Center for Embedded Computer Systems
Technical Report

Cycle-Approximate Estimation-Based Mapping

Kyoungwon Kim and Daniel D. Gajski

CECS Technical Report # 14-10
June 2014

WWW.CECS.UCI.EDU

ABSTRACT

This report describes Cycle-Approximate Estimation-Based Mapping (CAEBM) of a Model of Computation (MoC) to the given multi-core platform. In previous work, estimation precedes mapping, while cycle-approximate estimation follows mapping. Therefore, the estimates given by the existing estimation techniques is not vey accurate, lowering the quality of mapping. Cycle-approximate estimation improves the mapping process but requires changes to it.
Our CAEBM is driven by recent changes in estimation technologies that allow fast cycle-approximate estimation. With an initial mapping, CAEBM iteratively uses cycle-approximate estimation and heuristics to improve the mapping in terms of execution time. A case study is performed with a multimedia application, in parallel, an MP3 decoder and JPEG encoder. CAEBM reduces the execution time by at least 36.3%.

University of California, Irvine

Cycle-Approximate Estimation-Based Mapping

Technical Report CECS-14-10

Kyoungwon Kim and Daniel D. Gajski

Center for Embedded Computer Systems,
University of California, Irvine
Irvine, CA, 92697, USA
{kyoungk1,gajski}@uci.edu

June, 3, 2014

Abstract

This paper describes Cycle-Approximate Estimation-Based Mapping (CAEBM) of a Model of Computation (MoC) to the given multi-core platform. In previous work, estimation precedes mapping, while cycle-approximate estimation follows mapping. Therefore, the estimates given by the existing estimation techniques is not vey accurate, lowering the quality of mapping. Cycle-approximate estimation improves the mapping process but requires changes to it.

Our CAEBM is driven by recent changes in estimation technologies that allow fast cycle-approximate estimation. With an initial mapping, CAEBM iteratively uses cycle-approximate estimation and heuristics to improve the mapping in terms of execution time. A case study is performed with a multimedia application, in parallel, an MP3 decoder and JPEG encoder. We use two different algorithms to find the initial mapping for CAEBM. CAEBM reduces the execution time by at least 36.3%.

Contents

List of Figures

List of Tables

1 Introduction

A serious design challenge is posed by the enormous growth in the design complexity of embedded system designs has been led by ever-increasing user demands and the non-functional constraints that embedded systems are required to meet [1]. A popular way to meet the challenge, according to major technology roadmaps for semiconductors, is to raise the level of abstraction of system designs to the Electronic System Level (ESL) [2].

Many ESL design methodologies and tools begin the design process by capturing the functionality of the given applications in a Model of Computations (MoC). Capturing the system's functionality is followed by analysis, platform selection, and mapping. A survey of such design methodologies and tools can be found [3].

Yet in the mapping process of such design methodologies and tools, a dilemma arises. The dilemma is not unlike the "chicken and egg" conundrum. We call it the "estimation or the mapping?" dilemma. Estimation should precede mapping, while cycle-approximate estimation follows mapping.

Previous works take on the dilemma by selecting "estimation." This is partially because cycle-approximate estimation took a long time.

In recent years, fast cycle-approximate estimation technologies such as [4] and [5] have been proposed. Nonetheless, cycle-approximate estimation still calls for mapping to be done and thus cannot be fed to the input of the existing mapping processes.

