An Evolutionary Approach to Hardware/Software Partitioning

Xiaobo (Sharon) Hu[1*], Garrison Greenwood[1*] Joseph G. D'Ambrosio[2]

[1] Western Michigan University Kalamazoo MI 49008, USA
[2] General Motors R&D Center, Warren MI 48090, USA

Abstract. In this paper, we present an approach to hardware/software codesign of real-time embedded systems. Two of the difficulties associated with codesign are handling tradeoffs among multiple attributes and exploring a large design space. We use a combination of techniques from the evolutionary computation and utility theory fields to address these problem areas. A real-time microcontroller-based design example is presented to illustrate our approach.

1 Introduction

The objective of Hardware/software codesign is to produce computer systems that have a balance of hardware and software components which work together to satisfy a requirements specification. This balance between hardware and software implementations is referred to as the *partitioning problem*. Developing efficient means of performing hardware/software partitioning is key to the automatic design of complex computer systems.

A number of hardware/software partitioning approaches have been presented (*e.g.*, see [1, 2]). Most of these partitioning approaches start with a pure software or hardware specification at the behavioral level and then attempt to find a hardware/software implementation which meets the overall constraints and optimize certain attributes. Such fine-grained approaches have difficulty in exploring a large design space consisting of multiple choices of different microprocessors or microcontrollers. Additionally, these approaches have a limited means of handling tradeoffs among multiple attributes such as cost, power consumption and design time.

The exponential size of the search space for the hardware/software partitioning problem establishes the need for some type of heuristic search technique. This paper describes the combination of an *evolutionary algorithm* (EA) with *Imprecisely Specified, Multiple Attribute Utility theory* (ISMAUT) to develop real-time embedded systems. Our use of imprecise value functions based upon a designer's preferences for establishing a survival criteria is what differentiates our approach from conventional Pareto optimal searches that use EAs. The goal in most other approaches is to enumerate the entire Pareto optimal set [3]. While certainly

* Research supported in part by an External Research Program Grant from Hewlett-Packard Laboratories, Bristol, England.

laudable, this assumes that all Pareto optimal solutions are equally preferable which is often not the case. Using preferences to influence the probability of survival allows the designer to drive the search process of the EA. This helps to prevent the EA from conducting a simple blind exploration of the tradeoff space. A design example is presented to illustrate our approach.

2 Definitions & Preliminaries

Optimization problems are essentially search problems. Each individual in the population of an EA constitutes a potential solution or *alternative* within the problem space of an optimization problem. *Objectives* define desirable properties of a good alternative and *attributes* are used to determine the degree to which a specific objective is met. An objective is normally formulated as an *objective function* with the attributes as the function's arguments. The class of such problems are called *Multipleobjective Optimization Problems* (MOPs).

The hardware/software partitioning problem is an instance of a MOP. All system specifications can be modeled as a set of functions where each function has one or more performance constraints (*e.g.*, timing). Functions can be implemented in either hardware or software. Software implemented functions acquire attributes such as the number of instructions required for execution on a specific processor. Conversely, a function implemented in hardware affects the cost and power consumption of the system. The objective is to have a designer select components from a hardware library (containing microprocessors, application specific integrated circuits (ASICs), etc.) and then assign the functions as hardware or software implementations in some optimal way. The decision of whether functions in a real-time embedded system should be implemented in hardware or software forms a tradeoff between cost and performance.

In the terminology of EAs, attribute levels (dollar cost, computational power in MIPs, etc.) are used to quantify the fitness of an alternative and the objective function is equivalent to a fitness function. The goal of the EA is to find an alternative which optimizes the fitness. More formally, suppose we are given a MOP with $\mathcal{X}$ representing the set of all feasible alternatives. Further, let $\mathcal{A}$ represent the set of n attributes. Each alternative $x \in \mathcal{X}$ has an assigned level for each $a_i \in \mathcal{A}$. We let $\mathcal{A}_x$ denote the set of attribute levels associated with the alternative x. We would like to represent the fitness of x by defining an appropriate objective function $f : \mathcal{A}_x \rightarrow \Re_+$ where $\Re_+ \in [0, \infty)$.

