Soft Comput (2008) 12:1157–1167
DOI 10.1007/s00500-008-0295-y

FOCUS

A parallel evolutionary algorithm to optimize dynamic data types in embedded systems

José L. Risco-Martín · David Atienza · J. Ignacio Hidalgo · Juan Lanchares

Published online: 29 April 2008

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 searching space of possible DDTs implementations. That implies the minimization of embedded design variables (memory accesses, power consumption and memory usage). Up to know, some very effective heuristic algorithms have been developed in order to solve this problem, but it is unknown how good the selected DDTs are since the problem is NP-complete and cannot be fully explored. In these cases the use of parallel processing can be very useful because it allows not only to explore more solutions spending the same time, but also to implement new algorithms. This paper describes several parallel evolutionary algorithms for DDTs optimization in Embedded Systems, where parallelism improves the solutions found by the corresponding sequential algorithm, which indeed is quite effective compared with other previously proposed procedures. Experimental results show how a novel parallel multi-objective genetic algorithm, which combines NSGA-II and SPEA2, allows designers to reach a larger number of solutions than previous approximations.

J. L. Risco-Martín (✉) · D. Atienza · J. I. Hidalgo · J. Lanchares
DACYA/UCM, C/ Prof. José García Santesmases,
s/n, 28040 Madrid, Spain
e-mail: jlrisco@dacya.ucm.es

D. Atienza
e-mail: datienza@dacya.ucm.es

J. I. Hidalgo
e-mail: hidalgo@dacya.ucm.es

J. Lanchares
e-mail: julandan@dacya.ucm.es

1 Introduction

Latest multimedia embedded devices are enhancing its capabilities and are currently able to run applications reserved to powerful desktop computers few years ago (e.g., 3D games, video players) (Catthoor et al. 2002). 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 (Antonakos and Mansfield 1999) (dynamic arrays, linked lists, etc.) for each variable of the application, according to some specific constraints of the target device and typical embedded design metrics, such as memory accesses, memory usage and energy consumption (Atienza et al. 2007).

This task has been typically performed in the past using a pseudo-exhaustive evaluation of the design space of DDTs, including multiple executions of the application, to attain a *Pareto front (PF)* of solutions (Coello 1999), which tries to cover all the optimal implementation points for the some required design metrics. The construction of this PF has been proven to be a very time-consuming process, sometimes even unaffordable (Daylight et al. 2004).

Extensive work has been performed in the field of embedded memory subsystem optimization. Benini and de Micheli (2000) and Panda et al. (2001) presented two thorough surveys on static data and memory optimization techniques for embedded systems presented during the last years of the past century. More recently, in Catthoor et al. (2002) and Daylight et al. (2004), 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 (Wuytack et al. 1996; Panda et al. 2001). However, these approaches require the development of efficient pruning function costs and fully manual optimizations. In addition these works are not able to capture the evaluation of inter-dependencies of multiple DDTs implementations operating together, as the method proposed in this paper achieves by means of *evolutionary algorithms (EAs)* (Michalewicz 1996). Atienza et al. (2007) have already outlined the potential of EAs for dynamic memory optimizations. Nevertheless, their work only performed an initial analysis of one single type of EA and does not provide a complete analysis of tradeoffs between different technologies of sequential and parallel EAs, as we perform in this paper.

Also, according to the characteristics of certain parts of multimedia applications, several transformations for DDTs and design methodologies (Catthoor et al. 2002; Wuytack et al. 1996) have been proposed for static data profiling and optimization considering static memory access patterns to physical memories. In this context, the use of EA-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 (Panda et al. 2001). However, such techniques are not applicable to DDT implementations, due to the unpredictable nature at compile-time of the stored data.

In this paper we propose a novel framework to explore the design space of DDT implementation including a set of parallel procedures based on multi objective evolutionary algorithms (MOEAs) (Deb 2001). The development of parallel evolutionary algorithms for multi-objective problems involves the analysis of different paradigms for parallel processing and their corresponding parameters. In Van Veldhuizen et al. (2003) a generic formulation for *parallel multi-objective evolutionary algorithms (pMOEA)* is proposed and questions related with migration, replacement and niching schemes in the context of pMOEA are discussed. In Van Veldhuizen et al. (2003) four basic pMOEA based on the island paradigm are described: (1) islands execute the same MOEA (Xiong and Li 2003); (2) islands execute different MOEA (Fernandez et al. 2007); (3) each island evaluates a different subset of objective functions (Wilson and Moore 2005); and (4) each island considers a different region of the search domain (de Toro Negro et al. 2004). Taking into account this classification, our parallel design may be included in the second group. Since our migration policy is synchronous, we have combined two elitist evolutionary algorithms with different complexity, namely Strength Pareto Evolutionary Algorithm 2 (SPEA2) (Zitzler et al. 2002) and Non-dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al. 2002), implementing three variations of a pMOEA. SPEA2 is $O(N^3)$ and NSGA-II is $O(mN^2)$, where N is the population size and m is the number of objectives.

Our experiments in a real-life dynamic embedded application show that: (1) NSGA-II and SPEA2 reach important speed-ups (up to 955x faster) with respect to other traditional heuristics; (2) the parallel algorithm can achieve significant speed-ups (68% faster) with respect to the sequential versions in a multi-core architecture. Moreover, we compare the sequential and parallel approaches by means of multiple metrics, showing that the quality of the solutions is improved by the combination of NSGA-II and SPEA2 in a parallel implementation; and (3) such combination is executed on a cluster of 16 workstations of two cores each, where several population sizes are used in our experiments. The experimental results obtained are very promising. In particular, we show that if we increase the size of the population, the performance of the pMOEA improves as we increase the number of clusters used.

The rest of the paper is organized as follows. In Sect. 2 the Dynamic Data Types optimization problem is explained. In Sect. 3, we present our multi-objective optimization process. A description of the MOEAs, including an explanation of our parallel proposal, which combines NSGA-II and SPEA2 algorithms, is also detailed. Section 4 shows some performance and quality metrics used in our experiments discussed in Sect. 5. Finally, in Sect. 6 we summarize the main conclusions of this paper.

