J Supercomput (2017) 73:302–315
DOI 10.1007/s11227-016-1862-0

Using low-power platforms for Evolutionary Multi-Objective Optimization algorithms

J. J. Moreno[1] · G. Ortega[1] · E. Filatovas[2] · J. A. Martínez[1] · Ester M. Garzón[1]

Published online: 23 September 2016

Abstract Nowadays, the application of Evolutionary Multi-Objective Optimization (EMO) algorithms in real-time systems receives considerable interest. In this context, the energy efficiency of computational systems is of paramount relevance. Recently, the use of embedded systems based on heterogeneous (CPU + GPU) platforms is consistently increasing. For example, NVIDIA Jetson cards are low-power computers designed for development of embedded applications. They incorporate Tegra processors which feature a CUDA-capable GPU. This way, Jetson cards can be considered as a prototype of low-power computer of High-Performance Computing. In this work,

This work has been partially supported by the Spanish Ministry of Science throughout projects TIN2015-66680 and CAPAP-H5 network TIN2014-53522, by J. Andalucía through projects P12-TIC-301 and P11-TIC7176, and by the European Regional Development Fund (ERDF). Ernestas Filatovas has been partially granted by the European COST Action IC1305: Network for sustainable Ultrascale computing (NESUS).

✉ Ester M. Garzón
gmartin@ual.es

J. J. Moreno
jrm069@inlumine.ual.es

G. Ortega
gloriaortega@ual.es

E. Filatovas
ernest.filatov@gmail.com

J. A. Martínez
jmartine@ual.es

[1] Group of Supercomputation-Algorithms, Department of Informatics, University of Almería, ceiA3, 04120 Almería, Spain

[2] Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, Sauletekio av. 11, 10223 Vilnius, Lithuania

our interest is focused on the NSGA-II algorithm, a well-known representative of EMO algorithms. The strength of NSGA-II lies in its Non-Dominated Sorting (NDS) procedure of a population of individuals. Our purpose on the low-power computers is twofold: to define and evaluate the parallel NSGA-II versions with major focus on NDS procedure on the Jetson platforms and to determinate the size of NSGA-II problems which can be solved. The results show that the parallel version which achieves the best performance depends on the objectives functions and the frequencies of the clocks of the cores and memory of the GPU. The analysis of the results shows the capability of the Jetson as a low-consumption platform which allows to accelerate the execution of instances of the state-of-the-art EMO algorithm—NSGA-II.

Keywords Evolutionary Multi-Objective algorithms · Energy efficiency · Low-power platform · Jetson · NSGA-II

1 Introduction

Currently, real-time monitoring and control systems based on Multi-Objective Optimization (MOO) are actively developed and applied in different areas. Recently, several real-time traffic signal control adaptive systems that are able to totally adapt the signal timing to the traffic situation based on multi-objective optimization have been developed [3,12,20,24]. An Evolutionary Multi-Objective (EMO) algorithm was applied for operation optimization in a real-time water supply system [32]. In [2], a controller based on a EMO algorithm was used in simulation optimization methodology to solve a real-time multi-objective dispatching decision problem. In [26], the NSGA-II [6] algorithm was applied in microcontroller-based polarization control system. In [10], a microcontroller-based on Artificial Neural Network and the NSGA-II algorithm was developed. It could be applied for different fast real-time intelligent control applications with a non-linear model predictive strategies. There are still many more real-time systems in which optimization problems are solved by EMO algorithms [5,25]. Summarizing, MOO real-time systems are of great interest and the integration of EMO algorithms enables to provide fast enough good solutions. Determination of the Pareto front is the main goal of MOO. However, it is impossible for some problems to identify the exact Pareto front due to reasons such as continuity of the front, nonexistence of analytical expression or complexity of the problem being solved. On the other hand, in real-world applications it is usually not necessary to find the whole Pareto front, but rather its approximation.

Some well-known EMO algorithms to approximate the Pareto front are NSGA-II [6], PAES [21], MOAE/D [39], IBEA [41], SPEA2 [42], etc. Several phases can be identified in most of them: evaluation of an objective function, Pareto dominance ranking (Non-Dominated Sorting) and genetic operations. The most computationally expensive phase is the Pareto dominance ranking. Examples of EMO approaches based on Pareto dominance ranking are PESA-II [4], NSGA-II [6], R-NSGA-II [7], Synchronous R-NSGA-II [15], etc. In this work, NSGA-II is taken as an example of EMO algorithm to be adapted to low-power platforms and evaluated on them.

Currently, embedded systems for real-time computing combine low-consumption and heterogeneous systems (CPU and GPU). Representative examples of this kind of platforms are the NVIDIA Jetson platforms, widely used in real-time systems.[1] In the context of real time, the use of NVIDIA Jetson platforms can be useful to solve MOO problems using EMO algorithms for two main reasons: (1) the low-energy consumption because of the specific technology of this kind of platforms; (2) the exploitation of the multicore and the GPU of the NVIDIA Jetson allows the acceleration of programs. For this aim, OpenMP and CUDA can be used as interfaces to develop parallel programs.

