Design Flow of Dynamically-Allocated Data Types in Embedded Applications Based on Elitist Evolutionary Computation Optimization

José L. Risco-Martín, David Atienza, J. Ignacio Hidalgo and Juan Lanchares
Department of Computer Architecture and Automation
Complutense University of Madrid
C/Prof. José García Santesmases, s/n, 28040 Madrid, Spain
{jlrisco,datienza,hidalgo,julandan}@dacya.ucm.es

Abstract

New multimedia embedded applications are increasingly dynamic, and rely on Dynamically-allocated Data Types (DDTs) to store their data. The optimization of DDTs for each target embedded system is a time-consuming process due to the large design space of possible DDTs implementations. Thus, suitable exploration methods for embedded design metrics (memory accesses, memory usage and power consumption) need to be developed. This paper presents a design flow to tackle the optimization of DDTs in multimedia applications. By profiling of the original desktop application and using evolutionary algorithms, the proposed approach is able to find solutions 1584× faster than other state-of-the-art heuristics in an automated way. Moreover, we study the use of elitist Multi-Objective Evolutionary Algorithms (MOEAs) to explore DDT implementations, which offer 75% more optimal solutions to the system designer for the implementation of the final embedded application. To this end, we analyze the quality of the solutions by comparing three MOEAS and other optimization heuristics. Our results in two object-oriented multimedia embedded applications show that elitist MOEAs (NSGA-II and SPEA2) offer better solutions than simple non-elitist schemes (VEGA) and alternative well-known optimization heuristics.

1. Introduction

Latest multimedia embedded devices are enhancing their capabilities and are currently able to run applications reserved to powerful desktop computers few years ago (e.g., 3D games, video players). As a result, one of the most important problems designers face nowadays is the integration of a great 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 [1] (dynamic arrays, linked lists, etc.) the best one in each case, according to the specific restrictions for each target device and typical embedded design metrics, such as memory accesses, memory usage and energy consumption.

This task is typically performed using a pseudo-exhaustive evaluation of the design space of DDTs, including multiple executions of the application, to attain a Pareto front [6], which tries to cover all the optimal implementation points for the aforementioned required design metrics. The construction of this Pareto front is a very time-consuming process, sometimes even unaffordable [6], [15].

This paper presents a design flow by means of which multimedia desktop applications can be optimized by minimizing memory accesses, memory usage and power consumption for the particular hardware architecture of each target embedded system. To this end, our design space is formed by a very large number of multi-level DDTs proposed in the literature [21][6]. Our proposed design flow includes: (1) an automatic exploration composed by three independent phases, i.e., profiling, parameters estimation and optimization, (2) a 3-objective fitness function defined to minimize memory accesses, memory usage and energy consumption, and (3) the exploitation of a fast optimization method using elitist *Multi-Objective Evolutionary Algorithms (MOEAs)* [7], which offers more optimal alternatives to the system designer than other optimization heuristics and non-elitist evolutionary computation methods. In order to validate the proposed methodology, we applied our desing flow in two real-life dynamic embedded applications using six algorithms. Three traditional heuristics, namely breadth-first-search, depth-first-search and branch&bound, and three representative MOEAs, namely *Vector Evaluated Genetic Algorithm (VEGA)* [17], *Non-dominated Sorting*

Genetic Algorithm II (NSGA-II) [8], and *Strength Pareto Evolutionary Algorithm 2 (SPEA2)* [22]. On the one hand, our experiments show that evolutionary algorithms can achieve significant speed-ups compared to other traditional heuristics (up to 1584× faster in the case of VDrift and 1560× faster in the case of a 3D Physics engine). On the other hand, the quality of the solutions found by elitist algorithms is quite better than those found by the non-elitists ones. In addition, elitist algorithms offer 75% more optimal solutions.

The rest of the paper is organized as follows. Section 2 reviews the work related to memory optimizations for embedded systems. In Section 3, we present our multi-objective design process. For self-contained purposes, a brief background on multi-objective optimization and evolutionary algorithms is given in Section 4. Section 5 details our experimental setup. In Section 6, we discuss the obtained results in two real-life embedded applications. Finally, Section 7 summarizes the main conclusions of this paper.

2. Related Work

Up today extensive work has been performed in the field of embedded memory subsystem optimization. [15] and [3] include a thorough survey of static data and memory optimization techniques for embedded systems. In [4] and [21], 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.

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 [21][15]. 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 [2] it has already been outline the potential use of non-elitist MOEAs for dynamic memory optimizations. Nevertheless, this work only performed an initial analysis of one type of MOEA and does not provide a complete analysis of trade-offs between different technologies of MOEAs, as we perform in this paper.

Also, according to the characteristics of certain parts of multimedia applications, several transformations for DDTs and design methodologies [4][21] have been proposed for static data profiling and optimization considering static memory access patterns to physical memories. In this context, the use of MOEA-based optimization has been applied to solve linear and non-linear problems by exploring all regions of the state 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][15]. However, such techniques are not applicable in DDT implementations, due to the unpredictable nature at compile-time of the stored data.

3. DDTs Multi-Objective Optimization Flow

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 run-time 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.

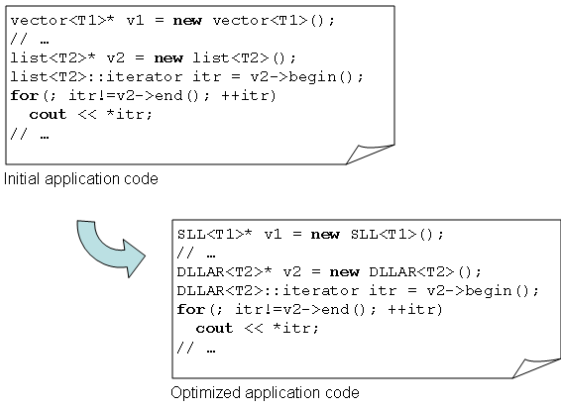

Figure 1. Code before and after the exploration of Dynamic Data Types

Figure 1 shows an example of a DDTs exploration. The initial code contains two variables, `v1` and `v2`, instantiated as vector and list, respectively. After the exploration process, one candidate solution gives `v1` to be instantiated as *Singly-Linked List (SLL)* and `v2` as *Doubly-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 the case of the 3D Physics engine, we optimized memory accesses, memory usage and

power consumption for more than 3000 variables.

In general, the application to optimize contains a set of n variables ($\{V\}$) which are candidates to be instantiated as a certain DDT from the set of possible implementation of DDTs library ($\{D\}$) (we used those presented in [2][6]). Thus, the goal of our optimization flow is to obtain a set of pairs (variable, DDT) or $(\vec{v}, \vec{d}), v_i \in \{V\}, d_i \in \{D\}, 1 \leq i \leq n$, 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 design flow uses three different phases to perform the automatic exploration of DDTs using MOEAs. Figure 2 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 exploration algorithm selected.

Next, we describe the three phases of our flow in detail.

3.1. Profiling

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 a first pre-characterization phase the equations to evaluate the behavior of DDT implementations by means of parameters such as the number of sequential accesses, random accesses , average size , etc., must be defined. In our case we have used the classification and equations defined in [2]. Table 1 lists such DDTs.

Next, we obtain a profiling report of the application where the following information is logged: accessing of an element, addition of an element, removal of an element, the clearing of the container, iterator operations such as pre-increment or dereference, constructor, destructor, copy constructor and swap operation. To this end, we replace all the candidate variables in the application by our vector DDT implementation, which logs all the needed information. Such replacement is done automatically, where the designer may choose the variables to include in the optimization.

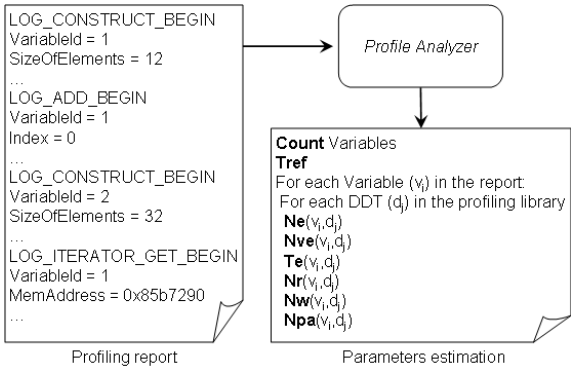

Figure 3. Example of profiling report and parameters estimation

The left side of Figure 3 shows an illustrative example on how the profiling report logs all the variables selected in the original application. The report shows two entries, labeled as LOG_CONSTRUCT_BEGIN, where two variables are instantiated and identified as VariableId 1 and 2. Such entries state that the variables will contain elements of size 12 and 32 Bytes, respectively. Another entry, labeled as LOG_ADD_BEGIN, shows that an element is added to the variable 1, at index 0. The last entry shows that an iterator accesses to a element stored in variable 2, and located at the address 0x85b7290.

3.2. Parameters estimation

In the following phase, we extract all information needed from the profiling report. The purpose is to evaluate the fitness of a candidate solution $\vec{v}, \vec{d}$ in the DDT exploration, using parameters such as the number of candidate variables ($Count$ in Figure 3), number of elements stored in the DDT in the worst case ($N_e(\vec{v}, \vec{d})$), average of the number of elements stored ($N_{ve}(\vec{v}, \vec{d})$), size of the elements ($T_e(\vec{v}, \vec{d})$, in bytes), size of the pointers (T_{ref}, in bytes), number of read accesses ($N_r(\vec{v}, \vec{d})$), number of write accesses ($N_w(\vec{v}, \vec{d})$) and cache misses ($N_{pa}(\vec{v}, \vec{d})$). All the parameters are obtained using the information logged in the profiling report and the analytical characterization. We have developed a tool called *Profile Analyzer*, which automates all the process. The right side of Figure 3 depicts an illustrative example on how the Profile Analyzer saves such parameters in an external file, used by the optimization phase.

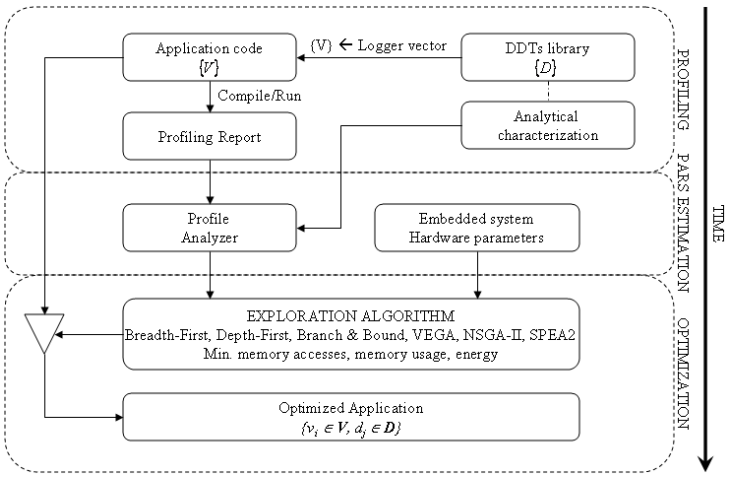

Figure 2. Proposed Embedded application design flow

3.3. Optimization

The last phase is the optimization process. It takes as input the parameters obtained in the previous phase and minimizes three objectives: memory accesses (MA), memory usage (MU) and energy (E), defined by equation 1, where Hw represents the effect that hardware parameters (memory architecture, CPU power, line sizes, memory access time, etc.) have on the optimization, and $(\vec{v}, \vec{d})$ represents one candidate solution.

$$\begin{aligned} MA(\vec{v},\vec{d}) &= f_{MA}(N_e, N_{ve}, N_r, N_w) \\ MU(\vec{v},\vec{d}) &= f_{MU}(T_e, T_{ref}, N_e) \\ E(\vec{v},\vec{d}) &= f_E(N_r, N_w, N_{pa}, Hw) \end{aligned} \quad (1)$$

Memory accesses of the system f_{MA} is given by the following equation:

$$f_{MA} \propto N_e \times (N_r + N_w) + N_{ve} \quad (2)$$

The exact form of equation 2 depends on each DDT selected in $(\vec{v}, \vec{d})$. 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 variable.

Memory usage f_{MU} is given by the following equation:

$$f_{MU} \propto T_{ref} + N_e \times (T_{ref} + T_e) \quad (3)$$

As in equation 2, the exact form of equation 3 depends on each DDT selected. It calculates the ammount of memory used by each element stored in the DDT.

Finally, energy equation of the system is given by the following equation:

$$\begin{aligned} f_E = \; & t_{ex} \times CPU_{pow} + \\ & (N_r + N_w) \times (1 - N_{pa}) \times C_{accE} + \\ & (N_r + N_w) \times N_{pa} \times C_{accE} \times C_{lineS} + \\ & (N_r + N_w) \times N_{pa} \times DRAM_{accP} \times \\ & \left(DRAM_{accT} + \frac{C_{lineS}}{DRAM_{bandW}} \right) \end{aligned} \quad (4)$$

where t_{ex} is the system's total execution time, CPU_{pow} is the total processor power excluding the cache power, C_{accE} is the cache access energ, C_{lineS} is the cache line size, $DRAM_{accP}$ 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 four components in the energy equation 4. The first term $t_{ex} \times CPU_{pow}$ calculates the processor energy given that execution time takes t_{ex} amount of time. The second term, $(N_r + N_w) \times (1 - N_{pa}) \times C_{accE}$ calculates the amount of energy consumed by the cache. The third term, $(N_r + N_w) \times N_{pa} \times C_{accE} \times C_{lineS}$ calculates the energy cost of writing to cache for each cache miss. The last term, calculates the energy cost of the DRAM to service all the cache misses.

Units for time variables in the equations are in seconds, bandwidth is in Bytes/sec., cache line size is in Bytes, power variable is in Watts, and energy unit is in Joules.

These equations are used by the optimization algorithm to evaluate que fitness of the solutions found in the exploration proccess. When the optimization process ends, it gives the DDT instantiation policy, i.e., which variable should be instantiated by which DDT. We also obtain the

gain on memory accesses, memory usage and energy consumption.

4. Multi-Objective Evolutionary Algorithms

Multi-objective optimization aims at simultaneously optimizing several contradictory objectives. For such kind of problems, it does not exist a single optimal solution, and some compromises have to be made. Thus, without any loss of generality, we can assume the following N-objective minimization problem:

$$\begin{array}{ll} Minimize & \vec{z} = (f_1(\vec{x}), f_2(\vec{x}), \ldots f_N(\vec{x})) \\ subject\ to & \vec{x} \in S \end{array} \quad (5)$$

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

$$\begin{array}{l} f_i(\vec{x}) \leq f_i(\vec{y}), \forall i \in \{1, 2, \ldots, N\} \\ f_i(\vec{x}) < f_i(\vec{y}), \exists i \in \{1, 2, \ldots, N\} \end{array} \quad (6)$$

A decision vector $\vec{x} \in S$ is called *Pareto-optimal* if there does not exist another $\vec{y} \in S$ that dominates it. An objective vector is called Pareto-optimal if the corresponding decision vector is Pareto-optimal.

The non-dominated set of the entire feasible search space S is the *Pareto-optimal set (POS)*. The Pareto-optimal set in the objective space is called *Pareto-optimal front (POF)* of the multi-objective problem at hand: it represents the best possible compromises with respect to the contradictory objectives.

A Multi-objective optimization problem is solved, when all its POS is found. MOEAs directly search for the whole POF, in contrast to classical optimization methods, that generally find one of the Pareto optimal solutions. Thus, MOEAs allow decision makers to choose one of the Pareto solutions with more complete information [7].

Up to date, many MOEAs have been developed. Generally speaking, they can be classified into two broad categories: non-elitist and elitist, also called first and second generation evolutionary algorithms. With the elitist approach, MOEAs store in an external set the best solutions of each generation. This set will then be a part of the next generation. Thus, the best individuals in each generation are always preserved, and this helps the algorithm to get close to its POF. Algorithms such as PESA-II [5], MOMGA-II [24], NSGA-II and SPEA2 are examples of this category. In contrast, the non-elitist approach does not guarantee preserving the set of best individuals for the next generation [23]. Examples of this category include MOGA [9], HLGA [10], NPGA [12] and VEGA.

For the purpose of this research, we have implemented three popular MOEAs, called VEGA, NSGA-II and SPEA2. VEGA was selected as a very optimized example of non-elitist MOEAs, where the concept of POF is not directly incorporated into the selection mechanism of this algorithm. NSGA-II and SPEA2 were utilized as two representative elitist-based MOEAs.

5. Experimental methodology

In this section we describe the experimental setup used to validate our design flow and the use of elitist MOEAs while optimizing two real-life dynamic embedded applications.

5.1. HW/SW Specification

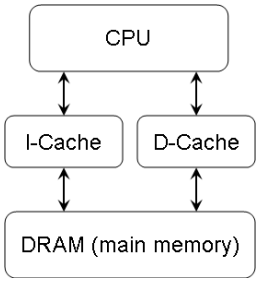

Figure 4. System architecture

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. The system architecture is illustrated in Figure 4.

Table 2. System specification

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

To analyze the effect of MOEAs on embedded system's memory accesses, memory usage and energy consumption, we utilized processor energy from [4], and the access time and energy values for caches of 32KB and embedded 16MB DRAM main memory from [19] and [11], respectively. The processor and memory specifications are summarized in Table 2.

