Adaptive Metrics for System-Level Functional Partitioning

Giovanni Agosta, Marco Santambrogio
Politecnico di Milano, Italy
Dipartimento di Elettronica e Informazione
Seda Ogrenci Memik
Northwestern University, IL USA
Department of EEECS

Abstract

Codesign is a design methodology aiming at the integration of both hardware and software into the same System-on-Chip architecture. This paper presents a new methodology to address the Codesign problem based on the adaptive computation approach. Adaptivity implies that due to input changes the output of the system is updated by only re-evaluating those portions of the program affected by the changes. This concept has been applied successfully to acceleration of software in the past. In this work, we propose to utilize the concept of adaptivity in a different context as a means to collect information on the dynamics of a complex computation. We introduce a new metric called codesign intensity ordering metric. The codesign intensity ordering metric is instrumental in efficiently identifying the computationally intensive portions in a computation as well as relatively static regions.

1 Introduction

In a recent study [Wol03], presenting the milestones of the last ten years of codesign, it is stated that the primary task of HW/SW Codesign is to increase the predictability of embedded system design, providing both analysis and synthesis methods. The former to inform the designers whether their system meets the specifications, the latter to support them in the evaluation of many different design methodologies. The need of this integration becomes clear as the decision of implementing a functionality as a dedicated piece of hardware or as software running on a general purpose processor depends mostly on the estimation of the performances of the overall system and on its cost. This paper introduces the codesign intensity ordering metric to guide the solution for a functional partitioning at the system level. The codesign intensity ordering metric is particularly intended to differentiate and categorize across different portions in a computation between a range of computationally intensive and relatively static regions.

Camposano and Brayton [CB87] have been among the first to introduce a new methodology for defining the hardware (HW) and the software (SW) side of a system. Their techniques take into account a partitioner driven by the closeness metric. This metric introduces a new way of tackling the codesign problem. It provides the designer a means to measure the quality of a solution that implements two different components on the same side of a partition; HW or SW. The bottleneck of this approach is the granularity of the input specification, since it is based on a fine-grain partitioning. Vahid and Gajsky [VG95] propose a set of closeness metrics for a functional partitioning at the system level. These metrics are based on a functional specification, the Access Graph, which represents the access relations among the processes that had to be partitioned. Another alternative approach [GSV03] presents the benefits and feasibility of dynamic partitioning using prototyping tools, executing on a lean embedded processor. The evaluation metric we propose in this work falls into the category of tools that can guide designers or automated partitioning tools during the functional partitioning of a complex computation within codesign. However, our approach is inspired by a different paradigm called adaptive computation.

2 Overview of Adaptive Computing Paradigm

Our methodology is a novel way of approaching the partitioning problem based on the adaptive computation. It is based on the following idea. To find a good solution, given a system description, for its codesign problem, it is not only necessary to have data describing possible implementations for different parts of the description but also to have information on the computation that has to be performed by system. Instead of resorting to computationally expensive profiling techniques we aim to analyze the nature of computation by quickly identifying parts of the system that are affected (or not affected) by the run of different input data sets. To accomplish this, we propose a new metric based on the adaptive computation. Adaptive computation is a technique that maintains the relationship between the input of a program and its output as the input changes [ABH03]. An adaptive program, [ABH02], responds to input changes by updating its output while only re-evaluating those portions of the program affected by the change. Adaptive programming is useful in situations where input changes lead to relatively small changes in the output. In limited cases one cannot avoid a complete re-computation of the output, but in many practical applications the results of the previous computation may be re-used to obtain the updated output more quickly than a complete re-evaluation. We will provide examples of such applications in the later
sections. Based on this observation we define the set of adaptive metrics able to provide information on how the system is going to be affected by an input change. This will help us categorize various subsets in a computation into sets of nodes representing those portions of the system that have to be re-evaluated due to input changes. The first phase consists of the transformation of a non-adaptive program to an adaptive one. Basically this phase involves two steps, [ABH02]:

- first, the data structures are made modifiable by placing desired elements in modifiables. A modifiable reference is essentially a write-once reference cell that records the value of an expression whose value may change as a direct or indirect, result of changes to the inputs, [ABH02];

- second, the original program is updated by making the reads of modifiables explicit and placing the results of each expression that depends on a modifiable into another modifiable. This means that all values that directly or indirectly depend on modifiable inputs are placed in modifiables.

The key issue is to support change propagation efficiently with minimal overhead. To this end, an adaptive program records the adaptive activity during execution by means of a dependency graph. In such a dependency graph, each node represents a modifiable and each edge represents a read. We define an augmented dependency graph (ADG) as a Directed Acyclic Graph (DAG) in which each edge has an associated reader and time stamp, and each node has an associated value and time stamp. We denote a node (and corresponding modifiable) as an input if it has no incoming edges.

The second stage performs the computation of the adaptive metrics, as described in Section 2.1. Finally this phase produces as output the HW and the SW sides partitioning, guided by the Codesign intensity ordering phase. Once we have the correct high-level functional description of the system that can support the change propagation we are going to show in Section 2.1 how to use this technique for defining a new metric for the Codesign.

2.1 Change Propagation as a Novel Metric

Given an augmented dependency graph and a set of changed input modifiables, the change propagation algorithm proposed in [ABH02] updates the ADG and the output by propagating changes in the ADG. We say that an edge, or corresponding read, is invalidated if the source of the edge changes its value. We say that an edge is obsolete if it is contained within an invalidated edge. At each run of the system under study it will be possible to identify the functions that are going to be affected at each run. Creating a training environment is going to be possible to run the description under test with a known training set. In this scenario, we will be able to define, for each function, \( f_i() \), defining the system, a value \( m \) which represents the number of runs \( m_i \), computed over all the training set, in which \( f_i() \) has been affected. Knowing the training set dimension \( ts \), which is the number of tests used for defining the training environment, we define the percentage of affection for each function as \( m_i/\text{ts} \). The \( m_i/\text{ts} \) function is defined between 0 and 1. The boundary conditions as follows: 0 will be used for a function that is not going to be affected by any input sets, while 1 is assigned to a function that has to be re-evaluated for each input. Therefore, it is possible to introduce an ordering function based on the \( m_i/\text{ts} \) values, the Codesign Intensity Ordering Value (CIOV). According to this formulation it is possible to define the nature, HW or SW, of a component by considering its CIOV. The two extremes 0 and 1 represent the two well-defined cases; a component with a 0-CIOV, that means high flexibility, has to be implemented in SW, while a functionality with a 1-CIOV is a part of the system that is going to be intensively used during the life-time of the system therefore it is desirable to implement it in HW which is likely to provide a faster computation. The computed CIOV values can be used by the designer to choose the best solution or they can be used as inputs for a decision tool to drive the system partitioner to find the system description while considering further aspects such as communication and interconnect.

3 Conclusions

It has been demonstrated that for purely software acceleration purposes adaptive computing and change propagation can be performed in real-time. Many application, such as convex-hull algorithms, have been proved to speedup their execution using this approach [ABH03], [ABH02]. In this paper, we show how adaptive computation could be use to support codesign activities. After this preliminary study, we will test the characterization change propagation and the computation of CIOV metric on a set of suitable applications, e.g. frame-based video.

References