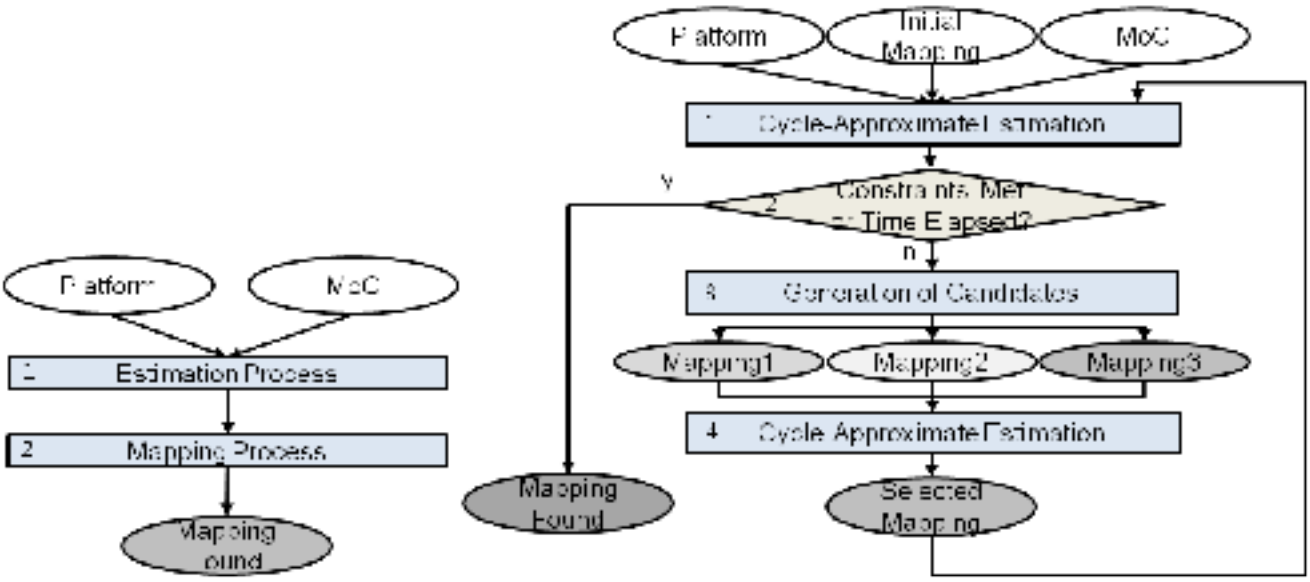
CAEBM is the first approach that is driven by the new fast cycle-approximate estimation. CAEBM is an intuitive way of thinking as far as the facilitating of a mapping process via fast cycle-approximate estimation. Since cycle-approximate estimation needs mapping and is not so fast at covering the entire design space, CAEBM is a local search iteratively using cycle-approximate estimation and heuristics.

In this paper, we minimize the execution time of the system as much as possible with the help of cycle-approximate performance estimation technologies. The platform is given under cost constraints. A case study is performed with a multimedia application running an MP3 decoder and JPEG encoder in parallel. For comparison, we choose two algorithms to find initial mappings. The first is a variant of Strength Pareto Evolutionary Algorithm (SPEA) [6]. The second is a Load Balancing algorithm [7] that iteratively finds the local optimal in terms of computation.

The organization of the rest of this paper is as follows. Section 2 reviews the previous work. Section 3 describes the MoC we use in this paper. Section 4 explains CAEBM. Sections 5 discusses the case study and Section 6 offers the conclusion.

2 Previous Work

The problem of mapping a MoC to the given heterogeneous or homogeneous platform is a complicated one. In the past, a large volume of research has tried addressing the problem. Many different MoCs are used to capture and analyze the system. Various algorithms such as Integer Linear Programming (ILP), graph-based approaches, Ant Colony optimization, various multi-objective evolutionary algorithms such as SPEA or SPEA II [8] are used to perform single or multi objective optimizations with the given MoC and the platform. [9], [10], [11] and [12] focus on Synchronous Data Flow (SDF) [13]. [6] [14] target process network models such as a Kahn Process Network (KPN) [15]. [16] uses dynamic data flow models to specify and analyze the



(a) Previous Work (b) CAEBM

Figure 1: The Flow of A Mapping Process

system. [17] assumes that the given application is converted to a Hierarchical Task Graph (HTG) [18].

As depicted in Fig. 1a, these works required the estimation process to precede the mapping process. In such a case, several crucial factors impacting the optimization goals may go unconsidered. Examples include bus contention, RTOS scheduling policies and overhead of each RTOS operations. Nonetheless, since cycle-approximate estimation was very slow, it might be the only way to answer to "estimation-or-the-mapping" dilemma.