2 The DDTs exploration problem

Dynamic data types are software abstractions by means of which we can manipulate and access data. The implementation of a DDT has two main effects on the performance of an application. First, it involves 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 shows an example of DDTs exploration. The initial code contains two variables, $v1$ and $v2$, instantiated as a vector and a list, respectively. After the exploration process, we can obtain for example a candidate solution that recommends $v1$ to be instantiated as *single linked list (SLL)* and $v2$ as *double linked list of arrays (DLLAR)*.

More generally we can state that the application to optimize contains a set of variables $\boldsymbol{V}$, which are candidates

```
vector<T1>* v1 = new vector<T1>();
// …
list<T2>* v2 = new list<T2>();
list<T2>::iterator itr = v2->begin();
for(; itr!=v2->end(); ++itr)
  cout << *itr;
// …
```

Initial application code

```
SLL<T1>* v1 = new SLL<T1>();
// …
DLLAR<T2>* v2 = new DLLAR<T2>();
DLLAR<T2>::iterator itr = v2->begin();
for(; itr!=v2->end(); ++itr)
  cout << *itr;
// …
```

Optimized application code

Fig. 1 Code before and after the exploration of DDT

Table 1 DDT library

Representation	DDT	Description
1	AR	Array
2	AR(P)	Array of pointers
3	SLL	Single-linked list
4	DLL	Doubly-linked list
5	SLL(O)	Single-linked list with roving pointer
6	DLL(O)	Doubly-linked list with roving pointer
7	SLL(AR)	Single-linked list of arrays
8	DLL(AR)	Doubly-linked list of arrays
9	SLL(ARO)	Single-linked list of arrays and roving pointer
10	DLL(ARO)	Doubly-linked list of arrays and roving pointer

to be instantiated as a certain DDT from the set of possible implementation of DDTs library $\boldsymbol{D}$ presented in Atienza et al. (2007) and Daylight et al. (2004). Thus, the goal of our optimization flow is to obtain a set of pairs (variable, DDT) $\{v_i \in \boldsymbol{V}, d_j \in \boldsymbol{D}\}$, such that minimizes memory accesses, memory usage and power consumption for the target embedded system. Additional constraints as the minimum and maximum values for all three objectives may be defined. Clearly, this is a multi-objective optimization problem.

To measure the quality of a solution, we have defined 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. 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 (Atienza et al. 2007).

Once we have fixed the problem optimization process for DDTs, we can describe the whole process shown in Fig. 2. It has three main steps: profiling of the application, estimation of the parameters and multi-objective optimization algorithms execution. These three steps are described in the next paragraphs.

2.1 Profiling

In order to evaluate the different metrics we need to obtain the real execution information from the application. Unfortunately, the execution of the whole application is not a viable solution. An alternative good solution recently proposed Daylight et al. (2004) is to obtain a profiling report of the application where the following information is logged: number and location of the accesses 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 need to replace all the candidate variables in the application by our vector DDT implementation, which logs all the information needed to evaluate them the using equations developed in Atienza et al. (2007).

2.2 Parameters estimation

In this phase, we extract all information needed from the profiling report. The purpose is to measure the quality of a solution (v_i, d_j) in the DDT exploration, using several parameters, namely, the number of candidate variables, number of elements stored in the DDT (the worst case), average of the number of elements stored, size of the elements (in bytes), size of the pointers (in bytes), number of read accesses, number of write accesses and cache misses. All these parameters can be extracted from the profiling report. To this end, we have developed a tool called *Profile Analyzer*. Cache misses are also obtained by means of simulation, generating memory traces from the profiling report and the DDT library, using them as input for the *Dinero IV* cache simulator (Edler and Hill 2007) for the particular memory configuration of the target embedded system. This phase is the most-time consuming part of the exploration, although it is done only once for each target architecture, and for each tested applications.

2.3 Optimization

The last phase is the optimization process and this is where MOEAs work. They take as input the information previously estimated and try to minimize three different objectives: memory accesses, memory usage and energy consumption. To this end, we have implemented sequential versions of two elitist MOEAs, called SPEA2 and NSGA-II and three pMOEAs, which combine SPEA2 and NSGA-II. A detailed explanation is presented in the next section.

When the three phases of the optimization process end, we obtain a set of DDT instantiation policies, i.e., which variable should be instantiated by which DDT. We also obtain

