Mixed Heuristic and Mathematical Programming Using Reference Points for Dynamic Data Types Optimization in Multimedia Embedded Systems

José L. Risco-Martín*, J. Ignacio Hidalgo*, David Atienza†, Juan Lanchares*, Oscar Garnica*
*Dept. of Computer Architecture and Automation, Complutense University of Madrid
C/ Prof. José García Santesmases, s/n, 28040 Madrid, Spain
{jlrisco, hidalgo, julandan, ogarnica}@dacya.ucm.es
†Embedded Systems Laboratory (ESL), Ecole Polytechnique Fédérale de Lausanne (EPFL)
EPFL-STI-IEL-ESL, Station 11, 1015 Lausanne, Switzerland
david.atienza@epfl.ch

ABSTRACT

New multimedia embedded applications are becoming increasingly dynamic. Thus, they cannot only rely on static data allocation, and must employ *Dynamically-allocated Data Types (DDTs)* to store their data and efficiently use the limited physical resources of embedded devices. However, the optimization of the DDTs for each target embedded system is a very time-consuming process due to the large design space of possible DDTs implementations and selection for the memory hierarchy of each specific embedded device. Thus, new suitable exploration methods for embedded design metrics (memory accesses, usage and power consumption) need to be developed. In this paper we analyze the benefits of two different exploration techniques for DDTs optimization: *Multi-Objective Particle Swarm Optimization (MOPSO)* and a *Mixed Integer Linear Program (MILP)*. Furthermore, we propose a novel MOPSO exploration method, OMOPSO*, which uses MILP solutions, as reference points, to guide a MOPSO exploration and reach solutions closer to the real Pareto front of solutions. Our experiments with two real-life embedded applications show that our algorithm achieves 40% better coverage and set of solutions than state-of-the-art optimization methods for DDTs (MOGAs and other MOPSOs).

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*Heuristic methods*

General Terms

Design, Performance

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

Keywords

Particle Swarm Optimization, Multi-Objective Optimization, Evolutionary Computation, Mathematical Programming, Embedded Systems Design

1. INTRODUCTION AND RELATED WORK

Latest multimedia embedded devices are enhancing its capabilities and currently are able to run applications reserved to powerful desktop computers few years ago (e.g., 3D games or video players). As a result, one of the most important problems designers face nowadays is the integration of a large amount of applications coming from the general-purpose domain in a compact and highly-constrained device. One major task of this porting process is the optimization of the dynamic memory subsystem. Thus, the designer must choose among a number of possible dynamically-allocated data structures or *Dynamic Data Types (DDTs)* implementations (dynamic arrays, linked lists, etc.) the best one in each case, according to the specific restrictions of the target device and typical embedded design metrics, such as memory accesses, memory usage and energy consumption [3].

So far extensive work has been performed in the field of embedded memory subsystem optimization. [14] includes a thorough survey of static data and memory optimization techniques for embedded systems. Also, in [4], authors have explored a coordinated data and computation reordering for array-based data structures in multimedia applications. They use a linear-time algorithm reducing the memory subsystems needs by 50%. Nevertheless, they are not suitable for exploration of complex DDTs employed in modern multimedia applications.

According to the characteristics of certain parts of multimedia applications, several transformations for DDTs and design methodologies [4] have been proposed for static data profiling and optimization considering static memory access patterns to physical memories. The use of *Multi-Objective Evolutionary Algorithms (MOEAs)* has been applied to solve linear and non-linear problems by exploring all regions of the design space in parallel. Thus, it is possible to perform optimizations in non-convex regular functions, and also to select the order of algorithmic transformations in concrete types of source codes [14]. However, such techniques are not applicable in DDT implementations, due to the unpredictable nature at compile-time of the stored data.

Regarding dynamic embedded software, suitable access methods, power-aware DDT transformations and pruning strategies based on heuristics have started to be proposed for multimedia systems [14]. However, these approaches require the development of efficient pruning function costs and fully manual optimizations [6]; otherwise they are not able to capture the evaluation of inter-dependencies of multiple DDTs implementations operating together, as our proposed method using evolutionary computation achieves. Also, in [3, 9] it has already been outline the potential use of MOEAs for dynamic memory optimizations. However, regarding DDTs optimization the true Pareto front is unknown, and in many cases the system designer is not able to define a set of preferences in the resultant set of non-dominated solutions.

In this paper we address this problem by applying *Mixed Integer Linear Programming (MILP)* and *Multi-Objective Particle Swarm Optimization (MOPSO)*. First, we find the optimal points for each design optimization objective in and individual fashion using MILP. Second, we use these points to guide the MOPSO algorithm towards solutions closer to the real Pareto front. Thus, as the experimental results of this paper indicate, our methodology is able to obtain better approximation fronts than state-of-the-art approaches for DDTs optimization.

The remainder of the paper is organized as follows. In Section 2, we briefly describe the background material for our DDTs optimization study, including the formal definitions of multi-objective optimization, MILP methods, particle swarms and the reference point method. In Section 3, we explain the DDTs optimization problem and present our design framework. In Section 4, we describe the optimization model for both MILP and MOPSO. In Section 5, we present our DDTs optimization algorithm. Then, the real applications used to evaluate our algorithm are described in Section 6. Finally, in Section 7 we summarize the main conclusions of this work.

