Rapid, High-Level Performance Estimation for DSE using Calibrated Weight Tables

omitted for blind review

omitted for blind review

Abstract. Automated Design Space Exploration (DSE) is a critical part of system-level design. It relies on performance estimation to make rapid decisions. However, since a plethora of design alternatives need to be compared, the run-time of performance estimation itself may pose a bottleneck. In DSE, fastest performance estimation is of essence while some accuracy may be sacrificed.
Fast estimation can be realised through capturing application demand, as well as Processing Element (PE) characteristics (later on called weight table) in a matrix each. Then, performance estimation (retargeting) is reduced to a matrix multiplication. However, defining the weight table from a data sheet is in-feasible due to the multitude of (micro-) architecture aspects.
This paper introduces a novel methodology for automatically generating Weight Tables in context of C source-level estimation using dynamic profiling and Linear Programming. (LP). LP solving is based on the measured performance of training benchmarks on an actual PE. We validated the proposed approach using a synthetic processor and benchmark model, and also analyse the impact of non-observable features on estimation accuracy. We evaluate the efficiency of the proposed methodology using 49 benchmarks on 2 different processors with varying configurations (multiple memory configurations and software optimizations). On a 3.1 GHz i5-3450 Intel host, 25 million estimations / second can be obtained regardless of the application size and PE complexity. The accuracy is sufficient for early DSE with a 24% average error.

1 Introduction

Recent advances in technology have expanded the design options in terms of number and type of processors as well as their configurations such as interconnects and memory hierarchy. When this flexibility of design is coupled with the increasing pressure of time to market, performance exploration of the design space becomes exponentially difficult. Current approaches try to automate the Design Space Exploration (DSE). In any DSE, two questions need to be addressed. One is how to traverse the design space and other is how to assess the fitness of each design instance - all unique combinations of platforms and mappings. One approach for DSE is genetic algorithm. With genetic algorithm, millions of design options will be traversed before making a design decision. Evaluating the fitness of each design option falls on the time critical path of DSE and

is at most importance here. Simulation based approaches are highly accurate but too slow for DSE. New approaches are needed for rapid high-level performance estimation in context of DSE. This paper revisits on an earlier approach of DSE and improves upon it.

An earlier approach of DSE [1] is based on performance estimation using retargetable profiling. This approach uses a weight table, which is a matrix of Processing Elements (PE) performance cost (cycles) of each high-level operation for all data types. One of the main challenges of retargetable profiling is that the weight tables need to be manually defined. The accuracy of weight table determines the accuracy of final estimation. Due to manual extraction of these weight tables from the data sheet, this process is error prone. Moreover, because of the high-level abstraction, only a few features of the processor are observable (can be quantified). For example, C statements do not reveal from where an operand needs to be fetched within the memory hierarchy. Therefore, the cycles captured in the weight tables have to statistically include these non-observable characteristics such as memory accesses and pipeline stalls. These elements which can affect the performance but are not observed during execution make it nearly impossible to come up with estimations sufficiently accurate for DSE. In this paper, we present a methodology and it's framework to automatically populate more realistic weight tables which lead to efficient DSE.

We validate the proposed approach using a synthetic processor and benchmark model, and also analyse the limitations of this approach. We evaluate the efficiency of the proposed methodology using 49 benchmarks on 2 different processors with varying configurations (multiple memory configurations and software optimizations).

The rest of this paper is organised as follows: In Section 2, an overview of related work and the motivation for this approach is presented. Retargetable profiling is introduced in Section 3. Our proposed approach is detailed along with the implementation specifications in Section 4. A synthetic model to validate the approach is presented in Section 6. Experimental results are shown in Section 7. Section 8 concludes the paper.

2 Related Work

In the recent years, many estimation methods are proposed with each one trying to solve different challenges in estimation such as accuracy, speed and being application specific. One of the most important features which distinguish these methods from one another is the abstraction level at which performance is estimated. In general there are three main levels: source-level (high-level), intermediate-level and binary-level (low-level). At high-level, less amount of details are taken into account. It is faster and retargetable but less accurate in terms of absolute performance numbers. On the other side low-level estimation delivers more information about target architecture that can be used to increase the accuracy at the cost of simulation speed. While low-level estimations are used for analysing and improving the performance at a finer level, high-level estimation is more suitable for DSE.