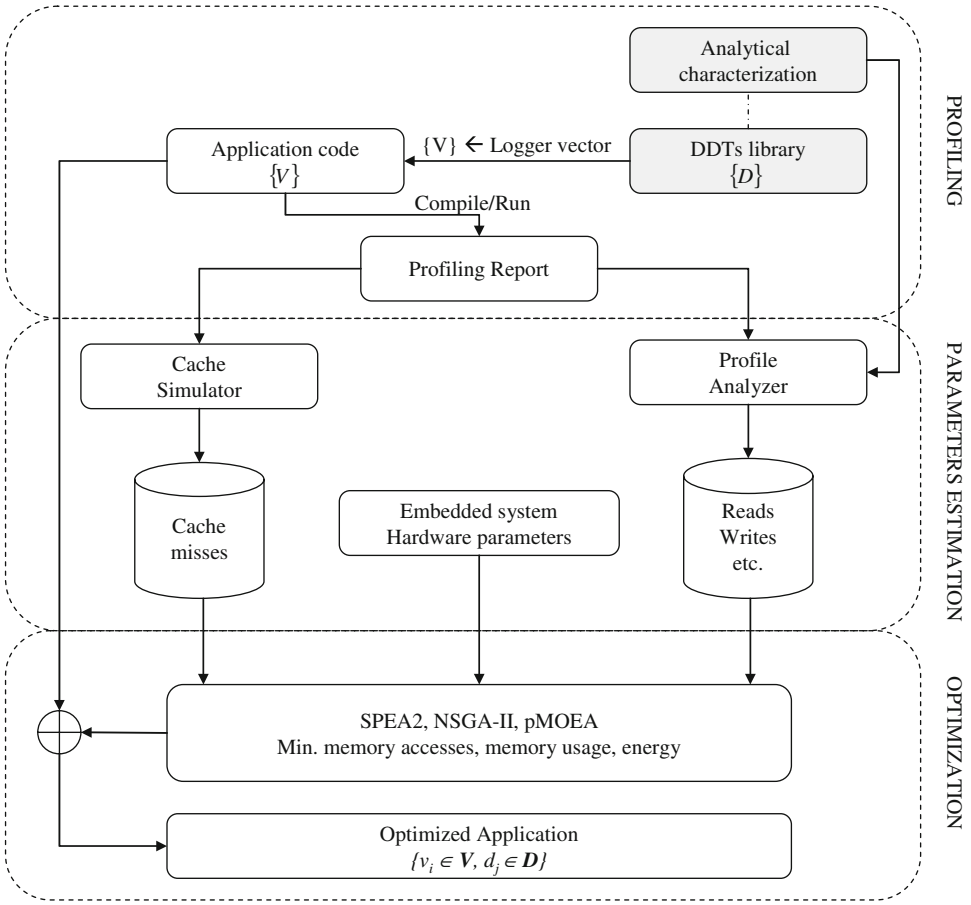

Fig. 2 DDTs optimization flow

the gain on memory accesses, memory usage and energy consumption.

3 Evolutionary algorithms

In this Section we describe the implementation of the two sequential MOEAs, which are capable of finding a solution to the DDTs exploration problem in a moderate time. In the next section, we report several parallel versions that combine these two algorithms.

Multi-objective optimization aims at simultaneously optimizing several objectives sometimes contradictory (memory accesses, memory usage and energy consumption for our problem). For such kind of problems, it does not exist a single optimal solution, and some compromises have to be made. Without any loss of generality, we can assume the following N-objective minimization problem:

$$\begin{aligned} &Minimize\ \mathbf{z} = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_N(\mathbf{x})) \\ &subject\ \text{to} \qquad\qquad \mathbf{x} \in \mathbf{X} \end{aligned} \tag{1}$$

where $\boldsymbol{z}$ is the objective vector with N objectives to be minimized, $\boldsymbol{x}$ is the decision vector, and $\boldsymbol{X}$ is the feasible region in the decision space. A solution $\boldsymbol{x} \in \boldsymbol{X}$ is said to dominate another solution $\boldsymbol{y} \in \boldsymbol{X}$ if the following two conditions are satisfied:

$$\begin{aligned} &\forall i \in \{1, 2, \ldots, N\}, f_i(\mathbf{x}) \leq f_i(\mathbf{y}) \\ &\exists i \in \{1, 2, \ldots, N\}, f_i(\mathbf{x}) < f_i(\mathbf{y}) \end{aligned} \tag{2}$$

If there is no solution which dominates $\boldsymbol{x} \in \boldsymbol{X}$, $\boldsymbol{x}$ is said to be a *Pareto optimal solution (POS)*. The set of all elements of the search space that are not dominated by any other element is called the *Pareto optimal front (POF)* of the multi-objective problem: it represents the best possible compromises with respect to the contradictory objectives. In both algorithms, the sequential and parallel versions, we attempt to reach the higher number of solutions of the Pareto front as possible.

Nowadays, many MOEAs have been developed. They can be classified into two broad categories: non-elitist and elitist also called first and second generation MOEAs (Coello 1999). In the elitist approach, EAs 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 (Corne et al. 2001), MOMGA-II (Zydallis et al. 2001), NSGA-II and SPEA2 are examples of this category. In contrast, the non-elitist approach does not guarantee preserving the set of

Table 2 Common evolutionary algorithm framework

1. Initialize the Population **P**
2. (elitist EAs) Select elitist solutions from **P** to create external set **EP**
3. Create mating pool from one or both **P** and **EP**
4. Reproduction based on the pool to create the next generation **P** using evolutionary operators
5. (elitist EAs) Combine **EP** into **P**
6. Go to step 2 if the terminated condition is not satisfied

best individuals for the next generation (Zitzler and Thiele 1999). Examples of this category include MOGA (Fonseca and Fleming 1993), HLGA (Hajela and Lin 1992), NPGA (Horn et al. 1994) and VEGA (Schaffer 1985).

When implementing a MOEA, the designer has to overcome two major problems (Zitzler and Thiele 1999). The first problem is how to get close to the Pareto optimal front (POF) (Deb 2001). The second problem is how to keep diversity among the solutions in the obtained set. These two problems become common criteria for most current algorithmic performance comparisons and they will be used in the experimental results section.

Although all the cited MOEAs are different from each other, we can find some common steps in these algorithms, which are summarized in Table 2. As we have already mentioned, two representative elitist algorithms, namely, SPEA2 and NSGA-II were selected (Zitzler et al. 2002; Deb et al. 2002).

3.1 SPEA2

SPEA2 is the second version of the elitist MOEA called *strength Pareto evolutionary algorithm* (Zitzler and Thiele 1999). This section just concentrates on the main points of SPEA2 (Zitzler et al. 2002). The initial population, representation and evolutionary operators are standard: uniform distribution, binary representation, binary tournament selection, single-point crossover, and bit-flip mutation. However, the real power of SPEA2 lies in the elitist-preserved operation.