2. BACKGROUND

2.1 Multi-objective optimization

Multi-objective optimization aims at simultaneously optimizing several objectives sometimes contradictory. For such kind of problems, there does not exist a single optimal solution, and some trade-offs need to be considered. Without any loss of generality, we can assume the following m-objective minimization problem:

$$\begin{aligned} &\textit{Minimize} \quad \vec{z} = (f_1(\vec{x}), f_2(\vec{x}), \dots f_m(\vec{x})) \\ &\textit{subject to} \qquad\quad \vec{x} \in X \end{aligned} \tag{1}$$

where $\vec{z}$ is the objective vector with m objectives to be minimized, $\vec{x}$ is the decision vector, and X is the feasible region in the decision space. A solution $\vec{x} \in X$ is said to dominate another solution $\vec{y} \in X$ (denoted as $\vec{x} \prec \vec{y}$) iff the following two conditions are satisfied.

$$\begin{aligned} &\forall i \in \{1, 2, \dots, m\}, f_i(\vec{x}) \leq f_i(\vec{y}) \\ &\exists i \in \{1, 2, \dots, m\}, f_i(\vec{x}) < f_i(\vec{y}) \end{aligned} \tag{2}$$

A decision vector $\vec{x} \in X$ is non-dominated with respect to $S \subseteq X$ if another $\vec{x}' \in S$ such that $\vec{x}' \prec \vec{x}$ does not exist. A solution $\vec{x}^* \in X$ is called Pareto-optimal if it is non-dominated with respect to X. An objective vector is called Pareto-optimal if the corresponding decision vector is Pareto-optimal.

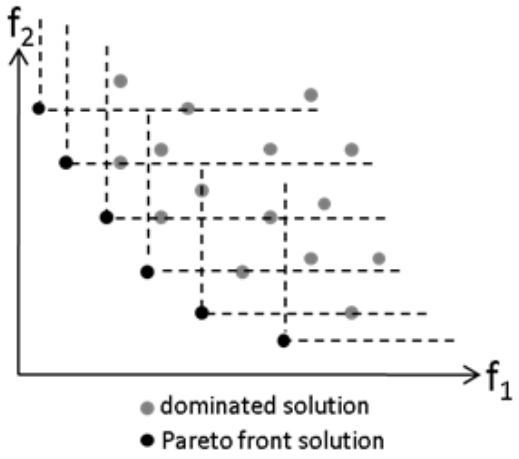

Figure 1: Non-dominated solutions of a set of solutions in a two objective space

The non-dominated set of the entire feasible search space X is the *Pareto-Optimal Set (POS)*. The image of the POS in the objective space is the P*areto-Optimal Front (POF)* of the multi-objective problem at hand. Figure 1 shows a particular case of the POF in the presence of two objective functions. A multi-objective optimization problem is solved, when its complete POS is found.

2.2 Mixed Integer Linear Program

A *mixed-integer linear program (MILP)* is a mathematical program with linear constraints in which a specified subset of the variables are required to take on integer values. Although MILPs are difficult to solve in general, the past ten years has seen a dramatic increase in the quantity and quality of software -both commercial and noncommercial- designed to solve MILPs.

To formally specify a MILP, let a polyhedron

$$\mathcal{P} = \{x \in \Re^n | Ax = b, x \geq 0\} \tag{3}$$

be represented in standard form by a constraint matrix $A \in \mathbb{Q}^{m \times n}$ and a right-hand side vector $b \in \mathbb{Q}^m$. Without loss of generality, we assume that the variables indexed 1 through $p \leq n$ are the integer-constrained variables (the integer variables), so that the feasible region of the MILP is $\mathcal{P}^I = \mathcal{P} \cap \mathbb{Z}^p \times \mathbb{R}^{n-p}$. In contrast, the variables indexed $p+1$ through n are called the continuous variables. A subset of the integer variables, called binary variables, may additionally be constrained to take on only values in the set $0, 1$. We denote here the set of indices of binary variables by $B \subseteq 1, 2, \dots, p$. The mixed-integer linear programming problem is then to compute the optimal value

$$z_{IP} = \min c^T x \tag{4}$$

where $x \in \mathcal{P}^I$, and $c \in \mathbb{Q}^n$ is a vector that defines the objective function. The case in which all variables are continuous ($p = 0$) is called a linear program (LP). Associated with each MILP is an LP called the LP *relaxation*, obtained by relaxing the integer restrictions on the variables. For an in-depth treatment of the theory of integer programming, we direct the reader to [20].

2.3 Particle swarm optimization

Particle Swarm Optimization (PSO) is a heuristic search technique that simulates the movements of a flock of birds that aim to find food. The relative simplicity of PSO and the fact that is a population-based technique have made it a natural candidate to be extended for multi-objective optimization [16].

In PSO, particles are "flown" through a hyper-dimensional search space. Changes to the position of particles within the search space are based on social-psychological tendencies of individuals to emulate the success of other individuals.