In this paper, we apply our design flow to two representative multimedia embedded applications. The first benchmark is VDrift, which is a driving simulation game [20].

The game includes 19 tracks, 28 cars, artificial intelligence players, networked multiplayer mode, etc. We logged 49 variables in its source code. The second benchmark is a 3D Physics Engine for elastic and deformable bodies [13], which is a 3D engine that displays the interaction of non-rigid bodies. It includes 3128 dynamic variables in its source code for which we select the optimal DDT implementation.

5.2. Quality of solutions

To compare the quality of the solutions offered by different MOEAs, we need to evaluate the obtained set of non-dominated solutions considering: (1) Convergence to POF. (2) Diversity on POF. Since the size of possible DDT implementations is large and it is not possible to cover the exact set of the POF, we compare the obtained *Pareto Front (PF)* with each other. In this direction, we select the following metrics to evaluate the performance of our approach.

Coverage (C): We use the coverage metric [23] to measure convergence. Let PF', PF'' be two sets of non-dominated solutions. The coverage metric can be defined as follows:

$$C(PF', PF'') = \frac{|p'' \in PF''; \exists p' \in PF' : p' \preceq p''|}{|PF''|} \quad (7)$$

where if $C(PF', PF'') > C(PF'', PF')$, the rate of dominated solutions in PF' is higher than in PF''.

Spread (D): A spread metric (D) determines in each objective space the maximum range represented by the non-dominated solutions. It was introduced by Ranjithan [16]. A higher value of the spread metric indicates a better performance. It is defined as:

$$D = \sqrt{\sum_{i=1}^{m} \left(\max_{j=1}^{|PF|} f_i(x_j) - \min_{j=1}^{|PF|} f_i(x_j) \right)^2}$$
$$x_j \in PF, j = 1, 2, \ldots, |PF| \quad (8)$$

where m is the number of objectives.

Spacing (S): Schott proposed a metric which allows to measure the distribution of vectors throughout PF [18]. It is defined as:

$$S = \sqrt{\frac{1}{|PF|} \sum_{j=1}^{|PF|} \left(d_j - \bar{d}\right)^2}$$
$$d_j = \min_{x_k \in PF \wedge k \neq j} \sum_{i=1}^{m} |f_i(x_j) - f_i(x_k)| \quad (9)$$

where m is the number of objectives, and $\bar{d}$ is the mean of all d_j. A zero value for this metric means that all members of PF are equidistantly spaced.

We compare the obtained sets of non-dominated solutions by means of the above three criteria.

5.3. Coding a solution

In order to apply a MOEA correctly we need to define a genetic representation of the design space of all possible DDT implementations alternatives. Moreover, to be able to cover all possible inter-dependencies of DDT implementations for different dynamic variables of an application, we must guarantee that all the individuals represent real and feasible solutions to the problem and ensure that the search space is covered in a continuous and optimal way [7].

Table 3. Example of an individual

d_1	d_2	d_3	...	d_n	$d_j \in \{D\}$
v_1	v_2	v_3	...	v_n	$v_j \in \{V\}$

Table 3 shows the representation of a chromosome. Genes are represented in the first row (gray shaded cells). Each of the chromosomes represents the set of DDT that should be used to instantiate all the corresponding variables in the application from Table 1. For example, the second variable $v_2 \in \{V\}$ will be instantiated by $d_2 \in \{D\}$. A chromosome contains n genes, where n is the number of the variables logged in the application, $n = size(\{V\})$. We may use an integer to represent the values of a gene, and the constraint a gene must satisfy is:

$$1 \leq d_j \leq size(\{D\}) \quad (10)$$

Consequently, if an application contains n variables, each individual (chromosome) has to be constituted by n integer fields (i.e., n genes). Our current implementation of the exploration framework optimizes up to 3128 variables using variations of the 10 possible DDTs contained in Table 1 for each of them; Thus, it can cover large real-life dynamic embedded applications.

To compare the performance of evolutionary algorithms, all parameters are set equally. After different tests, we have fixed them to the values indicated in Table 4. The number of non-dominated solutions to preserve between generations is set to the initial population size in the elitist algorithms.

Table 4. Parameters for evolutionary algorithms.

