Optimization of Dynamic Data Types in Embedded Systems using DEVS/SOA-based Modeling and Simulation

José L. Risco-Martín, *Saurabh Mittal, David Atienza, J. Ignacio Hidalgo, Juan Lanchares

Department of Computer Architecture
and Automation
Complutense University of Madrid
C/Prof. José García Santesmases, s/n
28040 Madrid (Spain)
(+34) 91 394 7603

{jlrisco, datienza, hidalgo, julandan}@dacya.ucm.es

*Arizona Center for Integrative Modeling
and Simulation
University of Arizona
ECE Building, 1230 E Speedway
Tucson AZ 85721
(+1) 520 204 2641

saurabh@ece.arizona.edu

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. This results in the minimization of embedded design variables (memory accesses, power consumption and memory usage). Till date some effective heuristic algorithms have been developed in order to solve this problem, however unreported, as 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 research work provides a methodology to use *Discrete Event Systems Specification (DEVS)* to implement a parallel evolutionary algorithm within a *Service Oriented Architecture (SOA)*, where parallelism improves the solutions found by the corresponding sequential algorithm. This algorithm provides better results when compared with other previously proposed procedures. In order to implement the parallelism the DEVS/SOA framework in utilized. 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. This research also establishes DEVS/SOA as a platform for conducting complex distributed simulation experiments.

Keywords

Embedded Systems Design, Evolutionary Computation, Discrete Event System Specification, Service Oriented Architecture, DEVS/SOA.

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) [6]. As a result, one of the most important problems designers face today 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 [2] (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 [3].

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 [7], which tries to cover all the optimal implementation points for the required design metrics. The construction of this PF has been proven to be a very time-consuming process, sometimes even unaffordable [9].

Extensive work has been performed in the field of embedded memory subsystem optimization. Benini et al. [4] and Panda et al. [23] presented two thorough surveys on static data and memory optimization techniques for embedded systems presented during the last decade. More recently, in [6] and [9], authors have explored a coordinated data and computation reordering for array-based data structures in multimedia applications. They used a linear time algorithm reducing the memory subsystems requirements 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 gain ground for multimedia systems [31] [23]. However, these approaches require the development of efficient pruning cost-functions and fully manual optimizations. In addition, these works are not able to capture the evaluation of inter-dependencies of multiple DDTs implementations operating together, in contrast to the method of *Evolutionary Algorithms (EAs)* as proposed in this paper [19]. Atienza et al. [3] 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. We tackle this problem in the present research work.

Also, according to the characteristics of certain parts of multimedia applications, several transformations for DDTs and design methodologies [6] [31] have been proposed for static data profiling and optimization with 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 the entire state space in parallel. Thus, it is possible to perform optimization in non-convex regular functions, and also to select the order of algorithmic transformations in concrete types of source codes [23]. 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 framework to explore the design space of DDT implementation including a set of novel parallel procedures based on *Multi Objective Evolutionary Algorithms (MOEAs)* [10] and *Discrete Event System Specification (DEVS)* [33]. The development of parallel evolutionary algorithms for multi-objective problems involves the analysis of different paradigms for parallel processing and their corresponding parameters. In [29] 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 [29] four basic pMOEA based on the island paradigm are described: (1) islands execute the same MOEA [32]; (2) islands execute different MOEA [13]; (3) each island evaluates a different subset of objective functions [30]; and (4) each island considers a different region of the search domain [22]. 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)* [35] and *Non-dominated Sorting Genetic Algorithm II (NSGA-II)* [11], 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 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 of this paper.

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 exists a single optimal solution, and some trade-offs need to be considered. Without any loss of generality, we can assume the following N-objective minimization problem:

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

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:

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

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 solution 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 [7]. 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 [8], MOMGA-II [36], 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 [7]. Examples of this category include MOGA [14], HLGA [15], NPGA [17] and VEGA [26].

When implementing a MOEA, the designer has to overcome two major problems [34]. The first problem is how to get close to the Pareto Optimal Front (POF) [10]. 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 1. As we have already mentioned, two representative elitist algorithms, namely, SPEA2 and NSGA-II were selected.