Some parallel versions of NSGA-II have been developed: in [9,11,13] parallel versions of NSGA-II algorithm based on master–slave paradigm are presented, where population is distributed among the workers to speed up the process of functions evaluation; in [22] several parallel strategies where the Pareto ranking is parallelized in NSGA-II are proposed, and are experimentally investigated when solving the competitive facility location problem in [23]; in [40] a parallel version with individual migration of NSGA-II is investigated; in [38] an EMO parallel algorithm for GPU was developed and tested on several two- and three-objective benchmark problems in [37]; a multi-objective version of Differential Evolution was parallelized on GPU in [33]. Usually, the cost of the evaluation of the objective function in EMO approaches is not very high in the sense of computation time. Hence, most of the computation time is used by checking the Pareto dominance ranking, which is implemented in the Fast Non-Dominated Sorting (FNDS) procedure [6]. The complexity of the FNDS procedure is $O(MN^2)$, where M is the number of objectives and N the total size of population. As FNDS consumes most of the NSGA-II runtime, the acceleration of the FNDS procedure is mandatory to speed up NSGA-II, it has been justified in [35]. The reduction of the complexity order of the FNDS has been a focus of interest for researchers [14,19,27,34,36]. Some improvements were implemented by developing more efficient sorting strategies, however, computational burden of the FNDS procedure has a complexity $O(MN^2)$ in the worst case for all the approaches. Thus, parallel strategies should be considered to accelerate the computation of the procedure.

Recently, a novel NSGA-II parallel implementation on a GPU, focusing on the FNDS procedure and achieving promising speedups has been proposed in [16]. The NDS version of complexity $O(MN^3)$ is accelerated on GPU and every thread computes the dominance of every individual without writing in auxiliary structures. Therefore, it is a GPU parallel NDS version with redundant dominance comparisons (hereinafter this version is referred to as Gupta-NDS). Moreover, an efficient parallel version of the FNDS procedure has been formally presented in [35], but its experimental analysis is very limited. Both works are very related to our proposal, since our objective is the acceleration of the Pareto dominance ranking on the integrated GPU of the embedded systems. However, our proposal, referred to as Efficient Fast NDS (EFNDS), defines a new data structure to store the dominance information which efficiently allows to compute the individuals that dominate the population by using shuffled reductions on modern GPUs. Additionally, another GPU version of NDS

[1] http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html.

based on Gupta-NDS and similar to the proposal [16] has been also developed. The performance of both implementations on Jetson has been evaluated. Comparative results show that the performance on GPU of EFNDS overcomes Gupta-NDS when the number of fronts and the memory bandwidth are high.

Bearing in mind the limited computational resources on the Jetson platform and our proposal to efficiently compute the Pareto dominance ranking (EFNDS), it is relevant to identify the scale of the problem that can be solved with NSGA-II algorithm on this kind of platforms.

The main contributions of this work are: (1) a novel proposal (EFNDS) to compute the Pareto dominance ranking on the GPU with a better performance than others proposed in the literature. EFNDS is very appropriate for embedded systems because it only consumes energy to compute useful computation with memory requirements of medium size which are supplied by these kind of systems. Results show that this proposal achieves better performance than Gupta-NDS when several fronts are computed; (2) the estimation of the problem sizes that can be solved on a Jetson platform as a prototype of modern embedded system. The rest of this paper is organized as follows. In Sect. 2, the definition of the multi-objective problem is provided. Section 3 describes the Efficient Fast Non-Dominated Sorting procedure (EFNDS) as well as its parallel implementations. An experimental evaluation of the parallel implementations on a Jetson platform is discussed in Sect. 4. Finally, Sect. 5 shows the conclusions of this work.

2 Description of the problem

A multi-objective minimization problem is formulated as follows [28]:

$$\min_{\mathbf{x}\in\mathbf{S}} \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})]^T \tag{1}$$

where $\mathbf{z} = \mathbf{f}(\mathbf{x})$ is an *objective vector*, defining the values for all objective functions $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})$, $f_i : \mathbb{R}^V \rightarrow \mathbb{R}, i \in \{1, 2, \ldots, M\}$, where $M \geq 2$ is the number of objective functions; $\mathbf{x} = (x_1, x_2, \ldots, x_V)$ is a vector of variables (*decision vector*) and V is the number of variables $\mathbf{S} \subset \mathbb{R}^V$ is *search space*, which defines all feasible decision vectors.

A decision vector $\mathbf{x}' \in \mathbf{S}$ is a *Pareto optimal solution* if $f_i(\mathbf{x}') \leqslant f_i(\mathbf{x})\, i = 1, \ldots, M$ for all $\mathbf{x} \in \mathbf{S}$ and $f_j(\mathbf{x}') < f_j(\mathbf{x})$ for at least one j. Objective vectors are defined as optimal if none of their elements can be improved without worsening at least one of the other elements. An objective vector $\mathbf{f}(\mathbf{x}')$ is Pareto optimal if the corresponding decision vector $\mathbf{x}'$ is Pareto optimal. The set of all the Pareto optimal decision vectors is called the *Pareto set.* The region defined by all the objective function values for the Pareto set points is called the *Pareto front*.

