In Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors, Galveston, Texas, September 2004

CHARMED: A Multi-objective Co-synthesis Framework for Multi-mode Embedded Systems

Vida Kianzad, Shuvra S. Bhattacharyya
ECE Department and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
{vida, ssb}@eng.umd.edu

Abstract

In this paper, we present a modular co-synthesis framework called CHARMED that solves the problem of hardware-software co-synthesis of periodic, multi-mode, distributed, embedded systems. In this framework we perform the synthesis under several constraints while optimizing for a set of objectives. We allow the designer to fully control the performance evaluation process, constraint parameters, and optimization goals. Once the synthesis is performed, we provide the designer a non-dominated set (Pareto front) of implementations on streamlined architectures that are in general heterogeneous and distributed. We also employ two different techniques, namely clustering and parallelization, to reduce the complexity of the solution space and expedite the search. The experimental results demonstrate the effectiveness of the CHARMED framework in computing efficient co-synthesis solutions within a reasonable amount of time.

1. Introduction

Modern embedded systems must increasingly accommodate dynamically changing operating environments, high computational requirements, flexibility (e.g., for the emergence of new standards and services), and tight time-to-market windows. Such trends and the ever-increasing design complexity of embedded systems have challenged designers to replace traditional ad-hoc approaches with more efficient techniques such as hardware-software (HW-SW) co-synthesis. HW-SW co-synthesis is the automated process of concurrently defining the hardware and software portions of an embedded system. Current and emerging embedded systems often involve multiple application subsystems. Such applications may either run concurrently (single-mode) or in a mutually exclusive fashion, depending on operational modes (multi-mode). A high-frequency (HF) radio communications system is one example of such a multi-mode application. It provides a single integrated system solution to current and future HF voice and data communications requirements for military airborne operations. The integrated multi-mode system provides data communications capability over HF with modems, video imaging systems, secure voice devices, and data encryption devices, while continuing to provide voice HF communications capability. Mobile telephony, audio decoding systems and video encoding systems are other examples of multi-mode applications.

Embedded systems have a variety of constraints and optimization goals such as memory, performance, price, area, power, runtime, number of physical links, etc. to be accommo-

dated. To satisfy such pressing design demands, researchers have shifted from optimal approaches such as MILP that could handle only a subset of such requirements for small task graphs [12] to deterministic heuristic [10][2][7][6] and probabilistic search heuristic [13][4][14] approaches. Many of these approaches only focus on single-mode systems or regard multi-mode systems as multiple single-mode systems and optimize them separately ([2] [9] [4] [14]). However, if a task is commonly used across different modes then these algorithms may not be able to find the most efficient solutions. Additionally, sharing of hardware resources among tasks that are not active simultaneously can greatly reduce system cost. [13], [10] and [7] consider multi-mode applications while optimizing only for a small set of costs concerning embedded systems. All of the deterministic heuristic methods, convert the multi-dimensional optimization problem to a single-dimensional optimization problem by forming a linear combination of the objectives (e.g. power, cost). The main disadvantage of this technique is that it cannot generate all Pareto-optimal solutions with non-convex trade-off surfaces, which typically underlie hardware-software co-synthesis scenarios. Furthermore, forming these linear combinations can be awkward because they involve computing weighted sums of values associated with heterogeneous metrics. There is no clear methodology for formulating the linear combinations, and their physical interpretation is ambiguous. [13], [4] and [14] employ evolutionary algorithms to overcome the two drawbacks mentioned above and target significantly larger problem instances. However, they optimize for a significantly smaller set of system costs compared to what we consider in this paper.

Our contribution in this work is as follows: We propose a modular co-synthesis framework that synthesizes multi-mode, multi-task embedded systems under a number of hard constraints; optimizes a comprehensive set of objectives; and provides a set of alternative trade-off points, generally known as Pareto-optimal solutions. Our framework allows the designer to independently configure each dimension of the design evaluation space as an optimization objective (to be minimized or maximized) or as a constraint (to be satisfied). Furthermore, we employ a pre-processing step of clustering and a parallelization technique to expedite the co-synthesis process.

The remainder of this paper is organized as follows: In Section 2 we present preliminary concepts and definitions. In Section 3 we describe the implementation details of our co-synthesis algorithm. In Section 4 we introduce the parallel version of our synthesis algorithm. We give the experimental results in Section 5 and conclude the paper with Section 6.