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

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 [33]. An atomic model is defined by the following equation:

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

where,

- X is the set of input values
- S is the state space
- Y is the set of output values
- $\delta_{\text{int}} : S \to S$ is the internal transition function
- $\delta_{ext} : Q \times X^b \to S$ is the external transition function
  - $Q = \{(s,e) : s \in S, 0 \le e \le ta(s)\}$ 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
- $\delta_{con} : S \times X^b \to S$ is the confluent transition function, subject to $\delta_{con}(s,\varnothing) = \delta_{\text{int}}(s)$
- $\lambda : S \to Y$ is the output function
- $ta(s) : S \to \Re_0^+ \cup \infty$ is the time advance function.

The formal definition of a coupled model is described as:

$$N = \langle X, Y, D, EIC, EOC, IC \rangle$$

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.

Figure 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).

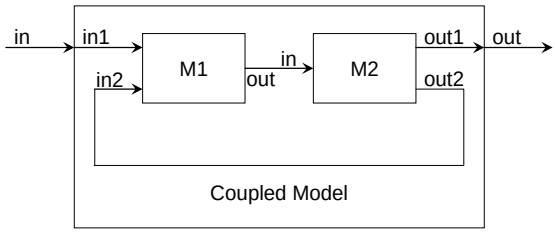

Figure 1. Coupled DEVS model

The DEVSJAVA [1] is a Java based DEVS simulation environment. It provides the advantages of Object Oriented framework such as encapsulation, inheritance, 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 [33].

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).

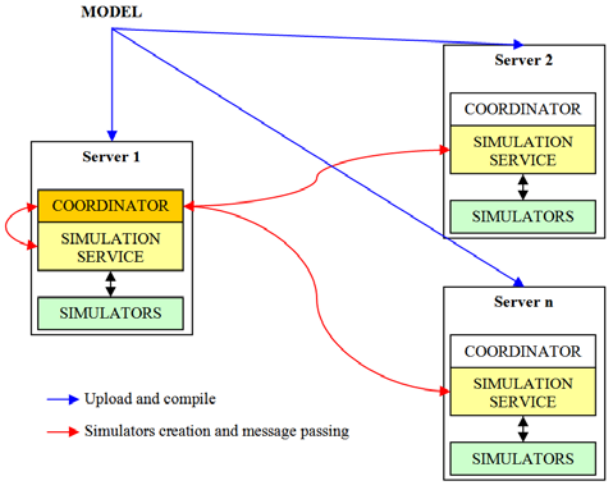

Figure 2. DEVS/SOA distributed architecture

Figure 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 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 [20][21].

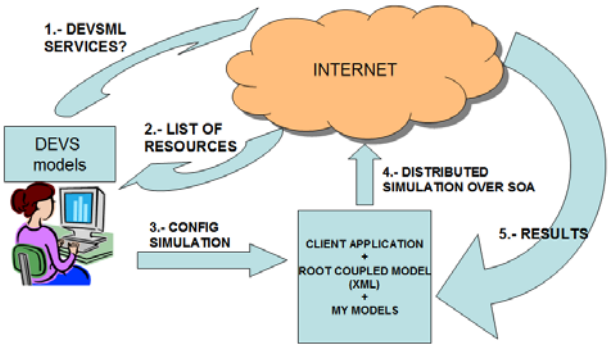

Figure 3. Execution of DEVS SOA-Based M&S

Summarizing from a user's perspective, the simulation process is done through three steps (Figure 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.

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.

```
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

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

Figure 4 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)*.

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

More generally we can state that the application to optimize contains a set of variables ***V***, which are candidates to be instantiated as a certain DDT from the set of possible

implementation of DDTs library ***D*** presented in [3] [9]. 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 2 contains the DDTs implemented [3].

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

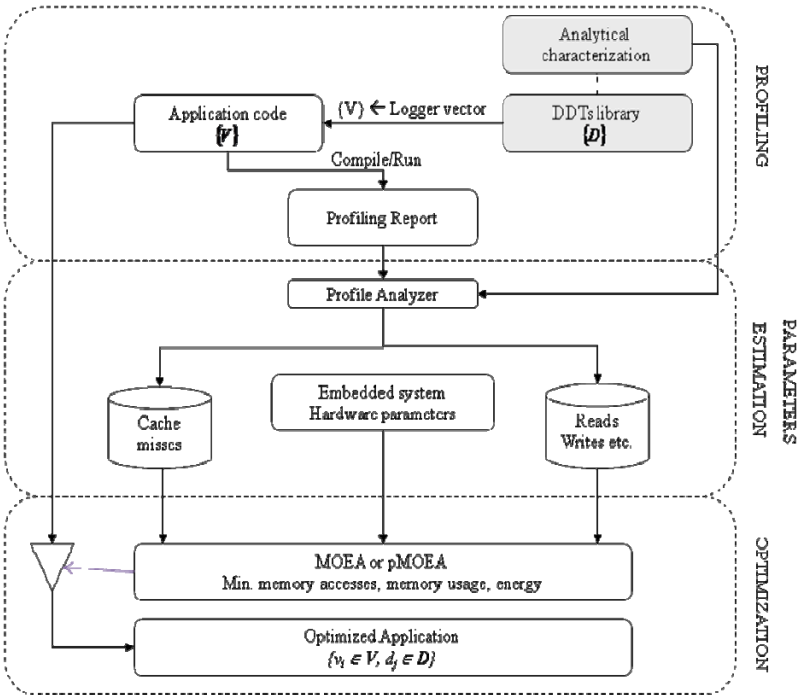

Figure 5. DDTs optimization flow

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 [9] 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 (v_i, d_j) in the DDT exploration, using several parameters, namely, the number of candidate variables, number of elements stored in the DDT in the worst case (*Ne*), average of the number of elements stored (*Nve*), size of the elements in bytes (*Te*), size of the pointers in bytes (*Tref*), number of read accesses (*Nr*), number of write accesses (*Nw*) and cache misses (*Npa*). 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 [12] 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. We are in a process of automating this data mining process using XML.

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.

$$MA\left(\vec{v},\vec{d}\right) = f_{MA}\left(Ne, Nve, Nr, Nw\right)$$
$$MU\left(\vec{v},\vec{d}\right) = f_{MU}\left(Te, Tref, Ne, Nve\right)$$
$$E\left(\vec{v},\vec{d}\right) = f_{MA}\left(Nr, Nw, Npa, Hw\right)$$

For more details about the mathematical model, see [25].

4. PARALLEL DEVS AND DEVS/SOA IMPLEMENTATION

In this section we describe how to use parallel MOEAs in a DEVS environment to solve the exploration of DDTs in embedded applications described in Section 3. 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 DEVS atomic model runs a different population as a thread. The number of individuals is the same as in the sequential version.

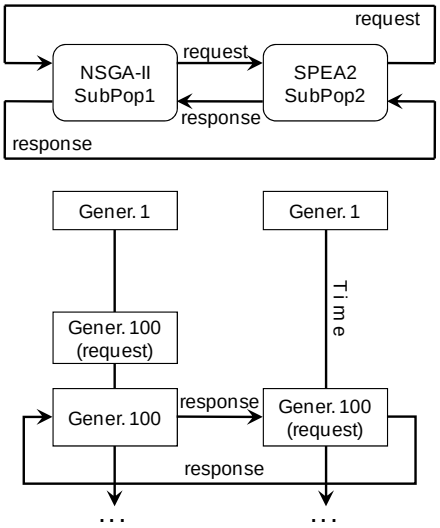

Figure 6. A graphic representation of the DEVS model (multi-core architecture) and it evolution over time

Figure 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 include 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 Figure 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 neighbour on a ring communication topology [5]. 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 atomic models; (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, and extract the first individual. Moreover, since NSGA-II is faster than SPEA2 ($O(mN^2)$ vs. $O(N^3)$, where N is the population size and m is the number of objectives), NSGA-II it finishes first while SPEA2 is still exploring early generations. Thus, as Figure 6 depicts, our migration policy is synchronized every 100 generations.

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

(1) NS^2 configuration: Four atomic models executing NSGA-II, SPEA2, NSGA-II and SPEA2.

(2) S^4 configuration: Four atomic models, but running all of them SPEA2 algorithm

(3) N^4 configuration: Four atomic models using the NSGA-II algorithm.

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

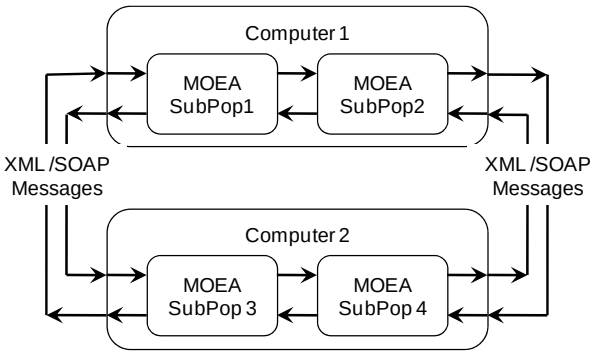

Figure 7. A graphic representation of the DEVS model (multi-core/distributed architecture)

The algorithm shown in Figure 6 follows a multi-threaded design, which is suitable to be executed in multi-core architectures. The 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 (models) are sent between different workstations using web services [20]. Figure 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. For simplicity our atomic models are suited for a ring topology. Experiments with other topologies are left for future study.

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 [18]. For this application we logged 3128 variables and the 10 possible DDTs contained in Table 2, which can cover almost all of the real-life embedded multimedia applications.

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 Figure 8.

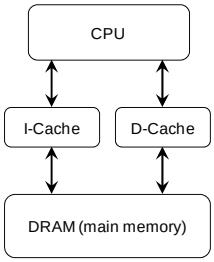

Figure 8. System architecture

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 [28] and [16], respectively. The processor and memory specification is described in Table 3.

Table 3. System specification

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

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 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 [34] to measure convergence. Let *PF′*, *PF″* be two sets of non-dominated solutions. The coverage metric can be defined as follows:

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

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′*. It is noted that both *C(PF′,PF″)* and *C(PF″,PF′)*, has 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 that in *PF′*.

5.2.2 Spread

A spread metric determines the maximum range represented by the nondominated solutions in each objective space. It was introduced by Ranjithan [24]. A higher value of the spread metric indicates a better performance. It is defined as

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

where *N* is the number of objectives.

5.2.3 Spacing

Schott proposed a metric which allows to measure the distribution of vectors throughout PF [27]. It is defined as:

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

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

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

5.3 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 [10].

Table 4. Example of an individual

D	*AR*	*AR*	*SLL*	*…*	*DLL*
V	v_1	v_2	v_3	…	v_m

Table 4 shows the representation of a chromosome. Genes are represented in the first row (gray shaded cells). Each of the chromosomes represents the set of DDT that should be used to instantiate all the corresponding variables in the application from Table 2. For example, the second variable $v_2 \in \mathbf{V}$ will be instantiated by an array (AR). A chromosome contains *m* genes, where *m* is the number of the variables logged in the application, *m=size(**V**)*. We may use an integer to represent the values of a gene, and the constraint a gene must satisfy is: $1 \leq ddt \leq size(\mathbf{D})$

Consequently, if an application contains *m* variables, each individual (chromosome) has to be constituted by *m* integer fields (i.e., *m* 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.

6. EXPERIMENTAL RESULTS

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 5. 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 atomic model, and the number of generations is set to 8000.

Table 5. Parameters for both sequential algorithms

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

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 three platforms: (1) AMD Sempron 3600+ 2GHz with 1GB DDR memory, (2) Intel Core 2 CPU 6600 2.40GHz with 2GB DDR memory, and (3) 16 workstations AMD Opteron Dual Core 2GHz with 4GB DDR memory connected via 100Mbps Ethernet network.

6.1 Sequential DEVS 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+ 2GHz with 1GB DDR memory. 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 [31], [9]. 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 variables) 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, 954× faster.

Table 6. 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 [3]	10.80×10^3 s.
NSGA-II	1.90×10^3 s.
SPEA2	3.03×10^3 s.

6.2 Multi-core DEVS 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 2.40GHz with 2GB DDR memory. The coverage, spread and the spacing values are calculated by averaging results of 100 trials.

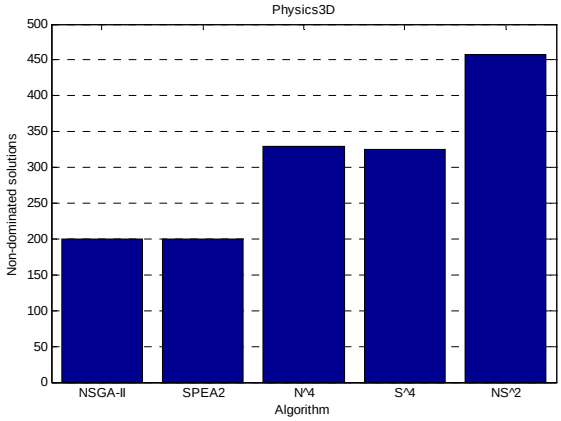