For two objective vectors $\mathbf{z}$ and $\mathbf{z}'$, $\mathbf{z}'$ *dominates* $\mathbf{z}$ (or $z' \succ z$) if $z'_i \leqslant z_i$ for all $i = 1, \ldots, M$ and there exists at most one j such that $z'_j < z_j$. In EMO algorithms, the subset of solutions in a population whose objective vectors are not dominated by any other objective vector is called the *non-dominated set*, and the objective vectors are called the *non-dominated objective vectors*. The main aim of the EMO algorithms

is to generate well-distributed non-dominated objective vectors as close as possible to the Pareto front.

NSGA-II [6] is the most widely used and well-known EMO algorithm for approximating the Pareto front that is based on NDS. It has been considered to analyse the energy efficiency of EMO algorithms when different number of CPU cores and/or GPU cards are exploited. The steps of NSGA-II are described in Algorithm 1.

Algorithm 1 NSGA-II

Step 1: Generate a random initial population P_0 of size N.
Step 2: Sort the population to different non-domination levels (fronts) using, and assign each individual a fitness equal to its non-domination level (1 is the best level).
Step 3: Create an offspring population of size N using binary tournament selection, recombination and mutation operations (parents with larger crowding distance are preferred if their non-domination levels are the same).
Step 4: Combine the parent and the offspring populations and create a population R.
Step 5: Reduce the population R to the population P of size N: sort the population R into different non-dominated fronts; fill the population P with individuals from population R starting from the best non-dominated front until the size of P is equal to N; if all the individuals in a front cannot be picked fully, calculate a crowding distance and add individuals with the largest distances into the population P.
Step 6: Check if the termination criterion is satisfied. If yes, go to Step 7, else return to Step 2.
Step 7: Stop.

Step 2 and Step 5 of the Algorithm 1 are devoted to computing the NDS procedure, which is the most computationally expensive task in the NSGA-II.

3 Efficient Fast Non-Dominated Sort

The procedure referred to as 'Fast Non-Dominated Sort' was proposed by Deb et al. in [6] to compute the Steps 2 and 5 of Algorithm 1. In such proposal, the dominance information of every individual consists of: the number of dominator individuals, the number and the list of dominated individuals. Then, the fronts are computed from this dominance information. The complexity order of this process is $O(MN^2)$ and it requires a storage of $O(N^2)$. Previous proposals of NDS have a high complexity $O(MN^3)$ and a high number of redundant comparisons between pairs of individuals, but as a counterpart their storage requirements grows up as $O(N)$. As mentioned in Sect. 1, there are several proposals to define efficient sorting strategies which reduce the redundant comparisons, with a computational burden of complexity $O(MN^2)$ in the worst case. The redundancy level of NDS increases as the number of fronts increases and can be an efficient approach if the population is classified in few fronts. So, two kinds of approaches to compute NDS can be distinguished: FNDS with a previous dominance computation which has few redundant comparisons and an intensive memory use; NDS with redundant comparisons to evaluate the dominance without additional memory requirements.

In this work, a new approach to compute and store the dominance information is proposed. The corresponding FNDS procedure is described in Algorithm 2 and referred to as Efficient FNDS (EFNDS). It selects N individuals as 'Elite population'

from the initial population P_0 with $2N$ individuals (step 5 of Algorithm 1). Algorithm 2 includes two phases. The Phase 1 computes the dominance matrix, D, of dimensions $2N \times 2N$, for the initial population and the Phase 2 selects the individuals of the first fronts until half of them are selected in the 'elite' set. The structure of the Phase 2 is well known. Our proposal is based on the previous computation of a dominance matrix D whose elements, $D_{i,j}$, store if the individual P_i is dominated by P_j; that is, $D_{i,j} = 1$ if P_j dominates P_i and $D_{i,j} = 0$ in other case.

This way, the idea for checking if P_i is dominated by the population P (line 14 of Algorithm 2) is based on the computation of the number of individuals of P which dominate P_i as $d_i = |\{D_{i,j} = 1;\ \forall P_j \in P\}|$, so if $d_i = 0$ then P_i is not dominated by P and it belongs to the new front. Therefore, the computation of d_i can be carried out reading the corresponding values $D_{i,j}$ on the ith row of D.

Algorithm 2 Efficient Fast Non-Dominated Sorting procedure (EFNDS) to compute Step 2 of Algorithm 1

Require: P_0: population M: number of objective functions

Phase 1: Compute the information of dominance

1: $2N \leftarrow |P_0|$
2: **for** $i \leftarrow 1$ **to** $2N$ **do**
3: **for** $j \leftarrow 1$ **to** $2N$ **do**
4: **for** $l \leftarrow 1$ **to** M **do**
5: Check dominance between individuals P_i and P_j for the objective l
6: $D_{i,j} = 0$
7: **if** P_j dominates P_i **then**
8: $D_{i,j} = 1$

Phase 2 Compute fronts from the dominance information of Phase 1

9: $P = P_0$; $Elite \leftarrow \emptyset$; $rank = 0$
10: **while** $|Elite| < N$ **do**
11: $rank = rank + 1$
12: $F_{Rank} \leftarrow \emptyset$
13: **for** each $P_i \in P$ **do** (Check if P_i dominates P from the matrix D)
14: $d = \sum_{P_j \in P} D_{ij}$
15: **if** $d = 0$ (i.e. Pi is a dominator of P) **then**
16: $P \leftarrow P \backslash \{P_i\}$
17: $F_{Rank} \leftarrow F_{Rank} \cup \{P_i\}$
18: **if** $|Elite| + |F_{Rank}| > N$ **then**
19: $F_{Rank} \leftarrow N - |Elite|$ individuals of F_{Rank} with higher crowding distance
20: $Elite \leftarrow Elite \cup F_{Rank}$
21: **return** $Elite$