2. Preliminaries

2.1 System specification

We represent the embedded system applications that are to be mapped into parallel implementations in terms of the widely-used *task graph model*. Each system is characterized by multiple modes of functionality, where each mode can comprise of several task graphs. The directed acyclic graph $G_{m,i}(V, E)$ represents the ith task graph of mode m. We assume there are M different modes of operation and $|G_m(V, E)|$ task graphs in each mode. V is the set of task nodes, which are in one-to-one correspondence with the computational tasks in the application ($V = \{v_1, v_2, ..., v_{|V|}\}$) and E is the set of communication edges. There is also a period associated with each task graph ($\pi(m, i)$). For each mode we form a hyper

task graph $G_{Hm}(V, E)$ that consists of copies of high rate task graphs that are active for the hyper-period given in the following equation:

$$\pi_H(m) = \underset{i=0}{\overset{|G_m(V,E)|-1}{LCM}} \{\pi(m, i)\}. \tag{1}$$

The target architecture we consider, consists of different processing elements (PE) such as programmable microprocessors and ASICs and communication resources (CR) such as links and buses. PEs and CRs can be of various types. A final solution may be constituted of multiple instances of each PE or CR type. We represent the number of available PE and CR types by $|PE_{type}|$ and $|CR_{type}|$ respectively. Each PE can be characterized by the following attribute vector:

$$PE_{attr} = [\alpha, \kappa, \mu_d, \mu_i, \wp_{idle}]^T, \tag{2}$$

where α denotes the area of the processor, κ denotes the price of the processor, μ_d denotes the size of data memory, μ_i denotes the instruction memory size and $\wp_{idle}$ denotes the idle power consumption of the processor. $\mu_i = 0$ for ASICs. Each CR also has an attribute vector:

$$CR_{attr} = [\wp, \wp_{idle}, \vartheta]^T, \tag{3}$$

where $\wp$ denotes the average power consumption per each unit of data to be transferred, $\wp_{idle}$ denotes idle power consumption and ϑ denotes the worst case transmission rate or speed per each unit of data. Each task is also characterized by a set of attributes given in the following equation:

$$V_{attr} = [type, \mu_i, WCET, \wp_{avg}]^T, \tag{4}$$

where *type* denotes the type of the task or its functionality, and μ_i denotes the amount of instruction memory required to store the task. $WCET$ and $\wp_{avg}$ denote the worst-case execution time and average power consumption, respectively. These values depend on the PE the task is running on. Task-PE relations are provided in two matrices Γ_V and Φ_V of size $|V_{type}| \times |PE_{type}|$, where $|V_{type}|$ denotes the number of available task types. Each edge is also characterized by a single attribute given in the following equation:

$$E_{attr} = [\delta], \tag{5}$$

where δ denotes the size of data communicated in terms of the data unit used to characterize CRs. Once the edge is assigned to a CR, the worst case communication time and average power consumption can be computed using the corresponding CR_{attr}.

2.2 Problem statement

The co-synthesis problem considered in this paper is defined as the problem of optimally mapping the task-level specification of the embedded system onto a heterogeneous hardware software architecture. The optimization goal is to find a set of implementations that simultaneously minimize multiple objectives for which the corresponding objective vectors cannot be improved in any dimensions without degradation in another. An implementation is described by selection of a set of processing elements (PE) and communication resources (CR) (allocation), mapping of the application onto the selected architecture (assignment) and scheduling each task and data communication on the system resources. Each implementation, represented by solution vector $\vec{x}$, is evaluated with respect to a set of objectives

that are as follows: area ($\alpha(\vec{x})$), price ($\kappa(\vec{x})$), number of links ($\ell_n(\vec{x})$), memory requirement of each PE ($\overrightarrow{\mu}(\vec{x})$), power consumption ($\wp(\vec{x})$) and parallel-time or completion-time ($\tau_{par}(\vec{x})$).

Initially all of these goals are defined explicitly as separate optimization criteria, however, our framework allows the designer to formulate any of these goals as a constraint, e.g., that the size of the system must not exceed given dimensions. The algorithm always takes the upper bound (not to be violated if the optimization goal is formulated as a constraint) for each optimization goal as an input. However, the input vector $\Omega_0 = [\alpha_0, \kappa_0, \ell_{n_0}, \overrightarrow{\mu}_0, \wp_0, \tau_{par_0}]$ will determine the optimization/constraint setting. An entry of 0 means strictly optimization, an entry of 1 means formulate as a constraint and an entry of 2 means optimize while satisfying the constraint. An entry of -1 means to discard that goal for the current problem instance. For example $\Omega_0 = [\alpha_0, \kappa_0, \ell_{n_0}, \overrightarrow{\mu}_0, \wp_0, \tau_{par_0}] = [0, 0, 1, -1, 0, 2]$ configures CHARMED to find an implementation such that it minimizes the area, dollar cost, power consumption and parallel-time; meets the deadline; and does not exceed the given number of inter-processor links.