An external set (archive) is created for storing primarily non-dominated solutions. It is then combined with the current population to form the next archive that is then used to create off-spring for the next generation. The size of the archive is fixed. It can be set to equal to the population size. Therefore, there exist two special situations when filling solutions in the archive. The first situation occurs when the number of non-dominated solutions is smaller than the archive size. In this case, other dominated solutions taken from the remainder part of the population are filled in. The selection is carried out according to a fitness value, specifically defined for SPEA. That individual fitness value defined for a solution $\boldsymbol{x}$, is the total of the SPEA-defined strengths of solutions which dominate $\boldsymbol{x}$, plus a density value. The second situation happens when the number of non-dominated solutions is over the archive size. In this case, a truncation operator is applied, such that, the solution with the smallest distance to the other solutions is removed from the set. If solutions have the same minimum distance, the second nearest distance is considered, and so forth. This is called the kth *nearest distance rule*. For a deeper explanation of details about the distance concept we refer the reader to Zitzler et al. (2002).

3.2 NSGA-II

As SPEA2, NSGA-II is also an elitist algorithm (Deb et al. 2002). The differences of NSGA-II from SPEA2 are mainly because of the elitist-preservation operation. In the case of NSGA-II, the archive size is set equal to the initial population size. The current archive is then determined based on the combination of the current population and the previous archive. Thus, NSGA-II uses dominance ranking to classify the population into a number of layers, such that the first layer is the best layer (non dominated solutions) in the population. The archive is created based on the order of ranking layers: the best rank being selected first. Moreover, if the number of individuals in the archive is smaller than the population size, the next layer (non dominated solutions excluding those solutions on the first layer) is taken into account and so on. If adding a layer would increase the number of individuals in the archive to exceed the initial population size, a truncation operator is applied to that layer based on the crowding distance. The crowding distance of a solution $\boldsymbol{x}$ is defined as the average objective-value differences between two adjacent solutions of solution $\boldsymbol{x}$, where the population is sorted according to each objective to find adjacent solutions and where also boundary solutions have infinite values. The truncation operator removes the individual with the smallest crowding distance. An offspring population of the same size as the initial population is then created from the archive by using crowded tournament selection, crossover, and mutation operators. The crowded tournament selection rule selects, as winner of two same-rank solutions, the one with the greater crowding distance value.

3.3 Encoding a solution

In order to apply our 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 (Deb 2001).

Table 3 Example of an individual

D	1	2	4	...	2
V	v_1	v_2	v_3	...	v_n

An individual is represented by a chromosome containing n genes, where n is the number of the variables logged in the application (i.e., the number of variables which need a DDT assignation), $n = size(\boldsymbol{V})$. Table 3 shows the representation of an individual, genes are represented in the first row (gray shaded cells). Each gene represents the DDT that should be used to instantiate the corresponding variable in the application from Table 1. In our example, the second variable $v_2 \in \boldsymbol{V}$ will be instantiated by $d_2 \in \boldsymbol{D}$, an array of pointers if we use the order of Table 1. We use an integer to represent the values of a gene, and the constraint a chromosome must satisfy is:

$$1 \leq d_i \leq size\,(\mathbf{D}) \tag{3}$$

Consequently, if an application contains n variables, each individual has to be constituted by n integer fields. The advantages of this representation are simplicity, as DDTs can be directly obtained from Table 1, and safety, since all genetic operators will always produce correct and real solutions.

3.4 Evaluation of the solutions

Once the solutions are encoded we need to evaluate the solutions represented by individual. Note that on the second phase we made a profiling of the real application for a realistic input set. After that step, all the information required for the analytical characterization of the DDTs implementations considered is available. Using equations from Atienza et al. (2007) and the number of read (*Nr*) and write (*Nw*) accesses to each DDT during execution (supported by the profiling) for each DDT implementation, we can compute the number of accesses (*NumAcess)* to layers of the memory hierarchy, memory footprint (*AvMem* in Bytes) and energy values (*Energy* in nJ). In this work we consider a basic memory hierarchy that consists of a main shared memory and a L1 data-cache. Other memory hierarchies could be modeled as well by modifying the cited equations.

Regarding the energy calculations, we assume the use of in-place sharing, as the DDTs lifetimes are short. Energy is computed as next equation indicates:

$$\begin{aligned} Energy = {} & (Npa * Epa) + (Nrw * Erw) \\ & + (AvMem * Eest) \end{aligned}$$

where *Nrw* is the number of reads/writes to the L1 data-cache memory, *Npa* is the number of misses in the data-cache. *Epa* is the energy consumed per access to main memory, *Erw* is the energy consumption per access to the cache, and *Eest* is the static energy consumed by the main memory. We assume (according to empirical validations) an average miss rate of the cache memory below 5% of the overall memory accesses in our energy calculations. However, this value is user-configurable in our MOEA-based exploration process and even additional multi-level cache miss rate effects can be configured. In addition, it is possible to introduce some constraints and weights for the metrics to be optimized.

3.5 Parallel implementation

In this section we describe how to use two sequential MOEAs in a parallel environment to solve the exploration of DDTs in embedded applications described in Sect. 1. The search process could be improved by using several threads to apply the operators in a larger number of individuals. We propose a coarse-grained pMOEA where each thread runs a different population, using an island model. The number of individuals is the same as in the sequential version. Figure 3 provides a scheme of the parallel procedure with two islands. The specific MOEA (NSGA-II or SPEA2) is applied to each subpopulation separately, and the best partial results are periodically sent from one subpopulation to its neighbor on a ring communication topology (Cantú-Paz 2000). As in most of the pMOEAs, migration from one subpopulation to another is controlled by several parameters specified at the beginning of the execution and remains unchanged, as (a) the topology defined by the connections between subpopulations; (b) a migration rate that controls how many individuals migrate, in our case the best individual; and (c) a migration interval that determines the migration frequency, every 100 generations. The best individual is selected in the following way. First, we extract the set of non-dominated solutions in the current population. Second, we sort the resulting set with respect to one random objective, from where we extract the first individual. Moreover, since NSGA-II is faster than SPEA2 ($O(mN^2)$ versus $O(N^3)$, where N is the population size and m is the number of objectives), NSGA-II could finish while SPEA2 is still exploring early generations. Thus, as Fig. 3 depicts, our migration policy is synchronized every 100 generations.