3.1 EFNDS on GPU

Compute Unified Device Architecture (CUDA) is the parallel interface introduced by NVIDIA to help develop parallel codes using C or C++ language. CUDA provides some abstraction to the GPU hardware, and it provides the SIMT (Single Instruction, Multiple Threads) programming model to exploit the GPU [31]. However, the programmer has to take into account several features of the architecture, such as the

Algorithm 3 Host pseudocode to compute EFNDS on GPU based on the kernels $CuDominance$ and $CuFronts$

Require: P_0: population M: number of objective functions
Phase 1: Compute the information of dominance
1: $2N \leftarrow |P_0|$
2: $D = CuDominance()$ (**Algorithm 4 is computed on GPU**)
Phase 2 Compute fronts from the dominance information of Phase 1
3: $P = P_0$; $Elite \leftarrow \emptyset$; rank=0
4: **while** $|Elite| < N$ **do**
5: $rank = rank + 1$
6: $F_{Rank} \leftarrow \emptyset$
7: $F_{Rank} = CuFronts()$ (**Algorithm 5 is computed on GPU**)
8: **if** $|Elite| + |F_{Rank}| > N$ **then**
9: $F_{Rank} \leftarrow N - |Elite|$ individuals of F_{Rank} with higher crowding distance
10: $Elite \leftarrow Elite \cup F_{Rank}$
11: **return** $Elite$

topology of the multiprocessors and the management of the memory hierarchy. For the execution of the program, the CPU (called host in CUDA) performs a succession of parallel routine (kernels) invocations to the device. The input/output data to/from the GPU kernels are communicated between the CPU and the 'global' GPU memories. Integrated GPUs (such as the GPUs in Tegra processors) share the low levels of cache memory with the CPU and it is not necessarily the replication of the input–output data of the GPU [30]. GPUs have hundreds of cores which can collectively run thousands of computing threads. Each core, called Scalar Processor (SP), belongs to a set of multiprocessor units called Streaming Multiprocessors (SM). The SMs are composed of 192 (or 128) SPs on Kepler (or Maxwell) GPU architectures [17,29,31]. This way, the GPU device consists of a set of SMs and each kernel is executed as a batch of threads organized as a grid of thread blocks [1].

It is relevant to underline that the computation of dominance consumes most of the runtime of FNDS procedures when it is executed in sequence [16,35]. Our approach to compute D with a high level of parallelism, can efficiently compute the dominance information on the GPU architecture. Moreover, the fronts computation can also be accelerated on GPU by the use of the fast shuffled reductions of CUDA. This way, our parallel EFNDS version efficiently computes D and the corresponding fronts on GPU.

Algorithm 3 shows the host pseudocode to compute EFNDS on GPU. It is based on two kernels: $CuDominance$ (line 2) which computes the matrix D and $CuFronts$ (line 7) which computes one front from the dominance information stored in D. $CuDominance$ is executed once and $CuFronts$ is iteratively computed until a $Elite$ population of N individuals is classified. To fit the classified population to N individuals, a subset of last front is selected according to the crowding distance on CPU.

Algorithm 4 shows the procedure to compute the dominance matrix on GPU, $CuDominance$. The input is the population of $2N$ individuals. This way, $4N^2$ threads are activated to check concurrently the dominance between all pairs of individuals and update the elements of D without any synchronization among them.

Algorithm 4 $CuDominance$ kernel to compute the dominance matrix of EFNDS on GPU

Require: P_0: population M: number of objective functions
CuDominance Kernel
1: $2N \leftarrow |P_0|$
2: **if** $idx < 2N \times 2N$ **then**
3: $i = \lfloor idx/2N \rfloor$; $j = idx\%2N$
4: **for** $l \leftarrow 1$ **to** M **do**
5: Check dominance between individuals P_i and P_j for the objective l
6: $D_{i,j} = 0$
7: **if** P_j dominates P_i **then**
8: $D_{i,j} = 1$

Algorithm 5 $CuFronts$ kernel to compute the set of fronts of EFNDS on GPU

1: $idB = blockIdx.x$; Cuda block identification
2: $idx = threadIdx.x$; Cuda thread identification
3: $BlockSize = blockDim.x$; number of threads in every Cuda block
4: **if** $idx < 2N \times BlockSize$ AND $P_{idB} \in P$ **then**
Check if P_{idB} dominates the population P from the matrix D
5: **for** $i = 0; i < 2N; i = i + BlockSize$ **do**
6: **if** $P_i \in P$ **then**
7: $d = d + D_{idB,i}$
8: $d_{idB} = BlockReduceSum(d)$ (Fast Shuffled Reduction into Threads Blocks)
9: **if** $d_{idB} = 0$ (i.e. P_{idB} is a dominator of P) **then**
10: **if** $idx == idB \times BlockSize$ (First Thread in every Cuda Block) **then**
11: $P \leftarrow P \backslash \{P_i\}$
12: $F_{Rank} \leftarrow F_{Rank} \cup \{P_i\}$
13: **return** F_{Rank}

When $CuDominance$ finishes, the matrix D is on memory and the fronts can be computed by successive executions of $CuFronts$. Algorithm 5 describes the procedure to compute one front on GPU. One thread block is activated for every individual. Therefore, the threads of every block compute the reduction of a particular row of D and a fast reduction scheme based on shuffled functions is applied.[2] The shuffled functions enable a thread to directly read a register from another thread in the same warp; therefore, threads can compute fast reductions avoiding memory accesses.