An intuitive metric for the optimality of MOPs would be a weighted sum of attributes. Specifically,

$$f = \sum_k w_k \, a_k \tag{1}$$

where a_k is the k-th attribute and $w_k > 0$ is its associated weight. (The weights must satisfy $\sum_k w_k = 1$. Higher weight values reflect greater importance.) Unfortunately, this format for a fitness function is a bit naive and has a number of problems. For example, there often exists conflicting goals between optimizing

attributes; some attributes should be maximized while others should be minimized. These factors make direct summation inappropriate. Also, there is the difficulty in specifying a precise weight of each attribute. Intuitively, it will be difficult to assign meaningful weights for more than 3 or 4 attributes.

The first problem above can be handled by *scaling* the attribute levels which maps the raw attribute levels to a convenient subset of $\Re_+$ (normally $0 \rightarrow 1$). Mapping all attribute levels to the same scale also facilitates an attribute trade-off analysis. As before, let $\mathcal{A}_x = \{a_1, a_2, \cdots, a_n\}$ represent the set of n attributes associated with the alternative x. Suppose there exists a set of real-valued functions $\{v_1, v_2, ..., v_n\}$ on $\mathcal{A}$ such that $v_i : a_i \rightarrow [0, 1]$ where $v_i \rightarrow 1$ as the attribute level of a_i improves[3]. The set of real-valued functions are referred to as *attribute value functions*. These functions should (as the attribute value increases) monotonically increase for a "more-is-better" attribute and monotonically decrease for a "less-is-better" attribute. Let $\tilde{a}_i$ and $\hat{a}_i$ denote the maximum and minimum levels, respectively, of the attribute a_i. Then for a "more-is-better" attribute

$$v_i(a_i) = \frac{a_i - \hat{a}_i}{\tilde{a}_i - \hat{a}_i} \tag{2}$$

and for a "less-is-better" attribute

$$v_i(a_i) = \frac{a_i - \tilde{a}_i}{\hat{a}_i - \tilde{a}_i} \tag{3}$$

Replacing each a_k in equation (1) with the corresponding $v_k(a_k)$ and assuming $\sum_k w_k = 1 \ ; \ w_k > 0$, the fitness function is now well defined.

To handle the difficulty of assigning precise weights, we resort to techniques from utility theory (which will be discussed shortly). Here we introduce the important concepts of dominance and preference. Let x and x' be two alternatives from $\mathcal{X}$ with their associated attribute level sets $\mathcal{A}_x = \{a_1, ..., a_n\}$ and $\mathcal{A}_{x'} = \{a'_1, ..., a'_n\}$, respectively. We say x *dominates* x' if $\forall i$ a_i is better than or equal to a'_i and, in addition, there exists at least one a_j such that a_j is strictly better than a'_j. The set of non-dominated alternatives lies on a surface in attribute space known as the *Pareto optimal frontier*[4]. In each generation of an EA there exists a set of non-dominated alternatives. We call this set of alternatives the *phenotypical Pareto front*.

Given two non-dominated alternatives, a designer may still prefer one over the other. This concept is expressed with the following two relationships:

R1: $x \succ x'$ (read as "x is preferred-to x'")

R2: $x \sim x'$ (read as "x is indifferent-to x'")

x and x' are indifferent when $x \not\succ x'$ and $x' \not\succ x$ indicating that there is no clear preference between them. Relationships **R1** and **R2** together establish a *partial order* on $\mathcal{X}$. If relationship **R2** does not exist (*i.e.*, $\forall\ x, x' \in \mathcal{X}$, either $x \succ x'$ or $x' \succ x$), then a *total order* on $\mathcal{X}$ is established. Preference is purely subjective and thus is different from dominance (which is purely objective). It is easily shown that our preference relationship does not violate dominance relationships [4].

[3] Some authors in this context will refer to v_i as a "utility function".

[4] Some authors refer to this as the Pareto optimal set.