Hence, the position of each particle is changed according to its own experience and its neighbors. Let $\vec{x}_i(t)$ denote the position of particle p_i, at time step t. The current position of p_i is then changed by adding a velocity vector $\vec{v}_i(t)$ to the previous position, i.e.:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \tag{5}$$

The velocity vector reflects the socially exchanged information and, commonly, is defined in the following way:

$$\begin{aligned} \vec{v}_i(t) &= W\vec{v}_i(t-1) \\ &\quad +C_1\vec{r}_{i1}\left(\vec{x}_{ipbest} - \vec{x}_i(t-1)\right) \\ &\quad +C_2\vec{r}_{i2}\left(\vec{x}_{ileader} - \vec{x}_i(t-1)\right) \end{aligned} \tag{6}$$

where:

- W is the inertia weight. It controls the impact of the previous history of velocities.
- C_1 and C_2 are the learning factors. C_1 is the cognitive learning factor and represents the attraction that a particle has toward its own success. C_2 is the social learning factor and represents the attraction that a particle has toward the success of its neighbors.
- $\vec{r}_{i1}, \vec{r}_{i2}$ are random vectors, each component in the range $[0, 1]$.
- $\vec{x}_{ipbest}$ is the personal best position of p_i, namely, the position of the particle that has provided the greatest success.
- $\vec{x}_{ileader}$ is the position of the particle that is used to guide p_i towards better regions of the search space.

Particles tend to be influenced by the success of any other element they are connected to. These neighbors are not necessary particles close to each other in the decision variable space, but instead are particles that are close to each other based on a neighborhood topology, which defines the social structure of the swarm [16].

2.4 Reference point method

User-preference methods, as described in the multi-criteria decision making literature, come in two forms. 1) A *priori* methods, where a system designer gives preferences first then the algorithm finds solutions considering those preferences; 2) A *posteriori* methods, where after an algorithm provides all possible solutions the system designer selects the interesting ones [13]. The reference point method is an a *priori* approach, where first reference points are provided in the objective-space and the search algorithm will concentrate around those points to find solutions. The advantage is that most computing effort can be spent on the preferred areas, instead of the entire search-space. This is especially important as the number of objectives increases.

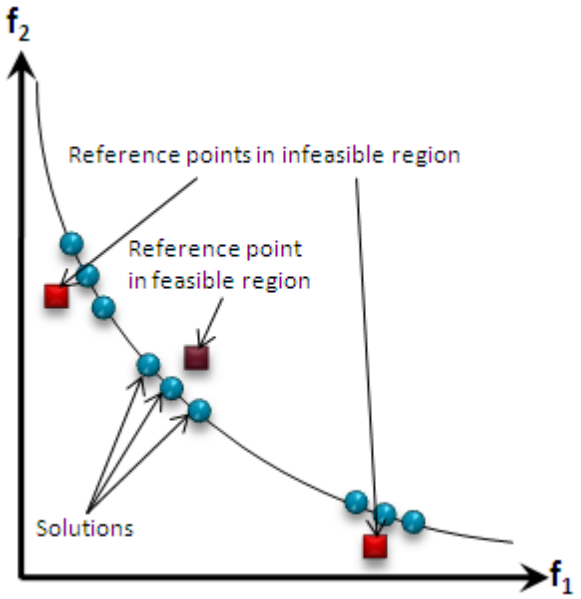

Figure 2: Reference point method

The classical reference point method was first described by Wierzbicki [13]. A reference point $\vec{z}$ for a multi-objective problem is a point consisting of aspiration values for each objective. This reference point is used to construct a single objective function (equation 7), which is to be minimized over the entire search-space S, where $\vec{x} \in S$.

$$minimize \max_{i=1\ldots m} \{w_i \left(f_i\left(\vec{x}\right) - \vec{z}_i\right)\} \tag{7}$$

where $\vec{z}_i$ is the i^{th} component of the reference point and w_i is a weight associated with the i^{th} objective. The system designer can assign a value for this weight, which represents any bias towards that objective.

The system designer is presented with the objective-space where preferred regions can be indicated to the algorithm with the use of reference points. Figure 2 illustrates the classical reference point method in a two-objective space. In our research we use the reference point methodology described in [19], where authors define a reference point as an array of aspiration values. The number of elements in the array corresponds to the number of objectives for a given problem. In the general sense such a reference point would indicate a potential solution point consisting of values for each objective.