It is noteworthy that the output of $CuFronts$ is written in a vector F of dimension $2N$, where every element F_i stores the front number for the individual i. So, when $CuFronts$ is executed, a new selection of individuals is classified in the new front. Additionally, F defines the set of individuals in the population P since if $F_i = 0$ then the individual i has not classified and it is a potential individual of posterior fronts, i.e. $i \in P$. Posterior fronts classifications will be computed only for the individuals with $F_i = 0$. Therefore, F also defines the population P to compute a new front. Figure 1 illustrates the computation of the fourth front on the GPU. It shows the data structures and their threads mapping involved in the computation of a front.

[2] https://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/.

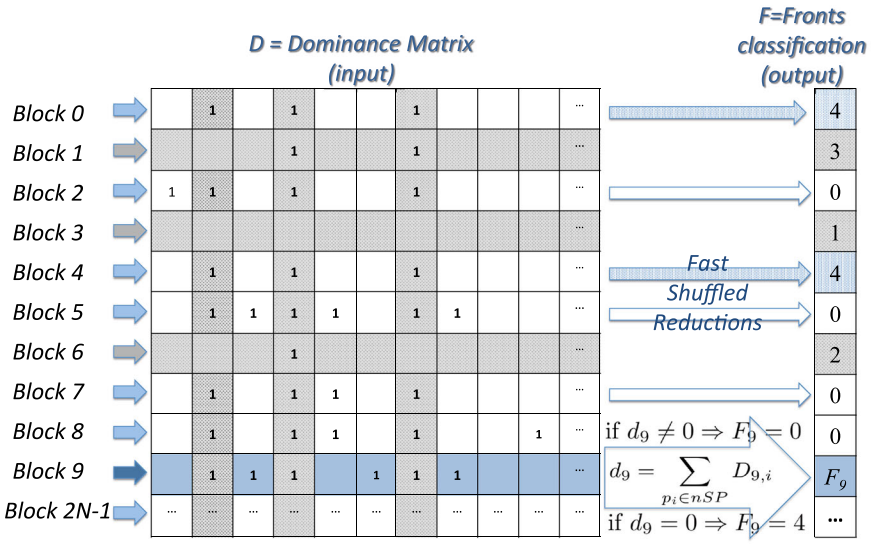

Fig. 1 Computation of the fourth front from D matrix when $CuFronts$ kernel is executed on GPU

Next section evaluates the improvement in the performance of NSGA-II on a Jetson platform when the EFNDS is computed on the GPU integrated in the Tegra processor.

4 Experimental evaluation

In this section, the performance of two parallel GPU versions of NSGA-II has been evaluated on the Jetson platform. These parallel versions are based on: (1) the EFNDS algorithm described above; (2) the Gupta-NDS version developed in [16]. Additionally, the sequential NSGA-II version 1.1.6 of K. Deb[3] has also been evaluated.

DTLZ2 test problem has been considered in the evaluation because the number of objective functions can be defined by the user [8,18]. DTLZ2 problem was specially designed for evaluating multi-objective algorithms in this context. Moreover, it can be configured to have high computational demands by establishing a large number of objectives and variables. The formulation of the problem is as follows:

$$\min f_1(\mathbf{x}) = (1 + g(x)) \prod_{i=1}^{v-1} \cos\left(\frac{x_i \pi}{2}\right)$$

[3] http://www.iitk.ac.in/kangal/codes.shtml.

$$
\begin{aligned}
&\min f_2(\mathbf{x}) = (1 + g(x))\sin\left(\frac{x_{v-1}\pi}{2}\right)\prod_{i=1}^{v-2}\cos\left(\frac{x_i\pi}{2}\right)\\
&\ldots\\
&\min f_l(\mathbf{x}) = (1 + g(x))\sin\left(\frac{x_{v-l+1}\pi}{2}\right)\prod_{i=1}^{v-l}\cos\left(\frac{x_i\pi}{2}\right)\\
&\ldots\\
&\min f_v(\mathbf{x}) = (1 + g(x))\sin\left(\frac{x_1\pi}{2}\right)
\end{aligned}
\tag{2}
$$

where $g(\mathbf{x}) = \sum_{i=v}^{n}(x_i - 0.5)^2$, $x_i \in [0, 1]$. The number of variables n is selected according to the equation $n = v + k - 1$, with a suggested value of $k = 10$.

Other test problems have been evaluated obtaining similar results (DTLZ5 and DTLZ7). However, for the sake of clarity they have not been included in this study.

The Jetson platform has been selected as a prototype of a low-power platform because: (1) its Tegra processor contains a multicore-CPU and a GPU and they can support the parallel programming interfaces as extended as CUDA or OpenMP; (2) the clock frequencies of cores and memory buses of the GPU and the CPU of the Jetson can be modified to control the energy consumption.