3 Overview of the Design Approach

The level of abstraction that we have adopted for modeling hardware and software components is called the *configuration level* [2]. At the configuration level, hardware is modeled as resources with no detailed functionality and software is modeled as tasks utilizing the resources. At this high level of abstraction, we are able to evaluate various partitioning schemes up front, which, in turn, guides lower level design efforts. The goal of configuration-level design is to determine which functions should be implemented in dedicated hardware circuits and which should be in software, what processors and ASICs should be used, and which software components should be executed by which processor. Our

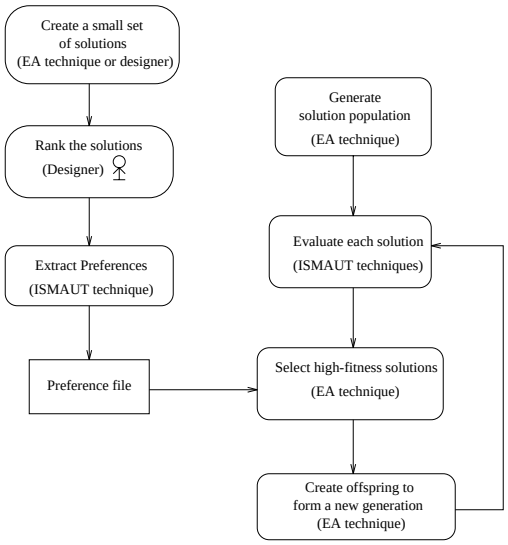

Fig. 1. Hardware/Software Partitioning System

approach to performing hardware/software partitioning (shown in Figure 1) can be summarized as follows. Initially, we obtain a small set of alternatives which can be either extracted from the initial population of an EA or which can be given by the designer. The designer then qualitatively ranks these alternatives. ISMAUT can then be used to define a set of designer's preferences which are stored into a file. (The reader is referred to [5] for details on ISMAUT.)

The EA works in a conventional manner of using genetic operators to generate new potential solutions. ISMAUT returns the dominant solutions (*i.e.*, solutions which are at least as good as any other solutions) which the EA uses to determine which individuals survive. This process continues until a satisfactory solution has been found or a fixed number of generations have been produced and evaluated. We have implemented this approach as a software package called **EvoC** (Evolutionary Codesign).

4 Evaluating Alternatives

To reflect the designer's preferences in the tradeoff of different attributes, we make use of imprecise value functions which are taken from the field of utility theory [6]. An imprecise multi-attribute value function corresponding to the alternative x has the following form:

$$V_x = \sum_k w_k \; v_k(a_k) \tag{4}$$

where $w_k \in \Re_+$ is the weight and $v_k(a_k)$ is the attribute value function for attribute a_k. All weights must satisfy $\sum_k w_k = 1 \; ; \; w_k > 0$.

V_x is imprecise in the sense that each w_k does not have a specific assignment, but is constrained by preferences among attributes. Such constraints can be formulated based upon preferences between distinct alternatives (provided by the designer or generated by an EA). For example, given two alternatives $x, x' \in \mathcal{X}$ with corresponding attribute level sets $\mathcal{A}_x$ and $\mathcal{A}_{x'}$ for which the designer has decided that $x \succ x'$. This preference relationship is captured as follows [4]:

$$x \succ x' \iff V_x - V_{x'} = \sum_{k=1}^{n} w_k[v_k(a_k) - v_k(a'_k)] > 0. \tag{5}$$

Such an expression defines a constraint for the attribute weights. When several alternative pairs are ranked by the designer, a series of such constraints are defined. The set of possible w_k values are confined to a subspace $W \subset \Re_+^n$ where $\Re_+^n$ is the n-dimensional space of positive real numbers.

