Trading-off Accuracy vs Energy in Multicore Processors via Evolutionary Algorithms Combining Loop Perforation and Static Analysis-Based Scheduling

Zorana Banković[1], Umer Liqat[1], and Pedro López-García[1,2(✉)]

[1] IMDEA Software Institute, Madrid, Spain
{zorana.bankovic,umer.liqat,pedro.lopez}@imdea.org
[2] Spanish Council for Scientific Research (CSIC), Madrid, Spain

Abstract. This work addresses the problem of energy efficient scheduling and allocation of tasks in multicore environments, where the tasks can permit certain loss in accuracy of either final or intermediate results, while still providing proper functionality. Loss in accuracy is usually obtained with techniques that decrease computational load, which can result in significant energy savings. To this end, in this work we use the loop perforation technique that transforms loops to execute a subset of their iterations, and integrate it in our existing optimisation tool for energy efficient scheduling in multicore environments based on evolutionary algorithms and static analysis for estimating energy consumption of different schedules. The approach is designed for multicore XMOS chips, but it can be adapted to any multicore environment with slight changes. The experiments conducted on a case study in different scenarios show that our new scheduler enhanced with loop perforation improves the previous one, achieving significant energy savings (31 % on average) for acceptable levels of accuracy loss.

1 Introduction

Task scheduling and allocation for energy efficiency in multicore environments is a well-known NP-hard problem which can be efficiently solved with heuristic algorithms, such as evolutionary algorithms. One example is our approach for scheduling and allocation, which is based on evolutionary algorithms (EAs) [1]. The algorithm was shaped for its application to XMOS multicore chips, which give support for dynamic voltage and frequency scaling (DVFS) at chip level, i.e., all cores have the same voltage and frequency. However, the approach can be adapted to any multicore environment with slight modifications. In this work we want to deal with optimally scheduling tasks which can permit certain accuracy loss.

As a matter of fact, the great majority of today's processors are designed in a way that can provide a high level of accuracy. However, there are numerous applications that allow certain accuracy loss, which still permits them to function

E. Onieva et al. (Eds.): HAIS 2015, LNAI 9121, pp. 690–701, 2015.
DOI: 10.1007/978-3-319-19644-2_57

properly, such as video streaming, machine learning, etc. Since decreasing the accuracy is usually achieved by reducing the computational load, this can lead to both increase in performance and decrease in energy consumption, so here we deal with a trade-off between accuracy on one side and performance and/or energy on the other. One technique that achieves this is loop perforation [7], which in essence consists in skipping every n-th loop iteration, for a given n. Broadly speaking, accuracy can be considered as one aspect of quality of service (QoS), so we can say that in this work we deal with the QoS/energy trade-off.

Thus, in this work we solve the following scheduling problem: given a set of tasks with known release time and number of cycles to compute them, find proper allocation and scheduling of the tasks, as well as a (V, f) assignment (i.e., voltage and frequency pair) to the cores in a way the total energy is minimised, while accuracy is maximised, meeting a minimal acceptable level of accuracy. Different levels of accuracy are achieved by applying the loop perforation technique with different n, where every n-th loop iteration is skipped.

Hence, we deal with two objectives: accuracy and energy. Accuracy is defined in terms of deviations of the output signal after applying the loop perforation, while in order to estimate energy consumption, we use an existing static analysis which, at compile time, with no need of executing the programs, and in a matter of seconds, gives a safe estimation of the energy consumed by programs. The energy consumption often depends on (the size of) input data, which is not known at compile time. For this reason, the static analysis provides the energy as a function of the input parameters, which is evaluated when input values are known at runtime. The energy consumption estimated by using the static analysis for a given scheduling is calculated as the sum of energies of the tasks running on different cores. This gives a safe upper bound on the total energy consumption, which is good enough for deciding which schedule consumes less energy, and can provide acceptable estimations of energy savings.

The rest of the paper is organised as follows. Section 2 gives more details of our proposed approach. Section 3 presents an experimental evaluation of it. Some related work is discussed in Sect. 4 and finally, some conclusions are drawn in Sect. 5.

2 Proposed Approach

2.1 Loop Perforation

The loop perforation technique consists in skipping some loop iterations, for example skipping every n-th iteration [7], where n can be varied in order to trade accuracy with energy, i.e., for higher n, less instructions are skipped, so the accuracy is higher, while more energy is saved for lower values of n. This trade-off between accuracy and energy consumption justifies the usage of a multiobjective algorithm. As we will see in the following, in this work the loop perforation technique is implemented as one possibility for the mutation operator.

