Performance Estimation in Heterogeneous MPSoC Based on Elementary Operation Cost

Nikolina Frid, Danko Ivošević and Vlado Sruk
University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
nikolina.frid@fer.hr, danko.ivosevic@fer.hr, vlado.sruk@fer.hr

Abstract – Effective use of resources available on heterogeneous MPSoC platforms can only be achieved through careful resource allocation and scheduling. The diversity of processing and memory elements will manifest itself in the total time and resources required to perform a task or execute an application. Choosing the right platform element is the key and the first step is performance estimation.
This paper tackles the issue of finding the most suitable processing element for each part of the software application through a novel approach – elementary operation cost. The cost of each elementary operation is experimentally determined through a set of carefully devised benchmarks and is used for estimating duration of complex functions found in common applications such as JPEG, AES etc. By raising the abstraction level on which the execution time is calculated from instruction to operation level, common problems in performance estimation such as pipelining and branch prediction can be avoided and estimation accuracy is improved. Demonstrated results show that the average error rate of estimated execution times for various benchmarks is around six percent compared to the actual execution times.

Keywords: Embedded Computer Systems, Heterogeneous Computing, FPGA, Design Space Exploration, Performance Estimation

I. INTRODUCTION

Heterogeneous multiprocessor systems have continuously been used in embedded systems for almost a decade due to their ability to provide better performance at lower cost, area and power demand. Much work has been done to achieve, in each new generation of systems, even better performance at lower cost and design time while maintaining software flexibility. Exploring large design space poses the greatest challenge in MPSoC design. Choosing the right platform architecture which will meet software application demands in early stages of design process is the key to reducing development cost and time-to-market. The widely accepted approach [1] to overcome this challenge is to start the design space exploration at high levels of abstraction allowing for fast evaluation of available options. In early design stages the hardware platform is still not finished and performance must be evaluated either through simulation or analytical methods. This article proposes a novel analytical method for software performance estimation on heterogeneous MPSoC platforms based on elementary operation cost concept. This method ensures high accuracy of results while maintaining high level of abstraction and thus enabling fast design space exploration.

II. RELATED WORK

In early days of heterogeneous platforms research and design, Instruction Set and Cycle-Accurate Simulators (ISS/CAS) have been invaluable tools for early performance estimation [2]. However, their high accuracy came at the cost of long simulation time and large design effort. More recent work has adopted the TLM [3] approach which allows much faster functional and performance (i.e. timing) evaluation with possibility to even simulate the behaviour of RTOS. While TLM gives very good results in functional simulation, the issues arise in achieving high accuracy of timing estimation. Main challenge is to accurately describe software application performance on different processing elements and provide it as an input to TLM. Several different approaches exist, mostly relying on some form of application source code annotation and transformation into SystemC, which is suitable for later use in TLM.

Authors in [5] rely on source level annotation. They all break down the application structure into Basic Blocks (mostly using GCC compiler) and use ISS or CAS to estimate performance for each block. In later stages the application is simulated (speed comparable to real-time execution of application) with different inputs, mostly as a part of a transaction-level model. Rate error remains under ten percent, in most work around six percent, within acceptable boundaries.

A slightly different approach is presented in [4]. The authors introduce annotation at intermediate representation (IR) level, a lower level of abstraction, in order to capture compiler optimizations more accurately.

However, all these approaches have trouble with the inter-block dependency (pipeline effects). The problem is partially solved by introducing pair-wise execution of neighbour blocks or simulation of as many possible basic block combinations at the cost of total simulation time.

A rather different approach is presented in [9], where authors use analytical estimation instead of simulation.

This method employs neural networks which receive the number of each type of instruction of software application and give the estimation of number of cycles (i.e. total duration) as output. The results are little less accurate with around 17 percent average error rate, and the network training time is a bit longer, but the system is more flexible and faster in later stages.

III. Elementary Operation Concept

Fast and efficient exploration of the large design space, present in MPSoC design, requires early performance estimation, as mentioned earlier. Consequently, the performance estimation must be conducted at very high level of abstraction using application and platform models. It is also very important that this models not only enable fast execution but also retain a very high degree of accuracy in performance estimation. Thus it is extremely important to accurately describe the heterogeneous platform.