2.3 Evolutionary multi-objective optimization

The complex, combinatorial nature of the co-synthesis problem and the need for simultaneous optimization of several incommensurable and often competing objectives has led many researchers to experiment with evolutionary algorithms (EAs) as a solution method. EAs seem to be especially suited to multi-objective optimization as due to their inherent parallelism, they have the potential to capture multiple Pareto-optimal solutions in a single simulation run and may exploit similarities of solutions by recombination. Hence, we have adapted the Strength Pareto Evolutionary Algorithm (SPEA), an evolutionary algorithm for multi-objective optimization shown to have superiority over other existing multi-objective EAs [15]. SPEA uses a mixture of established and new techniques in order to find multiple Pareto-optimal solutions in parallel and it is characterized by

(a) Storing non-dominated solutions externally in a second, continuously updated population,

(b) Evaluating an individual's fitness dependent on the number of external non-dominated points that dominate it,

(c) Preserving population diversity using the Pareto dominance relationship, and

(d) Incorporating a grouping procedure in order to reduce the non-dominated set without destroying its characteristics.

The dominance relation used in the SPEA algorithm is defined as follows:
Definition 1: *Given two solutions a and b and a minimization problem, then a is said to* dominate *b iff*

$$\forall i \in \{1, 2, ..., n\} : f_i(a) \leq f_i(b) \wedge$$
$$\exists j \in \{1, 2, ..., n\} : f_j(a) < f_j(b). \quad (6)$$

All solutions that are not dominated by another solution are called *non-dominated*. The solutions that are non-dominated within the entire search space are denoted as *Pareto optimal* and constitute the *Pareto-optimal set*. SPEA however does not handle constraints and only concentrates on unconstrained optimization problems. Hence, we have modified

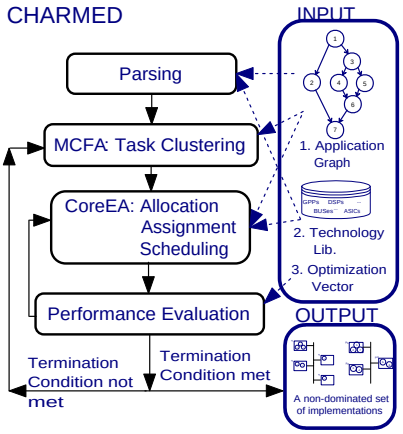

Figure 1. CHARMED framework.

this algorithm to solve constrained optimization problems by employing the constraint-dominance relation (in place of dominance relation) defined as follows [3]:

Definition 2: *Given two solutions a and b and a minimization problem, a is said to* constrained-dominate *b if*

1. Solution a is feasible and solution b is not, or

2. Solutions a and b are both infeasible, but solution a has a smaller overall constraint violation, or

3. Solutions a and b are feasible and solution a dominates solution b.

More details on the employed EA are given in the next section — this paper assumes familiarity with the fundamentals of evolutionary algorithms; for a tutorial introduction to these fundamentals the reader is referred to [1].

3. Our proposed algorithm

CHARMED or Co-synthesis of HARdware-software Multi-mode EmbeddeD Systems is a multi-objective evolutionary algorithm and is constituted of two main components of task clustering and task mapping. A high-level overview of CHARMED is depicted in Figure 1. CHARMED starts by taking input parameters that consist of: a system specification in terms of task graphs, PE and CR libraries, and an optimization configuration vector Ω_0. These inputs are then parsed and appropriate data structures such as attribute vectors and matrices are created. Next, the solution pool of the EA-based clustering algorithm (called multi-mode clusterization algorithm or MCFA) is initialized and task clustering is formed based on each solution. Solutions (clusters) are then evaluated using the coreEA, i.e. for each clustering, coreEA is initialized by a solution pool representing different mappings of the clustering onto different distributed heterogeneous systems. These mappings are evaluated for different system costs such as area, price, power consumption, etc. coreEA fine-tunes the solutions iteratively for a given number of generations and then returns the fittest solutions and fitness values. Once all the fitness values for all the clusters are determined, the clustering EA proceeds with the evolutionary process and updating the clustering population until the termination condition is met. The outline of this algorithm is presented in Figure 2.

INPUT: A set of task graphs $G_{m,i}(V, E)$, processing elements, communication resources library and an initial optimization vector Ω_0.
OUTPUT: A nondominated set (Å) of architectures which are in general heterogeneous and (distributed) together with task mappings onto these architectures.

