Parallel and Distributed Optimization of Dynamic Data Structures for Multimedia Embedded Systems

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

Abstract. Energy-efficient design of multimedia embedded systems demands optimizations in both hardware and software. Software optimization has no received much attention, although modern multimedia applications exhibit high resource utilization. In order to efficiently run this kind of applications in embedded systems, the dynamic memory subsystem needs to be optimized. A key role in this optimization is played by the Dynamic Data Types (DDTs) that reside in every real-life application. It would be desirable to organize this set of DDTs to achieve the best performance in the target embedded system. This 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. In this work, we propose a method that uses parallel processing and evolutionary computation to explore DDTs in the design of embedded applications. We propose a parallel *Multi-Objective Evolutionary Algorithm (MOEA)* which combines NSGA-II and SPEA2. We use *Discrete Event Systems Specification (DEVS)* to implement this parallel evolutionary algorithm over *Service Oriented Architecture (SOA)*. Parallelism improves the solutions found by the corresponding sequential algorithms, and it allows system designers to reach better solutions than previous approximations.

José L. Risco-Martín · J. Ignacio Hidalgo · Juan Lanchares
Dept. of Computer Architecture and Automation (DACYA), Complutense University of Madrid (UCM), C/Prof. José García Santesmases, s/n., 28040 Madrid, Spain
e-mail: {jlrisco,hidalgo,julandan}@dacya.ucm.es

David Atienza
Dept. of Computer Architecture and Automation (DACYA), Complutense University of Madrid (UCM), C/Prof. José García Santesmases, s/n., 28040 Madrid, Spain
e-mail: datienza@dacya.ucm.es
and
Embedded Systems Laboratory (ESL), Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

F. Fernández de Vega, E. Cantú-Paz (Eds.): Paral. and Distrib. Comp. Intel., SCI 269, pp. 263–290.
springerlink.com

1 Introduction

Latest multimedia embedded systems typically require reliable and powerful computing, superior graphical performance, multiple I/O configurations and long product life support. Currently, these systems are able to run applications initially designed for high performance desktop computers [6], which having large run-time memory management requirements, need to be mapped onto an extremely compact device. However, embedded systems struggle to execute these complex applications because they hold very different constraints regarding memory usage features. Thus, designers must pay attention in the porting process to the lack of memory of the target embedded system as well as the fact that such systems extensively employ the dynamic memory subsystem.

A desktop application is typically implemented using data structures or *Dynamic Data Types (DDTs)* [2] (dynamic arrays, linked lists, etc) to store their data. The DDT for each container is usually selected to achieve the best performance without bearing in mind other requirements such as power consumption, memory accesses and memory usage, an important factor in embedded systems. Thus, to map a desktop application, designers must reach the best set of DDTs that minimizes the system behavior according to some constraint of the target device, such as memory accesses, memory usage and energy consumption [3]. This is an NP-complete problem and cannot be fully explored.

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 [8], which tries to cover all the optimal implementation points for the required design metrics. The construction of this PF has been proven a very time-consuming process, sometimes even unaffordable [10]. To solve this problem we propose to use parallel processing based on Multi Objective Evolutionary Algorithms. This combination is very useful because we can explore more solutions in less time. Besides, the use of parallel processing allows us to design new algorithms that improve the number and the quality of solutions. However, before explaining deeply our proposal we begin by reviewing some related work.

Several works have been made in the field of embedded memory subsystem optimization, both in static and dynamic memory. In the case of static memory, Benini et al. [4] and Panda et al. [24] presented in the last decade two thorough surveys on static data and memory optimization techniques for embedded systems. More recently, in [6], [10] and [7], authors achieve to reduce the memory subsystems requirements by 50% using a linear time algorithm by exploring a coordinated data and computation reordering for array-based data structures in multimedia applications. Nevertheless, they are not suitable for exploration of complex DDTs employed in modern multimedia applications.

In the field of dynamic embedded software, there are some approaches that propose power-aware transformation and use pruning strategies based on heuristics to find the best solution [33] [24] [23]. These proposals have some weakness. On the one hand, they need to study and develop efficient pruning cost-function and a fully

manual optimization, which is not very efficient. On the other hand, these works do not bear in mind that the final behavior of the system presents inter-dependencies in the set of DDT implementations.

Regarding exploration methods, *Multi-Objective Evolutionary Algorithms (MOEAs)* started to be used to solve different CAD optimization problems (as proposed Michalewicz in [20]). In our case, the use of MOEAs to explore DDTs has become a good alternative. In [6] and [33] several transformations are established for DDTs and static data profiling and static memory access patterns to physical memories, and they are used to obtain information in order to find the best solution. In this context, MOEA-based optimization has been applied to solve linear-and non-linear problems by exploring the entire state space in parallel. Thus, it is possible to perform optimization in non convex regular functions, and to select the order of algorithmic transformations in concrete types of source codes [24]. However, such techniques are not applicable to DDT implementations due to it is not possible to know the DDT behavior (the number of elements stored in the DDT, number of read accesses, number of write accesses, etc.), at compile-time.

In the field of dynamic memory optimizations in embedded systems, Atienza et al. [3] have performed an initial analysis of one single type of MOEA showing the potential benefits of MOEAs for this kind of problems. Nevertheless, their work does not provide a complete analysis of tradeoffs between different technologies of sequential and parallel MOEAs. We tackle this problem in the present research work.

In order to be able to use parallel evolutionary algorithms for multi-objective problems, different paradigms of the parallel processing and their corresponding parameters have to be analyzed. In [31], Veldhuizen studies some important questions in the formulation of *parallel Multi-Objective Evolutionary Algorithms (pMOEA)* such as migration, replacement and niching schemes. Besides, he gives a classification of pMOEA based on the island paradigm: (1) islands execute the same MOEA [34]; (2) islands execute different MOEA [14]; (3) each island evaluates a different subset of objective functions [32]; and (4) each island considers a different region of the search domain [30].

In this work, we propose a new method that uses parallel processing and evolutionary algorithm to explore the design space of DDT implementation. To this end, we use Discrete Event Systems Specifications (DEVS) [35] over Service Oriented Architecture (SOA) [21], which offers DEVS-based simulations as a web service based on standard technologies, called DEVS/SOA. We explore several classical Multi-Objective Evolutionary Algorithms (MOEA) [11] and propose an algorithm which combines NSGA-II and SPEA2 within a DEVS/SOA framework. It allows designers to reach a larger number of solutions than classical approaches. Our parallel design may be included in the second group mentioned before (islands executing different MOEAs). Since our migration policy is synchronous, we have combined two elitist evolutionary algorithms with different complexity, namely *Strength Pareto Evolutionary Algorithm 2 (SPEA2)* [36] and *Non-dominated Sorting Genetic Algorithm II (NSGA-II)* [12], 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 469× faster) with respect to other traditional heuristics; (2) the parallel algorithm can achieve significant speed-ups 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 16 workstations of two cores each, where several population sizes were deployed as per our experiments. The experiments returned very promising results. In particular, we got empirical evidence that on increasing the size of the population, the performance of the pMOEA improves as we increase the number of workstations used.