The hardware setup we use is based on a NVIDIA Jetson TK1 development board,[4] embedding a Tegra K1 SoC (System on a Chip) processor with 2 GB of DDR3L RAM.

On the one hand, the Tegra K1 includes an NVIDIA GPU with 192 CUDA Kepler cores and an ARM quad-core Cortex-A15 variant of low-power architecture at 2.32 GHz. The board runs an extended version of Linux Ubuntu 14.04.4 LTS adapted for the ARM architecture. The codes for the ARM are compiled using gcc v.4.8.4, while the codes developed for the NVIDIA GPU are compiled with nvcc v.6.5.12.

The memory requirements of the EMO problems depend on three parameters: the size of the population, the number of variables and the number of objective functions. These parameters have to be defined bearing in mind that the Jetson board has only 2 GB of physical memory (shared by the ARM CPU and the CUDA GPU). Therefore, we have considered instances of the NSGA-II problem which fit on the Jetson platform.

As mentioned above , the Tegra processors allow us to control the clock frequencies of the GPU, CPU and memory controller, that is, the efficiency of cores and/or memory of the TK1 board can be configured. Therefore, NSGA-II based on EFNDS and Gupta-NDS can be evaluated with different frequency configurations. The analysis of these results can determine the most used resources by every NSGA-II version.

As shown in Fig. 2, our implementation is dramatically influenced by the memory controller clock, while the performance of the Gupta version does not differ much. However, when we change the frequency of the GPU core clock, we can observe that our implementation keeps its performance, while Gupta is strongly penalized. This is consistent with our theoretical considerations, as Gupta's implementation does not store any dominant information in memory, while EFNDS heavily uses memory to reduce the number of redundant comparisons needed to evaluate the dom-

[4] http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.htm.

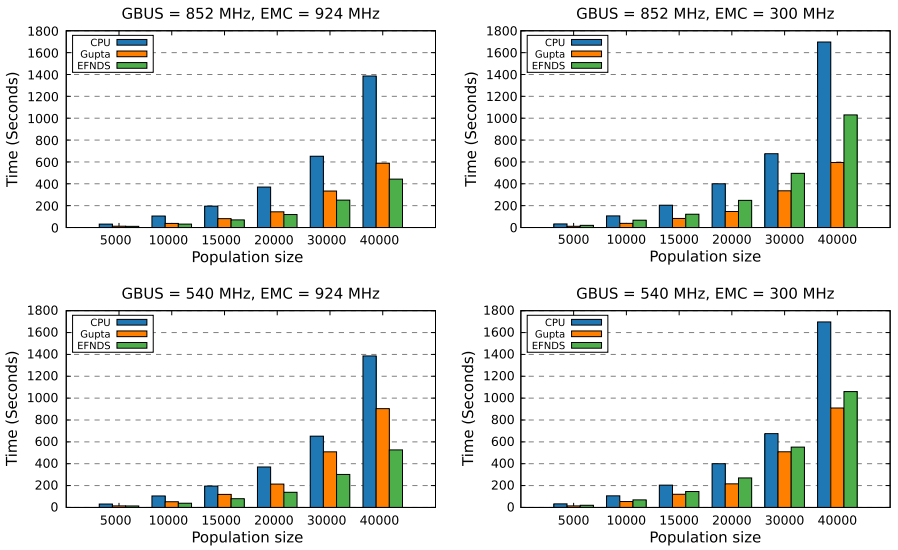

Fig. 2 Runtimes of the three versions of NSGA-II studied for DTLZ2 with $M = 5$, $N = 10{,}000$ and varying frequencies of the GPU core clock (GBUS) and the External Memory Controller clock (EMC)

inance between the individuals. Notice that the CPU performance is also affected by the EMC frequency. It proves that the memory bus is shared by CPU and GPU devices.

Table 1 shows the runtimes achieved by the NSGA-II versions based on sequence of K: Deb, parallel EFNDS, parallel Gupta-NDS and the parallel version which combines both. The results also show that there is no optimal implementation for all problems.

To always choose the best NDS implementation, we have also developed a hybrid version that switches between both implementations. As can be observed in the last column of Table 1, this strategy has remarkable results. This behaviour is shown in Fig. 3, where we plot the time spent on each generation for each strategy. As we can observe, EFNDS with many objectives and many fronts is faster while Gupta is better when there is only one front. We have concluded that this is a situation that commonly appears on the last generations of each execution, so we propose an adaptative approach which chooses the best one in each case.

It is assumed that the starting number of fronts is high. So, in both cases, the adaptative approach selects FNDS at the first generation. Then, it checks if the Gupta version can improve the performance when the number of fronts decreases. For DTLZ2 and $M = 5$, this happens in the last generations, so it switches from EFNDS to Gupta. For $M = 15$, EFNDS is always faster than Gupta even though most generations only compute one front, so the adaptative approach always selects FNDS.

Finally, the performance of all parallel GPU versions are improved over the sequential CPU version, with respectable acceleration factors, as shown in the last column of Table 1.