2.2 Evolutionary Algorithm (EA)

The work presented in this paper is an extension of our previous work where we developed a custom algorithm based on an NSGA-II multiobjective evolutionary algorithm [1]. The conflicting objectives are accuracy and energy consumption, since we want to decrease the energy consumption, while maintaining the accuracy level as high as possible (always above a given threshold).

The non-dominated solutions are generated using the well-known NSGA-II algorithm [2], while the EA follows the standard steps of evolutionary algorithms: initialisation, evolution, where the selection process is implemented as standard tournament selection, and our custom-made crossover and mutation operators are applied. In the following we give more detail on the particular improvements carried out in this work.

Individual. A solution to the problem we are solving has to contain information about scheduling and allocation of each task, how many cycles of each task are executed in the current run (since we support task migration), and voltage and frequency levels of the core at each moment. In this work we add a new dimension to the problem, which is the possibility to decrease accuracy through loop perforation and thus it also has to be encoded in the individual. For this reason, we add one more field after each task, which encodes n, i.e., the iterations which can be skipped in one or more loops previously identified in each task. An example of a part of an individual is given in Fig. 1, and can be read in the following way: on core 1 in state 2 we execute in this order,

- 48 cycles of task 1, without performing loop perforation on it, and
- 77 cycles of task 5, where we skip every 4th iteration in the loop previously defined.

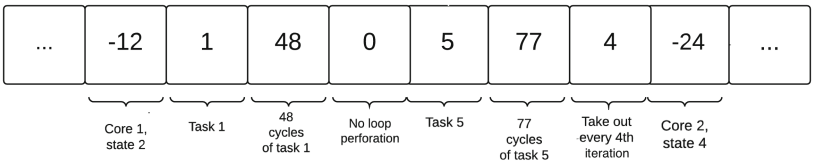

Fig. 1. Representation of an individual

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. The number of cycles of a task executed in each run, as well as the loop iterations to be skipped are also randomly chosen.

The Crossover Operator. Our custom crossover operator is designed in the following way:

– Each child preserves the order of appearance of the tasks, as well as their allocation from one of the parents,
– But, can take the distribution of the number of cycles, as well as the number of loop iterations to be skipped of one of them with equal probability.

The Mutation Operator. The mutation operator can perform different operations involving one or two tasks (designated as i and t in the following text). In each generation we perform one of the following operations with the same probability:

– *Swapping:* i and t, together with their corresponding number of cycles and loop iterations to be skipped, change their positions in the solution. However, in order to avoid creating solutions which are not viable, i and t have to belong to the cores which are executed in parallel.
– *Moving*: move i to a random position j. For the same reason as before, the position j has to belong to a core being executed in the same state as i's original state.
– *Changing the Cycle Distribution:* Randomly change distribution of the cycles of task i between its appearances on different cores.
– *Loop Perforation:* For a random task i, assign randomly the number of loop iterations to be skipped, update the total number of cycles, i.e., decrease the total number of cycles for the amount corresponding to the cycles of the skipped loops, and share them randomly between the existing appearances of the task i in the solution.

These operators are depicted in Fig. 2:

– *Swapping:* Tasks 1 and 2 are swapped between cores 1 and 2 while both in state 1.
– *Moving*: First part of task 1 (40 cycles) are moved to core 2 before task 2.
– *Changing the Number of Cycles:* Task 1 now executes 25 cycles on core 1 in state 1 and 45 cycles on core 2 and state 2.
– *Loop Perforation:* Task 1, where loop perforation has not been performed, now skips every 20th task in the defined loop, which results in decreased number of cycles, i.e., it has 60 cycles, where the first 35 cycles are executed in the first appearance of the task 1, while the remaining 25 cycles are executed in its second appearance.