Using the attribute value functions and the constraint subspace W, other configurations created by running the EA may be evaluated. More specifically, alternatives x'' and x can be compared by solving the following linear programming problem:

$$\begin{aligned} &\text{Minimize (w.r.t. } w_k\text{):} \qquad \textstyle\sum_k w_k[v_k(a''_k) - v_k(a_k)] \\ &\text{Subject to:} \qquad w_k \in W \end{aligned} \tag{6}$$

Then x'' is preferred to x if Equation (7) is true.

$$z = \min \sum_k w_k[v_k(a''_k) \; - \; v_k(a_k)] > 0 \tag{7}$$

However, knowing that $z \leq 0$ is not sufficient to determine preference. We must reverse the terms in Equation (7) as shown below.

$$\overline{z} = \min \sum_k w_k[v_k(a_k) \; - \; v_k(a''_k)] > 0 \tag{8}$$

Now, if Equation (7) is false and Equation (8) is true, then $x \succ x''$. If both equations are false, then x and x'' are pairwise indifferent. More detailed information on these concepts can be found in [4].

It is important to emphasize that the initial ranking of the selected alternatives is done merely to obtain the constraint subspace W. W is then used in the series of linear programming problems that must be solved to conduct pairwise comparisons between alternatives.

5 Evolutionary Algorithms for Partitioning

For the partitioning problem, individuals in the search space are design alternatives. The data structure for each individual consists of a binary vector with a bit position allocated to each function. A logic 0 (1) in a bit position indicates that the respective function is implemented in software (hardware). There is also an integer which indicates the selected processor. In some cases there are several choices for a hardware implementation (*e.g.*, coprocessor or ASIC). The binary vector is then appended with additional bits to account for these different choices. Of course appended bits are ignored if a software implementation is selected.

The EA terminates after a fixed number of generations (Γ) have been produced and evaluated or earlier if an acceptable assignment has been found. The EA algorithm is implemented as follows:

1. Create an initial population of μ design alternatives by randomly assigning functions as either hardware or software implementations.
2. Conduct a tournament to select alternatives for reproduction. Each selected alternative generates one offspring by applying mutation operators (described below). This creates a population with a total of 2μ alternatives.
3. Rank all alternatives according to their fitness.
4. Deterministically select the μ alternatives with the highest fitness.
5. Proceed to step 2 unless an acceptable solution has been found or Γ generations have been evaluated.

Offspring are created by applying one of three mutation operators. M_1 randomly selects one bit in the binary vector and complements it. The associated function is then reassigned from a hardware (software) implementation to a software (hardware) implementation. M_2 modifies hardware assignments which are identified by bits appended onto the binary vector. A third mutation operator M_3 modifies the integer field in the data structure to select a different processor. Operator M_i is applied to a parent with probability p_i where $\sum_k p_k = 1.0$.

We use the preference relationship discussed in Section 2 to assign fitness to each alternative. Alternative x is said to have a higher fitness over alternative x' if $x \succ x'$. However, this will typically establish only a partial order. A total ranking of the alternatives (based upon a technique described by Goldberg [8]) can be done as follows. Using ISMAUT, identify all preferred alternatives, assign them rank 1 and then remove them from further contention. A new set of preferred alternatives can then be found, ranked 2, and so on until all alternatives have been ranked. Note that any alternatives which violate constraints (*e.g.*, failure to meet a deadline) will not be preferred and thus ISMAUT will assign

these a high numerical rank. Note that fitness assignments based upon preference relationships preserves existing dominance relationships [4]. Therefore, the alternatives with rank 1 constitute the phenotypical Pareto optimal front. Tournament selection is used to select alternatives for the reproduction in the next generation [7]. Two distinct candidate alternatives are randomly selected from the current population and three additional distinct alternatives are randomly selected as a comparison set. If one candidate has a lower ranking than some alternative in the comparison set, and the other candidate does not, then the latter is selected for reproduction. If neither (or both) candidates have a lower ranking than some alternative in the comparison set, then a candidate is randomly chosen. (Equivalence class sharing [7] will be used in future **EvoC** versions.)