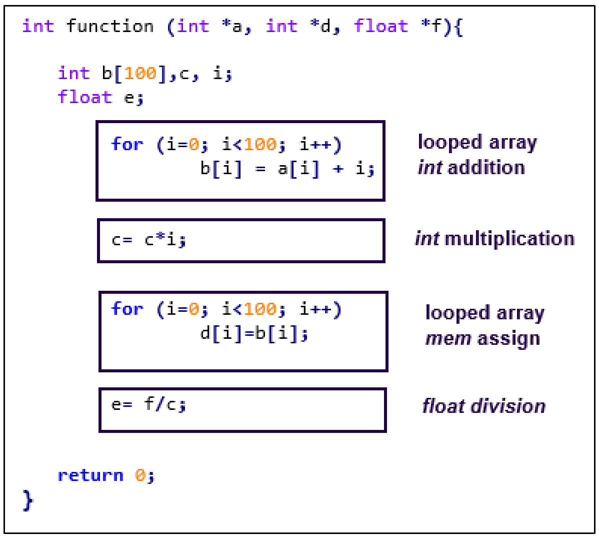
In earlier work regarding the impact of heterogeneity on MPSoC performance [10] it has been observed that the performance of a particular type of processing element in execution of simple tasks (e.g. calculation of square root of an integer number) highly correlates with the performance of that same element when executing a more complex task (e.g. JPEG compression). In this paper that concept is further elaborated.

A. *Concept Overview*

Great majority of authors agree that the key to source-level performance estimation is to identify a unit of code small enough to enable modular and reusable approach and in the same time large enough to diminish the pipeline effects. Assembly level instructions satisfy only the first condition. Basic blocks, at C source level, are much more resilient to pipeline effects but their reusability is often questionable – two different functions (tasks) almost never have two identical basic blocks. Thus ISS must be used each time a new function requires timing estimation. Moreover, pipeline effects (i.e. branch prediction) between two basic blocks must be separately handled.

On the other hand, it can be observed that there exist types of operations (not necessarily a single line of code) which are present in many different functions written in C. Although it is hard to list all possible types of operations which can be used in writing C functions it can be safely concluded that it is a finite set. And while the research discussed in this paper has certainly not identified all of them, through careful examination of different types of functions it is possible to do so.

At this point it can be concluded that there exist several distinct subsets of these elementary operations that accurately reflect the implicit features of a processing element architecture: integer and floating point arithmetic operations, logic and memory operations. All of these operation types are also supported by processor through dedicated parts of datapath. Further examination and experiments reveal that distinction must also be made between operations on local and global variables, and operations with arrays. Looped execution of a certain type of operation should also be taken into consideration. An example of several types of elementary operations is illustrated in Figure1.



```
int function (int *a, int *d, float *f){

   int b[100],c, i;
   float e;
        for (i=0; i<100; i++)
                b[i] = a[i] + i;

        c= c*i;

        for (i=0; i<100; i++)
                d[i]=b[i];

        e= f/c;

   return 0;
}
```

Figure 1. Elementary operation examples

Once a large enough set of elementary operations is defined, the cost is determined based on execution time of these operations for each type of processing element by using a simulator or the actual target. The data collected is reusable, without any additional measurements, for estimating performance of any software application task at hand. The granularity of elementary operation is large enough to encompass pipeline effects. Also, an elementary operation often crosses basic block boundaries thus cancelling inter-block effects.

Test benches for measuring elementary operation execution times have been carefully devised to include a large set of possible operation types. CHStone [10] had been used as a starting point, mostly to identify the distinction between integer and floating point arithmetic operations, and between simple (e.g. addition, shift) and more complex operations (e.g. multiplication, division). However, major modifications and improvements have been done to better suit the earlier described features of a heterogeneous architecture.

B. *Elementary operation cost measurement*

Experiments and measurements have been conducted on two different RISC processor architectures implemented on Xilinx ZedBoard Zynq®-7000 All Programmable SoC.

An ARM Cortex A9 processor was used in the following configuration: operating frequency @ 667MHZ, 32 KB L1 cache and 512 KB L2 cache with both instructions and data stored in local DDR3 SDRAM memory operating at 533 MHz.

MicroBlaze, a softcore processor was used in the following architecture configuration: 5-stage pipeline

TABLE I. ELEMENTARY OPERATION EXECUTION TIMES

Target processor		ARM Cortex A9 [ms]	MicroBlaze [ms]
INT ADD	loop	1,2E-5	1,15E-4
	unrolled loop	9,23E-6	2,93E-5
INT MUL	loop	1,352E-5	2,9E-4
	unrolled loop	6,9E-6	1,02E-4
FLOAT ADD	loop	1,35E-5	1,635E-3
	unrolled loop	1,18E-5	1,48E-3
FLOAT MUL	loop	1,5E-5	1,74E-3
	unrolled loop	1,414E-5	9,83E-4

with hardware multiplier, FPU and barrel shifter @ 200 MHz (200 MHz clock frequency is the maximum achievable frequency on PL side of Zynq system) with both the instructions and data stored in local BRAM memory.