Figure 9. Non-dominated individuals obtained by NSGA-II, SPEA2, N^4, S^4 and NS^2

Figure 9 depicts the number of non-dominated individuals obtained in the best population. NSGA-II offers the same non-dominated individuals as 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.

Table 7. 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	----

Regarding convergence comparisons, Table 7 shows that the coverage values of NS^2 are better than any other algorithm. For example, $C(NS^2,NSGA\text{-}II) > C(NSGA\text{-}II, NS^2)$ 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.

Table 8. Spread and spacing for the five algorithms

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

Similar results are obtained using the average spread and spacing metrics. Table 8 indicates that the higher spread is found by parallel algorithms in all cases.

Table 9. 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.

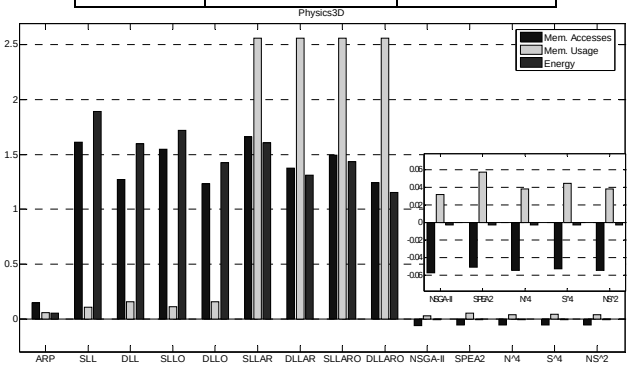

Figure 10. Overall results for different design metrics coming from various sets of DDT implementations (logarithmic scale)

Table 9 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 712 s. for NSGA-II when two processors are used and 1900 s. when one processor is used, giving a speed-up of 63%.

For comparative reasons, we present Figure 10 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 were 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 Figure 5. Furthermore, as this figure indicates, NS^2 offered the best solution among objectives.

6.3 Multi-core DEVS/SOA architecture

Finally, the NS^K configuration was distributed on a set of 16 workstations AMD Opteron Dual Core 270 2GHz with 4GB 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

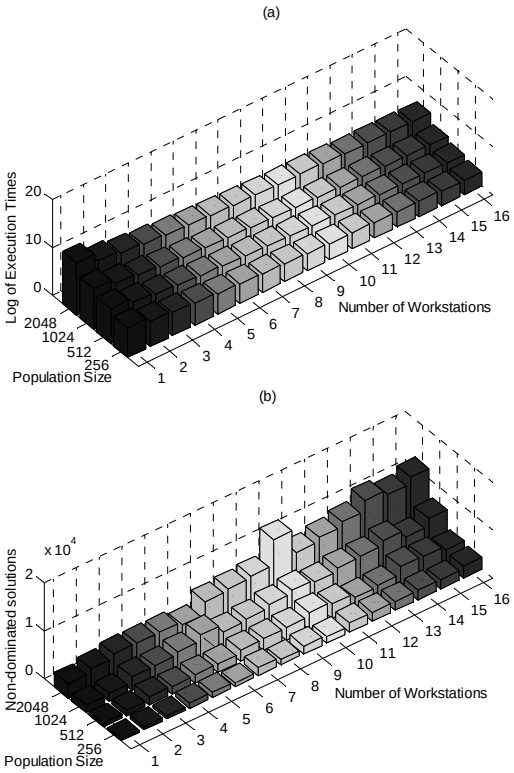

Figure 11. Execution times (a) and non-dominated solutions (b) as a function of the number of workstations. Each workstation executes two DEVS atomic models

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 (256, 512, 1024, and 2048). The tests were performed by changing only the number of workstations in order to observe and study the increase in performance (speedup). In all these cases the number of generations was set to 8000. The external archive size of each island was set to the entire population size, i.e., 256, 512, 1024 and 2048.

In light of the results presented in Figure 11a, as the size of the population increased, the performance of the parallel version improved proportionally to the number of islands. Also, Figure 11b 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.

7. CONSLUSIONS 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 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 63% with respect to the non parallel implementation.

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.

In addition to doing performance evaluation of proposed NS^K algorithm, we have attempted to evaluate the utility of DEVS/SOA infrastructure. This study is by far the first ever conducted study on distributed DEVS/SOA infrastructure. We used 16 workstations each running DEVS/SOA infrastructure. Not only it validated the DEVS/SOA architecture as a distributed simulation platform, it allows us to use it for benchmarking studies for various other applications. 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 linear 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 paper. Further, comparison of DEVS/SOA performance with other distributed simulation frameworks is underway. The experiments designed in this study will be performed on other frameworks to conduct benchmarks for DEVS/SOA simulation framework.

8. ACKNOWLEDGMENTS

Omitted for blind review

9. REFERENCES

[1] Arizona Center of Integrative Modeling & Simulation (ACIMS), http://www.acims.arizona.edu, 2008.

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

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

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