A reference point could be in any region either feasible or infeasible, because the system designer might not know beforehand where the real (or true) Pareto front is for a given problem. Thus, the algorithm will attempt to find a set of solution points on the Pareto front that is the closest to the given reference point. An advantage of using an evolutionary algorithm is that, unlike a classical optimization approach where a single solution is found, a set of solutions can be found in a single run near the reference point.

```
vector<T1>* c1 = new vector<T1>();
// …
list<T2>* c2 = new list<T2>();
list<T2>::iterator itr = c2->begin();
for(; itr!=c2->end(); ++itr)
  cout << *itr;
// …
```

Initial application code

```
SLL<T1>* c1 = new SLL<T1>();
// …
DLLAR<T2>* c2 = new DLLAR<T2>();
DLLAR<T2>::iterator itr = c2->begin();
for(; itr!=c2->end(); ++itr)
  cout << *itr;
// …
```

Optimized application code

Figure 3: Code before and after the exploration of Dynamic Data Types

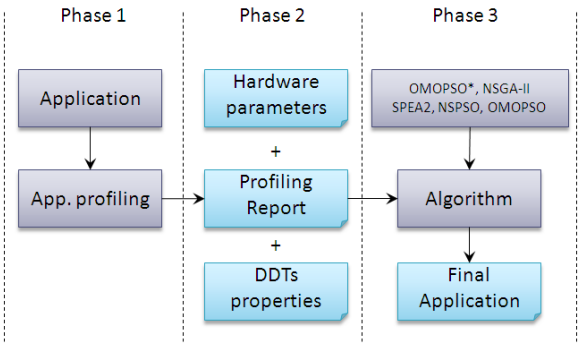

Figure 4: DDTs optimization flow.

3. THE DYNAMIC DATA TYPES EXPLORATION PROBLEM

A DDT is a software abstraction by means of which we can manipulate and access data. The implementation of a DDT has two main components. First, it has storage aspects that determine how data memory is allocated and freed at runtime and how this memory is tracked. Second, it includes an access component, which can refer to two different basic access patterns: sequential or iterator-based and random access.

Table 1: DDT library

DDT	Description
AR	Array
AR(P)	Array of pointers
SLL	Singly-linked list
DLL	Doubly-linked list
SLL(O)	Singly-linked list with roving pointer
DLL(O)	Doubly-linked list with roving pointer
SLL(AR)	Singly-linked list of arrays
DLL(AR)	Doubly-linked list of arrays
SLL(ARO)	Singly-linked list of arrays and roving pointer
DLL(ARO)	Doubly-linked list of arrays and roving pointer

In our case we have classified the DDT implementations in basic DDT and multi-layer implementations relevant for embedded multimedia applications. Table 1 contains the DDTs implemented.

Figure 3 shows an example of a DDTs exploration. The initial code contains two containers, *c1* and *c2*, instantiated as vector and list, respectively. After the exploration process, one candidate solution gives *c1* to be instantiated as *Single Linked List (SLL)* and *c2* as *Double Linked List of Arrays (DLLAR)*. Such instantiation policy tries to minimize memory accesses, memory usage and energy consumption of the final application. It is important to stress that it is unmanageable for the designer to get a totally complete exploration of all the possible DDT implementation combinations using the traditional way for real-life complex applications. For example, in one of the applications analyzed, we optimized memory accesses, memory usage and power consumption for 3128 containers. Whereas in previous works 41 containers were optimized in several days, our design framework based on evolutionary computation is able to optimize embedded applications including 3128 containers in just a few hours.

In general terms, the application to optimize contains a set of n containers $\mathbf{C}$ which are candidates to be instantiated as a certain DDT from the set of possible implementation of DDTs library $\mathbf{D}$ presented in [3]. Thus, the goal of our optimization flow is to obtain a set of pairs (container, DDT) or $(\vec{c},\vec{d}), c_i \in \mathbf{C}, d_i \in \mathbf{D}, 1 \leq i \leq n$, such that minimizes three objectives: memory accesses, memory usage and energy consumption. Additional constraints, such as minimum and maximum values for all three objectives may be defined.

The proposed optimization framework uses three different phases to perform the automatic exploration of DDTs. Figure 4 shows the different phases required to perform the overall DDTs optimization. In the first phase, we generate an initial profiling of the iterator-based access methods to the different DDTs used in the application. In the second phase, using this detailed report of the accesses, we extract all the information needed by the optimization phase. Finally, an exploration of the design space of DDTs implementation is performed using the algorithm selected. When the optimization process ends, it gives the DDT instantiation policy, i.e., which container should be instantiated by which DDT. We also obtain the gain on memory accesses, memory usage and energy consumption. In addition to MILP, we have solved the problem using four relevant *Multi-Objective Evolutionary Algorithms (MOEAs)*, i.e. NSGA-II [7], SPEA2 [22], OMOPSO [15] and NSPSO [11].

We apply these algorithms to two multimedia embedded applications. The first benchmark is VDrift, which is a driving simulation game. The game includes 19 tracks, 28 cars, artificial-intelligence players, networked multiplayer mode, etc. [2]. We logged 49 containers in its source code. The second benchmark is a 3D Physics Engine for elastic and deformable bodies [10], which is a 3D engine that displays the interaction of non-rigid bodies. It includes 3128 dynamic containers in its source code for which we choose the optimal DDT implementation.

4. OPTIMIZATION MODEL

As it has been stated above, the goal of our optimization process is to obtain a set of pairs (container, DDT), such that minimizes three objectives: memory accesses, memory usage and energy consumption. In the following we provide both the mathematical program and the heuristic method

to perform such exploration. Finally we show our MOPSO algorithm that using MILP solutions as reference points is able to find better solutions than other approaches.