The rest of the paper is organized as follows. Definitions of MOEAs and underlying technologies such as DEVS and DEVS/SOA are given in Section 2. In Section 3 the Dynamic Data Types optimization problem is explained. In Section 4, 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 5 details our experimental setup as well as shows some performance and quality metrics used in our experiments in Section 6. Finally, in Section 7 we summarize the main conclusions and future work.

2 Background

2.1 Multi-objective Evolutionary 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, 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:

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

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

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

If there is no solution which dominates $\mathbf{x} \in X$, $\mathbf{x}$ 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 solution with respect to the contradictory objectives.

In both algorithms, the sequential and parallel versions, we attempt to reach the higher number of non-dominated solutions 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 [8]. In the elitist approach, EAs store the best solutions of each generation in an external set. 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 [9], MOMGA-II [38], 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 [8]. Examples of this category include MOGA [15], HLGA [16], NPGA [18] and VEGA [28].

When implementing a MOEA, the designer has to overcome two major problems [37]. The first problem is how to get close to the POF [11]. 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.

Table 1. 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

Although all the cited MOEAs are different from each other, we can find some common steps in these algorithms, which are summarized in Table 1. As we have already mentioned, two representative elitist algorithms, namely, SPEA2 and NSGA-II were selected.

2.2 DEVS and DEVSJAVA

DEVS formalism consists of models, the simulator and the experimental frame. We will focus our attention to the specified two types of models i.e. atomic and coupled

models. The atomic model is the irreducible model definition that specifies the behavior for any modeled entity. The coupled model is the aggregation/composition of two or more atomic and coupled models connected by explicit couplings. The formal definition of parallel DEVS (P-DEVS) is given in [35]. An atomic model is defined by the following equation:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda \rangle \tag{3}$$

where,

- X is the set of input values
- S is the state space
- Y is the set of output values
- $\delta_{int} : S \rightarrow S$ is the internal transition function
- $\delta_{ext} : Q \times X^b \rightarrow S$ is the external transition function
  - $Q = \{(s,e) : s \in S, 0 \leq e \leq ta(s)\}$ is the total state set, where e is the time elapsed since last transition
  - X^b is a set of bags over elements in X
- δ_{con} is the confluent transition function, subject to $\delta_{con}(s, \oslash) = \delta_{int}(s)$
- $\lambda : S \rightarrow Y$ is the output function
- $ta(s) : S \rightarrow \Re_0^+ \cup \infty$ is the time advance function.

The formal definition of a coupled model is described as:

$$N = \langle X, Y, D, EIC, EOC, IC \rangle \tag{4}$$

where,

- X is the set of external input events
- Y is the set of output events
- D is a set of DEVS component models
- EIC is the external input coupling relation
- EOC is the external output coupling relation
- IC is the internal coupling relation.

The coupled model N can itself be a part of component in a larger coupled model system giving rise to a hierarchical DEVS model construction.

Fig. 1 shows a coupled DEVS model. M1 and M2 are DEVS models. M1 has two input ports: “in1” and “in2”, and one output port: “out”. The M2 has one input port: “in1”, and two output ports: “out1” and “out2”. They are connected by input and output ports internally (this is the set of internal couplings, IC). M1 is connected by external input “in” of Coupled Model to “in1” port, which is an external input coupling (EIC). Finally, M2 is connected to output port “out” of Coupled Model, which is an external output coupling (EOC).

The DEVSJAVA [1] is a Java based DEVS simulation environment. It provides the advantages of Object Oriented framework such as encapsulation, inheritance,

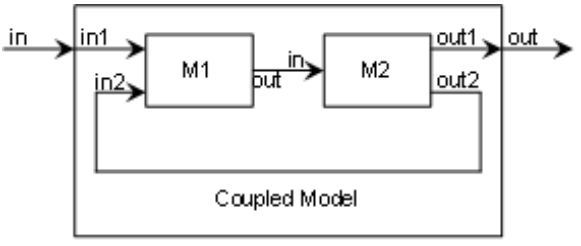

Fig. 1. Coupled DEVS model

and polymorphism. DEVSJAVA manages the simulation time, coordinates event schedules, and provides a library for simulation, a graphical user interface to view the results, and other utilities. Detailed descriptions about DEVS Simulator, Experimental Frame and of both atomic and coupled models can be found in [35].

2.3 DEVS/SOA

The Service oriented Architecture (SOA) is a framework consisting of various W3C standards, in which various computational components are made available as "services" interacting in an automated manner achieve machine-to-machine interoperable interaction over the network. Web-based simulation requires the convergence of simulation methodology and WWW technology (mainly Web Service technology). The fundamental concept of web services is to integrate software application as services. Web services allow the applications to communicate with other applications using open standards. We are offering DEVS-based simulators as a web service, which are based on these standard technologies: communication protocol (Simple Object Access Protocol, SOAP), service description (Web Service Description Language, WSDL), and service discovery (Universal Description Discovery and Integration, UDDI).

Fig. 2 shows the framework of our distributed simulation using SOA. The complete setup requires one or more servers that are capable of running DEVS Simulation Service. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the latest DEVSJAVA Version 3.1.

The Simulation Service framework is two layered framework. The top-layer is the user coordination layer that oversees the lower layer. The lower layer is the true simulation service layer that executes the DEVS simulation protocol as a Service. The lower layer is transparent to the modeler and only the top-level is provided to the user.

The top-level has three main services: upload DEVS model, compile DEVS model, and simulate DEVS model. The second lower layer provides the DEVS

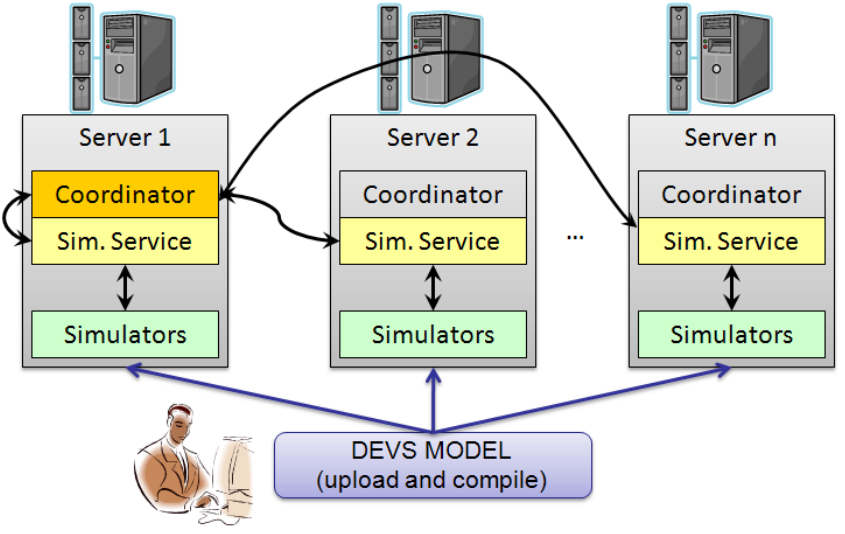

Fig. 2. DEVS/SOA distributed architecture

Simulation protocol services: initialize simulator *i*, run transition in simulator *i*, run lambda function in simulator *i*, inject message to simulator *i*, get time of next event from simulator *i*, get time advance from simulator *i*, get console log from all the simulators, and finalize simulation service.