In recent few years, fast cycle-approximate estimation technologies have been deployed [4] [5]. Now, cycle-approximate estimation achieves both accuracy in estimation and speed, contradictory achievements. The key is to annotate the application based on the given mapping and to generate a simulation model executed on the host machine(s) from the mapping. [19] added RTOS overhead estimation to the fast cycle-approximate estimation. However, mapping must still precede cycle-approximate estimation, which indicates that cycle-approximate estimation cannot be fed to the input of the existing mapping processes. Our CAEBM implements the changes in the mapping process that are required to utilize fast cycle-approximate estimation. Figure 1 shows the differences between CAEBM and previous work. CAEBM starts with an initial mapping instead of a given estimation. The given mapping is cycle-approximately evaluated and heuristics generates a small number of candidates to be cycle-approximately estimated. The loop terminates if the time limit is elapsed or the design contraints are met. Since the initial mapping is given by any previous work, CAEMB is also complementary to the previous work.

3 Application Model

In this paper, we use a Program State Machine (PSM) [7] to capture the application. A general MoC should have concurrency, the concept of states, imperative program-

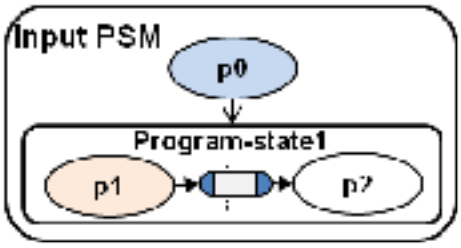


Figure 2: Application Model

ing languages, dynamic data-oriented behaviors, structural and behavioral hierarchy, etc. PSM is one such general MoC.

An application is a hierarchical program state. Each program state can be recursively defined as either a leaf program state called a process, a decomposition of parallel program states, or state transitions among sequential program states. Any pair of program states may communicate over one or more channels.

Figure 2 shows a simple PSM example. The PSM is sequentially decomposed into two program states: p0 and program state1. P0 is a process, or, interchangeably, a leaf program state. A process is a function written in an imperative programming language such as C. Program state1 is also decomposed into two parallel processes: p1 and p2. P1 and p2 communicate over a channel. Likewise, in general, p0 and p1 can also communicate over a channel such as a memory channel.

4 Cycle-Approximate Estimation Based Mapping

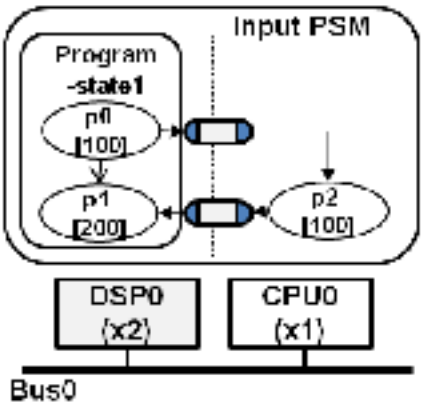
4.1 Overview

Figure 3 describes how CAEBM works. Figure 3a is the input PSM and the platform with two processing elements. DSP0 is twice as fast as CPU0. The numbers on each process is the execution time of the process on CPU0. In this example, the major problem is to select PE to locate p2.

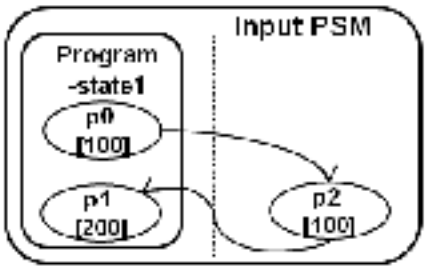
The assumption in this example is as follows: Although p2 seems to be in parallel with the others, p0, p2, and p1 run sequentially due to the data dependency. In this simple example, the actual order of execution may be easily known ahead of time. However, in general, there are at least several tens of processes and channels. The execution order may vary depending on RTOS scheduling policies, communication routing, bus arbitration policies and many others. Thus, the actual execution order is not predictable before cycle-approximate estimation.

Figure 3c shows a mapping based on estimation that ignores data dependencies. The mapping compares Figure 3e and Figure 3g, although Figure 3e is different from the actual timeline of Figure 3c. Even though the mapping algorithms resulting in Figure 3c is the optimal, the wrong estimation misleads the design decisions. The estimated execution time of mapping in Figure 3c is 150, while the actual execution time is 250.