Table 1 Runtimes of NSGA-II based on: sequential version of NDS (DebSeq), parallel EFNDS, parallel Gupta-NDS and adaptative version (with its acceleration factor AF*)

Test	N	T (Deb-Sec)	T (Gupta)	T (EFNDS)	T (adaptative*)	AF*
dtlz2_5	1000	2.16	1.71	1.49	1.48	1.46
	2000	7.92	3.66	3.33	3.30	2.40
	5000	31.07	11.80	10.87	10.84	2.87
	10,000	105.09	37.31	31.39	31.13	3.38
	15,000	195.51	81.51	69.65	69.88	2.80
	20,000	370.08	143.78	118.79	118.97	3.11
	30,000	652.31	333.63	250.76	250.80	2.60
	40,000	1386.06	588.17	443.00	441.71	3.14
dtlz2_10	1000	4.63	2.44	2.34	2.23	2.08
	2000	15.73	5.06	5.60	5.00	3.14
	5000	90.15	14.94	19.03	14.17	6.36
	10,000	317.87	36.97	52.06	37.71	8.43
	15,000	832.19	101.32	116.44	98.96	8.41
	20,000	1679.62	127.78	179.96	123.95	13.55
	30,000	4428.19	306.81	356.20	301.76	14.67
	40,000	8358.40	530.24	631.68	525.52	15.91
dtlz2_15	1000	6.22	3.20	3.30	3.24	1.92
	2000	25.89	7.44	8.15	7.44	3.48
	5000	114.66	25.58	26.40	24.38	4.70
	10,000	498.46	75.92	74.61	74.37	6.70
	15,000	1304.82	171.61	156.57	156.81	8.32
	20,000	2676.36	270.62	250.85	249.97	10.71
	30,000	7083.76	593.78	535.67	535.59	13.23
	40,000	13,043.91	1046.28	925.82	925.54	14.09

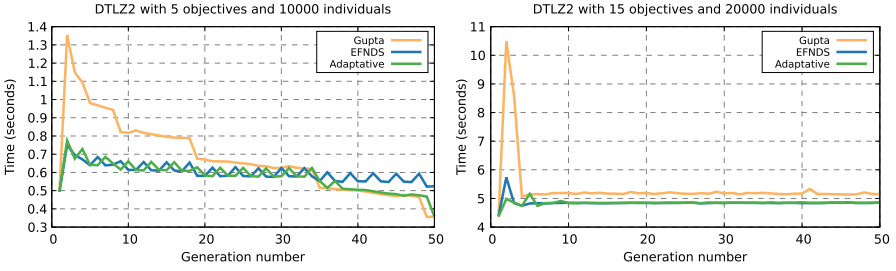

Fig. 3 Time evolution on each NSGA-2 iteration for the test problem DTLZ2 with $M = 5, 15$ and $N = 10{,}000, 20{,}000$

5 Conclusions and future works

In this work, a new parallel proposal to accelerate the NSGA-II for solving multi-objective problems has been proposed. It has been evaluated in comparison with the

recent proposal [16] on a Jetson TK1 as a prototype of modern embedded system of low power. The results have shown that EFNDS achieves better performance than Gupta when several fronts are computed and/or there is a high number of objectives. We also propose an adaptative version that switches between both of them, improving the performance of NSGA-II in all test cases.

Furthermore, we show that the Jetson TK1 embedded platform is adequate to accelerate EMO algorithms for large numbers of objectives and populations, achieving respectable performance and high acceleration factors.

As future works, we are planning to evaluate the energy efficiency of the adaptative approach on this platform. We are also testing the performance of our implementations on other GPU platforms.

References

1. Brodtkorb AR, Hagen TR, Sætra ML (2013) Graphics processing unit (GPU) programming strategies and trends in GPU computing. J Parallel Distrib Comput 73(1):4–13
2. Chang X, Dong M, Yang D (2013) Multi-objective real-time dispatching for integrated delivery in a fab using ga based simulation optimization. J Manuf Syst 32(4):741–751
3. Coll P, Factorovich P, Loiseau I, Gómez R (2013) A linear programming approach for adaptive synchronization of traffic signals. Int Trans Oper Res 20(5):667–679
4. Corne D, Jerram N, Knowles J, Oates M (2001) PESA-II: region-based selection in evolutionary multiobjective optimization. In: GECCO, pp 283–290
5. Cortés C, Sáez D, Milla F, Nuez A, Riquelme M (2010) Hybrid predictive control for real-time optimization of public transport systems' operations based on evolutionary multi-objective optimization. Transp Res Part C Emerg Technol 18(5):757–769
6. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
7. Deb K, Sundar J, Udaya Bhaskara Rao N, Chaudhuri S (2006) Reference point based multi-objective optimization using evolutionary algorithms. Int J Comput Intell Res 2(3):273–286
8. Deb K, Thiele L, Laumanns M, Zitzler E (2002) Scalable multi-objective optimization test problems. In: WCCI, pp 825–830
9. Dehuri S, Ghosh A, Mall R (2007) Parallel multi-objective genetic algorithm for classification rule mining. IETE J Res 53(5):475–483
10. Dendaluce M, Valera JJ, Gómez-Garay V, Irigoyen E, Larzabal E (2014) Microcontroller implementation of a multi objective genetic algorithm for real-time intelligent control. In: International Joint Conference SOCO13-CISIS13-ICEUTE13, pp 71–80. Springer
11. Domínguez J, Montiel O, Sepúlveda R, Medina N (2013) High performance architecture for NSGA-II. In: Recent Advances on Hybrid Intelligent Systems, pp 451–461. Springer
12. Dujardin Y, Vanderpooten D, Boillot F (2015) A multi-objective interactive system for adaptive traffic control. Eur J Oper Res 244(2):601–610
13. Durillo JJ, Nebro AJ, Luna F, Alba E (2008) A study of master-slave approaches to parallelize NSGA-II. In: IPDPS, pp 1–8. IEEE
14. Fang H, Wang Q, Tu YC, Horstemeyer MF (2008) An efficient non-dominated sorting method for evolutionary algorithms. Evol Comput 16(3):355–384
15. Filatovas E, Kurasova O, Sindhya K (2015) Reference point based multi-objective optimization using evolutionary algorithms. Informatica 26(1):33–50
16. Gupta S, Tan G (2015) A scalable parallel implementation of evolutionary algorithms for multi-objective optimization on GPUs. In: CEC, pp 1567–1574. IEEE
17. Harris M (2014) Maxwell: the most advanced CUDA GPU ever made. https://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/
18. Huband S, Hingston P, Barone L, While L (2006) A review of multiobjective test problems and a scalable test problem toolkit. IEEE Trans Evol Comput 10(5):477–506