The explicit transition functions, namely, the internal transition function, the external transition function, and the confluent transition function, are abstracted to a single transition function that is made available as a Service. The transition function that needs to be executed depends on the simulator implementation and is decided at the runtime. For example, if the simulator implements the *Parallel DEVS (P-DEVS)* formalism, it will choose among internal transition, external transition or confluent transition.

The client is provided a list of servers hosting DEVS Service. He selects some servers to distribute the simulation of his model. Then, the model is uploaded and compiled in all the servers. The main server selected creates a coordinator that creates simulators in the server where the coordinator resides and/or over the other servers selected. This whole framework is known as DEVS/SOA framework and details are available at [22], [21].

Summarizing from a user's perspective, the simulation process is done through three steps (Fig. 3): (1) write a DEVS model (currently DEVSJAVA is only supported), (2) provide a list of DEVS servers (through UDDI, for example). Since we are testing the application, these services have not been published using UDDI by now. Select *N* number of servers from the list available, and (3), run the simulation (upload, compile and simulate) and wait for the results.

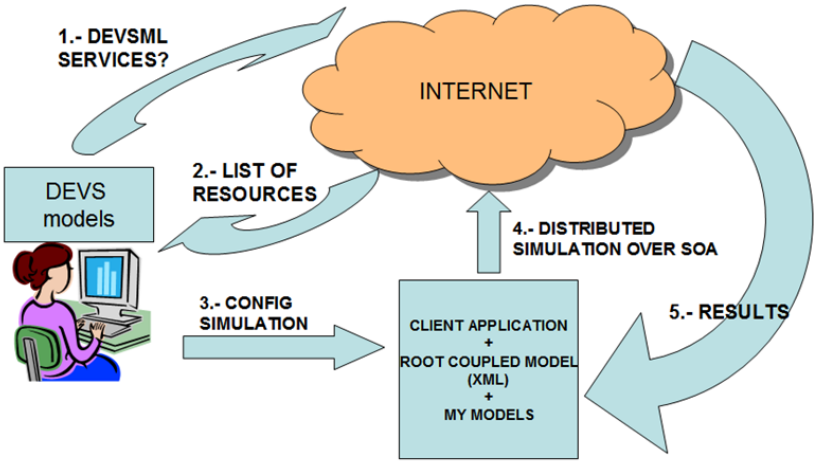

Fig. 3. Execution of DEVS SOA-Based M&S

3 The Dynamic Data Types Exploration Problem

DDTs are software abstractions by means of which we can manipulate and access data. The implementation of 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.

Fig. 4 shows an example of DDTs exploration. The initial code contains two containers, *c1* and *c2*, instantiated as a *vector* and a *list*, respectively. After the exploration process, we can obtain for example a candidate solution that recommends *c1* to be instantiated as *Single Linked List (SLL)* and *c2* as *Double Linked List of Arrays (DLLAR)*.