Parameter	VDrift	Physics
Population size	100	200
Number of generations	2000	4000
Probability of crossover	0.80	0.80
Probability of mutation	0.01	0.01

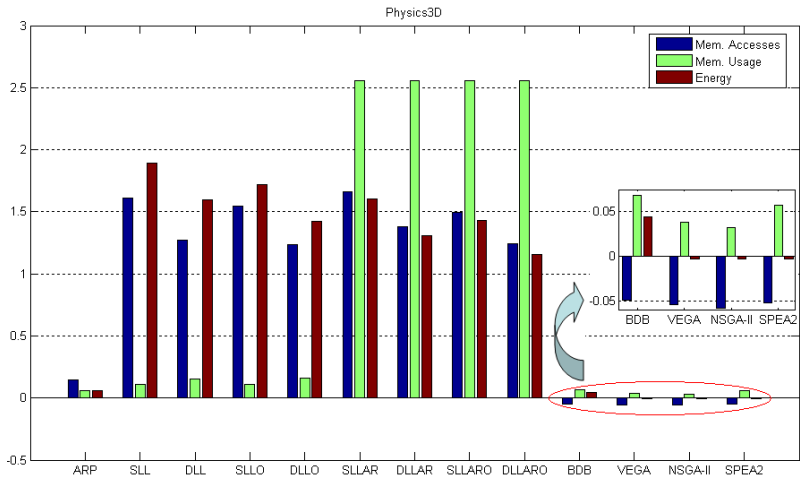

Figure 5. Overall results for different design metrics coming from various sets of DDTs for Physics (logarithmic scale)

6. Experimental results

In this Section we apply our design flow to VDrift and Physics, using state-of-the-art pruning and optimization methods for DDT implementations presented in [21][6][2], and two elitist MOEAs, NSGA-II and SPEA2.

First, we have tested the exploration speed by comparing six exploration methods. The results obtained are shown in Table 5, where the time measured starts from the profiling report in Figure 2. In the case of breadth-first, depth-first and branch&bound explorations we used a weighted sum of the three objectives as the fitness function. All experiments were executed in an AMD Sempron 3600+ 2GHz, with 1GB DDR memory.

Table 5. Time to explore DDT implementations for VDrift and Physics using six exploration algorithms.

DDTs exploration method	VDrift	Physics
Breadth-First search	22 hours	13 days
Depth-First search	8 minutes	5 days
Branch&Bound search	62 seconds	3 hours
VEGA	50 seconds	12 minutes
NSGA-II	64 seconds	17 minutes
SPEA2	159 seconds	25 minutes

The results in Table 5 outline that the exploration process with evolutionary algorithms is much faster than using other heuristics, namely, more than 1584× faster (50 seconds *vs* 22 hours) for VDrift and 1560× (12 minutes *vs* 13 days) for Physics.

For comparison reasons we present in Figure 5 the results of the optimization process that our methodology performs in the case of Physics. In this case, the set of DDTs was successively implemented using AR, ARP, SLL, etc. All the three objectives have been normalized to the AR DDT and represented in logarithmic scale. Since breadth-first, depth-first and branch&bound exploration methods offer the same solution, these results are grouped and labeled as BDB in Figure 5. In the case of evolutionary algorithms, the set of solutions obtained is averaged. The figure shows the achieved level of optimization and final gains after applying the proposed design flow shown in Figure 2. Furthermore, as this figure indicates, evolutionary algorithms offered the best compromise among objectives.

Finally, we show that although VEGA is faster than both NSGA-II and SPEA2 (30% faster than NSGA-II and 55% faster than SPEA2), elitist algorithms offer more alternatives to the system designer. In addition, the quality of the solutions found by elitist algorithms is quite better than those found by VEGA. Such quality is measured in terms of coverage, spread and spacing. These values are calculated by averaging results of 10 trials.

Figure 6 depicts the Pareto-front obtained in the best population of both applications. With respect to VDrift, this figure shows that both NSGA-II and SPEA2 offer the same number of non-dominated individuals and 61% more than VEGA. In the case of Physics, NSGA-II and SPEA2 offer the same optimal solutions. In addition, both NSGA-II and SPEA2 offer 92% more optimal solutions than VEGA. Thus, elitist algorithms cover better the set of possible optimal solutions than non-elitist ones.