Name	Number	Activation	Deadline	Period
DigitalFilter1 (DF1)	1	0.00	46.00	104.17
DigitalFilter2 (DF2)	2	9895.83	10000.00	10000.00
DecodeSPUB (DSB)	3	0.00	83.00	208.33
DecodeSPUA (DSA)	4	83.00	138.00	208.33
ReadCAM (RC)	5	0.00	416.67	10000.00
ServiceRoutine (SR)	6	0.00	208.33	416.67
FuelCalc (FC)	7	833.33	1333.33	2500.00
SparkCalc (SC)	8	1666.67	2500.00	2500.00
ReadMAP (RM)	9	0.00	312.50	416.67

Table 1. Primary set of functions. Activation, deadline and period are in μs.

Name	Function Implemented	Instructions Executed	RAM Required	ROM Required
DF1-S	DF1	64	100	100
DF2-S	DF2	32	100	100
DSB-S	DSB	30	200	300
DSA-S	DSA	30	200	300
RC-S	RC	30	100	100
SR-S	SR	20	200	200
FC-S	FC	480	500	400
SC-S	SC	100	400	300
RM-S	RM	40	100	100

Table 2. Software modules to implement functions. RAM and ROM are measured in bytes.

Name	Functions Implemented	Cost	MIPS Available
MC1-H	CPU, RAM(2K), ROM(2K), DF1,DF2,DSB,DSA	3.50	1.30
MC2-H	CPU, RAM(2K), ROM(2K), TC(32)	3.25	1.50
MC3a-H	CPU, RAM(4K), TC(16)	5.25	2.50
MC3b-H	CPU, RAM(4K), DF1,DF2, DSB,DSA	6.25	2.50
MC4a-H	CPU, RAM(2K), DF1, DF2, DSB, DSA, TC(14)	3.75	1.70
MC4b-H	CPU, RAM(2K), DF1, DF2, DSB, DSA, TC(14)	3.25	1.35
MC4c-H	CPU, RAM(2K),TC(16)	2.50	1.70
P1-H	CPU, RAM(2K), ROM(2K)	2.00	1.43
P2-H	CPU	13.00	13.50
ASIC1-H	DF1,DF2,DSB,DSA	2.50	-
PIO1-H	TC(16)	1.00	-
RAM1-H	RAM(2K)	2.00	-
ROM1-H	ROM(2K)	1.00	-

Table 3. Hardware modules to implement functions.

6 Design Example and Discussions

The example embedded system we used is similar to the one discussed in [2]. Table 1 gives the system specification which has nine functions with real-time constraints. Activation time indicates the earliest start time for the first execution while deadline is the time by which the function must be completed after this first activation. Period indicates how often the function is required to execute. The system's attributes are component cost, critical excess MIPS ($\triangle_c$), and feasibility factor (λ). Critical MIPs indicates the amount of computational power yet available for future expansion [2]. Feasibility factor reflects the ability of an implementation to meet all temporal requirements. λ is dependent on the scheduling algorithm used and indicates the probability that the target processor has sufficient computational power to meet all of the timing requirements of the tasks assigned to it. A methodology for calculating λ can be found in [2].

Modules in the software library are listed in Table 2. For this example, we assume the software characterization given are valid for every processor. The first 4 functions may be implemented in hardware. Table 3 lists some of the hardware modules available for this system. The modules include: microcontrollers (MC), processors (P), ASICs, standard peripherals (PIO), timing channels (TC), RAM, and ROM.

Number	Part Set	Cost	Feasibility Factor (λ)	Critical Excess Req. Ratio (Δ_c)
1	DF1toTC, DF2toTC, DSB-S, DSA-S, P1-H, PIO1-H	3.00	0.013	0.011
2	DF1toTC, DF2toTC, DSB-S, DSA-S, MC2-H	3.25	0.094	0.081
3	MC1-H	3.50	0.706	0.183
4	DF1toTC, DF2toTC, DSB-S, DSA-S, MC4c-H, ROM1-H	3.50	0.325	0.281
5	MC4b-H, ROM1-H	4.25	0.899	0.233
6	P1-H, ASIC1-H	4.50	1.000	0.313
7	MC4a-H, ROM1-H	4.75	1.000	0.583
8	DF1toTC, DF2toTC, DSB-S, DSA-S, MC3a-H, ROM1-H	6.25	1.000	1.081
9	MC3b-H ROM1-H	7.25	1.000	1.383
10	DF1-S, DF2-S, DSB-S, DSA-S, P2-H, RAM1-H, ROM1-H	16.00	1.000	11.460
11	DF1toTC, DF2toTC, DSB-S, DSA-S, P2-H, PIO1-H, RAM1-H, ROM1-H	17.00	1.000	12.080
12	P2-H, ASIC1-H, RAM1-H, ROM1-H	18.50	1.000	12.380