More generally we can state that the application to optimize contains a set of containers C, which are candidates to be instantiated as a certain DDT from the set of possible implementation of DDTs library D presented in [3] [10]. Thus, the goal of our optimization flow is to obtain a set of pairs (container, DDT) $\{c_i \in C, d_j \in D\}$, 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

```
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

Fig. 4. Code before and after the exploration of Dynamic Data Types

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 2 contains the DDTs implemented [3].

Table 2. DDT library

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

Once we have fixed the problem optimization process for DDTs, we can describe the whole process shown in Fig. 5. 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 sections.

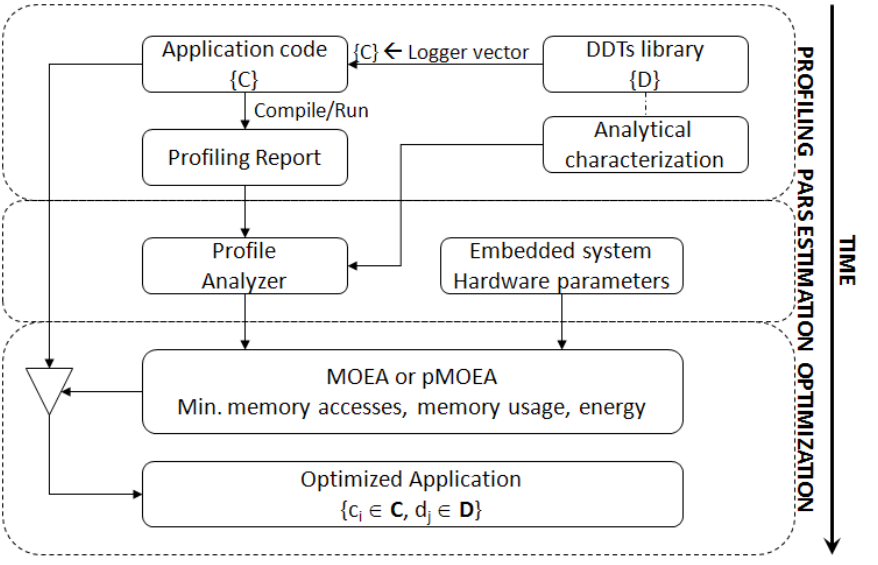

Fig. 5. DDTs optimization flow

3.1 Profiling of the Application

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 [10] 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 [3].

3.2 Parameters Estimation

In this phase, we extract all information needed from the profiling report. The purpose is to measure the quality of a solution (c_i, d_j) in the DDT exploration, using several parameters, namely, the number of candidate variables, number of elements stored in the container in the worst case (N_e), average of the number of elements stored (N_{ve}), size of the elements in bytes (T_e), size of the pointers in bytes (T_{ref}), number of read accesses (N_r), number of write accesses (N_w) and cache misses (N_{pa}). 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 [13] 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 application.

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 the following equations, where Hw represents the effect that hardware parameters (memory architecture, CPU power, line sizes, memory access time, etc.) have on the optimization [25].

$$\begin{aligned} MA(\mathbf{c},\mathbf{d}) &= f_{MA}(N_e, N_{ve}, N_r, N_w) \\ MU(\mathbf{c},\mathbf{d}) &= f_{MU}(T_e, T_{ref}, N_e) \\ E(\mathbf{c},\mathbf{d}) &= f_E(N_r, N_w, N_{pa}, Hw) \end{aligned} \tag{5}$$

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

$$f_{MA} \propto N_e \times (N_r + N_w) + N_{ve} \tag{6}$$

The exact form of equation 6 depends on each DDT selected in $(\mathbf{c},\mathbf{d})$. 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 container.

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

$$f_{MU} \propto T_{ref} + N_e \times \left(T_{ref} + T_e\right) \tag{7}$$

As in equation 6, the exact form of equation 7 depends on each DDT selected. It calculates the amount of memory used by each element stored in the DDT.

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

$$\begin{aligned} f_E = {} & t_{ex} \times CPU_{pow} + \\ & (N_r + N_w) \times (1 - N_{pa}) \times C_{accE} + \\ & (N_r + N_w) \times N_{pa} \times C_{accE} \times C_{lineS} + \\ & (N_r + N_w) \times N_{pa} \times DRAM_{accP} \times \\ & \left(DRAM_{accT} + \frac{C_{lineS}}{DRAM_{bandW}} \right) \end{aligned} \tag{8}$$

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 energy, 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 8. 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.

The equation for calculating the system's total execution time t_{ex} is given by:

$$\begin{aligned} t_{ex} = {} & (N_r + N_w) \times (1 - N_{pa}) \times C_{accT} + \\ & (N_r + N_w) \times N_{pa} \times DRAM_{accT} + \\ & (N_r + N_w) \times N_{pa} \times \frac{C_{lineS}}{DRAM_{bandW}} + \\ & T_{bus} \end{aligned} \tag{9}$$

where C_{accT} is the access time of the cache.

There exist four components in the system's execution time shown in equation 9. The first term $(N_r + N_w) \times (1 - N_{pa}) \times C_{accT}$ is for calculating the amount of time taken for the processor to access the cache. The second term $(N_r + N_w) \times N_{pa} \times DRAM_{accT}$ calculates the amount of time required for the DRAM to respond to each cache miss. The third term calculates the amount of time taken to fill a cache line on each cache miss. The bus communication time cost is supposed to be constant (T_{bus}). As the bus communication time is expected to be similar to other systems, such decision will not adversely affect the final results.

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 the fitness of the solutions found in the exploration process. When the optimization process ends, it gives the DDT instantiation policy, i.e., which container should be implemented by which DDT. We also obtain the gain on memory accesses, memory usage and energy consumption.

3.4 Encoding 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 [11].

Table 3 shows the representation of a chromosome. Genes are represented in the first row. Each of the chromosomes represents the set of DDT that should be used to instantiate all the corresponding containers in the application from Table 2.

For example, the second container $c_2 \in C$ will be instantiated by an array (AR). A chromosome contains n genes, where n is the number of the containers logged in the application, $n = size(C)$. We may use an integer to represent the values of a gene, and the constraint a gene must satisfy is: $1 \leq ddt \leq size(D)$.

Table 3. Example of an individual

Dynamic Data Type	*AR*	*AR*	*SLL*	...	*DLL*
Container	c_1	c_2	c_3	...	c_n

Consequently, if an application contains n containers, 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 2 for each of them. Thus, it can cover large real-life dynamic embedded applications.

4 Parallel Implementation

In this section we describe the parallel MOEA designed and how it is implemented in a DEVS environment.

4.1 pMOEA

We are employing pMOEAs for better performance when solving the exploration of DDTs in embedded applications described in Section 3, i.e., we are improving the quality of the solutions found and the time to obtain them. When developing pMOEA, some parameters must be defined [31]: MOEA(s) parallelized, topology, population size, migration rate, and replacement.