Step 1 **Initialization (MCFA):** Generate an initial population $P_I(t)$ of binary string of size $\sum_{m=0}^{M-1} \sum_{i=0}^{|G_m(V,E)|-1} |E_{m,i}|$. Randomly initialize with 0 and 1s. Create the empty archive (external set) $\overline{P_I}(t) = \emptyset$ and $t = 0$.
Step 2 **Task Clustering:** Decode each binary string, form the associated clusters.
Step 3 **Fitness Assignment:** Perform mapping and scheduling for each individual (representing a set of clusters). Compute different system costs as indicated by Ω_0 for each individual. Calculate fitness values of individuals in $P_I(t)$ and $\overline{P_I}(t)$.
Step 4 **Environmental Selection:** Copy all non-dominated individuals in $P_I(t)$ and $\overline{P_I}(t)$ to $\overline{P_I}(t+1)$.
Step 5 **Termination:** If $t > T$ or other stopping criterion is met then set $A = \overline{P_I}(t+1)$ and stop.
Step 6 **Mating Selection:** Perform binary tournament selection on $\overline{P_I}(t+1)$ to fill the mating pool.
Step 7 **Variation:** Apply crossover and mutation operators to the mating pool and set $P_I(t+1)$ to the resulting population. Increment generation counter $t = t+1$ and go to Step 2.

Figure 2. Flow of CHARMED

3.1 MCFA: Multi-Mode Clusterization Function Algorithm

Clustering is often used as a front-end to multiprocessor system synthesis tools [2][6]. In this context, clustering refers to the grouping of tasks into subsets that execute on the same PE. The purpose of clustering is thus to reduce the complexity of the search space and constrain the remaining steps of synthesis, especially assignment and scheduling. The clustering algorithms employed in earlier co-synthesis research have been designed to form task clusters only to favor one of the optimization goals, e.g. to cluster tasks along the critical path or higher energy-level path. Such algorithms are relatively simple and fast but suffer from a serious drawback, namely that globally optimal or near-optimal clusterings with respect to all system costs may not be generated. Hence in this work we adapt the clusterization function algorithm (CFA), which we introduced in [8]. The effectiveness of CFA has been demonstrated for the minimum parallel-time scheduling problem, that is, the problem of scheduling a task graph to minimize parallel-time for a given set of allocated processors. However, the solution representation in CFA is not specific to parallel-time minimization, and is designed rather to concisely capture the complete design space of possible graph clusterings. Therefore, it is promising to apply this representation in other synthesis problems that can benefit from efficient clustering. One contribution of this paper is to apply CFA in the broader contexts of multi-mode task graphs, co-synthesis, and multi-objective optimization. In doing so, we demonstrate much more fully the power of the clustering representation that underlies CFA. Our multi-mode extension of CFA is called MCFA and its implementation details are as follows:

Solution Representation: Our representation of clustering exploits the view of a clustering as a subset of edges in the task graph. The coding of clusters for a single task graph in MCFA is composed of a n-size binary string, where $n = |E|$ and E is the set of all edges in the graph. There is a one to one relation between the graph edges and the bits, where each

bit represents the presence or absence of the edge in a cluster. Assuming M modes and $|G_m(V, E)|$ task graphs for each mode, the total size of the binary string that would capture the clustering for all task graphs across different modes is $n_{all} = \sum_{m=0}^{M-1} \sum_{i=0}^{|G_m(V,E)|-1} |E_{m,i}|$.

Initial Population: The initial population of MCFA consists of N_I (to be set experimentally) binary strings that represent different clusterings. Each binary array is initialized randomly with equal probability for a bit of 1 or 0.

Genetic Operators: We will discuss the crossover and mutation operators in Section 3.3. For the selection operator we use binary tournament with replacement [1]. Here, two individuals are selected randomly, and the best of the two individuals (according to their fitness values) is the winner and is used for reproduction. Both winner and loser are returned to the pool for the next selection operation of that generation.

Fitness Evaluation: Clusterings are evaluated using coreEA, which is described in detail in the next section(3.2).

The key characteristic of MCFA (or CFA) is the natural, binary representation for clusterings that, unlike previous approaches to clustering in co-synthesis, is not specialized for one specific optimization objective (e.g., critical path minimization), but rather, can be configured for different, possibly multi-dimensional, co-synthesis contexts based on how fitness evaluation is performed.