4.1 MILP formulation

For the problem variables and parameters, the application in study can be represented initially by a set of n containers $\mathbf{C}$ which are candidates to be instantiated (or implemented) as a certain DDT from the set of possible implementation of DDTs library $\mathbf{D}$ in Table 1.

Let be x_{ij} a $0,1$ variable that indicates when a container $i \in \mathbf{C}$ of the original application is implemented by a DDT $j \in \mathbf{D}$ of the design space. Since every container in the application must be implemented by one and only one DDT, the following constraint must be fulfilled for each $i \in \mathbf{C}$:

$$\sum_{j \in \mathbf{D}} x_{ij} = 1 \tag{8}$$

Let be A the number of memory accesses performed by the application, U the total memory used and E the energy consumed by the application, it is clear that

$$A = \sum_{i \in \mathbf{C}, j \in \mathbf{D}} a_{ij} x_{ij} \tag{9}$$

$$U = \sum_{i \in \mathbf{C}, j \in \mathbf{D}} u_{ij} x_{ij} \tag{10}$$

$$E = \sum_{i \in \mathbf{C}, j \in \mathbf{D}} e_{ij} x_{ij} \tag{11}$$

In the previous equations, parameters a_{ij}, u_{ij} and e_{ij} are the number of memory accesses, total amount of memory used and energy consumed by the application when the container i is implemented by the DDT j, respectively.

Memory accesses a_{ij} is given by the following equation:

$$a_{ij} \propto N^e_{ij} \times \left(N^r_{ij} + N^w_{ij}\right) + N^{ve}_{ij} \tag{12}$$

where N^e is the number of elements stored in the container in the worst case, N^r is the number of read accesses, N^w is the number of write accesses, and N^{ve} is the average of the number of elements stored. The exact form of equation 12 depends on each DDT selected in (i, j). It takes into account the number of random and sequential accesses to the elements stored in the DDT, as well as the number of creations and destructions of the container.

Memory usage u_{ij} is given by the following equation:

$$u_{ij} \propto T^{ref} + N^e_{ij} \times \left(T^{ref} + T^e\right) \tag{13}$$

where T^{ref} is the size of the pointers in bytes and T^e the size of the elements in bytes. As in equation 12, the exact form of equation 13 depends on each DDT j selected to implement the container i. It computes the amount of memory used by each element stored in the DDT.

Finally, energy equation e_{ij} of the system is given by the following equation:

$$\begin{aligned} e_{ij} = {} & t^{ex}_{ij} \times CPU^{pow} + \\ & \left(N^r_{ij} + N^w_{ij}\right) \times \left(1 - N^{pa}_{ij}\right) \times C^{accE} + \\ & \left(N^r_{ij} + N^w_{ij}\right) \times N^{pa}_{ij} \times C^{accE} \times C^{lineS} + \\ & \left(N^r_{ij} + N^w_{ij}\right) \times N^{pa}_{ij} \times DRAM^{accP} \times \\ & \left(DRAM^{accT} + \frac{C^{lineS}}{DRAM^{bandW}}\right) \end{aligned} \tag{14}$$

Table 2: Coding a solution

$c_1 = 0.3$	$c_2 = 3.6$	$c_3 = 8.9$	...	$c_n = 1.1$
AR	DLL	$SLL(ARO)$	...	$AR(P)$

where t^{ex}_{ij} is the system's total execution time, CPU^{pow} is the total processor power excluding the cache power, C^{accE} is the cache access energy, C^{lineS} is the cache line size, N^{pa}_{ij} is the number of cache misses, $DRAM^{accP}$ is the active power consumed by the DRAM, $DRAM^{accT}$ is the DRAM latency time, and $DRAM^{bandW}$ is the bandwidth of the DRAM.

There exist several techniques to solve a multi-objective optimization problem in the field of mathematical programming, such as minimizing a weighted sums of objectives, homotopy techniques, goal programming, normal-boundary intersection, multilevel programming, etc [12]. However, whereas MILPs are extremely fast in minimizing single objectives, multi-objective techniques require intensive computation, and in many cases good compromises between objectives are not properly reached [5]. On the contrary, evolutionary algorithms offer a whole set of non-dominated solutions in one simulation run. In this work we use MILP solutions as reference points for MOPSO. It significantly improves the quality of solutions found by the swarm when it works without reference points.

4.2 MOPSO representation

Table 2 shows the representation of a candidate solution (gray shaded cells) used in our MOPSO representation. Each field of the candidate solution represents the DDT from Table 1 that should be used to instantiate the corresponding container in the application. For example, the second container $c_2 \in \mathbf{C}$ will be instantiated by our *Doubly-Linked List (DLL)* $\in \mathbf{D}$. A candidate solution contains n real fields, where n is the number of the containers logged in the application, $n = size(\mathbf{C})$. The constraint a field must satisfy is $0 \leq c_i < size(\mathbf{D})$. Rounded up to the next highest integer, it represents the DDT that must implement the corresponding container. For example, a value $c_i \in [0..1)$ means that the container will be instantiated by our first DDT $\lceil c_i \rceil = 1$, i.e. our array DDT (AR in Table 1).

The multi-objective function used to evaluate each particle in the swarm is the same that the one described in the MILP formulation (memory accesses, memory usage and power consumption). But, in the MOPSO algorithm the evaluation is largely simplified since we do not have to evolve a set of binary decision variables, i.e., the set of decision variables is reduced from $\mathbf{C} \times \mathbf{D}$ to $\mathbf{C}$. Thus, if x_i represents the i^{th} position of a particle, the 3-objective function is reduced to the following equations:

$$A = \sum_{i \in \mathbf{C}} a_{i\lceil x_i \rceil} \tag{15}$$

$$U = \sum_{i \in \mathbf{C}} u_{i\lceil x_i \rceil} \tag{16}$$

$$E = \sum_{i \in \mathbf{C}} e_{i\lceil x_i \rceil} \tag{17}$$

where a_{ij}, u_{ij} and e_{ij}, are defined by equations 12, 13 and 14 respectively.

5. MOPSO WITH MILP REFERENCE POINTS

The proposed reference point MILP-based MOPSO algorithm involves the following steps:

1. Obtain a set of MILP solutions. First, the MILP is solved for each objective separately. Thus, a set of four points are obtained to include in the list of reference points: $(A^*_{MILP}, 0, 0)$, $(0, U^*_{MILP}, 0)$, $(0, 0, E^*_{MILP})$ and a combination of them $(A^*_{MILP}, U^*_{MILP}, E^*_{MILP})$, where A^*_{MILP}, U^*_{MILP} and E^*_{MILP} are the optimal MILP points for memory accesses, memory usage and energy consumption respectively. A set of system designer preference points may be included as well. We do not have to worry about the feasibility of the reference points. Following the reference method proposed in [19], the system designer can also specify a spread as a preference. The spread defines the extent of the solutions on the Pareto front near the reference point. The spread is given by a value δ, which is defined as the maximum variance of the distance values of the population.

2. Initialize the particles. The population is first initialized. The particles are evaluated according to the objective functions and fitness is assigned. Each particle is assigned to the closest reference point. For any vector $\vec{x}$ the distance to a reference point $\vec{z}$ is defined by the following equation

$$dist\,(\vec{x}) = \max_{i=1\ldots m} \{w_i\,(f_i\,(\vec{x}) - \vec{z}_i)\} \tag{18}$$

As in [19], here $\sum_{i=1}^{m} w_i = 1.0$. A particle's assigned reference point will remain unchanged throughout the run. The particle will choose a leader, which also has the same reference point.

3. Obtain non-dominated solutions and rank according to the closeness to the reference points. The non-dominated particles are extracted from the population. Next, the non-dominated particles assigned for each reference point are ranked according to the ascending order of distance values. Particles with lower distance values are considered as candidates to be leaders.

4. Choose leaders from the assigned ranked non-dominated set and move the particles. Each particle in the population will choose a leader (global best) from the assigned set of non-dominated solutions. Here we apply a binary tournament in the ranked non-dominated particles to choose leaders with the same reference points. Then each particle will adjust their velocities and positions according to equations 5 and 6.

5. Evaluate particles using the objective functions. The entire population is evaluated and fitness values are assigned. This fitness value will be next used to determine the dominance. New distance values are assigned and then steps 3 to 5 are repeated until the stop criteria is met. The algorithm will stop once it has reached the desired spread of particles on the Pareto front near the reference points or the maximum number of iterations allowed.

6. EXPERIMENTS

In this research work we have explored DDTs for VDrift and Physics using two Multi-Objective Genetic Algorithms (MOGAs): NSGA-II [7] and SPEA2 [22], and two MOPSOs: NSPSO [11] and OMOPSO [15]. Finally, we incorporated the reference point method based on MILP solutions to OMOPSO (named OMOPSO*), and compared all the results. The library used to run all the evolutionary algorithms can be found at [17]. The MILP was executed using ILOG CPLEX [1]. All the values are computed by averaging 30 trials.

Table 3: System specification

Processor Energy	168 mW, 100MHz
Embedded DRAM	100MHz
Energy	19.5 mW
Latency	19.5 ns
Bandwidth	50MB/s

The model of the embedded system architecture consisted of a processor with an instruction cache, a data cache, and embedded DRAM as main memory. The data cache uses a write-through strategy. We utilized processor energy from [4], and the access time and energy values for caches of 32KB and embedded 16MB DRAM main memory from [18] and [8], respectively. The processor and memory specification is described in Table 3.

Since the size of all possible DDT implementations is large and it is unfeasible to cover the exact set of the POF, we compare the obtained approximated front with each other using the hypervolume indicator: I^-_H. This metric computes the volume (in the objective space) covered by members of a non-dominated set of solutions Q [21]. Let v_i be the volume enclosed by solution $i \in Q$. Then, a union of all hypercubes is found and its hypervolume (I_H) is calculated. If a set X has a greater hypervolume than a set Y, then X is taken to be a better set of solutions than Y. In this work, we consider the hypervolume difference to a reference set R, defined as

$$I^-_H(Q) = I_H(R) - I_H(Q) \tag{19}$$

where smaller values correspond to higher quality. Since the reference set cannot be computed, we take $I_H(R) = 0$.