Regarding MOEAs and topology, we propose a coarse-grained pMOEA where each island may execute a different MOEA, in our case either NSGA-II [12] or SPEA2 [36]. We have used these two MOEAs because of their different complexity, but other algorithms could be included. 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 islands are suited for a ring topology [5]. Experiments with other topologies are left for future study.

With respect to the population size of each island, few studies have been made in the literature[31]. For example, in [27], the sequential population is divided by the number of islands, remaining the size of the external set constant and equal to the initial population size. In this way, the number of islands increases, the execution time is reduced and the number of non-dominated solutions grows up. However,

some metrics such as hypervolume or spread loose quality. In this work, we apply the following equation to the population size [26]:

$$P_i = \frac{P}{I} + \alpha \times \left(P - \frac{P}{I} \right) \tag{10}$$

where P_i is the population size of island $i \in [1..I]$, P is the population size in the sequential approach, $I > 1$ is the number of islands and $\alpha \in [0..1]$ is a scaling factor. Note that when I tends to infinity, P_i is constant: $\alpha \times P$. It is a design parameter that depends on the migration rate and hardware configuration (i.e. network bandwidth, processor types, etc). The purpose is to obtain better solutions in less computing time when the number of islands is increased. After several tests, we have set $\alpha = \frac{2}{I}$.

Regarding the external file size, we apply the following equation [26]:

$$P_i^E = P_i + N_I \tag{11}$$

where N_I is the total number of immigrants that island i will receive.

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. These parameters are: (a) the topology defined by the connections between islands, a ring in our case; (b) a migration rate that controls how many individuals migrate; and (c) a migration interval that determines the migration frequency. Our migration rate is set to $P/100$, where P is the population size in the sequential algorithm. The best $P/100$ individuals are selected in the following way. First, we extract the set of non-dominated solutions in the current population P_i. Second, we sort the resulting set with respect to one random objective, and extract the first $P/100$ individuals. Moreover, since NSGA-II is faster than SPEA2 ($O(mN^2)$ vs. $O(N^3)$), NSGA-II could finish first while SPEA2 is still exploring early generations. Thus, our migration policy is synchronized every 100 generations.

4.2 DEVS and DEVS/SOA Implementation

Fig. 6 provides a scheme of the parallel procedure with two atomic models (top of the figure) and their execution over time (bottom of the figure). Each atomic model represents an island and includes two pair of *request, response* output and input ports. *Request* connections are used to ask for the best individual of the adjacent atomic model, and *response* connections are used to send this individual when available (every 100 generations, in Fig. 6). In other words, the specific MOEA (NSGA-II or SPEA2) is applied to each atomic model separately, and the best partial results are periodically sent from one atomic model to its neighbor on a ring communication topology.

We have implemented three variations that are tested in a multi-core and distributed architecture. The only difference between these variations is the MOEA algorithm that is controlling the subpopulation, i.e. running on each atomic model:

1. NS_K configuration: K atomic models executing NSGA-II and the same quantity running SPEA2, $2K$ islands in total.
2. SS_K configuration: $2K$ atomic models, but running all of them SPEA2 algorithm.
3. NN_K configuration: $2K$ atomic models using the NSGA-II algorithm.

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

The algorithm shown in Fig. 6 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 set of workstations connected over a LAN. To this end, using our DEVS/SOA framework, we have executed 32 atomic models on 16 workstations each of two cores. The algorithm is exactly the same, but each workstation executes two atomic models. Individuals are sent between different workstations using web services [22]. Fig. 7 depicts an illustrative example of two workstations each running two MOEAs. Every workstation executes two MOEAs as a DEVS coupled model. The coupled models are connected in the desired topology (a ring in our case), which again is another design parameter that could impact the performance. Our atomic models are suited for a ring topology as well.

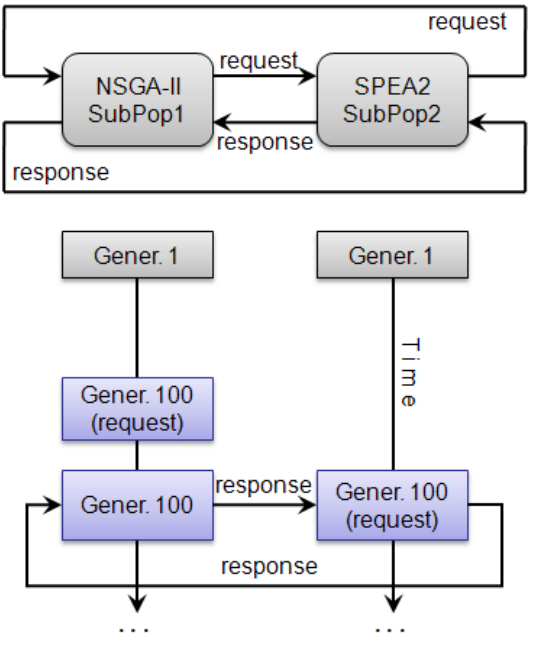

Fig. 6. A graphic representation of the DEVS model (multi-core architecture) and its evolution over time

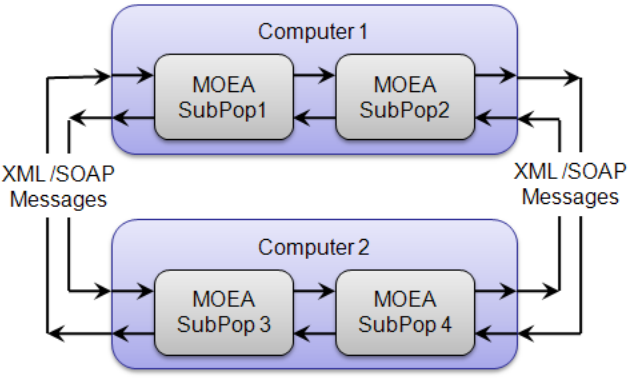

Fig. 7. A graphic representation of the DEVS model (multi-core/distributed architecture)

5 Experimental Methodology

In this section we describe the complete method applied to compare the different type of sequential and parallel MOEAs while optimizing a real-life dynamic embedded application.

We have evaluated the proposed optimization framework for a 3D Physics Engine for elastic and deformable bodies [19], 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 select the optimal DDT implementation in Table 2. It can cover all of the real-life embedded applications we are aware off.

5.1 Embedded System HW/SW Specification

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 Fig. 8.

To analyze the effect of MOEAs on embedded system's memory accesses, memory usage and energy consumption, we utilized processor energy from [6], and the access time and energy values for caches of 32KB and embedded 16MB DRAM main memory from [29] and [17], respectively. The processor and memory specification is described in Table 4.

5.2 Performance Metrics

To compare the performance of different MOEAs, we need to evaluate the obtained set of non-dominated solutions considering: (1) Convergence to POF. (2) Diversity

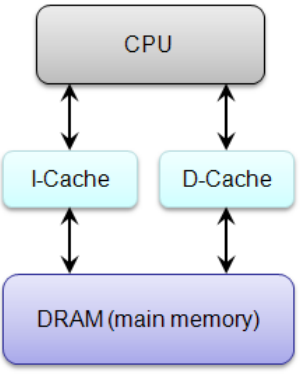

Fig. 8 System architecture: Instruction cache, data cache, and embedded DRAM as main memory

Table 4. System specification

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

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.

5.2.1 Coverage

We use the coverage metric [37] to measure convergence. Let PF', PF'' be two sets of non-dominated solutions. The coverage metric can be defined as follows:

$$C(PF', PF'') = \frac{|p'' \in PF''; \exists p' \in PF' : p' \preceq p''|}{|PF''|} \tag{12}$$

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

5.2.2 Hypervolume or S-Metric

This metric calculates the volume (in the objective space) covered by members of a nondominated set of solutions Q [37]. Let v_i be the volume enclosed by solution $i \in Q$. Then, a union of all hypercubes is found and its hypervolume (H_V) is calculated.

$$H_V = \bigcup_1^{|Q|} v_i \tag{13}$$

The hypervolume of a set is measured relative to a reference point, usually the anti-optimal point or "worst possible" point in space. (We do not address here the problem of choosing a reference point, if the anti-optimal point is not known or does not exist one suggestion is to take, in each objective, the worst value from any of the fronts being compared). 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. Since this metric is not free from arbitrary scaling of objectives, we have evaluated the metric by using normalized objective function values.

5.2.3 Non-dominated Solutions

Given that DDTs optimization is a difficult problem, finding a high number of non-dominated solutions could be itself a hard challenge for any multi-objective optimizer. In this sense, the number of non-dominated solutions can be considered as a measure of the ability of the algorithm for exploring difficult search spaces.

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

6 Experimental Results

To compare the performance of both sequential and parallel algorithms, the number of generations, and probability of crossover and mutation are set to the same values. After different tests, we have fixed them to the values indicated in Table 5. The sequential population size is set to 200 for each atomic model. In our parallel simulations, the population size follows equation 10. Migration rate and frequency are those described in Section 4. In all cases, the external archive size (where non-dominated solutions are stored) is set to the value given by equation 11.

Next, we summarize the results obtained by the sequential and parallel evolutionary algorithms. As it was mentioned in Section 4, we are able to run our MOEAs under three configurations: (1) a stand-alone atomic model (sequential architecture), (2) several atomic models running in separated threads (multi-core architecture) which utilize multiple processors when available, and (3) several atomic models running in separated threads and distributed amid a set of workstations (multi-core/distributed architecture). The distributed version is configured by using the DEVS/SOA framework. The experiments have been made using 16 workstations

Table 5. Parameters for evolutionary algorithms

Parameter	Value
Population size	200
Number of generations	8000
Probability of crossover	0.80
Probability of mutation	0.01

Intel® Core™ 2 CPU 6600 2.40GHz with 2GB DDR memory connected via 100Mbps Ethernet network.

6.1 Sequential DEVS Architecture

We have tested the sequential DDTs exploration speed in comparison to different alternative methods for the 3D Physics Engine application on a Intel® Core™ 2 CPU 6600 2.40GHz with 2GB DDR memory. Execution times are calculated by averaging results of 10 trials. The results obtained for the different tested exploration methods are shown in Table 6. We have compared our algorithms with state-of-the-art pruning and optimization methods for DDT implementations presented in [33], [10]. In these cases breadth-first, deep-first and branch & 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^{3128} feasible solutions (10 DDTs for 3128 containers) it is unfeasible to reach the complete POF by means of exhaustive exploration. The results in Table 6 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, 470× faster. Note that although in theory VEGA is faster than both NSGA-II and SPEA2, our design framework is able to obtain better speed-ups. This is because of our *Profile Analyzer* tool (Section 3), which can extract all

Table 6. Comparison between the proposed sequential algorithms and other techniques

Exploration method	Time (seconds)
Breadth-First	$11.23 \times 10^5 \pm 98.62$
Depth-First	$43.20 \times 10^4 \pm 87.13$
Branch&Bound	$10.80 \times 10^3 \pm 55.42$
VEGA [3]	$7.20 \times 10^3 \pm 103.20$
NSGA-II	$2.39 \times 10^3 \pm 0.78$
SPEA2	$3.83 \times 10^3 \pm 4.37$

the needed information from the profiling report, and it is done once for the target embedded application.

6.2 Multi-core DEVS Architecture

In order to exploit our 2-cores architecture, we have explored DDTs with some configurations of the three algorithms proposed (i.e., NN_K, NS_K and SS_K) on an Intel® Core™ 2 CPU 6600 2.40GHz with 2GB DDR memory. All the values presented are calculated by averaging results of 50 trials.

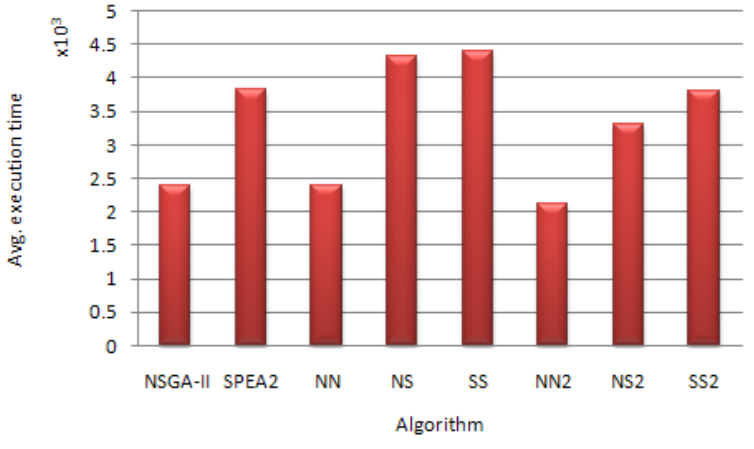

Fig. 9. Comparison between our sequential and multi-core algorithms

Fig. 9 shows the comparisons between the execution times of both sequential and parallel algorithms. Regarding pMOEAs, the number of islands is increased, the execution time is reduced. With respect to *NN*, *SS* and *NS*, we can see that the execution time is greater than in the sequential version. It is because equation 10 has been designed to balance the loss of non-dominated solutions when the number of islands grows up. However, as Fig. 9 depicts, NN_2 is faster than NSGA-II, and SS_2 is faster than SPEA2. To conclude, except in the case of two islands, all the pMOEAs are faster than the sequential version, even if more islands than cores are used (see SS_2 vs. SPEA2 in Fig. 9, for example). Between pMOEAs, the fastest one is NN_2, as each island uses the smallest population size with the fastest algorithm.