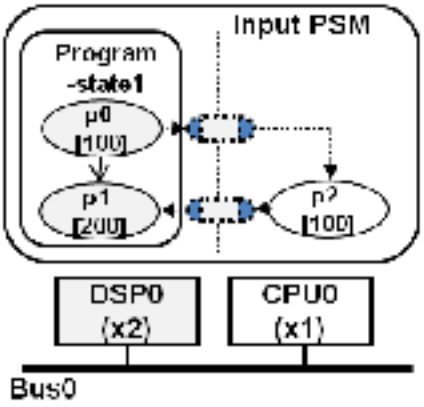
CAEBM fails to see every possible mapping. However, CAEBM compares mapping near the initial mapping depicted in Figure 3c based on cycle-approximate estimation. Thus, CAEBM makes a decision based on the right timelines: Figures 3f and 3g. Since these three processes are completely sequential, running all of them on the same PE is optimal in terms of execution time. Therefore, CAEBM move p2 to DSP0.



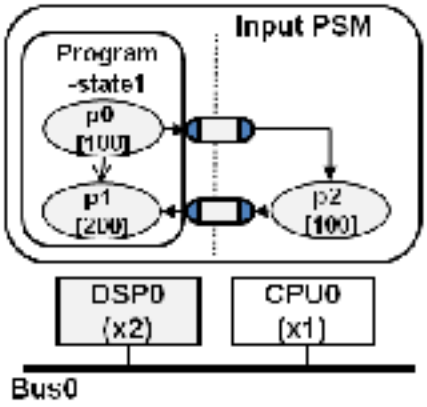
(a) Platform and MoC



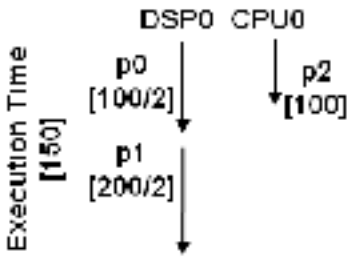
(b) Actual Execution Order Due to Data Dependencies



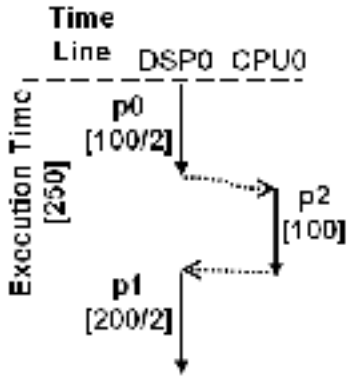
(c) Estimation Ignoring Data Dependencies and Mapping



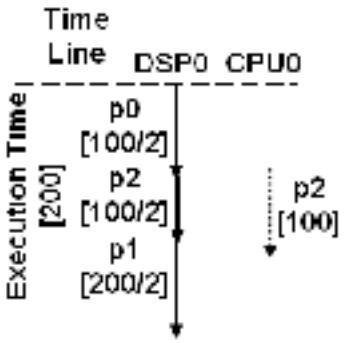
(d) Improvement by CAEBM



(e) Estimated Timeline of Mapping (c)



(f) Actual Timeline of Mapping (c)



(g) Estimated/Actual Timeline of Mapping (d)

Figure 3: CAEBM

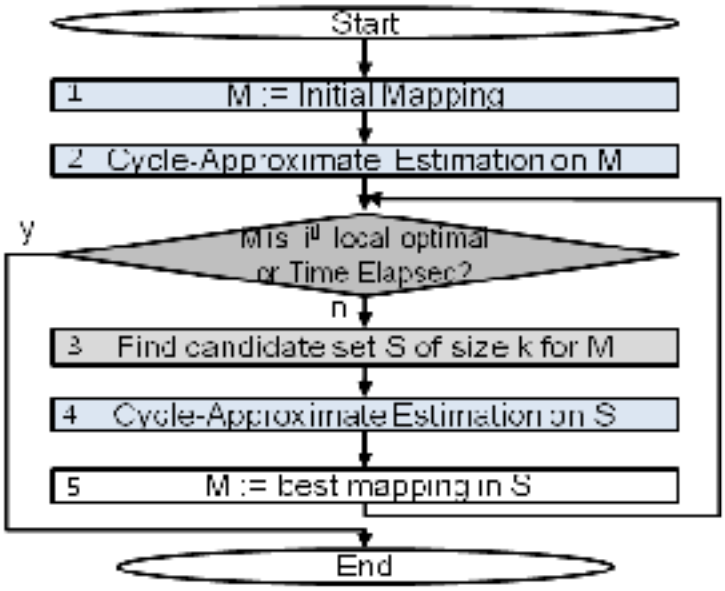


Figure 4: Flow Chart of CAEBM

4.2 Algorithm

Figure 4 describes the algorithm of CAEBM. An initial mapping is given. Cycle-approximate estimation is performed on the initial mapping. CAEMB uses heuristics and this cycle-approximate estimation to construct a set of candidates for the mapping that is fed to the input of the next iteration. From the constructed set, CAEBM identifies the best candidate. This decision is based on the cycle-approximate estimation on each candidate in the set.