There are various high-level estimation techniques. The authors of [2] propose an approach which has limitations due to compiler optimizations. Wang et al. [3] present an approach which takes compiler optimizations into account. However, both these approaches need execution on host machine for estimation. The authors of [4] propose a compiler-assisted technique to rapidly estimate the performance without the need of actual execution. However, this approach is developed for the FPGA based processors. Oyamada et al.[5] present an integrated approach for system design and performance analysis. An analytic approach based on neural networks is used for high-level software performance estimation. This approach takes about 17 seconds to estimate the performance of an MPEG4 encoder application. A hybrid simulation method is introduced in [6] which also uses a cache simulator to measure memory access delay. [9] presents a complementary method for increasing the accuracy of approaches that are annotating timing information into source code by mapping binary representation to source level. This approach requires the source code and the binary-level CFG. In [7], an estimation approach is proposed for transaction level. One drawback of this work is that the mapping between the C processes to PEs should be determined before using this estimation approach. These approaches are suitable for estimating the performance of a PE, but efficient design space exploration requires faster retargetability.

Javaid et al. [8] propose two estimation mechanisms whose goal is to minimise the estimation time. Though the approach is retargetable for pipelined MPSoCs, the performance estimation of individual component of design space is yet not retargetable. Mohanty et al. [10] present a mechanism for DSE using interpretive simulation which requires specific inputs to the proposed model.

The retargetable profiling approach made by the authors in [1] is most suitable for DSE as it is retargetable and does not involve simulating the target application across the design space. However, the estimation accuracy of this approach is largely dependent on the weight table entries of PEs in the design space.

3 Retargetable Profiling

Retargetable profiling [1] is a high-level estimation technique used for DSE. The flow is divided into two stages - Profiling and Retargeting.

In the profiling stage, system specification (defined in a specification language like SpecC) is instrumented and simulated in order to collect execution counts of all operations for each data-type executed. These frequencies are captured in the form of specification characteristic table. This specification characteristic table has the format same as weight table of a PE. Note, the profiling stage is done only once per application.

In the Retargeting stage, the designer decides the mapping between behaviour and PE. Performance of each PE is estimated by multiplying specification characteristics (obtained from profiling) with delay values stored in the weight table of that PE. The total performance (E) of that PE will be calculated

as follows:

$$E = \sum_i \sum_j (F_{ij} \times W_{ij}) \tag{1}$$

where W_{ij} is the weight and F_{ij} is the occurrence frequency of each operation i of data type j. Since retargeting consists of a pure static approach, it avoids the time-consuming steps of simulation and profiling.

Retargetable profiling includes some of the most important features needed for DSE such as fast traversing speed, being source level and target independence. However, in practice, this method requires a tedious manual step that involves extracting the weight table information from the data sheet of each PE. Considering that the IP vendors have their unique way of representing this information, data collection can be a time-consuming task. Furthermore, dedicating only one table for each processor limits the designer to use only the default configuration of each processor in terms of compiler optimizations and hardware configurations. This limitation makes the design space too simple and unrealistic. As a result of this, comparison between estimations and actual performance measured on the target processors show a low association. Moreover, some affecting elements are unknown, such as details of pipeline and data forwarding, because the vendors often do not release this information. Both non-observable elements like memory hierarchy and multi-issue parallel execution, and unknown functionalities are not compensated in the weight tables. In this paper, we try to overcome these limitations by providing a framework for automatically generating the weight tables by calibration.

4 Approach

In this section, we propose a technique for calibrating PE weight tables. It automatically populates the weight tables using a training set of benchmarks and Linear Programming Formulation (LPF). This methodology not only expands flexibility of the retargetable profiling approach, but also increases the accuracy by implicitly considering many architectural features which were overlooked by the approach in [1].

4.1 An Overview of the Framework

As shown in Figure 1, the proposed framework is engineered to produce weight tables for PEs according to a set of training benchmarks. Every benchmark is captured in *SpecC* language (which is a super-set of ANSI-C) and then profiled with *SCProf* profiler tool [1]. Output of the profiling tool is organised in the form of application computation demand. It includes the frequency of all operations for each data type. In order to extract the timing information, benchmarks are executed on the hardware of each PE to obtain benchmark execution cycles. A linear program is constructed by formulating information from the specification characteristic table and execution cycles for each benchmark as an inequality. The solution of this linear system is the weight table for target PE.