Fig. 10 depicts the number of non-dominated individuals obtained. NSGA-II offers the same non-dominated solutions as SPEA2. *NS* offers 49.3% more optimal solutions than both NSGA-II and SPEA2, and NS_2 4.69% more than *NS*. Thus, with respect to N_D, NS_K offers more optimal alternatives to the system designer for the implementation of the final embedded application.

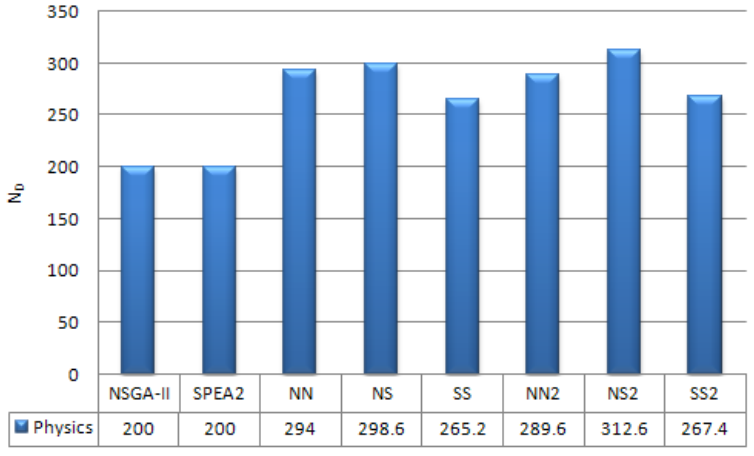

Fig. 10. Non-dominated individuals obtained by NSGA-II, SPEA2, NN_K, SS_K and NS_K, with $K = 1,2$

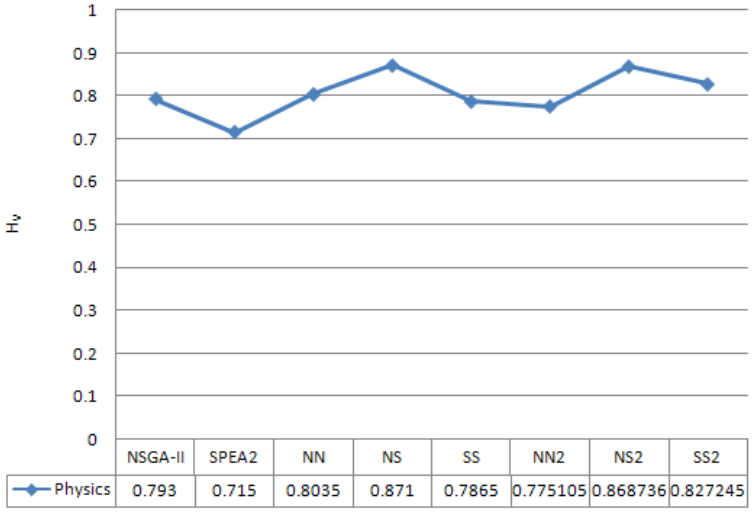

Fig. 11. Hypervolume or S-metric obtained by NSGA-II, SPEA2, NN_K, SS_K and NS_K, with $K = 1,2$

Fig. 11 shows the hypervolume or S-metric obtained. NS_K algorithms reach better values compared to the other MOEAs, sequential or parallel. Thus, the result set from NS_K algorithms is taken to be a better set of solutions than those obtained from other algorithms.

Table 7. Coverage metric

	NSGA-II	SPEA2	NN	NS	SS	NN_2	NS_2	SS_2	AVG
NSGA-II	–	0.0660	0.0959	0.0813	0.1132	0.1546	0.1208	0.1246	0.1081
SPEA2	0.2260	–	0.1719	0.1149	0.1684	0.1518	0.1100	0.1519	0.1564
NN	0.3030	0.2450	–	0.1476	0.1776	0.2122	0.1764	0.2130	0.2107
NS	0.3890	0.3180	0.4031	–	0.3213	0.2153	0.2207	0.2299	**0.2996**
SS	0.3440	0.3170	0.3223	0.1030	–	0.2404	0.1256	0.2677	0.2457
NN_2	0.3680	0.2810	0.2795	0.0693	0.2478	–	0.1092	0.1869	0.2202
NS_2	0.3620	0.3650	0.3131	0.1575	0.2429	0.2580	–	0.2897	**0.2840**
SS_2	0.3580	0.3570	0.2753	0.1723	0.2984	0.2937	0.1249	–	0.2685
AVG	0.3357	0.2784	0.2659	**0.1208**	0.2242	0.2180	**0.1411**	0.2091	–

Finally, Table 7 shows the coverage values obtained. Last row and last column show the averaged coverage over each column and each row, respectively. Regarding convergence comparisons, Table 7 shows that, in average, NS_K algorithms are better than any other algorithm. For example, $C_{avg}(NS,*) > C_{avg}(*,NS)$ is $0.2996 > 0.1208$ or $C_{avg}(NS_2,*) > C_{avg}(*,NS_2)$ is $0.2840 > 0.1411$. In the same way, $C_{avg}(NS,*) > C_{avg}(SS,*)$ is $0.2996 > 0.2457$ and $C_{avg}(*,NS) < C_{avg}(*,SS)$ is $0.1208 < 0.2242$.Thus, NS_K offers more optimal alternatives to the system designer for the implementation of the final embedded application.

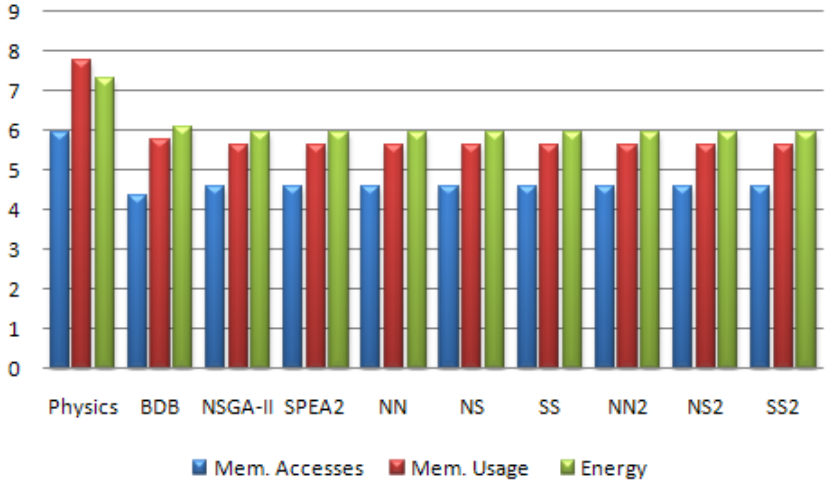

Fig. 12. Comparison of the real application with results obtained by our design framework (logarithmic scale).

For comparative reasons with the original application, we present Fig. 12 to illustrate the optimization process that our methodology performs. In this test, we compare the evaluation of our multi-objective function to the results obtained by all the algorithms used in our design framework. Since breadth-first, depth-first and branch & bound exploration methods offer the same solution, these results are grouped and labeled as BDB in Fig. 12. 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 Fig. 5. Furthermore, as this figure indicates, evolutionary algorithms offered the best compromise among objectives.

6.3 Multi-core DEVS/SOA Architecture

Finally, the NS_K configuration was distributed on a set of 16 workstations Intel® Core™ 2 CPU 6600 2.40GHz with 2GB DDR memory, connected via a 100Mbps Ethernet network. To this end, we placed two threads per workstation and the communication among workstations was made through our DEVS/SOA framework. All the values presented are calculated by averaging results of 10 trials.

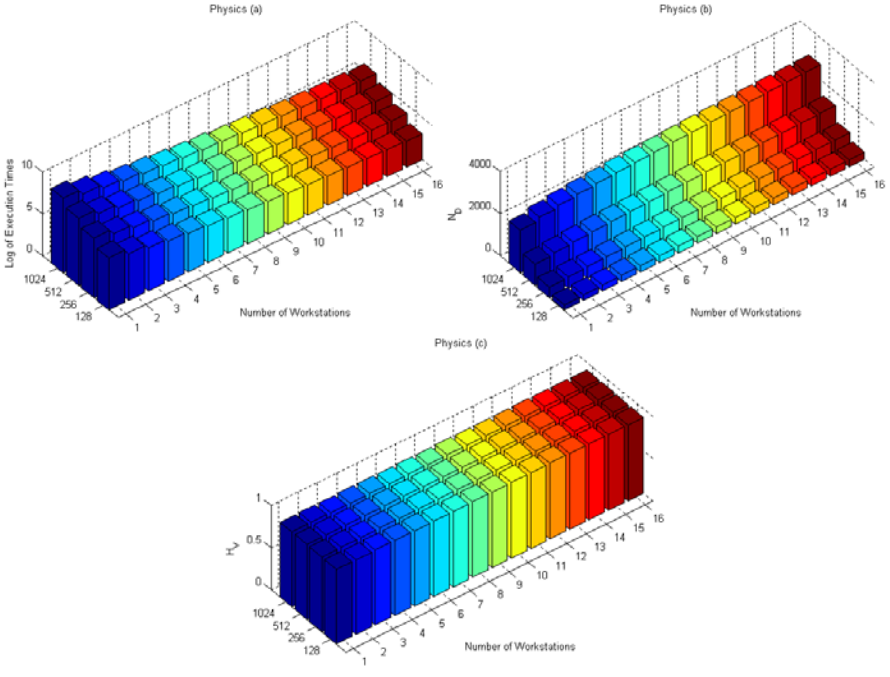

Fig. 13. Execution times (a), non-dominated solutions (b), and hypervolume (c) as a function of the number of workstations. Each workstation executes two DEVS atomic models

We tested our algorithm using from 1 to 16 workstations. This leads to 2, 4, 6, ..., 32 MOEAs running in parallel, namely NS_1, NS_2, NS_3, ..., NS_{16}, and different population sizes (128, 256, 512 and 1024). The tests were performed by changing only the number of workstations in order to observe and study the increase in performance (speed-up, N_D and H_V). In all these cases the number of generations was set to 8000. The population and external archive size of each island was set following equations 10 and 11, respectively.

In light of the results presented in Fig. 13, as the size of the population increased, the execution time of the parallel version improved proportionally to the number of islands (see Fig. 13a). Also, Fig. 13b indicates that the number of non-dominated individuals increased at logarithmic rate as the number of islands increased. Finally, as Fig. 13c depicts, the hypervolume remains constant along all the simulations, with non-significant variations.

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.

7 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 research work 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 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, NS_K reaches more optimal solutions than the other sequential and parallel algorithms, obtaining an execution time that decreases with the number of islands used.

We also have 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. Although we conducted our research experiments in a LAN setting, deploying the application over a grid enabled DEVS/SOA infrastructure allows us to capitalize on the speedup that we achieved in our proposed NS_K.

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 research work.

References

1. Arizona center of integrative modeling & simulation, acims (2008), http://www.acims.arizona.edu
2. Antonakos, J.L., Mansfield, K.C.: Practical Data Structures using C/C++. Prentice-Hall, Englewood Cliffs (1999)
3. Atienza, D., Baloukas, C., Papadopoulos, L., Poucet, C., Mamagkakis, S., Hidalgo, J.I., Catthoor, F., Soudris, D., Lanchares, J.: Optimization of dynamic data structures in multimedia embedded systems using evolutionary computation. In: SCOPES 2007: Proceedingsof the 10th international workshop on Software & compilers for embedded systems, pp. 31–40. ACM Press, New York (2007), http://doi.acm.org/10.1145/1269843.1269849
4. Benini, L., de Micheli, G.: System-level power optimization: techniques and tools. ACM Trans. Des. Autom. Electron. Syst. 5(2), 115–192 (2000), http://doi.acm.org/10.1145/335043.335044