Objective Functions: Energy Consumption. This objective represents the total energy consumption of the given schedule, and it should be minimised. It is given with the following formula:

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

where $P_{st,i}$ is the static power of the core i, T is the total execution time of the schedule, i.e., the moment when the last task finishes its execution, $\tau_{i,j}$ is

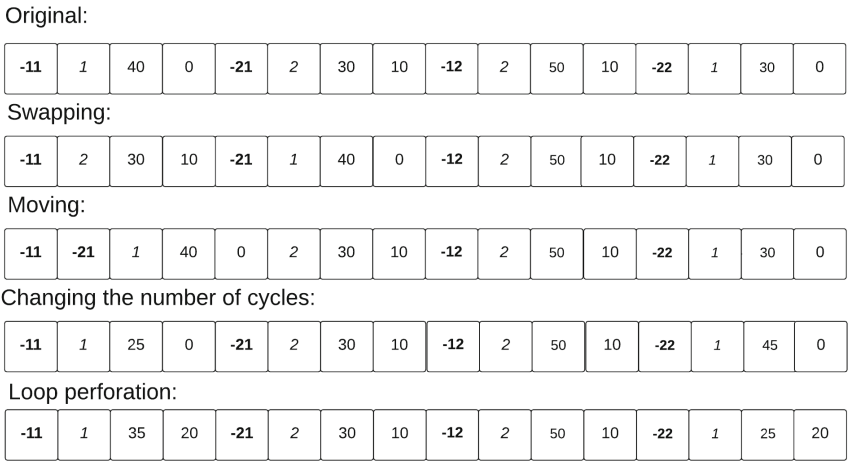

Fig. 2. Different possibilities for mutation

the execution time of task j on core i, $x_{i,j}$ is a binary value, $x_{i,j} \in \{0,1\}$, that represents whether the task j is executed on the core i ($x_{i,j} = 1$) or not ($x_{i,j} = 0$), and $p_{i,j}$ is the power of task j when executed on core i.

Objective Functions: Accuracy. In this work accuracy is defined as an average error of the output after applying loop perforation, and it should be minimised. If a task performs some sort of signal processing, where the output is a digital signal consisting of a number of samples, the error is calculated as the Euclidean distance between the outputs obtained with and without loop perforation.

2.3 Energy Static Analysis as Input

In order to statically estimate the energy consumed by programs we use an existing static analysis. It is a specialization of the generic resource analysis presented in [8] for programs written in a high-level C-based programming language, XC [9], running on the XMOS XS1-L architecture, that uses the instruction-level energy cost models described in [3]. The analysis is general enough to be applied to other programming languages and architectures (see [4,5] for details). It enables a programmer to symbolically bound the energy consumption of a program P on input data $\bar{x}$ without actually running $P(\bar{x})$. It is based on setting up a system of recursive cost equations over a program P that capture its cost (energy consumption) as a function of the sizes of its input arguments $\bar{x}$. Consider for example the following program written in XC:

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

The transformation based analysis framework of [4,5] 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 analyzer deals with this HC IR always in the same way, independent of where it originates from, setting up cost equations for all code blocks (predicates).

$$fact_e(N) = fact\_if_e(0 \leq N, N) + c_{entsp} + c_{stw} + c_{ldw} + c_{ldc} + c_{lss} + c_{bf}$$

$$fact\_if_e(B, N) = \begin{cases} fact_e(N-1) + c_{bu} + 2\ c_{ldw} + c_{sub} + \\ \qquad + c_{bl} + c_{mul} + c_{retsp} & \text{if } B \text{ is } \texttt{true} \\ c_{mkmsk} + c_{retsp} & \text{if } B \text{ is } \texttt{false} \end{cases}$$

The cost of the function $fact$ is captured by the equation $fact_e$ which in turn depends on the equation $fact\_if_e$, that captures the cost of the two clauses 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, we obtain the energy consumed by `fact` as a function of its input data size (N): $fact_e(N) = 13\ N + 8$.

3 Experimental Evaluation

3.1 Testing Environment

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. All threads have their own register set and up to 4 instructions per thread can be buffered, which are scheduled in a way to minimize simultaneous memory accesses by consecutive threads. 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 threads have the same voltage and frequency at the same time. All XMOS chips support frequency scaling. However, only the XS1-SU01A-FB96 [6] chip provides the possibility of voltage scaling enabled by two DC-DC converters whose output voltage belongs to the range (0.6V, 1.3V). In order to apply DVFS, we need list of Voltage-Frequency *(V,f)* pairs or ranges that provide a correct chip functioning.

Table 1. Viable (V, f) pairs for XMOS chips.

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

We have experimentally concluded that the XMOS chips can function properly with the voltage and frequency levels given in Table 1.

Task Set. We use two real world programs for testing:

- `fir(N)`: Finite Impulse Response (FIR) filter. In essence, it computes the inner-product of two vectors: a vector of input samples, and a vector of coefficients.
- `biquad(N)`: Part of an equaliser implementation, which uses a cascade of Biquad filters. The energy consumed depends on the number of filters in the cascade, also known as banks `N`.