3.2 coreEA: mapping and scheduling

coreEA is the heart of the CHARMED framework and its goal is to find a set of implementations for each member of the MCFA solution pool (or each clustering). It runs once for each member. coreEA starts by creating a PE and a CR allocation string for the given solution (or clustering). The lengths of these string are equal to the number of PE and CR types, respectively. We initialize them such that every cluster and every inter-cluster communication (ICC) edge has at least one instance of PE or CR that it can execute on. Each entry of the string represents the number of available instances of the associated PE type. Based on these allocation strings and the numbers of clusters and ICC edges we then initialize the population of coreEA. Further design details of this EA are as follows:

Solution Representation: Solutions in coreEA represent the assignment of clusters to PEs and ICCs to CRs. These assignments are encoded in two different binary matrices, hence each solution is represented with a pair of matrices. Using the allocation arrays we compute the total number of available PEs ($|PE_{avail}|$) and CRs ($|CR_{avail}|$)(including different instances of a same type). The assignment matrix for clusters is of size $|PE_{avail}| \times |clusters|$ and for ICCs is of size $|CR_{avail}| \times |ICC|$. $|clusters|$ denotes the number of clusters in the solution and $|ICC|$ denotes number of ICC edges. Each column of the cluster (ICC) assignment matrix corresponds to a cluster (ICC) that has to be assigned to a PE (CR). Each row of this matrix corresponds to an available PE (CR). Each column of the PE (CR) assignment matrix possesses exactly one non-zero row that determines the PE (CR) that the cluster (ICC) is assigned to.

Initial Population: The initial population of coreEA consists of N_{II} (to be set experimentally) pair of assignment matrices (one for clusters and one for ICCs). For each solution, exactly one 1 is randomly assigned to each column of each assignment matrix.

Genetic Operators: The crossover and mutation operators of coreEA are discussed in Section 3.3. For the selection operator, we use a technique similar to the one described in Section 3.1.

Fitness Evaluation: Each member of the solution pool of coreEA that is a pair of assignment matrices representing cluster-to-PE and ICC-to-CR mapping, is employed to construct a schedule for each clustering. Once the ordering and assignment of each task is known we calculate other objectives and constraints given in Ω_0. Since the modes are mutually exclusive, it is possible to employ scheduling methods that are used for single mode systems. Scheduling a task graph for a given allocation and for a single mode is a well-known problem which has been extensively studied and for which good heuristics are available. Hence, we employ a deterministic method that is based on the classic list scheduling heuristic.

Once clusters are mapped and scheduled to the target architecture, we compute different system costs across different modes and check for constraint violations of individual modes. Next, using the constrained-dominance relation we calculate the fitness value for each individual [15]. In certain problems, the non-dominated set can be extremely large and maintaining the whole set when its size exceeds reasonable bounds is not advantageous. Too many non-dominated individuals might also reduce selection pressure and slow down the search. Thus, pruning the external set while maintaining its characteristics, before proceeding to the next generation is necessary [15]. The pruning process is based on computing the phenotypic distance of the objective values. Since the magnitude of each objective criterion is quite different, we normalize the distance with respect to each objective function. More formally, for a given objective value $f_1(\vec{x})$, the distance between two solutions $\vec{x_i}$ and $\vec{x_j}$ with respect to f_1 is normalized as follows:

$$\frac{(f_1(\vec{x_i}) - f_1(\vec{x_j}))^2}{(\max(f_1(t)) - \min(f_1(t)))^2}. \tag{7}$$

For area, price, power consumption and parallel-time, the $\max(f_1(t))$ and $\min(f_1(t))$ denote the worst and best case values of the corresponding system cost among all the members in generation t, respectively. The maximum number of links or $\max(\ell_n(t))$ is computed from the maximum possible number of physical inter-processor links for the given graph set $G_{m,i}(V, E)$ and the processor configuration, i.e.

$$\max(\ell_n(t)) = \max(|PE_{used}| \times (|PE_{used}| - 1), |E_{m,i}|), \tag{8}$$

where $|E_{m,i}|$ is the number of edges in the graph. The equation implies that the maximum number of inter-processor links that make sense for a network is not necessarily that corresponding to a fully connected-network; depending on the number of edges in the graph, this number can be smaller. Minimum number of links is equal to $|PE_{used}| - 1$). The maximum value for the memory requirement is equal to the size of data and instruction memory. The minimum value is computed using the smallest amount of memory used among the solutions in generation t. If optimization goals are formulated as constraints, the maximum values are replaced by the given constraint values.

The flow of coreEA is given in Figure 3. coreEA can be employed without the preprocessing step of clustering by simply substituting groups of tasks (clusters) by individual tasks.