In Table 1 an example of elementary operation execution times for different types of addition and multiplication operation is presented. The distinction is made between integer and floating point operations and whether the operation is performed in a loop or not.

Besides using the obtained data for estimating duration of a complex task execution on a certain type of processor, this data can also be used for quick identification of implicit features of processor architecture and datapath. For example, Figure 2 illustrates the impact of loop unrolling for operations specified in Table 1. It is clearly visible that due to the internal architecture, it is much more beneficial to do loop unrolling on a MicroBlaze processor than on ARM Cortex A9 – speedup of integer addition goes up to four times, and floating point multiplication almost doubles in performance.

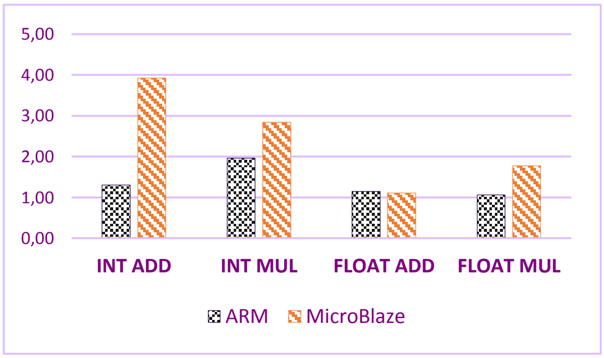


Figure 2. Loop unrolling speedup comparison

IV. EXPERIMENTAL RESULTS

Several different benchmark sets have been used to test the ability to accurately estimate task execution time based on elementary operation cost concept: MiBench [12], JPEG [13,14] and AES [15].

The total of seven benchmarks were used: two benchmarks from MiBench set (*Cubic* and *Sqrt*), three from JPEG set (*Shift*, *DCT* and *Zig-Zag*) and two from AES set (*SubBytes* and *ShiftRows*). These particular benchmarks have been chosen to test all elementary operation subsets. *Cubic and Sqrt* benchmarks test integer and floating point arithmetic operations. *Sqrt, Shift, DCT* and *Zig-Zag* integer, memory and logic operations, while AES benchmarks test memory operations.

The number of each type of elementary operation included in these benchmarks needed for timing estimation, has been analysed with the help of a previously developed tool [16]. This tool has been further improved to enable automation of the process of identifying total number of occurrence of each type of elementary operation in a given function.

After the analysis of elementary operations contained within each benchmark, timing estimation was calculated based on previously obtained operation execution times. The results are given in Table 2 under *Est.* These results were compared with the results of real-time execution of

TABLE II. COMPARISON OF ESTIMATED AND ACTUAL EXECUTION TIMES

Benchmark			*Cubic*	*Sqrt*	*JPEG Shift*	*JPEG DCT*	*JPEG Zig-Zag*	*AES SubBytes*	*AES ShiftRows*
Target	ARM Cortex A9	*Est. [ms]*	4,99E-03	1,77E-03	2,24E-03	1,34E-02	4,03E-03	1,33E-03	5,39E-04
		Act. [ms]	4,76E-03	1,69E-03	2,27E-03	1,54E-02	3,89E-03	1,44E-03	5,16E-04
		Error	***-4,83%***	***-4,73%***	***1,32%***	***12,99%***	***-3,60%***	***7,64%***	***-4,46%***
	MicroBlaze	*Est. [ms]*	2,77E-01	9,28E-03	1,35E-02	9,06E-02	1,42E-02	5,44E-02	5,64E-04
		Act. [ms]	2,56E-01	8,74E-03	1,47E-02	9,79E-02	1,38E-02	5,30E-02	5,07E-04
		Error	***-8,20%***	***-6,18%***	***8,16%***	***7,46%***	***-2,90%***	***-2,64%***	***-11,24%***

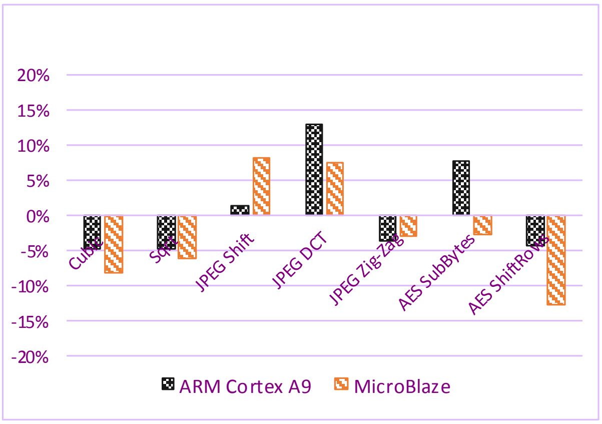


Figure 3. Estimation error rates

the same set of benchmarks on the Zynq platform for both target processors. Benchmark execution times on the Zynq platform are indicated in the Table 2 under *Act*.