Figure 4 shows the general structure of the pMOEA. We have implemented three variations to be tested in a multicore architecture. They differ on which MOEA algorithm is controlling the subpopulation, i.e., running on each thread.

(1) NS^2 configuration: Four islands executing NSGA-II, SPEA2, NSGA-II and SPEA2.
(2) S^4 configuration: Four islands, but running all of them SPEA2 algorithm
(3) N^4 configuration: Four islands using the NSGA-II algorithm.

The fitness function, the operators, and the stop criterion are the same as in the sequential version.

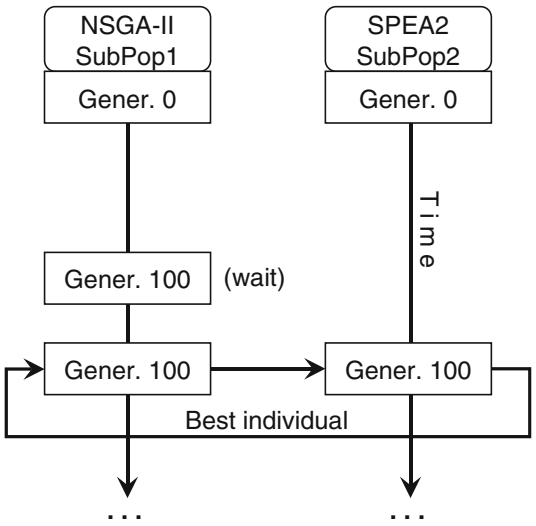

Fig. 3 A graphic representation of the coarse grained model

```
Begin
  create N subpopulations
  for each (subpopulation)
    assign algorithm to subpopulation ∈ {SPEA2, NSGA-II}
    assign neighbour to subpopulation
    Run(subpopulation) in a separate thread
End

Run(subpop)
  apply genetic operators
  if (currentGeneration mod 100 == 0)
    select best individual
    Send(subpop, individual, neighbour)
End

Send(subpop, ind, neighbour)
  while(subpop.currentGeneration>neighbour.currentGeneration)
    wait
  neighbour.receive(ind)
End
```

Fig. 4 Parallel multi-objective evolutionary algorithm for DDTs exploration

The algorithm shown in Fig. 4 follows a multi-threaded design, which is suitable to be executed in multi-core architectures. Another approach we have implemented consists of executing our proposed pMOEA in a cluster of workstations as well as PCs connected over a LAN. To this end, we have executed up to 32 islands on a cluster of 16 workstations of two cores each, using the proposed DEVS/SOA framework in Mittal et al. (2007). The algorithm is exactly the same, but each workstation executes two threads. Individuals are sent between different workstations using web services. Further details on how DEVS/SOA works are out of the scope of this paper.

4 Performance and quality metrics

We compare the obtained sets of non-dominated solutions by means of three criteria explained in this section: Coverage, Spread and Spacing.

To compare the performance and quality of the different MOEAs, we need to evaluate the obtained set of non-dominated solutions in terms of the following two aspects:

- Convergence to POF.
- Diversity on POF.

As this is an NP-hard problem, we cannot know the exact set of the POF, and we must evaluate the algorithms by means of a comparison of the non-dominated sets obtained by each algorithm. Thus, we use the coverage metric (Zitzler and Thiele 1999) to measure the convergence.

Let F', F'' be two sets of non-dominated solutions. The coverage metric can be defined as follows:

$$C\left(F', F''\right) = \frac{\left|p'' \in F''; \exists p' \in F' : p' \succeq p''\right|}{|F''|} \tag{4}$$

The value $C(F',F'')=1$ means that all points in F'' are dominated by or equal to points in F'. On the other hand, $C(F', F'') = 0$ means that no solutions in F'' are covered by the set F'. It is noted that both $C(F', F'')$ and $C(F'', F')$, has to be considered, since $C(F',F'')$ is not necessary equal to $C(F'', F')$. If $C(F', F'') > C(F'', F')$, the rate of dominated solutions in F'' is higher than that in F'.

Then, to evaluate the diversity of the obtained solution set, we employed two criteria (Deb 2001): spread D and spacing S. On one hand, D is calculated as the length of the diagonal line of the minimum N-dimensional hyper rectangle that includes the solution set:

$$D = \sqrt{\sum_{i=1}^{n} \left(\max_{j=1}^{|F|} f_i\left(x_j\right) - \min_{j=1}^{|F|} f_i\left(x_j\right)\right)^2} \qquad x_j \in F, j = 1, 2, \ldots, |F| \tag{5}$$

On the other hand Spacing (S), which is another measure of the diversity, is calculated as a standard deviation of the distance to the neighboring solutions in the N-objective space, as follows:

$$S = \sqrt{\frac{1}{|F|} \sum_{j=1}^{|F|} \left(d_j - \bar{d}\right)^2} \qquad d_j = \min_{x_k \in F \wedge k \neq j} \sum_{i=1}^{N} \left|f_i\left(x_j\right) - f_i\left(x_k\right)\right| \tag{6}$$

5 Experimental results

In this section we describe the complete method applied to compare the different types of sequential and parallel

Table 4 Parameters for both sequential algorithms

Parameter	Value
Population size	200
Number of generations	8,000
Probability of crossover	0.80
Probability of mutation	0.01

MOEAs proposed while optimizing one real-life dynamic embedded application. We have evaluated the proposed optimization framework for a 3D Physics Engine for elastic and deformable bodies (Kharevych and Khan 2002). For this application we logged 3128 variables and the 10 possible DDTs contained in Table 1, which can cover all the real-life embedded multimedia applications we are aware of. To analyze the effect of sequential and distributed algorithms on embedded system's memory accesses, memory usage and energy consumption, we utilized processor energy from Catthoor et al. (2002), and the access time and energy values for caches of 32 kB and embedded 16 MB DRAM main memory from Shivakumar and Jouppi (2006) and Hardee et al. (2004), respectively.