3.3 Multi-mode genetic operators

The design of genetic operators for manipulating multi-mode systems requires special considerations that take into account both the global aspects of the systems, while re-

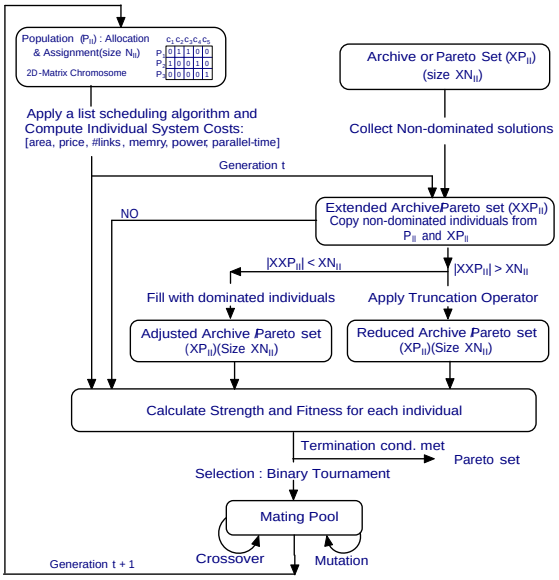

Figure 3. Flow of coreEA Algorithm.

specting the local properties associated with individual modes. This point is elaborated as follows:

- For a given single-mode application, all system costs (area, power, price, parallel time, etc.) are the direct result of the final assignment and scheduling of task graphs of that mode. However, for multi-mode applications, some system costs such as area and price are highly dependent on the influence of individual modes and are a combination of the areas and prices of individual modes considered in isolation. Hence to minimize these costs, each individual mode should be minimized as well. However, there are other system costs such as parallel-time and power that mostly depend only on individual modes. For example, if a set of clusterings of mode i does not meet the required deadline it can have several reasons as follows : i) improper clustering of task graphs of that mode, ii) inefficient cluster (ICCs) to PE (CR) assignment or iii) inefficient selection of PEs and CRs. The first two problems can only be fixed by intra-mode changes i.e. exchanging tasks among clusters or changing the cluster assignments (these can be achieved by applying evolutionary operators i.e. mutation and crossover to the part of the solution string representing this mode). The last problem can be fixed by changing the type or number of given PEs or CRs among different modes i.e. applying evolutionary operators across modes. On the other hand, for the same system, if the price is not minimized effectively, it is largely due to inefficient assignments across all modes and improvement can only be made by swapping bits across the modes.
- If a candidate solution has a high fitness, then it is reasonable to assume that it is made up of smaller parts that in turn conferred a certain degree of fitness on the overall solution. Therefore, by selecting highly fit solutions for the purpose of crossover to create the next generation, we are giving more chance to those solutions that contain good *building blocks*.

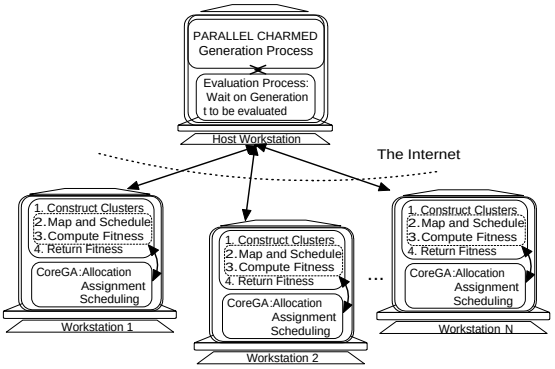

Figure 4. Parallel CHARMED framework.

Motivated by the above observations, we apply evolutionary operators once to each mode to preserve good local building blocks of that mode and across modes to preserve global building blocks.

The genetic operators for reproduction (mutation and crossover) that we use are the traditional two-point crossover and the typical mutator for a binary string chromosome where we flip the bits in the string with a given probability. However, we apply both operators once locally within the modes and again globally across different modes as opposed to some techniques that apply these operators once and only across all modes [13].

4. Parallel CHARMED

Fitness evaluation is usually very time-consuming. Fortunately, however, there is a large amount of parallelism in the overall fitness evaluation process. Therefore, we have developed a parallel version of the CHARMED framework that provides an efficient and highly scalable means for leveraging additional computational power to reduce synthesis time.

We employ the asynchronous master-slave parallelization model [5]. This method, also known as distributed fitness evaluation, uses a single population and the evaluation of the individuals and/or the application of evolutionary operators are performed in parallel. The selection and mating is done globally, hence each individual may compete and mate with any other. The evaluation process normally requires only knowledge of the individual being evaluated (not the whole population), so there is no need to communicate during this phase, and this greatly reduces overhead. The asynchronous master-slave algorithm does not stop to wait for any slow processors and/or until all the evaluations are returned and selection waits only until a fraction of the population has been processed. A high level description of parallel CHARMED is given in Figure 4.