[5] Cantú-Paz, E. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.

[6] Catthoor, F., Danckaert, K., Kulkarni, C., Brockmeyer, E., Kjeldsberg, P. G., Achteren, T. V. and Omnes, T. *Data access and storage management for embedded programmable processors*. Kluwer Academic Publishers, 2002.

[7] Coello, C. A Comparative Survey of Evolutionary-based Multiobjective Optimization Techniques. *Knowledge and Information Systems, 1* (1999), 269-308.

[8] Corne, D. W., Jerram, N. R., Knowles, J. D. and Oates, M. J. PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (2001), 283-290.

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

[10] Deb, K. Multiobjective Optimization using Evolutionary Algorithms. John Wiley and Son Ltd., 2001.

[11] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6, 2* (2002), 182-197.

[12] Edler, J. Dinero IV Trace-Driven Uniprocessor Cache Simulator. http://pages.cs.wisc.edu/markhill/DineroIV, 2008.

[13] Fernandez, J. M., Vila, P., Calle, E. and Marzo, J. L. 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)* (2007), 266-271.

[14] Fonseca, C. M. and Fleming, P. J. Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA 1993)* (1993), 416-423.

[15] Hajela, P. and Lin, C. Y. Genetic search strategies in multicriterion optimal design. *Structural Opt., 4* (1992), 99-107.

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

[17] Horn, J., Nafpliotis, N. and Goldberg, D. E. A niched Pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation* (1994), 82-87.

[18] Kharevych L. and Khan R. 3D Physics Engine for Elastic and Deformable Bodies. http://www.cs.umd.edu/Honors/reports/kharevych.html.

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

[20] Mittal, S., Risco-Martín, J. L. and Zeigler, B. P. DEVS-Based Web Services for Net-centric T&E. In *Summer Computer Simulation Conference, SCSC 2006* (2006).

[21] Mittal, S., Risco-Martin, J. L. and 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] de Toro Negro, F., Ortega, J., Ros, E., Mota, S., Paechter, B. and Martín, J. PSFGA: Parallel Processing and Evolutionary Computation for Multiobjective Optimisation. *Parallel Computing, 30, 5-6* (May-June 2004), 721-739.

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

[24] Ranjithan, S. R., Chetan, S. K. and Dakshima, H. K. Constraint Method-Based Evolutionary Algorithm (CMEA) for Multiobjective Optimization. In *First International Conference on Evolutionary Multi-Criterion Optimization* (2001), 299-313.

[25] Risco-Martin, J. L., Atienza, D., Hidalgo, J. I. and 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] 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* (1985), 93-100.

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

[28] Shivakumar P. and Jouppi N. P. *Cacti 3.0: An Integrated Cache Timing, Power, and Area Model*, Compaq Computer Corporation, 2001/2, 2001.

[29] Veldhuizen, D. A. V., Zydallis, J. B. and Lamont, G. B. Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation, 7, 2* (April 2003), 144-173.

[30] Wilson, L. and Moore, M. Cross-pollinating parallel genetic algorithms for multiobjective search and optimization. *International Journal of Foundations of Computer Science, 16, 2* (April 2005), 261-280.

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

[32] Xiong, S. and Li, F. Parallel Strength Pareto Multi-objective Evolutionary Algorithm for Optimization Problems. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)* (2003), 2712-2718.

[33] Zeigler, B. P., Kim, T. and Praehofer, H. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press, 2000.

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

[35] Zitzler, E., Laumanns, M. and 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* (2002), 95-100.

[36] Zydallis, J. B., van Veldhuizen, D. A. and Lamont, G. B. A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA-II. In *First International Conference on Evolutionary Multi-Criterion Optimization* (2001), 226-240.