To compare the performance of both sequential and parallel algorithms, all parameters are set to the same values. After different tests, we have fixed them to the values indicated in Table 4. The external archive size (where non dominated solutions are stored) is set to be equal to the initial population. The crossover and mutation probabilities are the same that in the sequential algorithms. The population size is set to 200 for each island, and the number of generations is set to 2,000.

Next, we summarize the results obtained by the sequential and parallel evolutionary algorithms. All the algorithms have been implemented using Java 6. The multi-threaded version is based on a multi-threaded Java application which will utilize multiple processors when available. The distributed version is made by adding the DEVS/SOA framework proposed in Mittal et al. (2007). The experiments have been made using three platforms: (1) AMD Sempron 3600+ 2 GHz with 1 GB DDR memory, (2) Intel Core 2 CPU 6600 2.40 GHz with 2 GB DDR memory, and (3) cluster of 16 workstations AMD Opteron Dual Core 2 GHz with 4 GB DDR memory.

5.1 Sequential architecture

We have tested the DDTs exploration speed in comparison to different alternative methods for a 3D Physics Engine application on the AMD Sempron 3600+ 2 GHz with 1 GB DDR memory. The results obtained for the different tested exploration methods are shown in Table 5. We have compared our algorithms with state-of-the-art pruning and optimization methods for DDT implementations presented in Daylight et al. (2004), Wuytack et al. (1996). In these cases breadth-first, deep-first and branch and bound exploration heuristics are used to minimize overall memory access, memory usage and energy consumption in embedded multimedia applications. In this context, we have used a weighted sum of the three objectives as the fitness function for these three algorithms. Since there are $10^{3,128}$ feasible solutions (10 DDTs for 3,128 variables) it is unfeasible to reach the complete POF by means of exhaustive exploration. The results in Table 5 outline that the exploration process with our method (using NSGA-II and SPEA2) is much faster than using directly the implementations of DDTs and other heuristics, namely, 954 $\times$ faster.

Table 5 Comparison between the proposed sequential algorithms and other techniques

DDT exploration method	Time (AMD Sempron)
Breadth-first	18.14×10^6 s.
Deep-first	36.00×10^3 s.
Branch & Bound	25.20×10^3 s.
VEGA [2]	10.80×10^3 s.
NSGA-II	1.90×10^3 s.
SPEA2	3.03×10^3 s

5.2 Multi-core architecture

We have also explored DDTs with each of the five algorithms proposed (i.e., SPEA2, NSGA-II, N^4, NS^2 and S^4) on an Intel Core 2 CPU 6600 2.40 GHz with 2 GB DDR memory. The coverage, spread and the spacing values are calculated by averaging results of 100 trials.

Figure 5 depicts the number of non-dominated individuals obtained in the best population. NSGA-II offers the same non-dominated individuals than SPEA2. NS^2 offers 64% more optimal solutions than both NSGA-II and SPEA2, 29% more than S^4 and 27% more than N^4.

Regarding convergence comparisons, Table 6 shows that the coverage values of NS^2 are better than any other algorithm. For example, $C(NS^2, \text{NSGA-II}) > C(\text{NSGA-II}, NS^2)$ is $0.083 > 0.016$ or $C(NS^2, N^4) > C(N^4, NS^2)$ is $0.384 > 0.153$. Thus, NS^2 offers more optimal alternatives to the system designer for the implementation of the final embedded application.

Similar results are obtained using the average spread and spacing metrics. Table 7 indicates that the lower spread is found by parallel algorithms in all cases. Note that a larger value of the spread does not always mean desirable since the aim of the problem is the minimization. Thus, the larger

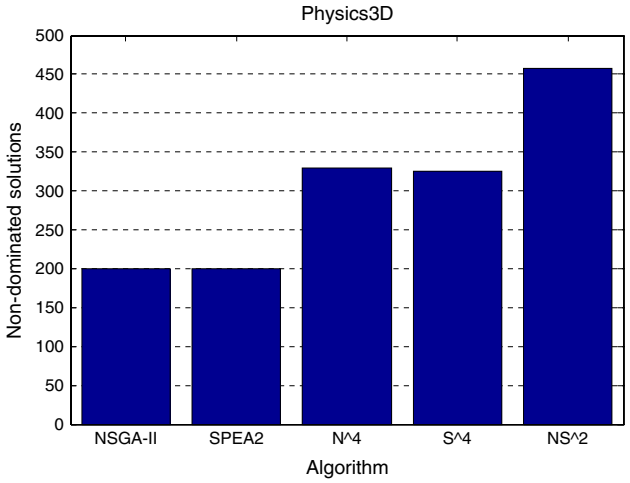

Fig. 5 Non-dominated individuals obtained by NSGA-II, SPEA2, N^4, S^4 and NS^2

Table 6 Coverage metric

	NSGA-II	SPEA2	N^4	NS^2	S^4
NSGA-II	–	0.065	0.020	0.016	0.023
SPEA2	0.045	–	0.010	0.001	0.017
N^4	0.071	0.139	–	0.153	0.267
NS^2	0.083	0.152	0.384	–	0.516
S^4	0.030	0.061	0.100	0.227	–

Table 7 Spread and spacing for the five algorithms

	NSGA-II	SPEA2	N^4	NS^2	S^4
Spread	0.142	0.188	0.112	0.127	0.136
Spacing	0.002	0.002	0.001	0.001	0.001

Table 8 Comparison between our sequential and distributed algorithms

	AMD Sempron	Intel Core 2
NSGA-II	1900.279 s.	712.236 s.
SPEA2	3026.896 s.	1328.312 s.
N^4	983.183 s.	421.77 s.
NS^2	1186.44 s.	546.682 s.
S^4	1701.113 s.	707.063 s.

spread can be calculated if the obtained set is far from the POF.