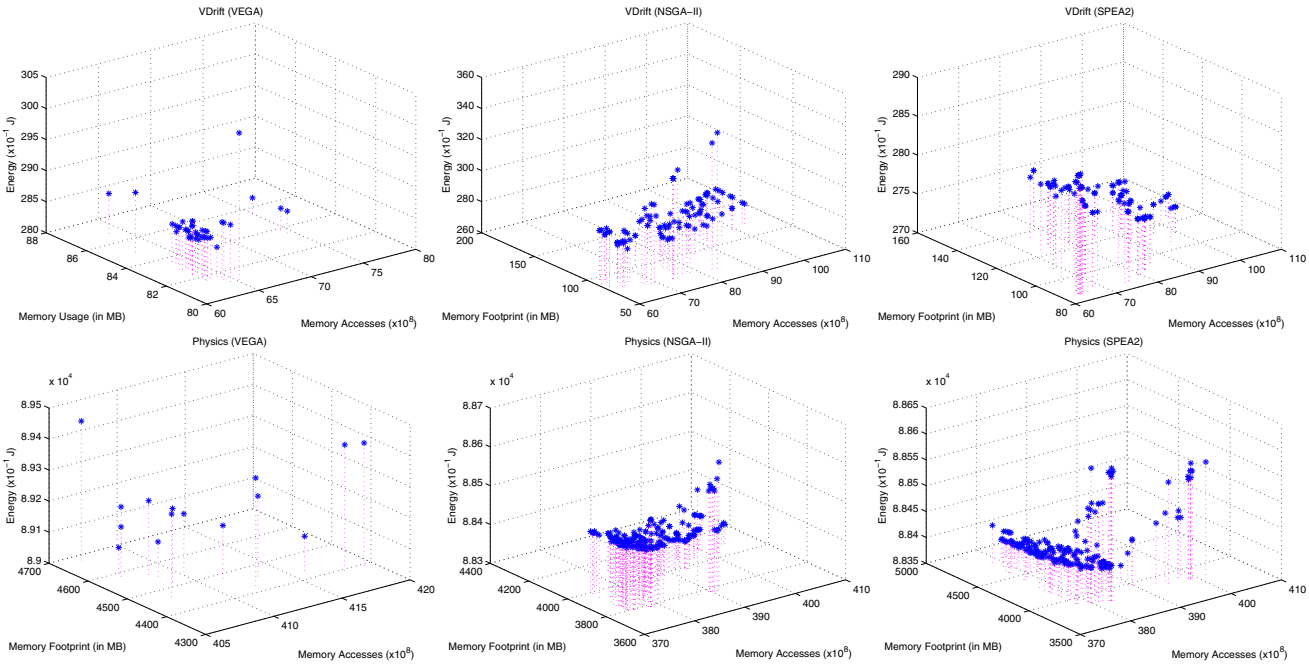

Figure 6. 3D Pareto-fronts obtained for VDrift and Physics using VEGA, NSGA-II and SPEA2. Elitist algorithms offer more alternatives to the system designer.

Table 6. Coverage metric for VDrift/Physics.

VDrift	**VEGA**	**NSGA-II**	**SPEA2**
VEGA	–	0.034	0.005
NSGA-II	0.132	–	0.369
SPEA2	0.023	0.215	–
Physics	**VEGA**	**NSGA-II**	**SPEA2**
VEGA	–	0.000	0.000
NSGA-II	1.000	–	0.462
SPEA2	1.000	0.002	–

Table 7. Spread and spacing for the three evolutionary algorithms and two applications.

	VDrift		**Physics**	
	Spread	δ^2	*Spread*	δ^2
VEGA	1.47E-01	2.23E-03	8.32E-02	1.85E-03
NSGA-II	7.12E-01	1.25E-04	1.26E-01	2.95E-04
SPEA2	5.42E-01	7.73E-04	2.21E-01	3.35E-04
	Spacing	δ^2	*Spacing*	δ^2
VEGA	1.72E-02	9.57E-05	1.11E-02	7.58E-05
NSGA-II	2.18E-02	4.94E-05	1.91E-03	2.40E-07
SPEA2	1.60E-02	2.47E-06	2.82E-03	2.34E-07