These filters are often used in signal processing, where some certain level of accuracy loss can be permitted. This makes them good candidates for experimenting with the accuracy/energy trade-off. 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 program, with different number of banks: 5, 7, 10 and 14. We have tested our approach in scenarios with 32 tasks, each one corresponding to one of the above mentioned implementations. The tasks corresponding to the same implementation have different release times.

The energy consumed by the programs is inferred at compile time by the static analysis described in Sect. 2.3. This energy is expressed as a function of an input parameter N, which is known at run time only. In the case of FIR, N is the number of coefficients, while in the case of the Biquad cascade, N is the number of banks. These functions are given in Table 2. The analysis assumes that a single program is running on one thread on the XMOS chip, while all other threads are inactive. This means that only the first stage of the pipeline is occupied with an instruction, while the rest are empty, i.e., occupied with NOPs. In this implementation, the EA algorithm approximates the total energy of a schedule taking the sum of the energies of all the tasks running on different cores, i.e., threads, as we have seen in Sect. 2.2. However, in reality if all the threads are active and execute a program, each pipeline stage will contain an instruction from a different thread. For this reason, we can say that the estimation produced by the static analysis of the energy consumed by a set of tasks is an upper bound on the actual energy consumption. However, this estimation provides precise enough information for the EA to decide which schedule is better.

3.2 Testing Scenario

We have tested our approach on a scenario of 32 tasks, where each task implements either an FIR or a Biquad cascade previously described. For the case of

Table 2. Energy functions for 3 different pairs of voltage (V) / frequency (F, in MHz)

	V = 0.70 F = 50	**V = 0.75 F = 100**	**V = 0.80 F = 150**
$fir(N)$	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 $N + 57.2$
	V = 0.80 F = 300	**V =0.87 F = 400**	**V = 0.95 F = 500**
$fir(N)$	20.14 $N + 33.2$	18.95 $N + 31.09$	19.15 $N + 31.3$
$biquad(N)$	104.3 $N + 34.4$	98.31 $N + 32.4$	99.48 $N + 32.7$

FIR, loop perforation takes out a few coefficients, while in the case of Biquad cascade, it takes out a few banks. All tasks have different release time. Task deadlines do not exist. However, we should bear in mind that in the case of DVFS it is not beneficial to scale down voltage and frequency indefinitely, since at some point static power consumption becomes more significant than dynamic power consumption. Thus, if we keep decreasing the dynamic power, the static power is increased at the same time, and as a result, the total energy consumption increases. The input signal to all tasks is a standardised set of input samples for testing in signal processing.

3.3 Obtained Results and Discussion

The EA has been trained with the following parameters: population of 200 individuals, evolved for 150 generations, crossover rate: 0.9, and mutation rate: 0.9 - since mutation introduces loop perforation, a high rate is needed.

In order to illustrate the energy savings provided by loop perforation (referred to as *Case 1* in the following text), we have trained another EA, where the objectives are to minimize energy and execution time, without the possibility of loop perforation (referred to as *Case 2* in the following). This algorithm has been trained with the same parameters given above. Since both algorithms are multiobjective, the result of the training of both is a Pareto front of possible solutions with different trade-off between the objectives. Examples of Pareto fronts obtained in *Case 1* and *Case 2* are given in Figs. 3 and 4 respectively. In *Case 1* we have picked a solution with the smallest energy objective value, whose maximal deviation from the final result (accuracy) is below (above) a given threshold, while in *Case 2* we have chosen a solution with the smallest energy objective. The results are presented in Table 3, with the following columns:

- *Column 1:* Maximal acceptable average error (or equivalently, minimal acceptable level of accuracy) of the final result.
- *Column 2:* Average energy of the final schedule obtained in a set of experiments of *Case 1* estimated by static analysis given in mJ (mili Joules).

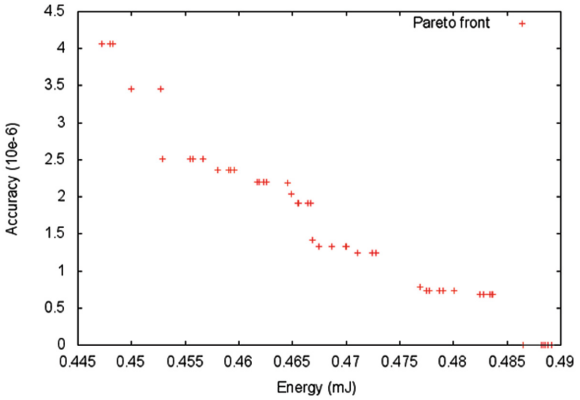

Fig. 3. Pareto front for Energy/Accuracy trade-off EA (Case 1)

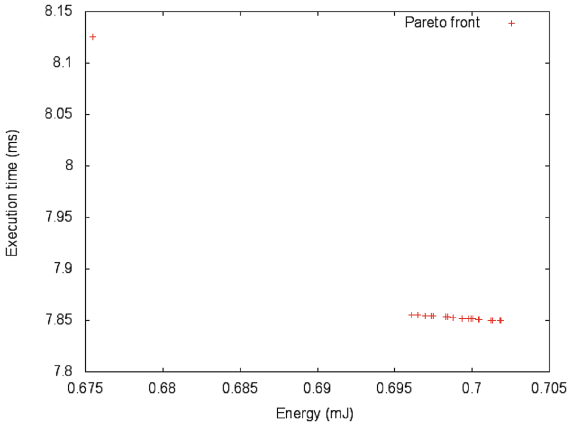

Fig. 4. Pareto front for Energy/Time trade-off EA (Case 2)

Table 3. Obtained savings with different levels of minimal acceptable accuracy.

Max.	Case 1:	Case 2:	Savings(%)	
Avg. Error	Avg. En.(mJ)	Avg. En.(mJ)	Avg.	CI0.05
10^{-6}	0.487	0.721	16.18	0.93–31.42
$2 \cdot 10^{-6}$	0.461	0.597	18.21	3.54–32.87
$3 \cdot 10^{-6}$	0.434	0.666	31.04	13.72–48.37

- *Column 3:* Average energy of the final schedule obtained in a set of experiments of *Case 2* estimated by static analysis given in mJ (mili Joules).
- *Column 4:* Obtained savings expressed as % and calculated as $\frac{Column3-Column2}{Column3} \cdot 100$.
- *Column 5*: Statistics of the experiments expressed as 0.05 confidence interval, i.e., we can claim with 95 % certainty that the final result will belong to this interval.

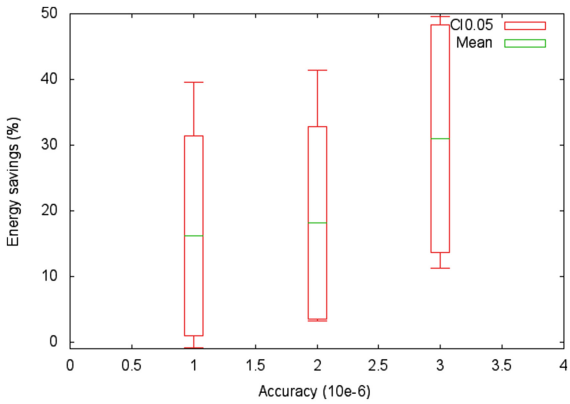

Fig. 5. Energy savings for different accuracy levels

As we can observe, energy savings that can be obtained with loop perforation are significant and range from 3 % to 40 % in different experiments, even with small permitted level of error. As we increase the accepted level of average error, the savings increase, as expected, which is clearly depicted in Fig. 5. However, the relationship between the accuracy and the energy savings depends on the application: some applications can preserve acceptable accuracy by skipping more loop iterations (and hence achieve bigger energy savings) than others that lose acceptable accuracy by skipping less loop iterations (and hence achieve smaller energy savings).

Fluctuations in the final result in different experiments appear due to the imprecision of the static analysis, since currently it gives an upper bound, rather than a realistic estimation of energy consumption. This can explain the big confidence intervals. Since the acceptable level of error is small, we could observe that in the final result only tasks that perform FIR could skip a few iterations, while some of the tasks that perform biquad could skip one iteration at most, since the number of iterations is bigger in FIR than in the case of the biquad cascade. In Table 4 we present an example of a part of an output containing tasks where loop perforation was applied, where the maximal acceptable error is 10^{-6}. In the table, for each task, we show the original number of loop iterations, the number of loop iterations after applying loop perforation, and N, where every N-th loop is skipped. The actual error of this example is $7.8 \cdot 10^{-7}$, but we still achieve significant energy savings.

Table 4. Result of an experiment: tasks whose final number of loop iterations has been changed.

Task	Original num. of loop iterations	Final num. of loop iterations	N
FIR97-1	97	87	9
FIR85-1	85	76	9
FIR121-1	121	108	9
FIR109-1	109	104	21
FIR97-2	97	96	96
FIR85-2	85	84	84
FIR121-2	121	120	120
FIR109-2	109	108	108
FIR97-3	97	87	9
FIR85-3	85	76	9
FIR121-3	121	108	9
FIR109-3	109	97	9
FIR85-4	85	84	1
FIR121-3	121	81	3
FIR109-3	109	97	9

4 Related Work

In the existing literature techniques that include QoS as an objective in scheduling are mainly designed for Grid or Cloud Computing environments, where QoS is measured as either execution time, cost, etc., which has to be provided according to the signed Service Level Agreement (SLA) between the provider and the customer [10–12]. Multiobjective genetic algorithms were used in [12] to minimize cost and execution time, since they can be in conflict. A similar approach is presented in [11]. However, in the recent past, energy consumption has become a bottleneck, so it has become very important to reduce it. One such work is given in [10], where the authors try to minimize energy and maximize QoS at the same time in a Cloud Computing environment. The multiobjective optimisation problem is solved using particle swarm optimisation.

However, as far as we know, none of the approaches in the literature propose to trade-off QoS (accuracy in our case) with energy or performance in a scheduling problem by transforming the code, in our case by using loop perforation.

5 Conclusions

In this work we have presented an approach for energy efficient scheduling in multicore environments, adapted to multicore XMOS processors, where significant additional energy can be saved if a certain level of accuracy reduction in final result is allowed. Accuracy reduction is performed by using the loop

perforation technique. Our experimental results show that, even with small acceptable levels of error in the result, significant energy savings can be obtained.

However, the energy estimation of different schedules is based on a static analysis that can only provide an upper bound. Although it is still capable of providing energy savings, better results could be achieved with more precise energy estimations. For this reason, we are developing an energy analysis of concurrent program, which is expected to provide additional savings.

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 Madrid TIC-1465 *PROMETIDOS-CM* project.

References

1. Banković, Z., Lopez-Garcia, P.: Stochastic vs. deterministic evolutionary algorithm-based allocation and scheduling for XMOS chips. Neurocomputing **150**, 82–89 (2014)
2. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**, 182–197 (2000)
3. Kerrison, S., Eder, K.: Energy modelling of software for a hardware multi-threaded embedded microprocessor. ACM Trans. Embed. Comput. Syst. (TECS) (2015, to appear)
4. Liqat, U., Kerrison, S., Serrano, A., Georgiou, K., Lopez-Garcia, P., Grech, N., Hermenegildo, M.V., Eder, K.: Energy consumption analysis of programs based on XMOS ISA-level models. In: Gupta, G., Peña, R. (eds.) LOPSTR 2013, LNCS 8901. LNCS, vol. 8901, pp. 72–90. Springer, Heidelberg (2014)
5. López-García, P. (ed.) Initial Energy Consumption Analysis. ENTRA Project: Whole-Systems Energy Transparency (FET project 318337), April 2014. Deliverable 3.2, http://entraproject.eu
6. XMos Ltd., Xs1-su01a-fb96 datasheet, November 2012
7. Hoffmann, H., Misailovic, S., Sidiroglou, S., Rinard, M.: Managing performance vs. accuracy trade-offs with loop perforation. In: Proceedings of FSE 2011. ACM Press (2011)
8. Serrano, A., Lopez-Garcia, P., Hermenegildo, M.: Resource usage analysis of logic programs via abstract interpretation using sized types. In: Theory and Practice of Logic Programming, 30th Int'l. Conference on Logic Programming (ICLP 2014) Special Issue, 14(4–5):739–754, (2014)
9. Watt, D.: Programming XC on XMOS Devices. XMOS Limited (2009)
10. Yassa, S., Chelouah, R., Chelouah, R., Granado, B.: Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. Sci. World J. 2013, Article ID: 350934, 1–13 (2013)
11. Ye, G., Rao, R., Li, M.: A multiobjective resources scheduling approach based on genetic algorithms in grid environment. In: Fifth International Conference on Grid and Cooperative Computing Workshops, GCCW 2006, pp. 504–509, October 2006
12. Yu, J., Kirley, M., Buyya, R.: Multi-objective planning for workflow execution on grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, GRID 2007, pp. 10–17. IEEE Computer Society, Washington, DC (2007)