The loop terminates if the given time is elapsed. In addition, the loop terminates if two local optimums are found. A local optimum is defined as the mapping satisfying the following two properties:

- A local optimum found at i^{th} iteration is better than the mapping constructed at $(i-1)^{th}$ iteration.
- A local optimum found at i^{th} iteration is better than any other mapping in the candidate set S constructed at i^{th} iteration.

The local optimum that comes first may not be sufficiently close to the global optimum in many applications. Thus, CAEBM finds i^{th} local optimum instead of the first local optimum. i could vary depending on the given applications and we use 2 in this paper. The size of the candidates set S is k. The greater k is, the smaller the number of iterations. k could also vary and we use 2 for it.

In construction of the candidate set S, heuristics are necessary. If P is the set of processes and V is the set of PEs, the number of different mappings can be as high as $\|P\|^{\|V\|}$. The $\|P\|^{\|V\|}-1$ mapping excluding the current mapping M could be elements of S. Performing cycle-approximate estimation on all of these mapping is not feasible. Therefore, we use heuristics using the cycle-approximate estimation on M, choose only k candidates among the $\|P\|^{\|V\|}$ mapping and perform cycle-approximate estimation on the selected candidates only.

Since CAEBM is a local search, we construct each element of S by selecting a single process and move it to an other PE. Therefore, the objective function f is defined with

domain P and codomain V as follows:

$$\begin{aligned} f(p_i, v_j) = w_0 \sum_{p \in v_j \cup \{p_i\}} ExecutionTime(p) \\ + w_1 \sum_{p \in v_j} CommOverhead(p_i, p) + w_2 Par(p_i, v_j), \\ where\, p_i \in P,\, v_j \in V,\, w_0,\, w_2 \geq 0,\, and\, w_1 \leq 0 \end{aligned} \tag{1}$$

A small f indicates p_i is recommended to be mapped to v_j. The idea is to balance computational loads, to map the processes together if there is large data transactions between the processes, and to exploit parallelism. Thus, w_0 and w_2 is a positive coefficient, while w_1 is negative.

We compute f for all pairs of processes and PEs, and finds k pairs of p_i and v_j that have the smallest possible f and meet the following conditions as well:

- p_i is not now on the v_j.
- Moving p_i to v_j does not fall into the mapping already visited.

Execution time of each process is in a table given ahead of CAEBM. We modified [5] to print the execution time of each process on each PE by a single TLM simulation. To compute the communication-overhead function, we record the total number of bytes in the transactions for every pair of processes respectively. This procedure can be completed by the same single TLM simulation as well. We divide the number of bytes by the bandwidth of the best route between p_i and v_j.

Communication and computation must be taken into consideration and that is what $ExecutionTime$ and $CommOverhead$ functions are for. However, we include the Par function in the objective function f.

Par function represents approximately how much the process p_i can be executed in parallel with the processes on v_j. To define Par, we define $ParProcesses$ first. For any given two different processes p and q, $ParProcesses(p, q)$ is the minimum execution time of the processes if they are parallel processes in the PSM. Otherwise, it is the sum of the execution time of p and that of q. Par is defined as follows:

$$\begin{aligned} Par(p_i, v_j) = \sum_{q \in v_j} ParProcess(p_i, q)\ , \\ where\, p_i \in P\, and\, v_j \in V \end{aligned} \tag{2}$$

5 Case Study

5.1 MP3 Decoder And JPEG Encoder

We have chosen a computation-intensive multimedia application running a MP3 decoder and JPEG encoder in parallel. The PSM capturing the entire application is decomposed into two parallel program states. Two program states, MP3 decoder and JPEG encoder, are running in parallel. The number of levels of hierarchy is up to 3. There are 13 processes and 17 channels. A platform is randomly given under cost constraints. The platform consists of five different PEs and two buses.