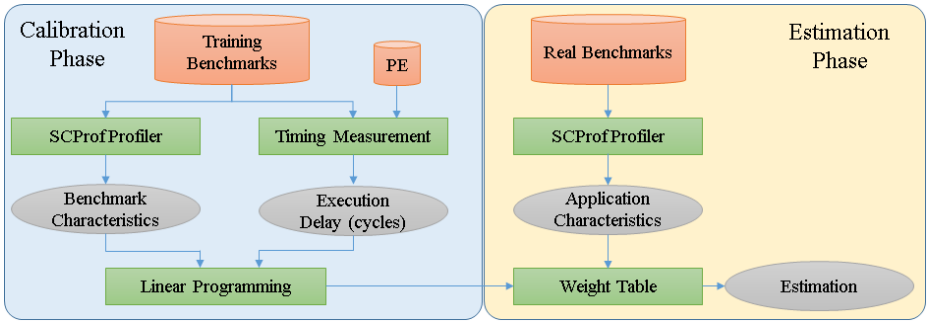

Fig. 1. Framework Flow

4.2 Linear Programming Formulation

In this section, we describe the LPF used to obtain the weight tables. For each benchmark i on a particular PE, (B_i) specifies the measured execution time (cycles). The recurrence j of every combination of operation and data-type in the benchmark i is denoted by R_{ij} (which is the computational demand as determined by profiler) and weight of that combination on the PE is denoted by W_j. CF is the Calibration Fudge factor which allows for error within the estimation. An equation is generated for each benchmark which includes the weights of each operation-data-type combination, occurrence frequency of each combination and total execution time. The number of equations would be equal to the number of benchmarks available. The weights in each equations are the unknown, which will be solved by LPF. Minimising this fudge factor will help in associating accurate weights.

Minimise:
Σ_i |CF(i)|
Subject to:
$Benchmark1 : R_{11}.W_1 + R_{12}.W_2 + ... + R_{1M}.W_M + CF_1 = (B_1)$
$Benchmark2 : R_{21}.W_1 + R_{22}.W_2 + ... + R_{2M}.W_M + CF_2 = (B_2)$
...
$BenchmarkN : R_{N1}.W_1 + R_{N2}.W_2 + ... + R_{NM}.W_M + CF_N = (B_N)$
where N is the number of benchmarks and M is the number of possible C operations of each data type.

As every benchmark in this formulation is represented by one equation, by increasing the number of training benchmarks, number of equations in LPF will increase. As a result, more information will be added to LP solver for finding the weights of that PE. Consequently, it would be possible to extract a more realistic weight table leading to a better future estimation.

5 Implementation

In this section, we present the benchmarks, tools, PEs and metrics that are used for evaluating the approach.

5.1 Benchmarks used for Calibration

Benchmarks used in the framework play an important role in estimation process. Balance in distribution of these benchmarks allows more accurate estimation for future applications. A major effort has been devoted to gather a suitable set of benchmarks that can cover most of operations, data types and coding structures. Table 2 shows a classification of the benchmarks. In Table 1, *Common* bench-

Table 1. List and categories of benchmarks

Category	Names of Benchmarks
MiBench [12] & DSP-Stone [13]	AES whetstone bcnt blit bubblesort cnt crc crc2 edn fft1 fir2 gamma hanoi heapsort linpack lms lms2 ludcmp matmult matrix basicmath ndes nsichneu peakSpeed1 prime queens v42 wavelt
Randomly Generated	frand1 frand10 frand12 frand13 frand16 frand17 frand18 frand19 frand2 frand20 frand3 frand4 frand6 frand8 frand9 rand3 rand5 rand6 rand7 rand8 rand9
Synthetic	synadddouble synaddfloat synaddlonglongint syndivdouble syndivint syndivlonglongint synmindouble synminfloat synminint synmulfloat synmulint synmullonglongint synmuldouble synaddint synminlonglongint syndivfloat
WCET	adpcm crc2 edn fft1 fir2 lms ndes nsichneu qurt

marks include many open source as well as commercial benchmarks available from the Mibench Suite [12] and DSP-Stone Suite [13]. *Randomly Generated* benchmarks have been generated using the *Randprog* tool citeEide08Volatiles. Synthetic benchmarks were generated for tuning particular operations and data types. These synthetic benchmarks were used only for training phase. Only other real benchmarks were used for estimation, as real benchmarks reflect the characteristics of an actual workload. The *WCET* benchmarks are a subset of those presented in [15].

5.2 Tools and Processors

We used the *SCProf* profiler to extract profiling information and *SCC* SpecC compiler to compile and execute the benchmarks on real hardware. In order to solve the linear programming model, we used the open source SCIP solver as discussed in [16]. We have used two different processors (Blackfin527 [14] and ARM9 [11]), with various hardware configurations (SRAM and SDRAM with Blackfin527 [14]) and compiler optimizations (O0, O1, O2, O3 with ARM9 [11]).

5.3 Metrics

As a metric for absolute error, we used the *Average Error* to compare real execution cycles with estimated cycles. However, absolute accuracy may not be needed always. During DSE, different design alternatives are compared. In this setting, the correctness of a relative comparison is sufficient. This is described by *Fidelity* quantifying the correctness of a relative comparison.

6 Validation through Synthetic Model

Any estimation approach is limited by the number of observable features (profiling restrictions), and by the complete availability of processor performance information (limited micro-architectural knowledge). To initially validate and optimise our approach under known conditions, we employ a synthetic model. We designed a statistical model which produces synthetic PEs and synthetic benchmarks for the framework. Goal of the synthetic model is to replicate the estimation flow with real framework, however using synthetic training benchmarks and synthetic PEs. This idealised approach increases visibility over real measurements and processors. Each processor is modelled by a set of elements that contribute to delay in execution (such as execution of an operation, cache hit and cache miss). To mimic the effect of partial observability by the profiler, we declare some of these effects as observable, while other effects as non-observable. The number of training benchmarks and non-observable elements are varied to study their impact on the estimation accuracy. In order to realise the non-observable elements, their effect was deliberately included when calculating the measured execution time. However, the occurrence of non-observable elements in benchmarks is hidden from the profiler. In result, the profiler counts only the observable elements, while the timing measurement includes delay (cycles) due to observable as well as non-observable elements. The LPF will attribute the effects of non-observable elements to the observable elements. As such, the number of cycles will increase for each observable feature. This is similar to other models which for example fold memory access delay statistically into operations.

Processor P is modelled as

$$P = [W_k][W_u] \tag{2}$$

where $[W_k]$ is the set of delays for known elements and $[W_u]$ is the set of delays of unknown elements. The particular values for $[W_k]$ and $[W_u]$ are randomly chosen (linear distribution) during the generation of a processor model.

Similarly, a synthetic benchmark is defined as recurrence of elements known and unknown to the Profiler.

$$B = [R_k][R_u] \tag{3}$$

where $[R_k]$ is the set of recurrence for known elements and $[R_u]$ is the set of recurrence of unknown elements. The particular values for $[R_k]$ and $[R_u]$ are randomly chosen (linear distributed) during the generation of a benchmark.

The delay of a specific synthetic benchmark i on a particular synthetic PE j is defined as

$$(Delay)_{ij} = [R_k]_i[W_k]_j + [R_u]_i[W_u]_j \tag{4}$$

Figure 2 shows the effect of non-observable elements evaluated by changing the ratio of observable elements to non-observable elements in the synthetic model. In this experiment, the total number of elements were kept to 60, while varying the non-observable elements as 2

In the upper left graph of Figure 2, the mean estimation error quickly converges to 2% when the number of training benchmarks reaches 60. In the upper right graph with 25% non-observable elements, the mean estimation error stays at 12% with more than 60 training benchmarks. When non-observable elements increases to 40% or even 60% , both the average and absolute estimation error are high. The estimation error is no longer improved with more than 60 training benchmarks.