The performance of the four algorithms are compared using the following parameters, obtained after several tests:

- Population/Swarm size: 100 in the case of VDrift and 200 in the case of Physics.
- Number of iterations: 2000 for VDrift and 4000 for Physics.
- Crossover: Real-parameter SBX crossover operator ($\eta_c = 20$). Crossover probability of 0.9 for NSGA-II and SPEA2.
- Mutation: Polynomial mutation operator ($\eta_m = 20$), with probability inversely proportional to the chromosome length for NSGA-II and SPEA2. Uniform and non-uniform mutation applied to OMOPSO.
- Coding strategy: Real encoding for all the algorithms.
- $W = 0.4$, $C_1 = 2.0$ and $C_2 = 2.0$ for NSPSO.
- Spread: $\delta = 0.001$.

Table 4 shows the hypervolume or S-metric obtained for VDrift and Physics. In both cases, OMOPSO* algorithm reaches better values compared to the other MOEAs. Thus, the result set from our reference point method is taken to be a better set of solutions than those obtained from other algorithms. Note than even when OMOPSO offers the worst

Table 4: Hypervolume metric for VDrift/Physics.

Algorithm	VDrift	Physics 3D
OMOPSO*	$\mathbf{-1.1913 \pm 0.0035}$	$\mathbf{-1.2892 \pm 0.0915}$
NSGA-II	-0.9356 ± 0.0231	-1.1600 ± 0.1902
NSPSO	-1.1360 ± 0.0363	-1.1830 ± 0.1074
OMOPSO	-1.1657 ± 0.0114	-0.7215 ± 0.2449
SPEA2	-1.1492 ± 0.0374	-0.9002 ± 0.1899

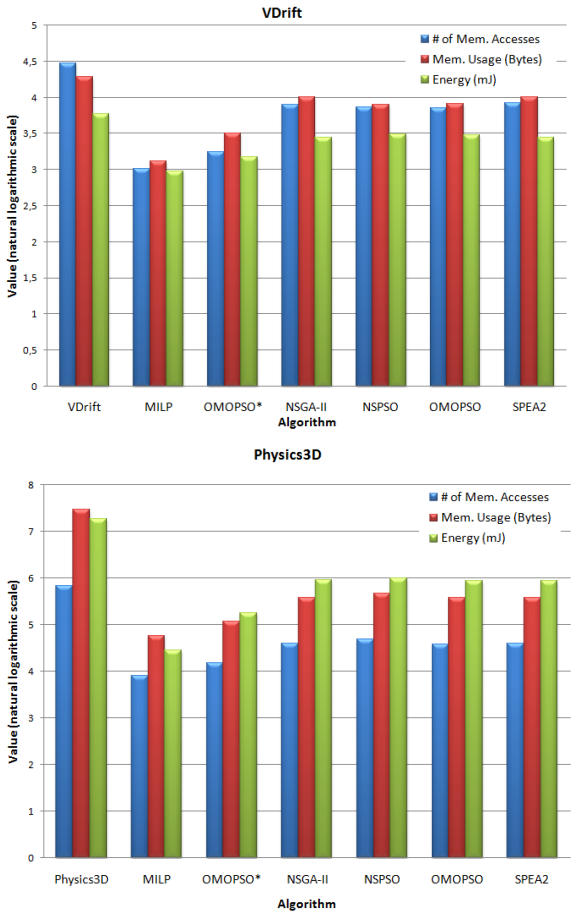

Figure 5: Comparison of the real application with results obtained by our design framework (logarithmic scale).

values for Physics in terms of hyper-volume (-0.7215 in average), results are improved when our MILP solutions are incorporated as reference points (-1.2892 in average).

We present Figure 5 to compare the number of memory accesses, memory usage (in Bytes) and energy consumed (in mJoules) of the original application with the resultant optimized application with all the algorithm under test. To this end, each application is evaluated with their original DDTs and compared to the combination proposed by our framework. The figure shows clearly the achieved level of optimization and final gains after applying the proposed optimization flow in Figure 4. In the case of evolutionary algorithms, the set of non-dominated solutions obtained is averaged. It should be noted that OMOPSO* offered the best gain in all the three objectives. Although MILP points are represented in Figure 5, they are independently optimized. As a result, these points are outside of the feasible region. However, they are represented to observe the degree of approximation of all the MOEAs to these true optimal points.