19. Jensen MT (2003) Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms. IEEE Trans Evol Comput 7(5):503–515
20. Khamis MA, Gomaa W (2014) Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multi-agent framework. Eng Appl Artif Intell 29:134–151
21. Knowles J, Corne D (2000) Approximating the non-dominated front using the Pareto archived evolution strategy. Evol Comput 8(2):149–172
22. Lančinskas A, Žilinskas J (2012) Approaches to parallelize pareto ranking in NSGA-II algorithm. In: PPAM, pp 371–380. Springer
23. Lančinskas A, Žilinskas J (2013) Solution of multi-objective competitive facility location problems using parallel NSGA-II on large scale computing systems. In: PARA, pp 422–433. Springer
24. Li Y, Canepa E, Claudel C (2014) Optimal traffic control in highway transportation networks using linear programming. In: ECC, pp 2880–2887. IEEE
25. Lokuciejewski P, Marwedel P (2011) Multi-objective optimizations. Springer, Netherlands, Dordrecht, pp 197–227. doi:10.1007/978-90-481-9929-7_7
26. Mamdoohi G, Abas AF, Samsudin K, Ibrahim NH, Hidayat A, Mahdi MA (2012) Implementation of genetic algorithm in an embedded microcontroller-based polarization control system. Eng Appl Artif Intell 25(4):869–873
27. McClymont K, Keedwell E (2012) Deductive sort and climbing sort: new methods for non-dominated sorting. Evol Comput 20(1):1–26
28. Miettinen K (1999) Nonlinear multiobjective optimization. Kluwer Academic Publishers, Boston
29. NVIDIA: NVIDIA's next generation CUDA compute architecture: Kepler GK110 (2012). https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf
30. NVIDIA: Tegra K1 technical reference manual (2014). https://developer.nvidia.com/gameworksdownload#?search=Tegra%20K1%20Technical
31. NVIDIA: CUDA C programming guide. version 7.0 (2015). http://docs.nvidia.com/cuda/cuda-c-programming-guide/
32. Odan FK, Ribeiro Reis LF, Kapelan Z (2015) Real-time multiobjective optimization of operation of water supply systems. J Water Resour Plan Manag 141(9):04015,011
33. de Oliveira FB, Davendra D, Guimarães FG (2013) Multi-objective differential evolution on the GPU with C-CUDA. In: Soft Computing Models in Industrial and Environmental Applications, pp 123–132. Springer
34. Shi C, Chen M, Shi Z (2005) A fast nondominated sorting algorithm. In: ICNN, vol 3, pp 1605–1610. IEEE
35. Smutnicki C, Rudy J, Żelazny D (2014) Very fast non-dominated sorting. Decis Mak Manuf Serv 8(1–2):13–23
36. Wang, Z, Ju G (2009) A parallel genetic algorithm in multi-objective optimization. In: Control and Decision Conference, 2009. CCDC'09. Chinese, pp 3497–3501. IEEE
37. Wong ML (2009) Parallel multi-objective evolutionary algorithms on graphics processing units. In: GECCO, pp 2515–2522. ACM
38. Yu Q, Chen C, Pan Z (2005) Parallel genetic algorithms on programmable graphics hardware. In: Advances in Natural Computation, pp 1051–1059. Springer
39. Zhang Q, Li H (2007) Moea/d: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
40. Zhang X, Ye T, Cheng R, Jin Y (2012) An efficient approach to non-dominated sorting for evolutionary multi-objective optimization. IEEE Trans Evol Comput 19(2):201–213
41. Zitzler E, Künzli S (2004) Indicator-based selection in multiobjective search. In: Parallel Problem Solving from Nature-PPSN VIII, pp 832–842. Springer
42. Zitzler E, Laumanns M, Thiele L (2001) SPEA2: improving the strength Pareto evolutionary algorithm. Tech. Rep. 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland