Journal of Circuits, Systems, and Computers
Vol. 27, No. 9 (2018) 1850145 (25 pages)

DOI: 10.1142/S0218126618501451

AC-DSE: Approximate Computing for the Design Space Exploration of Reconfigurable MPSoCs*

Arsalan Shahid

School of Computer Science, University College Dublin, Dublin 4, Ireland
arsalan.shahid@ucdconnect.ie

Muhammad Yasir Qadri† and Martin Fleury‡

School of Computer Science and Electronic Engineering,
University of Essex, Colchester, Essex CO4 3SQ, UK
†*yasirqadri@acm.org*
‡*fleum@essex.ac.uk*

Hira Waris§, Ayaz Ahmad¶ and Nadia N. Qadri‖

COMSATS Institute of Information Technology,
CIIT Wah, Wah Cantt, Pakistan
§*hira@emwi-tech.org*
¶*ayaz.ahmad@ciitwah.edu.pk*
‖*nadianqadri@googlemail.com*

Received 14 October 2016
Accepted 13 November 2017
Published 5 January 2018

This paper concerns the design space exploration (DSE) of Reconfigurable **M**ulti-**P**rocessor **S**ystem-on-**C**hip (MPSoC) architectures. Reconfiguration allows users to allocate optimum system resources for a specific application in such a way to improve the energy and throughput balance. To achieve the best balance between power consumption and throughput performance for a particular application domain, typical design space parameters for a multi-processor architecture comprise the cache size, the number of processor cores and the operating frequency. The exploration of the design space has always been an offline technique, consuming a large amount of time. Hence, the exploration has been unsuitable for reconfigurable architectures, which require an early runtime decision. This paper presents Approximate Computing DSE (AC-DSE), an online technique for the DSE of MPSoCs by means of approximate computing. In AC-DSE, design space solutions are first obtained from a set of optimization algorithms, which in turn are used to train a neural network (NN). From then on, the NN can be used to rapidly return its own solutions in the form of design space parameters for a desired energy and throughput performance, without any further training.

Keywords: Approximate computing; design space exploration; multi-core architecture.

*This paper was recommended by Regional Editor Emre Salman.
‡Corresponding author.

1. Introduction

In the past, computing platforms have always been developed with accuracy in mind, following the principle that every digital computation must be executed correctly. Conversely, there are now computing platforms capable of "Approximate Computing"[1] in order (say) to increase energy consumption efficiency at the cost of accuracy. This is possible owing to the fact that certain applications do not require strict accuracy but rather a limited accuracy or inexactness. For example, 12 applications, when analyzed for their resilience to inaccuracies,[2] were found to be suitable for approximate computing. These included the following: image searching, image segmentation, eye recognition and document searching, all of which are mainstream applications. By means of Approximate Computing for Design Space Exploration (AC-DSE) of Multi-Processor System-on-Chips (MPSoCs), we show how to obtain approximate design space solutions with respect to two desired objectives i.e., energy consumption and throughput. That is to say, DSE itself is treated as an application that is suitable for approximate computing. To our knowledge, this is the first implementation of approximate computing in the area of multi-objective DSE for MPSoC reconfiguration.

To identify design space solutions, we first apply various optimization algorithms, i.e., Nondominated Sorting Genetic Algorithm-II (NSGA-II), Multi-objective Simulated Annealing (MOSA), Multi-objective Particle Swarm Optimization (MOPSO), Adaptive Windows Pareto Random Search (APRS), Simple Evolutionary Multi-objective Optimizer (SEMO), Fair Evolutionary Multi-objective Optimizer (FEMO) and Greedy Evolutionary Multi-objective Optimizer (GEMO) (refer to Sec. 3.3). The existing optimization algorithms can be run in time-parallel fashion. These solutions in turn form the input for training a neural network (NN) and in return, we obtain approximate solutions for the desired objectives, i.e., improved energy consumption and throughput. The gain from this procedure is two fold: (1) the NN is trained as per knowledge of established artificial intelligence (AI) algorithms without implementing the algorithms in real-time, thus avoiding excessive computational complexity and (2) the *good-enough* approximate solution helps the real-time implementation of the DSE process, so that the MPSoCs can be configured for a particular application.

It is important to understand, that though existing optimization algorithms are still first employed, once the NN is trained, these algorithms are no longer needed. As a result, the NN can quickly find the design space parameters for any reasonable combined energy and throughput goals. Because it is usual to explore different tradeoffs between energy, throughput and the resources needed in an MPSoC configuration, the NN method saves time whenever a new configuration is sought. If only one configuration is sought, then obviously there is no time saving. However, if the MPSoC is to be reconfigured for many different applications each with different energy and throughput characteristics, then the saving in time will accumulate.

The saving will be greater if the original optimization algorithms are run in time-parallel fashion because obtaining the initial inputs take the duration of the longest algorithm, not the time for running each optimization algorithm in turn. The NN does not directly search for an optimal solution but obtains an approximate solution. Consequently, the main concern, which the results in this paper concentrate on, is the accuracy of an NN solution compared to the average solutions provided by the optimization algorithms. In this paper, the NN arrives at approximate solutions that differ by no more than 10% when compared to the average of the original inputs to the NN. In other words, AC-DSE represents a good compromise between accuracy and the desirable real-time operation.