7. CONCLUSIONS

Latest multimedia embedded systems include applications with a significant degree of dynamism. Therefore, they need to rely on *Dynamically-allocated Data Types (DDTs)* to efficiently store their data, rather than more traditional statically allocated variables in embedded systems. However, the selection of the best DDTs for each target embedded system is a very complex and time-consuming process due to the large design space of possible DDTs implementations. In this paper we have presented a new method to solve the combinatorial optimization problem of DDTs for embedded systems. Our results show that although multi-objective evolutionary algorithms are a relatively good choice to obtain reasonable approximation sets of optimal solutions for DDTs, other complementary techniques, such as the reference point method, may be incorporated, which allows the system designer to define specific preferences or optimization metric restrictions to guide the exploration algorithm towards particular optimization regions. More precisely, we exploit this approach in DDT optimization by including solutions of *Mixed Integer Linear Program (MILP)* as reference points to find better solutions. Indeed, our results have shown that MILP is able to obtain true optimal solutions when they are optimized independently, and then we have been able to incorporate these points to a new *Multi-Objective Particle Swarm Optimization (MOPSO)* algorithm that we have proposed (i.e., OMOPSO*) to achieve even better coverage of the multi-objective Pareto front of DDT implementations. The experimental results in two embedded applications, using a hyper-volume indicator, have shown that our proposed OMOPSO* offers better coverage results (up to 40%) than other state-of-the-art optimization algorithms to this problem (two variations of MOGA algorithms and two other MOPSOs). As a consequence, the overall performance of both applications under study has been compared and validated that OMOPSO* indeed outperformed the best DDT solutions found with the other four optimization algorithms compared with.

8. ACKNOWLEDGMENTS

This work has been supported by Spanish Government grants TIN2008-00508 and MEC Consolider Ingenio CSD00C-07-20811 of the Spanish Council of Science and Technology.

9. REFERENCES

[1] ILOG CPLEX. http://www.ilog.com, 2008.

[2] VDrift racing simulator. http://vdrift.net, 2008.

[3] D. Atienza, C. Baloukas, L. Papadopoulos, C. Poucet, S. Mamagkakis, J. I. Hidalgo, F. Catthoor, D. Soudris, and J. Lanchares. Optimization of dynamic data structures in multimedia embedded systems using evolutionary computation. In *SCOPES '07: Proceedings of the 10th international workshop on Software & compilers for embedded systems*, pages 31–40, New York, NY, USA, 2007. ACM.

[4] F. Catthoor, K. Danckaert, C. Kulkarni, E. Brockmeyer, P. G. Kjeldsberg, T. V. Achteren, and

T. Omnes. *Data access and storage management for embedded programmable processors*. Kluwer Academic Publishers, 2002.

[5] J. M. Cruz, J. L. Risco-Martín, A. Herrán-González, and P. Fernández-Blanco. Hybrid heuristic and mathematical programming in oil pipelines networks. In *CEC'04: Proceedings of the 2004 Congress On Evolutionary Computation*, volume 2, pages 1479–1486, June 2004.

[6] E. G. Daylight, D. Atienza, A. Vandecappelle, F. Catthoor, and J. M. Mendias. Memory-access-aware data structure transformations for embedded software with dynamic data accesses. *IEEE Transactions on VLSI Systems*, 12:269–280, 2004.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[8] K. Hardee, F. Jones, D. Butler, M. Parris, M. Mound, H. Calendar, G. Jones, L. Aldrich, C. Gruenschlaeger, M. Miyabayashil, K. Taniguchi, and I. Arakawa. A 0.6V 205MHz 19.5ns tRC 16Mb embedded DRAM. In *IEEE International Solid-State Circuits Conference (ISSCC)*, 2004.

[9] J. I. Hidalgo, J. L. Risco-Martín, D. Atienza, and J. Lanchares. Analysis of multi-objective evolutionary algorithms to optimize dynamic data types in embedded systems. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1515–1522, New York, NY, USA, 2008. ACM.

[10] L. Kharevych and R. Khan. 3D physics engine for elastic and deformable bodies. University of Maryland, College Park, 2002.

[11] X. Li. A non-dominated sorting particle swarm optimizer for multiobjective optimization. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 37–48, Chicago, 2003. Springer-Verlag.

[12] F. A. Lootsma. *Multi-Criteria Decision Analysis via Ratio and Difference Judgement*, chapter Multi-Objective Linear Programming, pages 1384–6485. Springer, 1999.

[13] K. Miettinen. Some methods for nonlinear multi-objective optimization. In *EMO'01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pages 1–20, London, UK, 2001. Springer-Verlag.

[14] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 6(2):149–206, 2001.

[15] M. Reyes-Sierra and C. A. C. Coello. Improving PSO-based multi-objective optimization using crowding, mutation and epsilon-dominance. In *EMO*, pages 505–519, 2005.

[16] M. Reyes-Sierra and C. A. C. Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[17] J. L. Risco-Martín. Java Evolutionary COmputation library (JECO). Available at: https://sourceforge.net/projects/jeco, 2008.

[18] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical Report 2001/2, Compaq Computer Corporation, 2001.

[19] U. K. Wickramasinghe and X. Li. Integrating user preferences with particle swarms for multi-objective optimization. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 745–752, New York, NY, USA, 2008. ACM.

[20] L. A. Wolsey. *Integer Programming*. John Wiley and Sons, 1998.

[21] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), 1999.

[22] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In *Proceedings of the Evolutionary Methods for Design, Optimization and Control with Application to Industrial Problems*, pages 95–100, Barcelona, Spain, 2002.