It indicates that the number of training benchmarks need to be necessarily larger than the number of elements in order to converge the estimation error. However, infinite number of training benchmarks does not improve the accuracy of the weight table and corresponding estimation when the non-observable elements are more than 40%. It is seen in Figure 2 that at least 50 training benchmarks are required for the estimation error to converge. The number of training benchmarks should be greater than the number of weights in the weight table for correct estimations.

Fig. 2. Effect of number of benchmarks on average error

The synthetic model allowed us to evaluate the effect of the non-observable elements and training benchmarks on the estimation error. The estimation accuracy improves with fewer non-observable elements. It is impossible to achieve a meaningful estimation with non-observable elements over 40% of all (observable and non-observable) architectural elements. Furthermore, the estimation error converges when the training benchmarks is slightly more than the desired observable elements. Adding more training benchmarks does not further improve the accuracy largely. In reality, since the widely used benchmarks are so limited, it is particularly important to find the minimal yet sufficient set of training benchmarks.

7 Experimental Results

In this section, we demonstrate the results in terms of accuracy and speed.

7.1 Evaluation

The efficiency of our approach affects the high-level design decisions. On one hand, accuracy is desired to correctly guide the DSE. On the other hand, performance estimation should be fastest to evaluate many design combinations.

Accuracy is evaluated is terms of average error and fidelity. In order to measure estimation error, we excluded one benchmark from the training benchmarks. The excluded benchmark was used as a target application, for which the estimation accuracy is determined. The procedure was repeated through all the real benchmarks by excluding each benchmark in every iteration. Real and synthetic benchmarks were used for calibrating the weight tables. However, only the real benchmarks were used as test applications as they better reflect the characteristics of an actual workload. The results are aggregated in Figure 3.

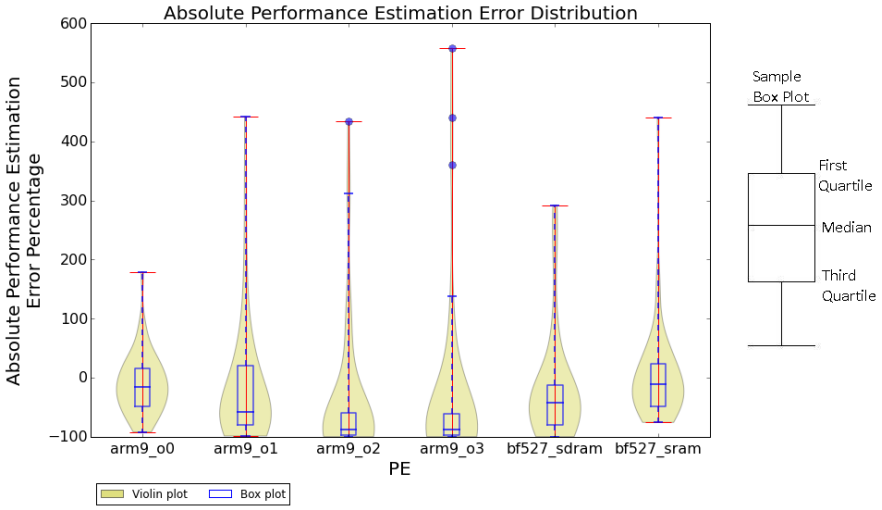

Fig. 3. Estimation Error on Real Platforms

Figure 3 shows the median, quartiles, minimum and maximum of the estimation errors among all benchmarks across all PEs. The box plot shows these static quantities, while the violin plots show the distribution of estimation error across the benchmarks. It also shows the effect of compiler optimizations and hardware configurations on the estimation error. The majority of benchmarks have an estimation error close to median (-6%) for Blackfin with SRAM. Moving from SRAM to SDRAM deteriorates the median to -30%. Most accurate results are achieved with low optimization. The ARM9 with O0 has the median at -11% with most of the benchmarks having an estimation error close to median. As the optimization increases, the association of source code with the execution time reduces. This is due to the weak correlation between source-level C code and binary at higher optimization levels. With an increased optimization to O1 for ARM9, the median is at -57%. The proposed approach is able to distinguish

between various software and hardware configurations. Some applications do not perform well in this approach in terms of the estimation accuracy. This is the cost being paid for estimating at the highest level of abstraction. The proposed approach has a poor absolute estimation accuracy than [1] because of the fact the [1] takes a very simplistic platform into consideration, which is devoid of caches. Our platforms are more realistic taking caches as well as compiler optimizations into account.