Comparison of the results, illustrated in Figure 3 shows that the average error rate is around six percent with the peak at twelve percent. Considering that timing estimation is done at very high level of abstraction, these results are within acceptable range and also comparable to the results of methods presented in related work mentioned previously.
It is important to note that while these experiments were conducted on a real physical platform, they could have also been conducted, without any modification on an ISS or cycle-accurate simulator. In this case the simulator for these processors was not available.

V. Conclusion

In this article a novel approach to source-level performance estimation has been proposed. The elementary operation cost concept enables very fast design space exploration with high accuracy of results. The average error rate at six percent is comparable to the results of methods presented in related work, and thus within acceptable limit, especially the performance estimation is done in a very early design stage where the main goal to detect the most suitable type of processing element for a certain type of software application task and determine the optimal total number of processing elements.
First steps in future evolution of this approach will be full automation of the entire process and providing support for evaluation of effects of different memory configurations. Further improvements will need to face the challenge of handling compiler-introduced optimizations. The possibility to use this approach for simulation-based software performance evaluation, possibly in a transaction-level environment, will also be considered.

References

[1] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, “System-level design: orthogonalization of concerns and platform-based design,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1523–1543, Dec. 2000.

[2] L. Cai and D. Gajski, “Transaction level modeling,” in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign & system synthesis - CODES+ISSS '03*, 2003, p. 19.

[3] S. Abdi, G. Schirner, Y. Hwang, D. D. Gajski, and L. Yu, “Automatic TLM generation for early validation of multicore sSystems,” *IEEE Des. Test Comput.*, vol. 28, no. 3, pp. 10–19, May 2011.

[4] S. Chakravarty, Z. Zhao, and A. Gerstlauer, “Automated, retargetable back-annotation for host compiled performance and power modeling,” in *2013 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2013*, 2013, pp. 1–10.

[5] K. L. Lin, C. K. Lo, and R. S. Tsay, “Source-level timing annotation for fast and accurate TLM computation model generation,” in *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 2010, pp. 235–240.

[6] Z. Wang and J. Henkel, “Accurate source-level simulation of embedded software with respect to compiler optimizations,” *Des. Autom. & Test Eur. Conf. & Exhib.*, vol. 0, pp. 382–387, 2012.

[7] E. Cheung, H. Hsieh, and F. Balarin, “Fast and accurate performance simulation of embedded software for MPSoC,” in *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 2009, pp. 552–557.

[8] P. Gerin, M. M. Hamayun, and F. Pétrot, “Native MPSoC co-simulation environment for software performance estimation,” in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis - CODES+ISSS '09*, 2009, p. 403.

[9] M. Oyamada, F. R. Wagner, M. Bonaciu, W. Cesario, and A. Jerraya, “Software Performance Estimation in MPSoC Design,” in *2007 Asia and South Pacific Design Automation Conference*, 2007, pp. 38–43.

[10] N. Frid, D. Ivosevic, and V. Sruk, “Heterogeneity impact on MPSoC platforms performance,” in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 1071–1076.

[11] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda and Hiroaki Takada, "Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis", *Journal of Information Processing*, vol. 17, pp.242-254, 2009.

[12] M. R. Guthaus, J.S. Ringenberg, D. Ernst, et al., “MiBench: A free, commercially representative embedded benchmark suite”, in *Proceedings of IEEE 4th Annual Workshop on Workload Characterization”*, Austin, TX, 2001., pp. 3-14.

[13] G. K. Wallace, “The JPEG Still Picture Compression Standard” *Communication of the ACM*, vol 34., Issue 4, ACM Press, New York, NY, USA, pp.30-44 , 1991.

[14] Arai Y., Agui T., Nakajima M., “A Fast DCT-sQ Scheme for Images”, Trans. IEICE, Vol. E71, No.11 pp. 1095-1097, 1988.

[15] NIST, “FIPS 197 Advance Encryption Standard (AES)”, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[16] D. Ivošević and V. Sruk, “Unified flow of custom processor design and FPGA implementation,” in *IEEE EuroCon 2013*, 2013, pp. 1721–1727.