For comparison, we choose two different algorithms to find the initial mappings for CAEBM. CAEBM starts each initial mapping, respectively. The first algorithm is Load Balancing (LB) [7]. LB is a heuristic that optimizes computation only. We also modify SPEA. SPEA [6] performs multi-criteria optimizations including approximation of execution time. We changed it to have a single optimization: execution time.

For each initial mapping, firstly, we measure the execution time by TLM simulation and compare the execution time to the estimate that each mapping uses. The objective function of SPEA is the maximum processing time, which is named by the authors. The processing time of a PE v is the sum of $f_v{}^e$ and $f_v{}^c$. $f_v{}^e$ is the sum of execution time of each process on the PE. f^c is the sum of the delays due to data transactions in which any PE on the process is involved. The objective function is the maximum of $f_v{}^e + f_v{}^c$. The objective function of LB excludes $f_v{}^c$ from the formula.

Table 1: Errors in Estimation: SPEA and LB

Mapping	Estimate	Execution Time	Error Percentage Time
SPEA	90.77 ms	112.17 ms	19.5%
LB	12.39 ms	65.71 ms	81.1%

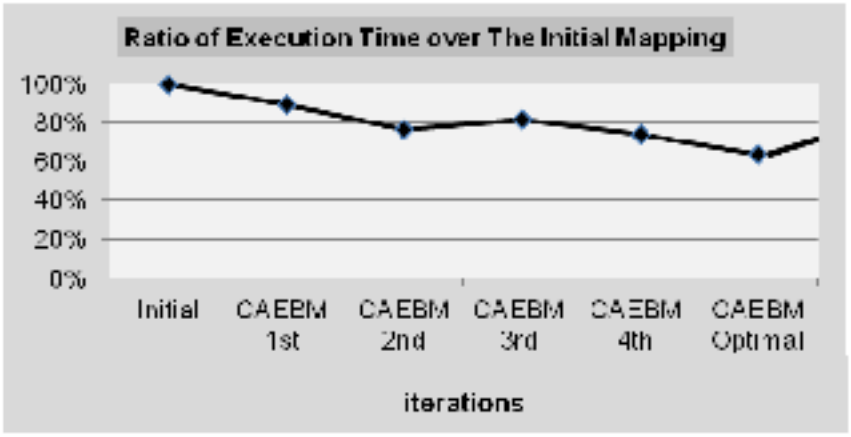
TABLE 1 shows the estimates and the execution times. The estimates are different from the execution time by 19.5% and 81.1%. In addition to communication and computation, process scheduling due to data dependencies and control dependencies also greatly impacts the execution time but is not included in either objective function. Therefore, the errors are not small, which indicates there is room for improvement.

Figure 5 shows the changes in execution time during the improvement process of CAEBM. In Figure 5a and Figure 5b, the initial mapping is LB and SPEA. CAEBM iteratively improves each of these initial mapping. The execution time is recorded at every iteration. The ratio of the execution time over the initial mapping is depicted in Figure 5a through Figure 5b. The X-axis is the number of iterations, while Y-axis represents the ratio of the execution time over the initial mapping. The CAEBM improvement process stops since the second local optimum is found. In all cases, there are improvements in execution time. The execution time is reduced at least by 36.3% compared to the initial mapping.

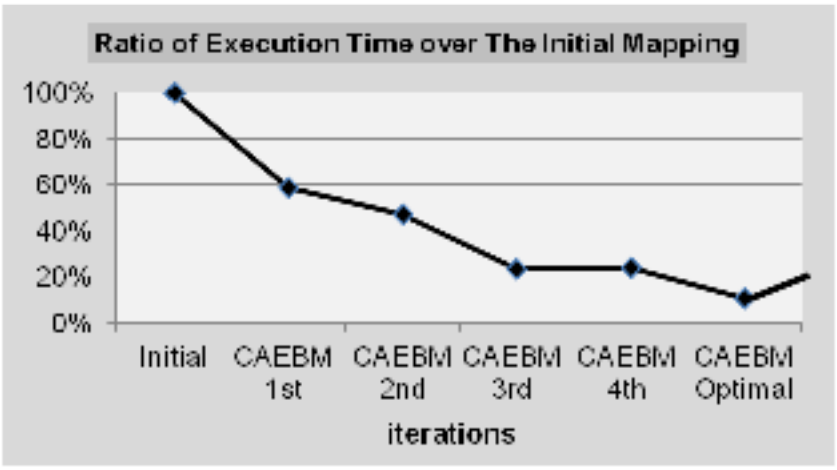
SPEA basically targets applications captured in a process network. In a process network, every task is assumed to run in parallel even though the tasks run sequentially due to data dependencies. In the estimation used by LB, niether communication overhead nor control flow of the application is considered. On the contrary, the heuristics used by CAEBM has a tendency to separate parallel tasks. Thus, the execution time is reduced during iterative iterations.