A lower absolute accuracy does not affect the early DSE as long as fidelity (i.e. relative comparison) is maintained. In order to analyse fidelity, we describe fidelity matrix, which shows the fidelity between all the possible pairs of PEs from the design space. Table 2 presents the fidelity matrix produced from our proposed approach. To further analyse fidelity, we plot fidelity over the measured performance gap between the investigated PE configurations. Fidelity depends on closeness between real performance of PEs being compared.

Table 2. Fidelity Matrix

	BF527 SRAM	BF527 SDRAM	ARM9 O0	ARM9 O1	ARM9 O2	ARM9 O3
BF527 SRAM	100%	90%	89%	76%	80%	80%
BF527 SDRAM	-	100%	76%	90%	93%	92%
ARM9 O0	-	-	100%	94%	96%	100%
ARM9 O1	-	-	-	100%	57%	61%
ARM9 O2	-	-	-	-	100%	55%
ARM9 O3	-	-	-	-	-	100%

Table 2 shows that performance estimation using calibrative weight tables has high fidelity with average fidelity being 82%. Fidelity increases when performance gap between the compared PEs is larger. The estimated comparison of ARM9 O0 with its higher optimization counterparts is correct in more than 94% of the cases. Note that as the importance of the decision (i.e. the performance gap between two comparison points) increases, the fidelity of our estimation approach also increases.

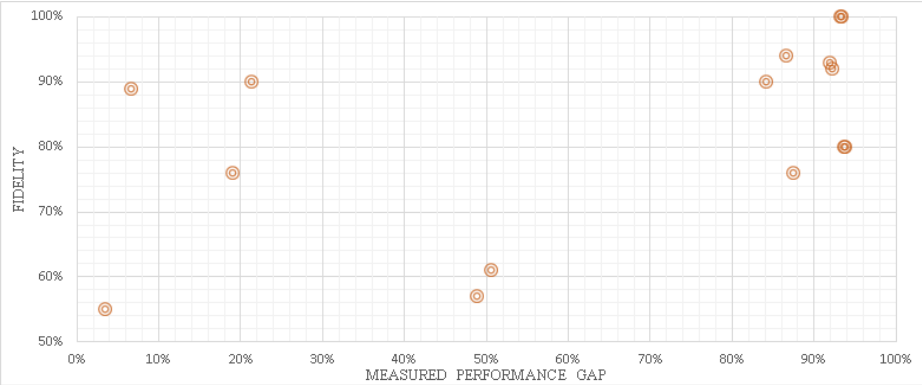

Fig. 4. Fidelity across Performance Gap

7.2 Estimation Speed

In addition to accuracy, the value of estimation methodology also depends on the time it takes for calibrating the weight table and making an estimation. The

weight table is determined only once in the lifetime of a PE. Hence, calibration takes place only once. Conversely, the retargeting to different PE configurations occurs very frequently during DSE. We have developed a framework to load and execute the benchmark on target PEs to measure the clock cycles. The average time it takes to load each benchmark is 2.5 seconds. The LPF took 0.6 seconds to generate the weight table for 48 given equations (48 benchmarks). The calibration phase took nearly 120 seconds in addition to the actual execution time of all benchmarks on the hardware. Estimation is merely a matrix multiplication of the PE weight table and application profile table. As the dimensions of weight table and application profile table are fixed, the estimation time is independent of the application size and PE complexity.

On a single core of 3.1 GHz i5-3450 Intel host, 25 million estimations / second can be obtained with our initial code regardless of the application size and PE complexity. Additionally, as the estimation is only a matrix multiplication, it is highly parallelizable and can be easily spread across all the available host cores to further enhance the estimation speed. The average error of 24% is acceptable for early DSE where fidelity and high speed of estimation are more prominent factors.

8 Conclusion

Rapid estimation with sufficient fidelity is essential for DSE. In this context, retargetable source-level profiling [1] is a promising approach. It profiles a specification once to determine the specification computation demand. Then, estimating the application's execution time is as simple as a matrix multiplication of the specification computational demand and weight table capturing the PE's computation supply. However, this approach heavily relies on quality and availability of weight tables.

The work presented in this paper proposes a calibration-based framework to automatically determine a processor's weight table(s). It avoids the manual and error-prone process of manual capturing processor characteristics (execution time). In particular, it mitigates the challenge of limited visibility of the source-level profiling (i.e. C statements) and the associated challenge of attributing non-visible characteristics into the accounted operations.

We devised a synthetic model in order to validate the approach and analyse the bounds. We measured efficiency of the proposed framework using 49 benchmarks (mainly MiBench and DSP Stone) on ARM9 and Blackfin BF527 processors and considered memory configurations (SRAM and SDRAM) and software optimizations (O0, O1, O2 and O3). With the weight table approach 25 million estimations / second can be performed on a single core of 3.1 GHz i5-3450 Intel host. The average estimation error was 24%. However, the approach offers higher fidelity especially with larger performance gap between (e.g. above 94% fidelity for comparing ARM0 at O0 with other optimizations). The high estimation speed with good fidelity makes this methodology an ideal cornerstone for an automated DSE.

References

1. Cai, Lukai and Gerstlauer, Andreas and Gajski, Daniel. Retargetable Profiling for Rapid, Early System-level Design Space Exploration. Proceedings of the 41st Annual Design Automation Conference, DAC 2004, 1-58113-828-8, San Diego, CA, USA 281–286
2. Lattuada, M. and Ferrandi, F. Performance modeling of embedded applications with zero architectural knowledge. IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010. 277-286.
3. Zhonglei Wang and Herkersdorf, A. An efficient approach for system-level timing simulation of compiler-optimized embedded software. 46th ACM/IEEE, Design Automation Conference, 2009, 220-225.
4. Yan Lin Aung and Siew-Kei Lam and Srikanthan, T. Compiler-assisted technique for rapid performance estimation of FPGA-based processors. IEEE International SOC Conference (SOCC), 2011, 341-346.
5. Oyamada, M. and Wagner, F. R. and Bonaciu, M. and Cesario, W. and Jerraya, A. Software Performance Estimation in MPSoC Design. Asia and South Pacific Design Automation Conference Proceedings of the 2007, 1-4244-0629-3, 38-43.
6. Gao, L. and Karuri, K. and Kraemer, S. and Leupers, R. and Ascheid, G. and Meyr, H. Multiprocessor performance estimation using hybrid simulation. 45th ACM/IEEE Design Automation Conference, 2008, 325-330.
7. Yonghyun Hwang and Abdi, S. and Gajski, D. Cycle-approximate Retargetable Performance Estimation at the Transaction Level. Design, Automation and Test in Europe, 2008. DATE '08, 3-8.
8. Javaid, H. and Janapsatya, A. and Haque, M.S. and Parameswaran, S. Rapid runtime estimation methods for pipelined MPSoCs. Design, Automation Test in Europe Conference Exhibition, 2010, 363-368.
9. Stattelmann, S. and Bringmann, O. and Rosenstiel, W. Fast and accurate source-level simulation of software timing considering complex code optimizations. Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE, 2011, 486-491.
10. Mohanty, S. and Prasanna, V.K. Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures. 15th Annual IEEE International ASIC/SOC Conference, 2002, 160-167.
11. Samsung Electronics. 32 bit CMOS Microcontroller User's Manual. S3C2440A, Jul, 2004, Rev 1.
12. Matthew Guthaus and Jeffrey Ringenberg and Dan Ernst and Todd Austin and Trevor Mudge and Richard Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. IEEE International Workshop on Workload Characterization WWC-4, Dec, 2001, 3-14.
13. V. Zivojnovic and J. Martinez and C. Schlager and H. Meyr. DSPstone: A DSP-Oriented Benchmarking Methodology, The International Conference on Signal Processing Applications and Technology, 1994, 715-720.
14. Analog Devices. Blackfin Embedded Processor. ADSP-BF527, 2013, Rev. D.
15. Jan Gustafsson and Adam Betts and Andreas Ermedahl and Bjorn Lisper. The Malardalen WCET Benchmarks – Past, Present and Future. WCET2010, 137–147, 2010, Brussels, Belgium.
16. Achterberg, T. Mathematical Programming Computation. SCIP: Solving Constraint Integer Programs, 2009, Jul, 1-41.