance estimation that is orders of magnitude faster than TLM Estimation since a single trace generation and alignment step replaces annotation and simulation of the entire platform. TDPE is as accurate as TLM Estimation since the data path of each PE is considered in the same way and TLM's abstract RTOS, bus protocol, memory hierarchy models and PE configurations are emulated during alignment. The case study showed TDPE is, without losing accuracy, 49.34 times faster than TLM Estimation when N is 100. TDPE is applicable to both software and custom hardware so re-targetable in a sense.

Acknowledgment

This work was supported in part by the National Science Foundation under NSF grant number 1136146.

6. REFERENCES

[1] Yonghyun Hwang, Gunar Schirner, Samar Abdi, and Daniel G. Gajski. Accurate timed rtos model for transaction level modeling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1333–1336, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[2] Yonghyun Hwang, Samar Abdi, and Daniel Gajski. Cycle-approximate retargetable performance estimation at the transaction level. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 3–8, New York, NY, USA, 2008. ACM.

[3] J.T. Russell and M.F. Jacome. Architecture-level performance evaluation of component-based embedded systems. In *Design Automation Conference, 2003. Proceedings*, pages 396–401, June 2003.

[4] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, Feb 2002.

[5] Moo-Kyoung Chung, Sangkwon Na, and Chong-Min Kyung. System-level performance analysis of embedded system using behavioral c/c++ model. In *VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT). 2005 IEEE VLSI-TSA International Symposium on*, pages 188–191, April 2005.

[6] S. Abdi, Yonghyun Hwang, Lochi Yu, Hansu Cho, I. Viskic, and D.D. Gajski. Embedded system environment: A framework for tlm-based design and prototyping. In *Rapid System Prototyping (RSP), 2010 21st IEEE International Symposium on*, pages 1 –7, june 2010.

[7] R. Plyaskin, A. Masrur, M. Geier, S. Chakraborty, and A. Herkersdorf. High-level timing analysis of concurrent applications on mpsoc platforms using memory-aware trace-driven simulations. In *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, pages 229–234, Sept 2010.

[8] K. Lahiri, A. Raghunathan, and S. Dey. System-level performance analysis for designing on-chip communication architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(6):768–783, Jun 2001.