For a computing platform architect, the values of parameters such as cache size, number of processor cores and operating frequency are critical to the success of a configuration. Most research in this domain has focused on a verification methodology based on either Transaction Level Modeling (TLM)[3] or virtualized platforms.[4,5] Unfortunately, a single configuration can take several hours for complete evaluation to take place, compared to a typical few seconds of NN operation. The process of exploring a number of configurable parameters in order to strike a balance between the energy and the throughput of the system is termed DSE.[6] DSE is normally considered to be a design time, offline technique. Thus, existing DSE methods can be time-consuming and offline, while the proposed NN-based method, once the NN is trained, is more rapid and can be performed in online fashion.

In the case of reconfigurable MPSoCs, online operation is particularly desirable. Reconfigurable multi-processor architectures, owing to their ability to adapt to per-application requirements, provide a better application-specific performance/power ratio[7] than static architectures of this type. An efficient multi-processor architecture according to the workload requirement will be the one that can optimize the number of processor cores, the size of cache memory and operating frequency (or some other parameter, for example, cache associativity or memory bandwidth). Thus, it is, that in the case of energy-aware reconfigurable architectures, an early decision which is often required to evaluate the impact of a particular configuration beforehand.[8] As a result, an efficient online technique has always been sought after in order to assist the reconfiguration engine. Furthermore, existing DSE mostly consists of solving a multi-objective optimization (MOO) problem,[9] which implies concurrent optimization of conflicting objectives. Hence, these problems mostly have various optimal solutions rather than one single, perfect solution. The solution set is called a Pareto set and the respective solutions are called Pareto-optimal solutions.[10] Therefore, approximate computing is indeed suitable for MPSoC DSE: not only are rapid configuration decisions required but a set of optimal solutions is obtainable as input to the approximate computing engine.

Multi-processor architectures have entered the mainstream as a way to increase performance within an SoC platform.[11] When it comes to the efficiency of an

MPSoC, energy consumption and throughput are considered to be the key objective parameters, judging by the number of researchers who have paid attention to these parameters (see Sec. 2). Moreover, there exists a trade-off between these two objectives.[12] It is a challenge to resolve that trade-off, which has led many researchers and engineers to apply themselves to this area. Thus, many techniques have evolved and been employed and some of these have been commercialized.[13,14] Approximate computing, through NNs, represents a new way to address the trade-off. It enables an efficient hardware and software implementation of a processor by approximation of the configuration calculations for an embedded application.[15]

This paper contributes to the state-of-the-art by

- solving the problem of parametric design of a multi-processor architecture using approximate computing for combined energy and throughput efficiency, which is to our knowledge the first such application;
- making a decision using a knowledge-base generated from the outputs of seven different optimization algorithms, i.e., NSGA-II, MOSA, MOPSO, APRS, SEMO, FEMO and GEMO;
- using the NN technique, the simulation latency of all the algorithms can be offset and the NN can generate quasi-optimal results in real-time.
- as the knowledge-base already supplies a range of optimal solutions (according to the seven existing algorithms), showing in the results how close the NN output approaches the input from these algorithms.

The rest of the paper is organized as follows. Section 2 is a review of the research literature with respect to approximate computing and DSE. Then, Section 3 explains why the chosen design space metrics were selected. Section 3 goes on to explain the methodology employed to construct and evaluate AC-DSE, including optimization algorithms, benchmarking applications and the NN training process. AC-DSE was then evaluated and selected results are presented in Sec. 4. As the NN runs much faster than any of the optimization algorithms, we concentrate in the evaluation on how closely the NN output approximates to optimal energy and throughput goals. Finally, Sec. 5 rounds off this paper with concluding remarks.

2. Related Work

The drive towards low-power processing has challenged designers and researchers to optimize every component of the processor.[16] However, if energy is optimized, throughput decreases, which overall may result in a limited gain. Thus, there is a need for joint optimization of energy and throughput. DSE serves as a tool for such system optimization by exploring multiple design parameters such as cache size, the number of cores and the operating clock frequency,[17] as indeed, we do in this paper. This section presents DSE research in the area of approximate

computing: approximate computing's usage for various applications; and the different optimization algorithms and tools that have been developed.

Approximate computing has attracted the attention of engineers and researchers, owing to its capability to trade-off computational accuracy for energy efficiency and/or throughput gains.[18] Thus, in this research, energy and throughput are emphasized at a cost in application accuracy, which differs from our usage of approximate computing in which computational accuracy is traded-off against real-time operation. Nevertheless, approximate computing for energy reduction remains indirectly relevant for the techniques employed.

2.1. *Approximate computing*

An under-designed architecture has been proposed[19] that guarantees the correctness of operations performed on the most significant data, while allowing approximation for the less significant data. The proposal achieved up to 35% energy savings with the significance-based computing. