J. of Mult.-Valued Logic & Soft Computing, Vol. 32, pp. 313–341
Reprints available directly from the publisher
Photocopying permitted by license only

A General Methodology for Energy-efficient Scheduling in Multicore Environments Based on Evolutionary Algorithms

ZORANA BANKOVIĆ[1,*], UMER LIQAT[1] AND PEDRO LÓPEZ-GARCÍA[1,2]

[1]*IMDEA Software Institute, Madrid, Spain*
E-mail: umer.liqat@imdea.org
[2]*Spanish Council for Scientific Research (CSIC), Spain*
E-mail: pedro.lopez@imdea.org

Received: August 16, 2016. Accepted: December 4, 2018.

In this work we present a general methodology for energy-efficient scheduling in multicore environments, which can be adapted to different features that the underlying environment exhibits, e.g., time and power consumption of different tasks, as well as to different requirements of the task scheduling, e.g., time or energy budgets, priorities, etc. We demonstrate how each of the different features can be used to obtain better scheduling results in terms of reducing the total execution time and power consumption. Given that task scheduling is in general a NP-hard problem, we rely on an Evolutionary Algorithm (EA) to provide an optimal solution in an acceptable amount of time. If the EA is not capable to provide a viable solution, in the sense that not all task deadlines are met, we resort to a modified version of the well known YDS algorithm. In this way, the methodology we propose always provides a viable solution to the scheduling problem. In the tested scenarios, our EA can save upto $55-90\%$ more energy on average than our modified YDS algorithm. We also model task dependence using copula theory in the EA and prove that better results can be obtained when the dependence is modeled. The resulting stochastic scheduler can save upto 18% more energy on average compared to the deterministic scheduler.

Keywords: Task scheduling, multicore, energy efficiency, evolutionary algorithm, YDS, probabilistic inference, copulas.

* Contact author: E-mail: zbankovic@gmail.com

1 INTRODUCTION

Energy efficient scheduling and allocation in multicore environments with enabled Dynamic Voltage and Frequency Scaling (DVFS) is a well-known NP-hard problem. Nevertheless approximated solutions can be efficiently found by heuristic algorithms, such as Evolutionary Algorithms (EAs). An example is our previous EA-based algorithm [8] and different extensions [4, 5, 7] to it.

In our setting, we want to solve the general scheduling problem where the tasks have arbitrary release times and deadlines, and where preemption and migration of tasks are allowed. It has been proven in [3] that the scheduling problem still remains NP-hard for arbitrary release times and deadlines of tasks that are not *agreeable*[†] . In this work we propose a general methodology for energy-efficient scheduling in multicore environments, which can be adapted to different features that the underlying environment exhibits, such as:

- Different ways of estimating time and power consumption, e.g., developed static analysis for a given device.
- The values of time and power estimated as either fixed values, or as probability distribution functions.
- Different ways of modeling task dependency, e.g., by using copulas [27] if they are estimated as probability distribution functions.

In addition, our methodology can be adapted to different requirements of the tasks to be scheduled:

- Meeting time and energy budgets.
- Dealing with task priorities, which can also be used to model dependencies between tasks, e.g., expressing that a given task cannot start before another task on which it depends on is not finished.

We have performed a set of experiments which prove that these features, either separately or combined where possible, can help improving the final result, i.e., obtaining a task scheduling which will consume less energy.

The main algorithm we used in our methodology is a multiobjective evolutionary algorithm (EA) with two objectives: the execution time and the total energy consumption, where both should be minimised. The objectives are clearly in conflict, since the application of Dynamic Voltage and Frequency Scaling (DVFS) reduces energy, but increases execution time. This justifies the usage of a multiobjective algorithm. The EA is based on the

[†] Two tasks are *agreeable* if the task with later release time also has a later deadline.

Nondominated Sorting Genetic Algorithm II (NSGA-II) [11]. NSGA-II is an efficient multiobjective evolutionary algorithm for problems where more than one objective functions needs to be optimized. It improves the adaptive fit of a population of candidate solutions to a Pareto front constrained by a set of objective functions. It has been proven in the literature to perform good when the number of objectives is small [8], and in our case we have only two. Since the output of the multiobjective algorithm is a set of solutions which form the (approximated) Pareto front, we can choose the solution that meets some given energy and/or time requirements. Usually we pick the solution with the minimal energy consumption among those that meet the given time bound (if applicable).

The first variant of our general scheduling algorithm in our proposed methodology (explained in Section 3) relies on the developed static analysis technique for estimating the energy consumption, which solves the time inefficiency problem. In addition, when this algorithm fails to produce a feasible solution, our methodology resorts to our modified YDS[‡] algorithm [32]. This way, we propose a combination of both the EA and the modified YDS algorithms that produces an energy efficient scheduling in reasonable time, and always finds a viable solution. The results presented in this section have been obtained by using real-world benchmarks.

We also consider task dependencies while scheduling (as explained in Section 4). Our results prove that representing input values (time and power/energy) as probability distribution functions, and modeling the dependency between them, can improve the results of the scheduling. The results are based on synthetic data, since the probabilistic static analysis that is needed to provide the necessary input to the algorithm is part of ongoing and future work.

Our methodology has been adapted for its application to multicore XMOS chips, but it could easily be adapted to any multicore environment, ranging from small scale embedded systems up to large scale systems, such as data centers.

The rest of the work is organised as follows. Section 2 presents the most important related work. Section 3 details the implementation and results of our general algorithm which relies on the developed static analysis technique for estimating the energy consumption, as well as using our modified YDS algorithm coupled with our developed EA. Section 4 presents the stochastic variant of our scheduling algorithm, that models the dependence among input values (time and power/energy) as probability distribution functions, including our experimental study. Finally, Section 5 draws the main conclusions and give some directions for future work.

[‡] The name is created using the first letter of the authors' last names.

2 RELATED WORK

Energy efficient scheduling has gained a lot of interest in the recent past. A great number of publications [14] try to present it as a mixed integer linear optimisation problem, which can be solved using mixed integer linear programming, or using a heuristic approach. However, these algorithms become impractical or fail to deliver a solution as the problem size grows.

Since DVFS can provide significant energy savings, its optimal usage has been extensively studied. Some examples divide scheduling and allocation in two separate steps, such as the one given in [28], where in the first step the allocation problem is solved using Linear Programming, while in the second one the scheduling problem is solved for separate processors using Bin Packing. Another approach [9] solves the scheduling problem using a GA that integrates DVFS in the fitness function. However, such a division of the problem reduces the search space, since it becomes limited by the optimal solution of the first part of the problem, which does not always correspond to the global optimum. For this reason, we believe that better solutions can be achieved by solving the scheduling and allocation problem at the same time, while also accounting for the DVFS. There is one example of GA-based scheduling [18] that combines scheduling, allocation and power management in one process. However, it only deals with voltage scaling.

There is also a significant group of publications on using EAs for the problem of optimal scheduling and allocation in multiprocessor systems that allow DVFS. For example, the approach presented in [25] aims to minimize both energy and makespan as a bi-objective problem. The same problem is solved in another work [26], but using the island model of parallel GA populations. Another approach [19] treats the problem from two opposite points of view: in the first one, it optimizes the energy given the scheduler length, while in the other one it optimizes the scheduling length given the energy bound. However, none of the solutions include the possibility of two levels of parallelism as in our work, where each processor can have a number of different threads executing in parallel. Moreover, as far as we know, none of the existing approaches introduce the possibility of task migration. Furthermore, they do not deal with the issue that the EAs cannot always find a viable solution, while we introduce an additional stage implemented as a modified YDS algorithm, which can always find a viable solution. Finally, using static analysis for estimating energy consumption to guide the EA is a relevant novelty of our methodology, which, as our experimental results show, achieves a significant speed-up and provides evidence about the practical applicability of such methodology.

On the other hand, stochastic scheduling has gained lots of interest over the years, since many different cases include uncertainty. In general,

approaches to optimisation under uncertainty include various modeling philosophies, the most important being the following ones:

- Expectation minimisation.
- Minimisation of deviations from goals.
- Minimisation of maximum costs.
- Optimisation over soft constraints.

Our algorithm clearly belongs to the first group. The solution presented in [15] is in the same group, however, it solves the stochastic scheduling problem by reducing it to the deterministic case. The benefit of this approach is the lower execution time, yet at the cost of decreased accuracy.

Copulas have been used in different versions of scheduling-related problems. For example, in [2] they are used in power supply system scheduling to model the presence of uncertain renewables, such as wind and solar energy. Another work given in [12] assesses the reliability of airport schedules using copulas. One more example is given in [31], where the authors use copulas to assess the schedule risk of a software development project. However, all of them use Gaussian or Student-t copulas, which can be applied only if the marginal distributions are either normal or Student-t, while in our work there is no restriction on the kind of marginal distributions. As far as we know, there have not been any attempts to use copulas in the way presented in this work.

3 ENERGY EFFICIENT SCHEDULING

The first practical implementation of our algorithm relied on an existing analytical model for calculating the energy consumption for programs running on multicore XMOS chips [17].

The energy model is limited to the cases when all the cores belong to the same chip, thus they must run at the same voltage and frequency level at each moment. For this reason, in this work we solve the problem of the so-called *global* DVFS, when all the cores always have the same voltage and frequency.

Hence in our first EA implementation, each individual in each generation is evaluated by using the above mentioned program energy model, which requires the execution traces of the programs. Given that the traces can be huge, even for small programs, such evaluation introduces a huge amount of overhead. In order to overcome this issue, we use an existing static analysis which, at compile time, without the need of executing the programs, and in a few seconds, gives a safe estimation of the energy consumed by programs. As energy consumption often depends on (the size of) input data, which is not

known at compile time, the static analysis provides the energy as a function of the input data sizes which are calculated once the input values are known at runtime. The energy consumption estimated by using the static analysis for a given scheduling is computed as the sum of the energies of the tasks running on different cores. This gives an approximated upper bound on the total energy consumption, although it may be less precise than the estimations computed with the program energy model mentioned previously. This may reduce possible energy savings, nevertheless, the information it provides is still good enough to decide which scheduling is better, and the gain in speed of the algorithm is huge: the simulation time is reduced from a few hours to a few minutes.

The EAs can have trouble in finding a viable solution, in the sense that not all task deadlines are met, if the task deadlines are too tight. In order to overcome this problem, we have used a modified YDS ¶ scheduling algorithm that we have adapted for multicore environments in [4]. As we will see later, our experimental results show that if the EA finds a viable solution, it is better than the one obtained by the YDS algorithm in terms of energy savings.

Hence our proposed methodology algorithm consists of the following steps:

1. Perform the static analysis of the input tasks to estimate the energy consumed by each of them.
2. Execute the EA using such estimations.
   - If the EA provides a viable solution, i.e., all the task deadlines are met, this is the final solution.
   - Otherwise, execute our modified YDS algorithm and take its output as the final solution.

In summary, when the EA fails to find a viable solution, the algorithm resorts to our modified YDS algorithm. In addition, the use of static analysis for energy estimation at compile time to guide the EA process implies a significant speed-up in solving the scheduling problem. Altogether, such techniques result in the practicability of our methodology, while still providing a solution with significant energy savings.

3.1 Energy Static Analysis as Input

In order to estimate the energy consumed by programs without actually running them we use an existing static analysis. It is a specialisation of the generic resource analysis presented in [29] for programs written in a high-level C-based programming language, XC [30], running on the XMOS

¶ The name is created using the first letter of the authors' last names.

XS1-L architecture, that uses the instruction-level energy cost models described in [21]. Nevertheless, the analysis is general enough to be applied to other programming languages and architectures (see [20, 21] for details). It is based on setting up a system of recurrence equations over a program that capture its cost (energy consumption) as a function of the sizes of its input arguments. Consider for example the following program written in XC (left hand side) and its compiled assembly representation (right hand side):

```
int fact(int N)
{
  if (N <= 0)
    return 1;

  return N * fact(N - 1);
}
```

```
<fact>:
   01: entsp 0x2
   02: stw   r0, sp[0x1]
   03: ldw   r1, sp[0x1]
   04: ldc   r0, 0x0
   05: lss   r0, r0, r1
   06: bf    r0, <08>

   07: bu    <010>
   10: ldw   r0, sp[0x1]
   11: sub   r0, r0, 0x1
   12: bl    <fact>
   13: ldw   r1, sp[0x1]
   14: mul   r0, r1, r0
   15: retsp 0x2

   08: mkmsk r0, 0x1
   09: retsp 0x21
```

The transformation based analysis framework of [20, 21] would transform the assembly (or LLVM IR) representation of the program into an intermediate semantic program representation (HC IR), that the analysis operates on, which is a series of connected code blocks, represented as Horn Clauses. The analyser deals with this HC IR always in the same way, independent of where it originates from, setting up cost equations for all code blocks in terms of their input data sizes.

$$fact_e(N_s) \quad = \quad fact\_if_e(N_s) + c_{entsp} + c_{ldw} + c_{ldc} + c_{lss} + c_{bf}$$

$$fact\_if_e(N_s) \quad = \quad \begin{cases} c_{mkmsk} + c_{retsp} & \text{if } N_s \leq 0 \\ fact_e(N_s - 1) + c_{bu} + 2\,c_{ldw} + & \text{if } N_s > 0 \\ + c_{sub} + c_{bl} + c_{mul} + c_{retsp} & \end{cases}$$

The cost of the function $fact$ is captured by the equation $fact_e(N_s)$ where N_s represents the size of the input argument to the function $fact$. $fact_e(N_s)$ in turn depends on the equation $fact\_if_e(N_s)$, that captures the cost of the two code blocks representing the two branches of the *if* statement, and a sequence of low-level instructions. The cost of low-level instructions, which constitute an energy cost model, is represented by c_i where $i \in \{ entsp, stw,$

ldw, ...} is an assembly instruction. Such costs are supplied by means of assertions that associate basic cost functions with elementary operations.

If we assume (for simplicity of exposition) that each instruction has unitary cost in terms of energy consumption, i.e., $c_i = 1$ for all i, solving the above system of recurrence equation we obtain the energy consumed by `fact` as a closed-form function of its input data size (N): $fact_e(N) = 13\ N + 8$.

The cost functions inferred by the static analysis are arithmetic functions of a wide range of types (polynomial, exponential, logarithmic, etc.), which depend on input data sizes (natural numbers), and we use them in our scheduling and allocation algorithm to estimate the energy consumed by the different tasks involved. Such estimation can be computed very efficiently once the input data sizes of the tasks are known, since all the basic arithmetic functions involved can be evaluated in little bounded time.

3.2 Our EA using Static Analysis

The problem that we are solving is the optimal allocation and scheduling of a set of tasks (in terms of energy or time, possibly under some requirements involving them), where each task is defined by its:

- Unique *ID*.
- *Release time*, i.e., the moment when the task becomes available.
- *Deadline*, i.e., the latest moment when the task has to finish.
- *Number of clock cycles*, as a good approximation of the execution time.

Individual Representation

The solution to the problem specified above has to contain the following information:

- The core(s) where each task will be executed. Since we allow task migration, a task can be allocated to more than one core.
- The current voltage and clock frequency (V, f) state to exploit DVFS.
- The time periods when the tasks are executed.
- The number of clock cycles each task will execute in the different periods marked by the preemption and migration of that task. This allows to express the number of cycles a task will execute before it is preempted, as well as the number of cycles it will execute after it is resumed in the same or in a different core, etc.

Having in mind these requirements, we have designed the solution representation as shown in Figure 1, which does not introduce significant overhead when executing the EA. Any given task has a positive (unique) number as its

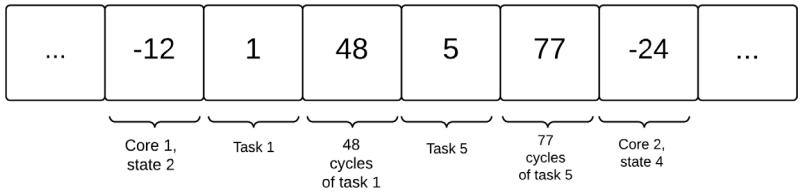


FIGURE 1
An example of (part of) a solution (i.e., individual) representation.

ID. Each gene representing a task ID is followed by a gene representing the number of cycles of the task that will be executed without any preemption. The order of task IDs represents the order of their temporal execution. We also use negative two digit numbers to encode the spatial allocation of the tasks. The first digit represents the core where the tasks are being executed and the second one an encoding of the (V, f) state of that core. As it will be explained in Section 3.4, Table 1, the number of different cores and states is finite, as well as the number of their combinations. The tasks following the allocation code are executed on that coded location. For instance, on Figure 1 we read: on core 1 in state 2, 48 cycles of task 1 will be executed, and 77 cycles of task 5, in this exact order, etc.

Our methodology allows a random order of allocation codes, in order to solve the most general problem. However, if two consecutive allocation codes have different (V, f) states, this means that the tasks allocated to the cores they represent will not be executed in parallel, since all of the cores have to run at the same (V, f) at any moment, and thus the (V, f) state has to be changed before the second group of tasks is executed. For example, in Figure 1 the allocation code following -12 is -24, which means that the chip will be first in the (V, f) state 2 and all tasks allocated to core 1 will be executed sequentially on that core. After they finish their execution, the core will change its (V, f) state from 2 to 4, and the tasks allocated to core 2 will be executed sequentially on core 2. If the second allocation code were -22 instead of -24, then the (V, f) state would not change, and the tasks allocated on cores 1 and 2 would be executed in parallel.

Population Initialisation Individuals in the initial population are created by randomly assigning tasks to random cores in random (V, f) settings with equal probability. However, in order to provide a load balanced solution (as much as possible), the probability of choosing a core decreases as its load increases, which is given by the following formula:

$$Prob = \frac{1}{NumberOfCores} - \frac{CurrentCoreLoad}{TotalLoad} \tag{1}$$

where $CurrentCoreLoad$ stands for the current load of the core expressed as the number of cycles, while $TotalLoad$ stands for the total number of cycles of all the tasks on all cores. According to that formula, at the beginning of the initialisation process, all cores have the same probability of being chosen, while this probability decreases as the core becomes loaded, and it is close to 0 when the load reaches the state where it is equally distributed in all cores. If during the initialisation process a newly calculated probability value of a core is below 0, the value is rounded to 0, and no new load will be assigned to that core. These random solutions do not always have to provide a viable solution, i.e., some of the tasks might miss their deadlines. For this reason, the mutation operator and the objectives are designed to deal with this problem.

Solution Perturbations Given the unique nature of the individual representation, we have designed new crossover and mutation operators. The individuals that participate in the crossover are selected by using the standard tournament selection process.

The Crossover Operator Since our solution allows task migration, a given task ID can appear more than once in an individual, so we cannot apply any of the existing permutation-based crossover operators. Thus, we have designed our own operator, where each child preserves the order of genes representing the task allocation and scheduling from one parent, and only the genes representing the number of cycles can be taken from the other parent. In this way, the produced offspring is a combination of both parents, and is at the same time a viable solution to the problem.

Example 1. *Consider Figure 2, which depicts a crossover operation for the most simple case of 2 cores, 2 tasks and 2 states, with one possible output. We can observe that the first child,* $C1$ *takes the scheduling and allocation from the first parent,* $P1$*, and the cycle distribution from the second parent,* $P2$*, while the second child,* $C2$*, takes the scheduling and allocation from* $P2$ *and the cycle distribution from* $P1$*.*

The Mutation Operator The mutation operator can perform different actions involving one or two tasks. Consider two tasks i and j. When choosing the first one we give higher probability to the tasks which miss their deadlines, in order to achieve a viable solution as soon as possible. In each generation we perform either one of the following actions with the same probability:

- *Swapping:* tasks i and j, together with their corresponding number of cycles, exchange their positions in the solution. In order to avoid creating

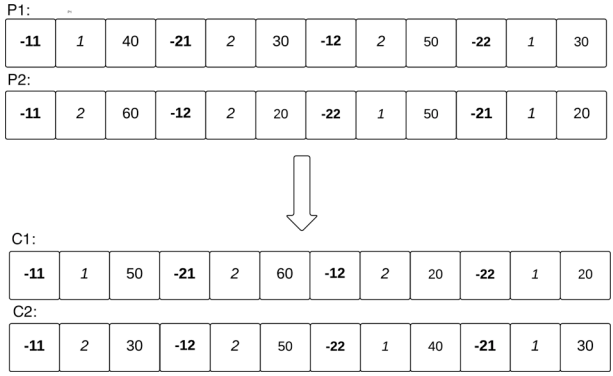


FIGURE 2
An example of a crossover operation.

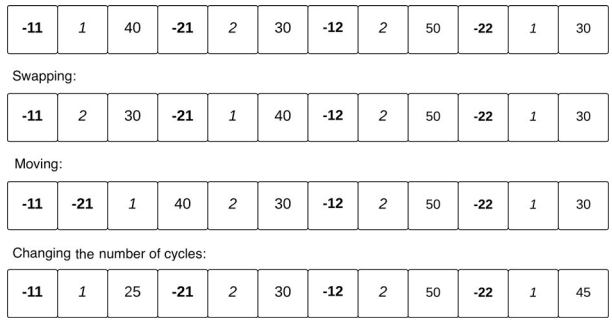


FIGURE 3
Examples of mutation operations.

solutions which are not viable, tasks i and j must be assigned to cores that are in the same (V, f) state.

- *Moving*: randomly move task i to another core and (V, f) state. For the same reason as before, the new (V, f) state must be the same as i's original state.
- *Cycle redistribution*: randomly change the distribution of the number of cycles of task i between its appearances on the different cores.

Example 2. *Consider Figure 3, which depicts different actions of the mutation operator:*

- *Swapping: tasks* 1 *and* 2 *are swapped between cores* 1 *and* 2, *and both cores are in state* 1 *(so that tasks* 1 *and* 2 *are executed in parallel).*
- *Moving: the first part of task* 1 *(40 cycles), originally allocated to core* 1, *is moved to core* 2, *before task* 2. *Both cores are in state* 1.

- *Cycle redistribution: task* 1 *now executes 25 cycles on core* 1 *in state* 1 *and 45 cycles on core* 2 *in state* 2.

Objective Functions

Execution Time One objective of our optimisation problem is to minimize the total execution time of the schedule, which is the time spent since the first task starts its execution until the last task finishes its execution. However, since the initial population is randomly created, it is possible that some of the tasks miss their deadlines, making the solution unviable. Assuming that these solutions can provide some quality genetic material, we do not want to discard them completely, but we penalize them by adding the amount of time the tasks have missed their deadlines to the objective function. Thus, the time objective function for n cores and k different tasks is the following:

$$\hat{T} = T + \sum_{1 \le i \le n} \Big(\sum_{1 \le j \le k} x_{i,j} \cdot y_j \cdot (s_{i,j} + \tau_{i,j} - deadline_j) \Big) \tag{2}$$

where T is the total execution time, given by:

$$T = \max_{\substack{1 \le i \le n \\ 1 \le j \le k}} (x_{i,j} \cdot (s_{i,j} + \tau_{i,j})) - \min_{\substack{1 \le i \le n \\ 1 \le j \le k}} (x_{i,j} \cdot s_{i,j}) \tag{3}$$

where $s_{i,j} \ge 0$ is the moment when task j is scheduled on core i ($s_{i,j} = 0$ if the task j is not scheduled on core i), $\tau_{i,j}$ is the execution time of task j on core i, $x_{i,j}$ is a binary value, that represents whether the task j is executed on core i ($x_{i,j} = 1$) or not ($x_{i,j} = 0$). The second part of formula (2) represents the penalisation, where y_j is another binary value that expresses whether the task j has missed its deadline, $deadline_j$, ($y_j = 1$) or not ($y_j = 0$).

Energy Consumption This objective represents the total energy consumption of the given schedule. In the most general case it is given by the following formula:

$$E = \sum_{1 \le i \le n} (P_{st,i} \cdot T + \sum_{1 \le j \le k} (x_{i,j} \cdot p_{i,j} \cdot \tau_{i,j})) \tag{4}$$

where $P_{st,i}$ is the static power of core i, T is the same as in formula (3), $p_{i,j}$ is the dynamic power of task j when executed on core i, and $x_{i,j}$ and $\tau_{i,j}$ are the same as in formula (2). Static analysis is used to estimate the energy consumption of single tasks, as explained in Section 3.1, while the energy is the sum of the energies of all the tasks, as given in formula (4).

3.3 EA based Scheduling with the YDS Algorithm

The Modified YDS Algorithm

YDS [32] is a well known algorithm for energy-efficient scheduling for single core DVFS-enabled environments. We have chosen it because it always finds a viable (and optimal) solution that minimises the total energy consumption, and it is simple and fast. However, it does not take into account the static power, which nowadays forms an important part of the total power of computing. YDS reduces the frequency and voltage in order to minimise the dynamic power in a way that the execution times of tasks are extended to their deadlines. However, this also results in an increase of the static energy. Thus, there is a critical point from which further reduction of voltage and frequency actually starts increasing the energy consumption. For this reason we have used our modified YDS algorithm reported in [4]. It is adapted for multicore environments, and does not turn off the chip, but instead, finds the critical $(V,\ f)$ point below which further decrease is not beneficial.

3.4 Experimental Evaluation of EAs with Static Analysis and YDS

XMOS Chips In this work we target the XS1-L architecture of the XMOS chips as a proof of concept. Although these chips are multicore and multithreaded, in this work we assume a single core architecture with 8 threads, which is the architecture for which we have an available energy model. In this case, in our experiments, we can use the algorithm and representation of individuals described in Section 3.2 by considering that a thread is conceptually equivalent to a core executing tasks sequentially, as described previously. We refer the reader to [8] for a description of a representation of individuals whose allocation codes include three digits, representing a core, a thread running in parallel on that core, and a $(V,\ f)$ state.

In the XS1-L architecture, the threads enter a 4-stage pipeline, meaning that only one instruction from a different thread is executed at each pipeline stage. If the pipeline is not full, the empty stages are filled with $NOPs$ (no operation). Effectively, this means that we can assume that the threads are running in parallel, with frequency F/N, where F is the frequency of the chip, and $N = max(4, numberOfThreads)$. DVFS is implemented at the chip level, which means that all the cores have the same voltage and frequency at the same time. In order to apply DVFS, we need a list of Voltage-Frequency $(V,\ f)$ pairs or ranges that provide a correct chip functioning. We have experimentally concluded that the XMOS chips can function properly with the voltage and frequency levels given in Table 1.

Task Set In order to test our proposed methodology, we use two different groups of task sets. The first group is made up of small tasks, where the EA training with the program-level energy model takes around one day to

$Voltage(V)$	0.95	0.87	0.8	0.8	0.75	0.7
$frequency(MHz)$	500	400	300	150	100	50

TABLE 1
Viable $(V,\ f)$ pairs for XMOS chips.

complete. This group is used to show the difference between the results obtained with the EA trained with the program-level energy model and the EA trained with the energy estimations obtained by the static analysis. In this group, we use four different arithmetic programs: `fact(N)`, for calculating the factorial of N, `fibonacci(N)` for calculating the Nth Fibonacci number, `sqr(N)` for computing N^2 and `power_of_two(N)` for computing 2^N. In total, we have created a set of 22 tasks to be scheduled, corresponding to the execution of the previous programs with different inputs N.

In the second group we use real world programs, where the EA training based on the program-level energy model is not practical: `fir(N)`, i.e., Finite Impulse Response (FIR) filter, which in essence computes the inner-product of two vectors of dimension `N`, a vector of input samples, and a vector of coefficients, and `biquad(N)`, which is a part of an equaliser implementation, based on a cascade of Biquad filters, whose energy consumption depends on the number of banks `N`. We have used four different FIR implementations, with different number of coefficients: 85, 97, 109 and 121. Furthermore, we have used four implementations of the biquad benchmark, with different number of banks: 5, 7, 10 and 14. We have tested our methodology in scenarios of 16 and 32 tasks, each one corresponding to such implementations. The tasks corresponding to the same implementation have different release times and deadlines.

The energy consumed by the programs is inferred at compile time by the static analysis described in Section 3.1. Such energy is expressed as functions of a parameter N, the size of the input, which is only known at runtime. Such functions are given in Table 2 for 3 of the 6 different voltage and frequency levels used in this work (for conciseness, since the functions for each program have the same complexity order, but different coefficients). The static analysis assumes that a single program (task) is running on one thread on the XMOS chip, while all other threads are inactive. In this implementation, the EA algorithm approximates the total energy of a schedule by adding the energies of all the tasks. Although in this way we loose precision, the estimation still provides precise enough information for the EA to decide which schedule is better.

Testing Scenarios In our current implementation, we assume no dependency between the tasks since it is not supported by the available energy models.

	V = 0.70 **F = 50**	**V = 0.75** **F = 100**	**V = 0.80** **F = 150**
$fact(N)$	$60.5\ N + 46$	$35\ N + 26.7$	$27\ N + 20.5$
$fib(N)$	$87.19 \times 1.62^N +$ $26.7 \times (-0.62)^N$ -74.7	$50.32 \times 1.62^N +$ $15.44 \times (-0.62)^N$ -43	$38.68 \times 1.62^N +$ $11.85 \times (-0.62)^N$ -33.2
$sqr(N)$	$21.3\ N^2 + 121\ N$ $+39.1$	$12.3\ N^2 + 69.8\ N$ $+22.5$	$9.48\ N^2 + 53.7\ N$ $+17.3$
$pow2(N)$	$55.1 \times 2^N - 39$	$63.7 \times 2^N - 39$	$24.49 \times 2^N - 30$
$fir(N)$	$74.93\ N + 124.5$	$43.36\ N + 71.9$	$33.41\ N + 55.2$
$biquad(N)$	$386\ N + 128$	$223.6\ N + 74.2$	$172.5\ N + 57.2$

TABLE 2
Energy functions inferred by static analysis for 3 different pairs of voltage (V)/frequency (MHz).

The release times and deadlines of the different tasks are set in different scenarios in order to experimentally show the benefits of DVFS and optimal scheduling, where all the tasks have different release times and deadlines, with tighter deadlines. It is also important to take into account the static power, especially in the case of loose deadlines.

Scenario 1: Tasks with Loose Deadlines In this scenario the *release time* of a task k, denoted T^k_{rel} is a random moment between 0 and the total execution time at the maximal frequency of all the tasks executed sequentially on a single core. Also, the *deadline* of a task is a random moment between $T^k_{rel} + 10 \times T^k_{maxf}$ and $T^k_{rel} + 20 \times T^k_{maxf}$, where T^k_{maxf} denotes the execution time of the task at maximum frequency. This way we achieve a scenario with loose deadlines even at a smaller frequency.

Scenario 2: Tasks with Tight Deadlines Here, the *release time* is the same as in Scenario 1. However, the *deadline* of a task is a random moment between $T^k_{rel} + 5 \times T^k_{maxf}$ and $T^k_{rel} + 7 \times T^k_{maxf}$. This way we get tighter deadlines, but also provide a set of tasks which are schedulable on the given platform. Note that the deadlines become even tighter as the frequency decreases.

Results: EA vs. Modified YDS Both EA and YDS are implemented in C++. EA extends the MOGAlib library [13] for multiobjective genetic algorithms. In our EA, the population of 200 individuals is evolved for 150 generations. The probability of both crossover and mutation is 0.9. The mutation operation is assigned higher probability than usual due to its important role for reaching a viable solution. Since the result of the optimisation process is a set of possible solutions which form the approximated Pareto front, we take

TABLE 3
EA vs. modified YDS in different scenarios.

	$EA_s(\mu J)$	$YDS_m(\mu J)$	$\frac{(YDS_m - EA_s)}{YDS_m}(\%)$	$\frac{(YDS_m - EA_m)}{YDS_m}(\%)$
A scenario with 22 small numeric tasks and loose deadlines				
Mean	14.3	33.1	56.8	76.57
CI 0.01	11.6 - 17	NA	48.64 - 64.95	67.87-85.27
CI 0.05	12.2 - 16.4	NA	50.45 - 63.14	70.05- 83.09
A scenario with 22 small numeric tasks and tight deadlines				
Mean	14.6	34.8	60.92	69.83
CI 0.01	11.5-17.7	NA	49.14 - 66.95	57.18-57.18
CI 0.05	12.2 - 17	NA	51.15 - 64.94	60.34-60.34
A scenario with 16 tasks made of Biquad and FIR filters and loose deadlines				
Mean	4.38	35.3	87.59	NA
CI 0.01	3.4 - 5.3	NA	85 - 90.37	NA
CI 0.05	3.7 - 5.1	NA	85.55 - 89.52	NA
A scenario with 16 tasks made of Biquad and FIR and tight deadlines				
Mean	14.5	35.4	59.04	NA
CI 0.01	9.4- 19.6	NA	44.63 - 73.45	NA
CI 0.05	10.6 - 18.4	NA	48.02 - 70.06	NA
A scenario with 32 tasks made of Biquad and FIR filters and loose deadlines				
Mean	17.85	68.16	73.81	NA
CI 0.01	10.8 - 25	NA	63.32 - 84.15	NA
CI 0.05	12.5 - 23.3	NA	65.82 - 81.66	NA
A scenario with 32 tasks made of Biquad and FIR filters and tight deadlines				
Mean	29.43	68.16	56.82	NA
CI 0.01	0.72 - 51.6	NA	24.3 - 89.44	NA
CI 0.05	12.5 - 46.3	NA	32.07 - 81.66	NA

the solution with minimal energy consumption such that all task deadlines are met.

Table 3 presents results comparing the EA trained with the energy estimations provided by static analysis (EA_s) and with the program-level energy model (EA_m), versus the modified YDS algorithm presented in Section 3.3.

The energy of the final solution calculated by using the energy estimations provided by static analysis is given in the first column (EA_s). The second column gives the energy of the final scheduling obtained by the modified YDS algorithm (referred to as YDS_m) using the program-level energy model, while the third column gives the energy saving of the EA trained with static analysis compared to YDS. Finally, the last column shows the energy saving obtained with the EA trained with the program-level energy model (EA_m) [6], which is only applicable in the scenarios with a small number of numeric tasks. Each row shows statistics for each scenario taken from 10-20 runs of the algorithm for the same scenario, where $CI0.01$ and $CI0.05$ represent 99% and 95% confidence intervals respectively, meaning that we can claim with 99(95)% certainty that the final result will fall in these intervals.

In order to perform the comparison between the EA (for both EA_s and EA_m) and YDS_m, in the case of EA and tight deadlines, we present the results when the EA can find a viable solution. However, the EA does not always provide a viable solution in all the scenarios with tight deadlines created as explained previously. As we can see in Table 3, if the EA finds a viable solution, it always performs better than the YDS_m. We can also observe that the EA trained with the program-level energy model (EA_m) achieves better results compared to the EA trained with the energy estimations by static analysis (EA_s). However, the EA_s gains are in training times that lasts around 10 minutes, compared to around 24 hours of training the EA_m, which makes it much more practical.

4 MODELING TASK DEPENDENCES: ENERGY-AWARE STOCHASTIC SCHEDULING

In this section we prove that modeling existing dependency between different variables can help obtain better scheduling results. Since probabilistic static analysis for estimating time and energy consumption as probability distribution functions has not been developed yet (it is planned as part of the continuation of this work), in this section we base our results on synthetic data.

The most common approach when dealing with a system which depends on a set of random variables is to assume that the variables are independent, mainly because the mathematical apparatus becomes too complex, or simply because it is not possible to mathematically describe the underlying dependence. However, this simplification often results in assuming an initial scenario which is very different from reality, which limits the usefulness of the final result. An example of this approach is stochastic scheduling, where the relevant characteristics of the tasks are represented as random variables with the corresponding probability distribution functions, and, as far as we

know, in the majority of existing works, the variables describing different tasks are considered to be independent, or modeled with normal distributions [12], which is often not realistic.

Our implementation is part of a bigger tool for scheduling based on EAs [8], to which we want to add the possibility of modeling dependence between the tasks. For this reason, we experiment with modeling dependence between the execution time and power of different tasks using *copulas* [27], in particular Archimedean copulas [24]. However, if we wanted a stand-alone implementation on copula-supported scheduling, a promising approach would be to use Estimation of Distribution Algorithms [16]. The main advantage of copulas when modeling dependence is the fact that they do not depend on the marginal distributions. In our implementation we have decided to study the applicability of Archimedean copulas for two important reasons: they have been extensively studied and the mathematical apparatus for their manipulation is quite mature, and they are known to properly model the existing tail dependency between two variables, i.e., the possibility of achieving extreme values at the same time, which can be expected in our case. In particular, in this work we experiment with Gumbel copulas [24] due to their proper modeling of positive right-tail dependence, e.g., if one task takes more time due to a prolonged memory access, it will lead to longer execution time of all the tasks that are related to it, as well as energy consumption, which is important when dealing with a time and/or energy budget. However, it does not model negative dependence, i.e., achieving small values at the same time, which is not important in our case since it does not affect the budgets mentioned above.

In the following we first give a short overview of the copula theory, necessary for understanding and reproducing the results of our work and later we detail our proposed methodology using copulas for modeling task dependence.

4.1 Copulas for Modeling Task Dependency

In this section we give a short survey on copulas [27]. In essence, the copula theory gives us a mathematical framework for describing dependence between the variables irrespectively of their underlying probability distribution functions.

Sklar's Theorem (1959) Let H be a *continuous* two-dimensional distribution function with marginal distribution functions F and G. Then there exists a copula C such that

$$H(x, y) = C(F(x), G(y)) \implies C(x, y) = H(F^{-1}(x), G^{-1}(y)) \quad (5)$$

Thus, for any two distribution functions F and G and copula C, the function H is a two-dimensional distribution function with marginals F and G.

Archimedean Copulas. An important group of copulas are the Archimedean copulas, where the dependence level depends on one parameter. They are defined in the following way: let ϕ be a continuous strictly decreasing function from $\mathbf{I}$ to $[0, \infty]$ such that $\phi(1) = 0$, and let $\phi^{[-1]}$ denote the "pseudo-inverse of ϕ:

$$\phi^{[-1]}(t) = \phi^{-1}(t) \text{ for } t \in [0, \phi(0)] \text{ and } \phi^{[-1]}(t) = 0 \text{ for } t \geq \phi(0) \quad (6)$$

Then, if ϕ is convex, the function

$$C(u, v) = \phi^{[-1]}(\phi(u) + \phi(v)) \quad (7)$$

is an *Archimedean* copula and ϕ is called its *generator function*. In the case of the Gumbel copula used in this work the generator function is the following:

$$\phi(t) = (-\ln t)^{\theta} \text{ where } \theta \in [1, \infty). \quad (8)$$

Monte Carlo for Copula-based Models. Since in our work we use Monte Carlo simulation to calculate the expected value of random variables, in the following we show how it is integrated in the copula model [23].

If we want to find the expected value of a function $g : \mathbb{R}^d \to \mathbb{R}$ applied to a random vector $(X_1, X_2, ..., X_d)$ whose cumulative distribution function (cdf) is H, the expected values we need are calculated in the following way:

$$\mathbb{E}[g(X_1, X_2, ..., X_d)] = \int_{\mathbb{R}^d} g(x_1, x_2, ..., x_d) \mathrm{d}H(x_1, x_2, ..., x_d) \quad (9)$$

If H is given by its copula model expressed with formula 5, formula 9 can be written as follows:

$$\mathbb{E}[g(X_1, X_2, ..., X_d)] = \int_{[0,1]^d} g(F_1^{-1}(u_1), ..., F_d^{-1}(u_d)) \mathrm{d}C(u_1, ..., u_d) \quad (10)$$

If copula C and marginals $F_1, .., F_d$ are known or estimated, the expected value can be approximated using the following Monte Carlo algorithm:

1. Draw a sample $(U_1^k, U_2^k, ..., U_d^k) \sim C, k = (1, 2, .., n)$ of size n from copula C.

2. Calculate a sample of $(X_1, X_2, ..., X_d)$ by applying the inverse cdf of marginal functions:

$$(X_1^k, X_2^k, ..., X_d^k) = (F_1^{-1}(U_1^k), F_2^{-1}(U_2^k), ..., F_d^{-1}(U_d^k)) \sim H(k = 1, ..., n)$$

3. Approximate $\mathbb{E}[g(X_1, X_2, ..., X_d)]$ with its empirical value:

$$\mathbb{E}[g(X_1, X_2, ..., X_d)] \approx \frac{1}{n} \sum_{k=1}^{n} g(X_1^k, X_2^k, ..., X_d^k)$$

Archimedean Copula Simulation. In order to draw a sample from Archimedean copulas (step 1 of the previous Monte Carlo algorithm), we follow the algorithm for Laplace transform Archimedean copulas, which are all the copulas whose generator function ϕ is a Laplace transform of some function G. The algorithm has been proven correct in [23] and consists on the following steps:

1. Generate a pseudorandom variable V whose *cdf* is G.
   – For the Gumbel copula used in this work, V is a stable distribution (class of probability distributions allowing skewness and heavy tails), $St(1/\theta, 1, \gamma, 0)$, with $\gamma = (cos(\pi/2/\theta))^\theta$ and $\hat{G} = exp(-t^{1/\theta})$.
2. Generate independent and identically distributed random variables $(X_1, X_2, ..., X_d)$.
3. Return $U_i = \hat{G}(-\frac{\ln X_i}{V}), i = 1, ..., d$.

4.2 Our Proposed Methodology using Copulas for Dependency Modeling

In the following we present the main aspects of our EA implementation.

Individuals. An individual, as a representation of a solution to the problem, has to contain information about temporal and spatial allocation of each task. A solution to the scheduling, i.e., temporal aspect of the problem is a permutation of the task identifiers (IDs), where their order also stands for the order of their temporal execution, assuming that each task has a unique ID. In order to solve the allocation problem, i.e., on which thread (and core) each task is executed, we add delimiters to the permutation of the task IDs that define where the tasks are being executed, i.e., core, thread and (V, f) setting (the tasks between two delimiters are executed on the right-side one). In order to be able to distinguish the delimiters from the tasks, delimiters are coded as negative three-digit numbers, where the first digit stands for the core, the second one for the thread on that core, and the third one for the core (V, f) setting (assuming that there is a finite number of settings, which is realistic).

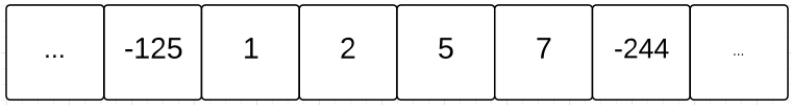


FIGURE 4
An example of (part of) a solution (i.e., individual) representation.

As an example, part of an individual is depicted in Fig. 4, where tasks with IDs 1, 2, 5 and 7 are executed in that order on thread 4 of core 2, with the (V, f) setting coded as 4. In the most general case, the order of delimiters is random. However, if two consecutive delimiters that belong to the same core have different (V, f) settings, this means that they are not being executed in parallel, since the voltage and/or frequency have to be changed. Representing individuals in the way described above has provided us with a relatively simple methodology, which does not introduce great overhead when executing the EA.

Population Initialisation. Since our problem setting in this work is simple (two cores with two thread each), individuals in the initial population are created by randomly assigning tasks to random threads in random (V, f) settings. However, in more complicated problem settings with more cores and threads per core, with task deadlines etc., we have to consider adding a heuristic in order to provide a viable solution in each run.

Solution Perturbations. Given that all the tasks and all the delimiters are different, different solutions are always permutations of the set of tasks and the set of delimiters. This gives us the opportunity to use some of the permutation-based crossover operators, and in this case we are using partial match crossover, since it performed better in terms of the objective function than cycle crossover, and slightly better than order crossover in terms of the objective function and the execution time. Since the order of delimiters is not important in the most general case, this operator provides at the same time variety in consecutive changes of (V, f) settings, and the capability of moving tasks from one thread to another. Regarding mutation, it is implemented in a way that two random threads exchange two random tasks with certain (low) probability.

Objective Functions. The objective of the scheduler is to minimize the total energy consumption, as well as the execution time. As explained before, the application of DVFS, which decreases energy, but increases time, makes these two objectives in conflict and hence require multiobjective optimisation. In general, these values are expressed with the following formulas for a

given set of n heterogeneous machines and a set of k tasks, for a particular machine-task assignment χ:

$$\begin{aligned} E(\chi) &= \sum_{1\le i\le n} (P_{st,i}\cdot T(\chi) + \sum_{1\le j\le k} x_{i,j}\cdot p_{i,j}\cdot \tau_{i,j}) \\ T(\chi) &= \max_{1\le i\le n}\{\sum_{1\le j\le k} x_{i,j}\cdot \tau_{i,j}\} \end{aligned} \tag{11}$$

where $P_{st,i}$ is the static power of the machine i, $x_{i,j}$ is a binary value, $x_{i,j} \in \{0, 1\}$, that represents whether the task j is executed on the machine i ($x_{i,j} = 1$) or not ($x_{i,j} = 0$), $p_{i,j}$ is the (dynamic) power consumption of the task j on the machine i, and $\tau_{i,j}$ is the execution time of the task j on the machine i.

The objective of the stochastic scheduler is to minimize the expected value of the following formulas:

$$\begin{aligned} &\min_{\chi\in\pi}\{\overline{E}(\chi)\} \\ &\min_{\chi\in\pi}\{\overline{T}(\chi)\} \end{aligned} \tag{12}$$

As in our previous work, these values are approximated using the Monte Carlo method, but here we have to introduce the necessary changes to account for the copula-based dependence:

- *Total Energy:* estimated in the Monte Carlo approximation presented in Section 4.1, taking $g(X_1, .., X_d) = \sum_{i=0}^{d} X_i$, where d is the total number of tasks and X_i are random variables representing the energy of each task.
- *Execution Time:* for each core approximated in the same way as the total energy, after that the maximum value between all the cores is taken.

In our implementation, we use the Mathematica system [1] to calculate the inverse cumulative distribution function (step 2 of the Monte Carlo method presented in Section 4.1) due to its capability to deal with all the probability distribution functions used in this work. For this purpose, C++ executes Mathematica as an external program in MathLink mode [22].

4.3 Results and Discussion

Input Data The input data to our scheduling algorithm consists of a set of tasks whose power consumption and execution time are given as random variables with a known probability density function. The following density functions are available at the moment: Uniform, Constant, Exponential, Normal Chi-squared, Gamma, Pareto, Poisson, Binomial, Negative Binomial, and any

combination of the previous ones expressed as a sum of products. A sampling from all the above density functions can be obtained by using a package implemented by Robert Davies [10]. In different (V, f) settings, the power consumption is scaled with V^2 and f, and the corresponding execution time is scaled with f. Finally, the energy of a task is a random variable obtained as the product of their corresponding execution time and power random variables.

Obtained Results and Discussion In this work we experiment with controlled dependency in synthetic data. Dependency is introduced in a way that some of the random variables which describe execution time and power of the tasks have the same fixed distribution. As we have previously mentioned, dependence in Archimedean copulas is controlled with the θ parameter, which in the case of the Gumbel copula belongs to the interval $[1, \infty)$, where $\theta = 1$ stands for independence, while $\theta \to \infty$ stands for comonotonicity, i.e., maximal positive dependence. However, $\theta \geq 10$ is already considered a significant level of dependence.

In order to check the possibility of improving the results of the stochastic scheduling by introducing copulas for modeling dependence, we have created an experiment where we fix the testing scenario, start with independence assumption, i.e., $\theta = 1$, and then increase the θ parameter in order to increase the level of dependence. Since the main claim of our work is that the stochastic scheduling may improve its deterministic counterpart, we check how the results are being improved as the level of dependence increases. In particular, we repeat the simulation for three different levels of dependence: $\theta = 1$, which is equivalent to independence, $\theta = 5$ and $\theta = 10$, which assumes a high level of dependence.

After performing the training and obtaining the Pareto front, in all the cases we took the solution with minimal energy consumption from all the solutions belonging to the Pareto front. The testing was performed on different sets of test cases, each having 10 test cases generated randomly. The results are summarised in Table 4, where we can observe (from left to right) the average percentage of test cases where the stochastic scheduler improves the deterministic one, along with the average improvement, and the average minimal and maximal improvement in all test cases. Figure 5 illustrates the evolution of different test cases (1-6 are different test case sets, each having 10 different test cases), sorted by their improvement. Note that negative numbers actually mean that the stochastic scheduler worsened the performances.

In the current implementation we rely on Mathematica to calculate the inverse cumulative distribution function, which significantly increases simulation time, since it takes 6–8 hours on a 2.5GHz Intel Core i5 with 4GB DDR3.

Test case ($\boldsymbol{\theta}$)	**Avg. % of improv. test cases**	**Avg. improv.** (%)	**Avg. min. improv.** (%)	**Avg. max. improv.** (%)
1 *(1)*	50	2.46	−4.58	12.1
2 *(5)*	81.66	18	−3.51	31.96
3 *(10)*	68.75	8.42	−3.95	22.75

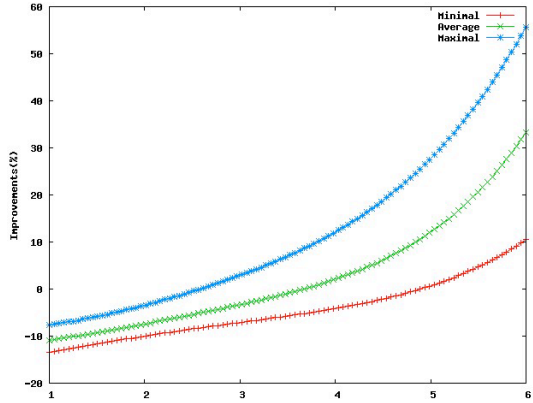
TABLE 4
Summary of average improvements.

From the table and the figures we can draw a few interesting conclusions:

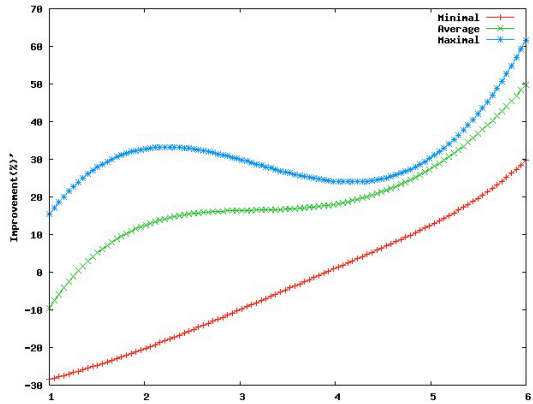
- The stochastic scheduler improves a significant number of test cases, even when assuming independence ($\theta = 1$).
- The best obtained results correspond to $\theta = 5$, which assumes certain level of dependence, although less than maximal positive dependence, which is close to $\theta = 10$. This confirms our expectation, since in our test cases some of the variables describing power and/or time are correlated, but not all of them.
- The maximal observed energy improvement achieved in a particular case is around 62%, while the maximal observed performance (in terms of energy savings) decrease is 28%, both corresponding to the case $\theta = 5$ (Figure 5).
- The test cases that obtained better performances with the deterministic scheduler, i.e., whose performances were decreased after applying the stochastic scheduler (and which can be observed as "negative" improvement in both Table 4 and Figure 5), had values which were close to the average values of the corresponding distributions, which were used to design the deterministic scheduler. This behaviour was also expected, since the main idea of the stochastic scheduler is to improve the scheduling when the real data deviate significantly from the expected ones used to create the deterministic scheduler. However, from this testing scenario we were not able to properly decide the threshold level which would tell us when it is beneficial to start using the stochastic scheduler.

5 CONCLUSIONS AND FUTURE WORK

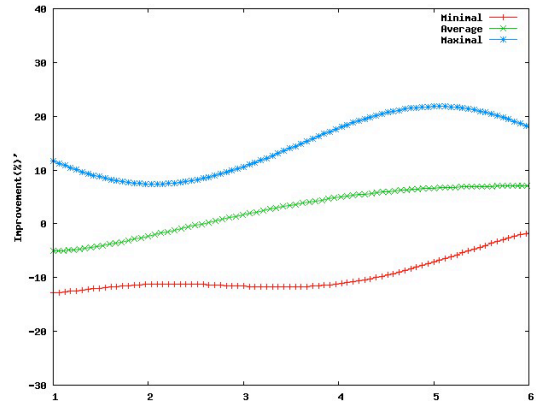
In this work we have presented a general methodology for energy-efficient scheduling in multicore environments, which can be adapted to different features of the underlying environment and to requirements of the task scheduling. We demonstrate how each of the different features can be used (often in



(a) $\theta = 1$



(b) $\theta = 5$



(c) $\theta = 10$

FIGURE 5
Evolution of Minimal, Average and Maximal Improvements (%) for $\theta = 1, 5, 10$

combination) to obtain better scheduling results in terms of reducing the total execution time and power consumption. In particular, we have shown that features such as the use of static analysis for estimating time and energy consumption, and the capability of modeling task dependence can significantly improve the results, since the final obtained scheduling which relies on these features can further reduce its energy.

The use of static analysis improves significantly the speed of the EA training process from hours to minutes, thus allowing its real world applicability. Furthermore, since in the case of very tight task deadlines the EA can fail in providing a viable solution, we add one more stage based on a modified YDS algorithm in order to always provide a scheduling solution. Our custom EA saves 55-90% more energy compared to YDS.

We have also studied the possibility of introducing dependence of both power consumption and execution time of the tasks which run on the same platform in order to improve the results of optimal task scheduling. Power consumption and execution time of the tasks are represented as random variables with known probability distribution functions, while their dependence is modeled with Gumbel copulas, whose θ parameter is varied in order to simulate different levels of dependence. As far as we know, this is the first work that uses copula-based dependence in the context of stochastic scheduling where the important aspects of the tasks are modeled using random distribution functions. If there is dependence present in the underlying data, our results show that the performance of the stochastic scheduler is significantly improved after assuming certain level of dependence: on average 18% of energy consumption can be saved compared to the results of the deterministic scheduler, along with 81% of improved test cases, versus 2.44% average savings when task independence is assumed, along with 50% of improved test cases.

In the future, we plan to develop a probabilistic static analysis, which will be able to estimate time and energy consumption as probability distribution functions, as well as their dependencies. This will allow the implementation of the unification of all the features in one single framework.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union 7th Framework Programme under grant agreement 318337, ENTRA - Whole-Systems Energy Transparency, Spanish MINECO TIN'12-39391 *StrongSoft* and TIN'08-05624 *DOVES* projects, and the Madrid M141047003 *N-GREENS* project.

REFERENCES

[1] Mathematica: the Way the World Calculates. http://www.wolfram.com/products/mathematica/index.html.

[2] Sajjad Abedi, Gholam Hossein Riahy, Seyed Hossein Hosseinian, and Mehdi Farhadkhani. (2013). Improved stochastic modeling: An essential tool for power system scheduling in the presence of uncertain renewables.

[3] S. Albers, F. Müller, and S. Schmelzer. (2007). Speed scaling on parallel processors. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, pages 289–298, New York, NY, USA. ACM.

[4] Z. Banković, U. Liqat, and P. López-García. (2015). A Practical Approach for Energy Efficient Scheduling in Multicore Environments by combining Evolutionary and YDS Algorithms with Faster Energy Estimation. In *The 11th International Conference on Artificial Intelligence Applications and Innovations (AIAI'15)*, volume 458 of *IFIP Advances in Information and Communication Technology*, pages 478–493. Springer.

[5] Z. Banković, U. Liqat, and P. López-García. (2015). Trading-off Accuracy vs. Energy in Multicore Processors via Evolutionary Algorithms Combining Loop Perforation and Static Analysis-based Scheduling. In Enrique Onieva, Igor Santos, Eneko Osaba, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems (HAIS 2015)*, volume 9121 of *Lecture Notes in Computer Science*, pages 690–701. Springer International Publishing.

[6] Z. Banković and P. López-García. (2015). Energy Efficient Allocation and Scheduling for DVFS-enabled Multicore Environments using a Multiobjective Evolutionary Algorithm. In *Genetic and Evolutionary Computation Conference (GECCO 2015)*, pages 1353–1354. ACM.

[7] Z. Banković and P. López-García. (2015). Improved Energy-aware Stochastic Scheduling based on Evolutionary Algorithms via Copula-based Modeling of Task Dependences. In Álvaro Herrero, Javier Sedano, Bruno Baruque, Héctor Quintián, and Emilio Corchado, editors, *International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2015)*, volume 368 of *Advances in Intelligent Systems and Computing*, pages 153–163. Springer International Publishing.

[8] Zorana Banković and Pedro Lopez-Garcia. (February 2015). Stochastic vs. Deterministic Evolutionary Algorithm-based Allocation and Scheduling for XMOS Chips. *Neurocomputing*, 150:82–89.

[9] Ying Chang-tian and Yu Jiong. (2012). Energy-aware genetic algorithms for task scheduling in cloud computing. In *7th ChinaGrid Annual Conference (CHINAGRID'12)*, pages 43–48.

[10] Robert Davies, (2013). Random distributions. http://www.robertnz.net/.

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. (2000). A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197.

[12] Tony Diana. (2011). Improving schedule reliability based on copulas: An application to five of the most congested {US} airports. *Journal of Air Transport Management*, 17(5):284–287.

[13] Balázs Gaál. (12 2009). *Multi-Level Genetic Algorithms and Expert System for Health Promotion*. PhD thesis, Univ. of Panonia, Faculty of Information Technology.

[14] M.E.T. Gerards, J.L. Hurink, P.K.F. Holzenspies, J. Kuper, and G.J.M. Smit. (Feb 2014). Analytic clock frequency selection for global dvfs. In *Parallel, Distributed and*

Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, pages 512–519.

[15] Chen Gong, Xiaodong Wang, Weiqiang Xu, and A. Tajer. (2013). Distributed real-time energy scheduling in smart grid: Stochastic model and fast optimization. *IEEE Transactions on Smart Grid*, 4(3):1476–1489.

[16] Yasser Gonzalez-Fernandez and Marta Soto. (2014). copulaedas: An r package for estimation of distribution algorithms based on copulas. *Journal of Statistical Software*, 58(9):1–34.

[17] S. Kerrison and K. Eder. (July 2014). Measuring and modelling the energy consumption of multithreaded, multi-core embedded software. *ICT Energy Letters*, pages 18–19.

[18] V. Kianzad, S.S. Bhattacharyya, and Gang Qu. (2005). Casper: an integrated energy-driven approach for task graph scheduling on distributed embedded systems. In *16th IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, pages 191–197.

[19] P.R. Kumar and S. Palani. (2012). A dynamic voltage scaling with single power supply and varying speed factor for multiprocessor system using genetic algorithm. In *2012 International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME)*, pages 342–346.

[20] U. Liqat, K. Georgiou, S. Kerrison, P. Lopez-Garcia, M. V. Hermenegildo, J. P. Gallagher, and K. Eder. (2016). Inferring Parametric Energy Consumption Functions at Different Software Levels: ISA vs. LLVM IR. In M. Van Eekelen and U. Dal Lago, editors, *Foundational and Practical Aspects of Resource Analysis. Fourth International Workshop FOPARA 2015, Revised Selected Papers*, Lecture Notes in Computer Science. Springer. In press.

[21] U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. Lopez-Garcia, N. Grech, M.V. Hermenegildo, and K. Eder. (2014). Energy Consumption Analysis of Programs based on XMOS ISA-level Models. In Gopal Gupta and Ricardo Peña, editors, *Logic-Based Program Synthesis and Transformation, 23rd International Symposium, LOPSTR 2013, Revised Selected Papers*, volume 8901 of *Lecture Notes in Computer Science*, pages 72–90. Springer.

[22] Mathematica. (1993). *MathLink Reference Guide*.

[23] A.J. McNeil, R. Frey, and P. Embrechts. (2010). *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton Series in Finance. Princeton University Press.

[24] Alexander J. McNeil and et al., (2009). Multivariate archimedean copulas, d-monotone functions and l_1-norm symmetric distributions.

[25] M. Mezmaz, Young Choon Lee, N. Melab, E. Talbi, and A.Y. Zomaya. (2010). A bi-objective hybrid genetic algorithm to minimize energy consumption and makespan for precedence-constrained applications using dynamic voltage scaling. In *IEEE Congress on Evolutionary Computation (CEC'10)*, pages 1–8.

[26] M.-S. Mezmaz, Y. Kessaci, Y.C. Lee, N. Melab, E.-G. Talbi, A.Y. Zomaya, and D. Tuyttens. (Oct 2010). A parallel island-based hybrid genetic algorithm for precedence-constrained applications to minimize energy consumption and makespan. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 274–281.

[27] Roger B. Nelsen. (2003). Properties and applications of copulas: A brief survey. In *First Brazilian Conference on Statistical Modelling in Insurance and Finance*, pages 10–28.

[28] F. Paterna, A. Acquaviva, A. Caprara, F. Papariello, G. Desoli, and L. Benini. (March). An efficient on-line task allocation algorithm for qos and energy efficiency in multicore multimedia platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6.

[29] A. Serrano, P. Lopez-Garcia, and M. Hermenegildo. (2014). Resource Usage Analysis of Logic Programs via Abstract Interpretation Using Sized Types. *Theory and Practice of Logic Programming, 30th Int'l. Conference on Logic Programming (ICLP'14) Special Issue*, 14(4-5):739–754.

[30] D. Watt. (2009). *Programming XC on XMOS Devices*. XMOS Limited.

[31] Dengsheng Wu, Hao Song, Minglu Li, Chen Cai, and Jianping Li. (June 2010). Modeling risk factors dependence using copula method for assessing software schedule risk. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*, pages 571–574.

[32] F. Yao, A. Demers, and S. Shenker. (1995). A scheduling model for reduced cpu energy. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:374.