Table 4. Pareto-optimal set of alternatives found by exhaustive search. Note that all alternatives also include: RM-S,SC-S, FC-S, SR-S, RC-S.

The only constraint used for this problem is that λ must be greater than zero which insures the design will meet real-time constraints. In most real-world problems additional constraints may be required to guarantee compatibility between the hardware modules (*e.g.*, coprocessors can only be interfaced to microprocessors from the same manufacturer). This size problem is small enough so that exhaustive search can be used to enumerate the Pareto optimal set within a reasonable amount of computational time. There are a total of 12 Pareto optimal alternatives which are identified in Table 4. Two tests were conducted using an EA with a population size of μ=20. (This example was small enough so that $\mu = 20$ was sufficient. For more complex problems μ should be several times larger.) The EA was run for Γ=50 generations with mutation probabilities of $p_1 = 0.8$, $p_2 = 0.05$, and $p_3 = 0.15$. After Γ generations had been processed, alternatives with rank 1 were output.

The first test ranked the three given alternatives according to cost (lower cost implies higher ranking). **EvoC** correctly identified alternatives 2 and 4 from Table 4. The second test ranked the three given alternatives according to Δ_c(higher value implies higher ranking). The EA consistently identified alternatives 10, 11, and 12 from Table 4. The significant aspect of these results is not simply that the EA can find *any* Pareto optimal solution, but rather that *specific* Pareto optimal solutions which correspond to a designer's preferences can be found.

7 Final Remarks

Complete enumeration of the Pareto optimal frontier $\mathcal{P}$ is rarely possible due to the high dimensionality of the tradeoff surface. Such a level of enumeration is often not even necessary as a designer's preferences really only demand enumeration of $\mathcal{P}' \subset \mathcal{P}$. This means that the progression of the phenotypical Pareto front should hopefully be towards $\mathcal{P}'$ rather than to some arbitrary subset of $\mathcal{P}$.

So how do we achieve this progression of the phenotypical Pareto front? Recall that in each generation the alternatives are ranked according to the designer's preferences. It can be shown that this preference relationship preserves Pareto optimality. *Then, if we choose the most preferred alternatives for reproduction, we are letting a designer's preferences drive the search process of the EA*. Our example has shown that we can achieve this goal with the combination of EAs and ISMAUT.

References

1. Special editions on hardware/software codesign appearing in *IEEE Design & Test of Computers*, vol.10, no.3 & no. 4, 1993
2. J.G. D'Ambrosio and X. Hu, "Configuration-level hardware/software partition for real-time embedded systems," *Proceedings of the Third International Workshop on Hardware-Software Co-Design*, 34-41, 1994
3. C. Fonseca and P. Fleming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization", *Evolutionary Computation*, Vol. 3, No. 1, 1-17, 1995
4. G. Greenwood, X. Hu, and J. D'Ambrosio, "Fitness Functions for Multipleobjective Optimization Problems: Combining Preferences With Pareto Rankings", *FOGA4* (to appear)
5. C. White, A. Sage, and S. Dozono, "A Model of Multiattribute Decisionmaking and Tradeoff Weight Determination Under Uncertainty", *IEEE Trans. Syst., Man, Cybern.*", Vol SMC-14, 223-229, 1984
6. R.L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, John Wiley & Sons, NY, 1976
7. J. Horn and N. Nafpliotis, " Multiobjective Optimization using the Niched Pareto Genetic Algorithm", IlliGAL Report 93005, University of Illinois at Urbana-Champaign
8. D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Pub. Co., 1989

This article was processed using the LaTeX macro package with LLNCS style