Table 8 shows the comparisons between the execution times of both sequential and parallel algorithms. The left column contains the execution time using 1 processor. The column on the right shows the same results, but using 2 processors. In the best case, we obtain an execution time of 422 s. for N^4 and two processors and 3027 s. for SPEA2 and one processor, giving a speed-up of 86%.

For comparison reasons we present Fig. 6 to illustrate the optimization process that our methodology performs. In this test, 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. Thus, in the end, compared to the combination proposed by our five algorithms, the figure shows the achieved level of optimization and final gains after applying the proposed optimization flow in Fig. 2. Furthermore, as this figure indicates, NS^2 offered the best compromise among objectives.

5.3 Multi-core/distributed architecture

Finally, the NS^K configuration was distributed on a cluster of 16 workstations AMD Opteron Dual Core 270 2 GHz with 4 GB DDR memory. To this end, we placed two threads per workstation and the communication among workstations was made through a DEVS/SOA framework (Mittal et al. 2007).

We tested our algorithm using from 1 to 16 workstations, which leads to 2, 4, 6, …, 32 islands running in parallel, namely NS^1, NS^2, NS^3, …, NS^{16}, and different population sizes (256, 512, 1,024, and 2,048) changing only the number of islands in order to observe and study the increase in performance (speedup). In all these cases the number of generations was set to 8,000. The external archive size of each island was set to the entire population size, i.e., 256, 512, 1,024 and 2,048.

In light of the results presented in Fig. 7a, it is possible to observe that, as the size of the population increased, the performance of the parallel version improved proportionally to the number of islands. Also, Fig. 7b indicates that the number of non-dominated individuals increased as the number of islands increased, especially for large populations.

This shows that the proposed pMOEA is better suited for large populations. It is also worthwhile to mention that with small populations a parallel and distributed version of a genetic algorithm is most likely to converge to a local minimum due to a small gene pool.

6 Conclusions and future work

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 studied several MOEAs to solve this problem. Particularly, we have proposed a new parallel algorithm (NS^K) which combines in a novel manner two widely used MOEAs. The problem is formulated as a multi-objective combinatorial optimization problem, for

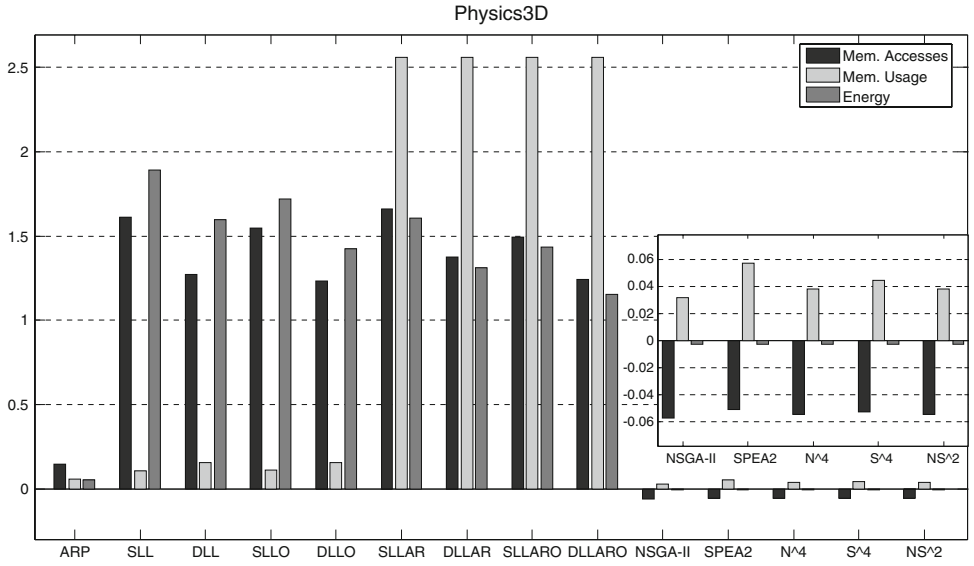

Fig. 6 Overall results for different design metrics coming from various sets of DDT implementations (logarithmic scale)

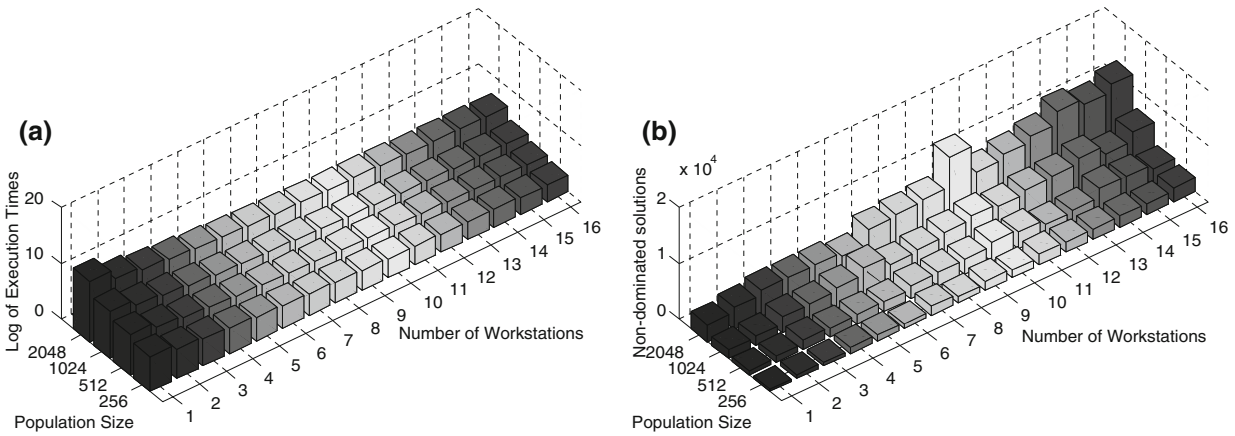

Fig. 7 Execution times (**a**) and non-dominated solutions (**b**) as a function of the number of workstations. Each workstation executes two islands

which we used three objective functions: memory accesses, memory usage and energy consumption. The results obtained shows that this parallel approach performs very well. In fact, the NS^K reaches more optimal solutions than the other sequential and parallel algorithm, obtaining a speed-up of 86% with respect to the non parallel implementation.