Regarding convergence comparisons, Table 6 shows that the coverage values of NSGA-II are better than VEGA or SPEA2 in both cases (e.g., for Physics $C(NSGA2, VEGA) > C(VEGA, NSGA2)$ is $1 > 0$, and $C(NSGA2, SPEA2) > C(SPEA2, NSGA2)$ is $0.462 > 0.002$). Thus, NSGA-II offers more optimal alternatives to the system designer for the implementation of the final embedded application. The situation is different for spread and spacing metrics, where NSGA-II does not obtain the best performance in all cases. Table 7 shows the spread and spacing for the three algorithms and both VDrift and Physics applications. Regarding VDrift the larger spread is found by NSGA-II (52% better, on average), and the best spacing by SPEA2 (22% better), whereas in the case of Physics SPEA2 obtains the larger spread (53% better) and the lower spacing is found by NSGA-II (64% better). Thus, we can conclude from our experiments that NSGA-II offers the larger coverage, whereas SPEA2 offers the better spread for large-scale explorations. However, the computational time needed to run SPEA2 is always larger.

7. Conclusions

New multimedia embedded applications are increasingly dynamic, and rely on DDTs to store their data. The selection of optimal DDT implementations for each variable in a particular target embedded system is a very time-consuming

process due to the large design space of possible DDTs implementations. In this paper we have proposed a framework to design optimized multimedia applications for a target embedded system. To validate our design flow, we have optimized two real-life multimedia applications using four exploration methods used in the literature (breadth-first, depth-first, branch&bound and VEGA). Moreover, we have included two elitist evolutionary algorithms to offer more alternatives to the system designer (NSGA-II and SPEA2). Comparing all the six algorithms, results show that evolutionary algorithms are able to find solutions 1584× faster that traditional heuristics. Comparing evolutionary algorithms, the obtained results show that VEGA is 30% quicker than the other two, but NSGA-II and SPEA2 achieve better results (75% on average) regarding covering of the design space of solutions. We have also automated all the design process. Thus, the designer only needs to select the set of variables to optimize in the application.

References

[1] J. L. Antonakos and K. C. Mansfield. *Practical Data Structures using C/C++*. Prentice Hall, 1999.

[2] D. Atienza, et al. Optimization of dynamic data structures in multimedia embedded systems using evolutionary computation. In *SCOPES '07: Proceedingsof the 10th international workshop on Software & compilers for embedded systems*, pages 31–40, New York, NY, USA, 2007. ACM.

[3] L. Benini and G. de Micheli. System-level power optimization: techniques and tools. *ACM Trans. Des. Autom. Electron. Syst.*, 5(2):115–192, 2000.

[4] F. Catthoor, et al. *Data access and storage management for embedded programmable processors*. Kluwer Academic Publishers, 2002.

[5] D. W. Corne, et al. Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In L. Spector, et al, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 283–290, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[6] E. G. Daylight, et al. 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. *Multiobjective Optimization using Evolutionary Algorithms*. John Wiley and Son Ltd., 2001.

[8] 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.

[9] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA 1993)*, pages 416–423, 1993.

[10] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Opt.*, 4:99–107, 1992.

[11] K. Hardee, et al. A 0.6v 205MHz 19.5ns tRC 16Mb embedded DRAM. In *IEEE International Solid-State Circuits Conference (ISSCC)*, 2004.

[12] J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1 of *IEEE World Congress on Computational Intelligence*, pages 82–87, 1994.

[13] L. Kharevych and R. Khan. 3D physics engine for elastic and deformable bodies. Available at http://www.cs.umd.edu/Honors/reports/kharevych.html, 2002. University of Maryland, College Park.

[14] Z. Michalewicz. *Genetic Algorithms + data structures = Evolution Programs*. Springer-Verlag, 1996.

[15] P. R. Panda, et al. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 6(2):149–206, 2001.

[16] S. R. Ranjithan, S. K. Chetan, and H. K. Dakshima. Constraint Method-Based Evolutionary Algorithm (CMEA) for Multiobjective Optimization. In E. Zitzler, et al, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 299–313. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[17] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, New Jersey, 1985.

[18] J. R. Schott. *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization*. PhD thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1995.

[19] 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.

[20] Sourceforge. Vdrift racing simulator. Available at http://sourceforge.net/projects/vdrift.

[21] S. Wuytack, F. Catthoor, and H. De Man. Transforming set data types to power optimal data structures. *IEEE Transactions on Computer-Aided Design*, 15(6):619–629, 1996.

[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.

[23] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computing*, 3(4):257–271, 1998.

[24] J. B. Zydallis, D. A. van Veldhuizen, and G. B. Lamont. A statistical comparison of multiobjective evolutionary algorithms including the momga-ii. In E. Zitzler, et al, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 226–240, 2001.