6 Conclusion

In this paper, we have described Cycle-Approximate Estimation-Based Mapping, which offers different answers to the "which comes first, estimation or the mapping?" dilemma. The mapping relies on the help of a recent remarkable change in estimation technologies: cycle-approximate estimation. With an initial mapping given by any previous work and its cycle-approximate estimate, CAEBM iteratively uses cycle-approximate estimation and heuristics to conduct a local search to meet the design constraints. The optimization goal is to minimize execution time. The case study is



(a) Initial Mapping: LB



(b) Initial Mapping: SPEA

Figure 5: Changes in The Ratio of Execution Time over The Initial Mapping During CAEBM's Process on MP3 + JPEG Application

performed with a multimedia application running an MP3 decoder and JPEG encoder. According to the case study, CAEBM improves, in terms of reduction of execution time, the design given by any initial mapping algorithm by 36.3%.

References

[1] Hermann Kopetz. The complexity challenge in embedded system design. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 3–12, 2008.

[2] International technology roadmap for semiconductor, 2011.

[3] A. Gerstlauer, C. Haubelt, A.D. Pimentel, T.P. Stefanov, D.D. Gajski, and J. Teich. Electronic system-level synthesis methodologies. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(10):1517–1530, 2009.

[4] K. Karuri L. Gao, Stefan Kraemer, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. Multiprocessor performance estimation using hybrid simulation. In *Pro-*

ceedings of the Design Automation Conference (DAC '08), Anaheim, CA, USA, jun 2008.

[5] Y. Hwang, S. Abdi, and D. D. Gajski. Cycle-approximate retargetable performance estimation at the transaction level. In *DATE*, Munich, Germany, April 2008.

[6] Cagkan Erbas, Selin C. Erbas, and Andy D. Pimentel. A multiobjective optimization model for exploring multiprocessor mappings of process networks. pages 182–187. ACM, 2003.

[7] Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[8] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, 2001.

[9] Jing Lin, A. Srivatsa, A. Gerstlauer, and B.L. Evans. Heterogeneous multiprocessor mapping for real-time streaming systems. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 1605–1608, 2011.

[10] Eckart Zitzler, Jrgen Teich, and Shuvra S. Bhattacharyya. Evolutionary algorithms for the synthesis of embedded software. *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, 8, 1999.

[11] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 897–902, 2010.

[12] Hyunok Oh and Soonhoi Ha. A hardware-software cosynthesis technique based on heterogeneous multiprocessor scheduling, 1999.

[13] S.S. Bhattacharyya, P.K. Murthy, and E.A. Lee. *Software Synthesis from Dataflow Graphs*. Springer, 1996.

[14] Mark Thompson, Hristo Nikolov, Todor Stefanov, Andy D. Pimentel, Cagkan Erbas, Simon Polstra, and Ed F. Deprettere. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '07, pages 9–14, New York, NY, USA, 2007. ACM.

[15] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[16] Christian Haubelt, Thomas Schlichter, Joachim Keinert, and Mike Meredith. Systemcodesigner: automatic design space exploration and rapid prototyping from behavioral models. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 580–585, New York, NY, USA, 2008. ACM.

[17] F. Ferrandi, C. Pilato, D. Sciuto, and A. Tumeo. Mapping and scheduling of parallel c applications with ant colony optimization onto heterogeneous reconfigurable mpsocs. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 799–804, Jan.

[18] M. Girkar and C.D. Polychronopoulos. Automatic extraction of functional parallelism from ordinary programs. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):166–178, 1992.

[19] Y. Hwang, S. Abdi G. Shirner, and D. D. Gajski. Accurate timed rtos model for transaction level modeling. In *DATE*, 2008.