We have also executed NS^K in a cluster of 16 workstations of two cores each. Our results show that if the size of the population is increased, the performance of the parallel version improves proportionally with respect to the number of available islands.

As a result, we can conclude that not only parallel implementations improve the speed of the optimization process, but also the quality and the variety of the solutions, especially for large populations.

Future work includes the development of dynamic control parameters, such as, the topology, and a deeper study of migration rates and frequency.

We are also working on exploring other alternatives with new combinations of different MOEAs to those used in this paper.

Acknowledgments This work has been supported by Spanish Government grants CICYT TIN2005-5619 and MEC Consolider Ingenio CSD00C-07-20811 of the Spanish Council of Science and Technology. We also thank the anonymous reviewers for their helpful and constructive comments.

References

Antonakos JL, Mansfield KC (1999) Practical data structures using C/C++. Prentice Hall, Englewood Cliffs

Atienza D, Baloukas C, Papadopoulos L, Poucet C, Mamagkakis S, Hidalgo JI, Catthoor F, Soudris D, Lanchares J (2007) 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. ACM, New York, pp 31–40

Benini L, Micheli Gde (2000) System level power optimization techniques and tools. ACM Trans Des Auto Electr Syst 5(2):115–192

Cantú-Paz E (2000) Efficient and accurate parallel genetic algorithms. Kluwer, Dordrecht

Catthoor F et al (2002) Data access and storage management for embedded programmable processors. Kluwer, Dordrecht

Coello C (1999) A Comparative Survey of Evolutionary-based Multiobjective Optimization Techniques. Knowl Inf Syst 1:269–308

Corne DW, Jerram NR, Knowles JD, Oates MJ (2001) PESA-II: region-based selection in evolutionary multiobjective optimization. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2001). Morgan Kaufmann Publishers, San Francisco, pp 283–290

Daylight EG, Atienza D, Vandecappelle A, Catthoor F, Mendias JM (2004) Memory-access-aware data structure transformations for embedded software with dynamic data accesses. IEEE Trans Very Large Scale Integr (VLSI) Syst 12(3):269–280

Deb K (2001) Multiobjective optimization using evolutionary algorithms. Wiley, New York

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evolut Comput 6(2):182–197

Toro Negro Fde , Ortega J, Ros E, Mota S, Paechter B, Martín J (2004) PSFGA: Parallel Processing and Evolutionary Computation for Multiobjective Optimisation. Parallel Comput 30:721–739

Edler J, Hill MD (2007) Dinero IV trace-driven uniprocessor cache simulator. http://pages.cs.wisc.edu/~markhill/DineroIV

Fernandez JM, Vila P, Calle E, Marzo JL (2007) Design of Virtual topologies using the elitist team of multiobjective evolutionary algorithms. In: Proceedings of international symposium on performance evaluation of computer and telecommunication systems (SPECTS'07), pp 266–271

Fonseca CM, Fleming PJ (1993) Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: Forrest S (ed) Proceedings of the fifth international conference on genetic algorithms, San Mateo, University of Illinois at Urbana-Champaign. Morgan Kauffman Publishers, San Francisco, pp 416–423

Hajela P, Lin CY (1992) Genetic search strategies in multicriterion optimal design. Struct Opt 4:99–107

Hardee K et al (2004) A 0.6v 205MHz 19.5ns tRC 16Mb embedded DRAM. In: IEEE international solid-state circuits conference (ISSCC)

Horn J, Nafpliotis N, Goldberg DE (1994) A niched Pareto genetic algorithm for multiobjective optimization. In: Proceedings of the first IEEE conference on evolutionary computation, IEEE World congress on computational intelligence, vol 1. IEEE Press, Piscataway, pp 82–87

Kharevych L, Khan R (2002) 3D Physics engine for elastic and deformable bodies http://www.cs.umd.edu/Honors/reports/kharevych.html, 2002

Michalewicz Z (1996) Genetic algorithms + data structures = evolution programs. Springer, Heidelberg

Mittal S, Risco-Martin JL, Zeigler BP (2007) DEVS-based simulation web services for net-centric T&E, In: Proceedings of the Summer computer simulation conference (SCSC 2007)

Panda PR et al (2001) Data and memory optimization techniques for embedded systems. ACM Trans Des Autom Electron Syst 6(2):149–206

Schaffer JD (1985) Multiple objective optimization with vector evaluated genetic algorithms. In: Proceedings of 1st international conference genetic algorithms, pp 99–100

Shivakumar P, Jouppi NP (2006) Cacti 4.0: integrated cache timing power-area model. Tech. Report, HP Labs

Van Veldhuizen DA, Zydallis JB, Lamont GB (2003) Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Trans Evolut Comput 7(2):144–173

Xiong S, Li F (2003) Parallel strength pareto multi-objective evolutionary algorithm for optimization problems. In: Proceedings of the 2003 congress on evolutionary computation (CEC'2003), vol 4. IEEE Press, New York, pp 2712–2718

Wilson L, Moore M (2005) Cross-pollinating parallel genetic algorithms for multiobjective search and optimization. Int J Found Comput Sci 16(2):261–280

Wuytack S, Catthoor F, Man HD (1996) Transforming set data types to power optimal data structures. IEEE Trans Comput Aided Des 15:619–629

Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evolut Comput 3(4):257–271

Zitzler E, Laumanns M, Thiele L (2002) 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, pp 95–100

Zydallis JB, Van Veldhuizen DA, Lamont GB (2001) A statistical comparison of multiobjective evolutionary algorithms including the MOMGA-II. In: First international conference on evolutionary multi-criterion optimization, pp 226–240