5. Cantú-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Dordrecht (2000)
6. Catthoor, F., Danckaert, K., Kulkarni, C., Brockmeyer, E., Kjeldsberg, P.G., Achteren, T.V., Omnes, T.: Data access and storage management for embedded programmable processors. Kluwer Academic Publishers, Dordrecht (2002)
7. Choi, Y., Kim, T., Han, H.: Memory layout techniques for variables utilizing efficient dram access modes in embedded system design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24(2), 278–287 (2005)
8. Coello, C.: A comparative survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems 1, 269–308 (1999)
9. Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J.: Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In: Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001), pp. 283–290. Morgan Kaufmann, San Francisco (2001)
10. Daylight, E.G., Atienza, D., Vandecappelle, A., Catthoor, F., Mendias, J.M.: Memory-access-aware data structure transformations for embedded software with dynamic data accesses. IEEE Transactions on VLSI Systems 12, 269–280 (2004)
11. Deb, K.: Multiobjective Optimization using Evolutionary Algorithms. John Wiley and Son Ltd., Chichester (2001)
12. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
13. Edler, J.: Dinero iv trace-driven uniprocessor cache simulator (2008), http://pages.cs.wisc.edu/~markhill/DineroIV
14. Fernandez, J.M., Vila, P., Calle, E., Marzo, J.L.: Design of virtual topologies using the elitist team of multiobjective evolutionary algorithms. In: Obaidat, M., Gburzynski, P. (eds.) Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2007), San Diego, USA, pp. 266–271 (2007)
15. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. In: Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA 1993), pp. 416–423 (1993)
16. Hajela, P., Lin, C.Y.: Genetic search strategies in multicriterion optimal design. Structural Opt. 4, 99–107 (1992)

17. Hardee, K., Jones, F., Butler, D., Parris, M., Mound, M., Calendar, H., Jones, G., Aldrich, L., Gruenschlaeger, C., Miyabayashil, M., Taniguchi, K., Arakawa, I.: A 0.6v 205mhz 19.5ns trc 16mb embedded dram. In: IEEE International Solid-State Circuits Conference, ISSCC (2004)
18. Horn, J., Nafpliotis, N., Goldberg, D.E.: A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the First IEEE Conference on Evolutionary Computation, vol. 1, pp. 82–87 (1994)
19. Kharevych, L., Khan, R.: 3d physics engine for elastic and deformable bodies. University of Maryland, College Park (2002), http://www.cs.umd.edu/Honors/reports/kharevych.html
20. Michalewicz, Z.: Genetic Algorithms + data structures = Evolution Programs. Springer, Heidelberg (1996)
21. Mittal, S., Risco-Martin, J.L., Zeigler, B.P.: Devs/soa: A cross-platform framework for net-centric modeling and simulation using devs. Submitted to SIMULATION: Transactions of SCS, in review (2007)
22. Mittal, S., Risco-Martín, J.L., Zeigler, B.P.: Devs-based web services for net-centric t&e. In: Summer Computer Simulation Conference, SCSC 2006 (2006)
23. Muttreja, A., Raghunathan, A., Ravi, S., Jha, N.K.: Automated energy/performance macromodeling of embedded software. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26(3), 542–552 (2007)
24. Panda, P.R., Catthoor, F., Dutt, N.D., Danckaert, K., Brockmeyer, E., Kulkarni, C., Vandercappelle, A., Kjeldsberg, P.G.: Data and memory optimization techniques for embedded systems. ACM Trans. Des. Autom. Electron. Syst. 6(2), 149–206 (2001), http://doi.acm.org/10.1145/375977.375978
25. Risco-Martin, J.L., Atienza, D., Hidalgo, J.I., Lanchares, J.: Analysis of multi-objective evolutionary algorithms to optimize dynamic data types in embedded systems. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008 (2008)
26. Risco-Martin, J.L., Atienza, D., Hidalgo, J.I., Lanchares, J., Mittal, S.: Optimization of multimedia embedded applications using genetic algorithms and discrete event simulation over soa. Submitted to IEEE Transactions on Computer-Aided Design
27. Risco-Martín, J.L., Atienza, D., Hidalgo, J.I., Lanchares, J.: A parallel evolutionary algorithm to optimize dynamic data types in embedded systems. Soft Computing - A Fusion of Foundations, Methodologies and Applications 12(12), 1157–1167 (2008)
28. Schaffer, J.D.: Multiple objective optimization with vector evaluated genetic algorithms. In: Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms, pp. 93–100. Hillsdale, New Jersey (1985)
29. Shivakumar, P., Jouppi, N.P.: Cacti 3.0: An integrated cache timing, power, and area model. Tech. Rep. 2001/2, Compaq Computer Corporation (2001)
30. de Toro Negro, F., Ortega, J., Ros, E., Mota, S., Paechter, B., Martín, J.: Psfga: Parallel processing and evolutionary computation for multiobjective optimisation. Parallel Computing 30(5-6), 721–739 (2004)
31. Veldhuizen, D.A.V., Zydallis, J.B., Lamont, G.B.: Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation 7(2), 144–173 (2003)
32. Wilson, L., Moore, M.: Cross-pollinating parallel genetic algorithms for multiobjective search and optimization. International Journal of Foundations of Computer Science 16(2), 261–280 (2005)
33. Wuytack, S., Catthoor, F., De Man, H.: Transforming set data types to power optimal data structures. IEEE Transactions on Computer-Aided Design 15(6), 619–629 (1996)

34. Xiong, S., Li, F.: Parallel strength pareto multi-objective evolutionary algorithm for optimization problems. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), vol. 4, pp. 2712–2718. IEEE Press, Canberra (2003)
35. Zeigler, B.P., Kim, T., Praehofer, H.: Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press, London (2000)
36. Zitzler, E., Laumanns, M., Thiele, L.: 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, Barcelona, Spain, pp. 95–100 (2002)
37. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computing 3(4), 257–271 (1998)
38. Zydallis, J.B., Van Veldhuizen, D.A., Lamont, G.B.: A statistical comparison of multiobjective evolutionary algorithms including the MOMGA-II. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 226–240. Springer, Heidelberg (2001)