5. Experimental results

We evaluated our proposed co-synthesis framework on several benchmarks to demonstrate its capability to produce high quality solutions in terms of different optimization goals. All algorithms were implemented using C++. The benchmarks consist of 10 random task graphs TG1-TG10 ($10 \sim 100$ nodes) that were generated using TGFF [4]. The

population size of MCFA and coreEA are 100 and 50. MCFA runs for 1000 generations and coreEA runs for 500 generations. In case of parallel CHARMED, the fraction of the population that the algorithm waits on, is 80% of the original population, in other words once it receives the results from 80 members (running remotely) it proceeds to the next generation.

To study the effectiveness of our clustering approach we first run CHARMED without the MCFA pre-processing step (coreEA only) and apply coreEA directly to tasks (instead of clusters of tasks). Next, we run CHARMED with the clustering step, i.e. MCFA + coreEA. The goal is to optimize price, power consumption and parallel-time. The results for a subset of graphs are given in Table 1. It can be seen from the table that CHARMED finds better quality solutions when it is employed with the pre-processing step of clustering. Additionally, as the size of the task graph increases, clustering step becomes more effective. Solutions found by CHARMED using clustering (MCFA + coreEA) dominate 50% to 100% of the solutions found using CHARMED without clustering (or coreEA). In order to study

Table 1. Effect of clustering: CHARMED without clustering step (coreEA only) vs. CHARMED with clustering step (MCFA + coreEA)

Task Graph	$\|V\|/\|E\|$	coreEA			MCFA + coreEA		
		Price	Power Consumption	Parallel-Time	Price	Power Consumption	Parallel-Time
TG2	18/25	*122*	*92.03*	*107*	122	87.9	107
		196	*102.6*	*101*	195	133.6	99
		226	121.5	92	196	98.7	100
		330	142.2	91	196	108.2	93
TG3	28/47	*226*	*199.3*	*195*	226	169.2	162
		226	*243.4*	*153*	226	173.1	151
		299	*241*	*153*	226	210.5	150
		330	*282.9*	*136*	330	270.4	135
TG5	48/85	*226*	*413.4*	*321*	122	250.8	274
		330	*374.6*	*241*	226	280.6	262
		330	419.4	230	226	353.1	241
		330	460.1	223	330	411.4	240
TG7	68/119	361	563.2	300	361	606.9	270
		361	*638.1*	*276*	361	631.3	261
		465	*805.5*	*267*	361	641.2	251
		495	*619.6*	*277*	361	691.4	245
TG9	88/161	*496*	*895.4*	*333*	465	927.3	302
		496	*1031.3*	*313*	465	938.2	288
		600	*881.5*	*358*	465	969.2	283
		600	*929.7*	*324*	496	873.4	299
TG10	98/181	630	1049.6	353	496	1054.6	343
		630	*1075.586*	*342*	496	1070.6	332
		630	*1092.6*	*330*	496	1107.9	324
		630	*1140.2*	*328*	496	1168.3	322

the effect of integrating multiple system modes and optimizing them jointly vs. optimizing modes separately, we created multi-mode applications by combining task graphs from the TG set (interpreted each task graph as a separate mode). We run CHARMED once for each mode and once for the combinations of modes. The optimization goals for these tests

are area and parallel-time. Results are given in Table 2. The M_is in the table represent individual modes, corresponding to task graphs listed in the first column respectively. "Total" represents the estimated system area. It can be seen from the table that optimizing all system modes simultaneously can significantly improve the results.

Table 2. Effect of optimizing modes separately vs. optimizing all modes together.

Task Graph	Separate Modes				Integrated Modes			
	Area				Area			
	M_1	M_2	M_3	Total	M_1	M_2	M_3	Total
TG1&TG2&TG3	0.509	0.809	0.509	1.118	0.509	0.818	0.509	0.818
TG3&TG5&TG6	0.509	0.818	0.780	1.39	0.509	0.818	1.09	1.09
TG2&TG6&TG7	0.809	0.780	1.08	1.39	0.818	1.09	1.09	1.09

We also compared the performances of parallel CHARMED and CHARMED. The results of our comparison clearly show the effectiveness of the parallelization techniques employed in parallel CHARMED. If we run both algorithms for the same amount of time, the solution quality of parallel CHARM-ED is significantly better than CHARMED's solution quality. Results of running CHARMED and parallel CHARMED on a subset of TG set to optimize for price and power, are given in Table 3. If we run both algorithms for the same number of generations, parallel CHARMED achieves a speedup between 4 to 8. This number however is a function of number of available remote hosts, network traffic and problem size. For these tests, we used a network of 24 workstations of Sun Ultra 5/10 UPA/PCI (UltraSPARC-IIi 440MHz) systems.

Table 3. Performance Comparison of CHARMED vs. parallel CHARMED

Task Graph	$\|V\|/\|E\|$	CHARMED		Parallel CHARMED	
		Price	Power Consumption	Price	Power Constumption
TG1	8/7	*61*	*37.7*	61	36.9
TG2	18/25	*92*	*103.9*	92	87.9
TG5	48/85	*239* *269*	*607.2* *522.4*	239	447.3
TG6	58/97	*269*	*714*	239 269	721 675.9
TG9	88/161	*509*	*1187.7*	496	1041.1
TG10	98/181	*540* *556*	*1421.4* *1274.3*	539	1299.1

6. Conclusion

In this paper, we have presented a modular framework called CHARMED for hardware-software co-synthesis of multi-mode embedded systems. At the heart of CHARMED is an efficient new evolutionary algorithm, called coreEA, for allocation, assignment, and scheduling of clustered, multi-mode task graphs. CHARMED also includes a novel integration of several synergistic techniques for multi-objective co-synthesis, including a hierarchical evolutionary algorithm architecture; the CFA-based binary representation for clustering [8]; the SPEA method for multi-objective evolutionary algorithms [15]; the constraint dominance

concepts of Deb et al. [3]; and optionally, the asynchronous master-slave parallelization model [5]. Our framework allows the designer to flexibly control the performance evaluation process by configuring design evaluation metrics as optimization goals or constraints, as desired. This flexibility enables the designer to narrow down the search to special regions of interest. Our parallelization technique, parallel CHARMED, is shown to provide an efficient means for applying additional computational resources to the co-synthesis process. This enables application of CHARMED to large instances of co-synthesis problems.

For future work, we are planning to add a refinement step that uses the possibly suboptimal solutions generated by the allocation/assignment phase as the starting point for its local search. We will also be looking into another method of parallelizing EAs that searches different subspaces of the search space in parallel and is less likely to get trapped in low-quality subspaces.

References

[1] T. Back, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: comments on the history and current state," IEEE Transactions on Evolutionary Computation, vol. 1, pp. 3-17, 1997.

[2] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," IEEE Trans. on VLSI Systems, vol. 7, pp. 92104, Mar. 1999.

[3] K. Deb, A. Pratap, and T. Meyarivan, "Constrained test problems for multi-objective evolutionary optimization," First International Conference on Evolutionary Multi-Criterion Optimization, pp 284–298. Springer Verlag, 2001.

[4] R. Dick, D. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," In Proc. Int. Workshop Hardware/Software Codesign, P.97-101, March 1998.

[5] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[6] J. Hou and W. Wolf, "Process partitioning for distributed embedded systems," in Proc. of Int. Workshop on Hardware/Software Co-Design, pp. 7076, Mar, 1996.

[7] A. Kalavade and P. A. Subrahmanyam, Hardware/Software Partitiong for Multifunction Systems. IEEE Trans. on Computer-Aided Design, 17(9):819836, Sep 1998.

[8] V. Kianzad and S. S. Bhattacharyya, "Multiprocessor clustering for embedded systems," Proc. of the European Conference on Parallel Computing, 697-701, Manchester, United Kingdom, August 2001.

[9] H. Oh and S. Ha, "Hardware-software co-synthesis technique based on heterogeneous multiprocessor scheduling," in Proc. of Int. Workshop on Hardware/ Software Co-Design, pp. 1831878, May 1999.

[10] H. Oh and S. Ha, "Hardware-software co-synthesis of multi-mode multi-task embedded systems with real-time constraints," in Proc. of the Int. symposium on Hardware/software codesign, pp. 133-138, May 2002.

[11] P. Marwedel and G. Goossens, Code Generation for Embedded Processors. Kluwer Academic Publishers, 1995.

[12] S. Prakash and A. Parker, "SOS: Synthesis of application-specific heterogeneous multiprocessor systems," J. of Parallel & Distributed Computing, vol. 16, pp. 338351, Dec. 1992.

[13] M. Schmitz, B. Al-Hashimi, and P. Eles, "A Co-Design Methodology for Energy-Efficient Multi-Mode Embedded Systems with Consideration of Mode Execution Probabilities," Proc. Design, Automation and Test in Europe, 2003.

[14] J. Teich, T. Blickle, and L. Thiele, "An evolutionary approach to system-level synthesis," in Proc. of Int. Workshop on Hardware/Software Co-Design, pp. 167 171, Mar. 1997.

[15] E. Zitzler, M. Laumanns, L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization," Evolutionary Methods for Design, Optimisation, and Control, pp. 95-100, 2002.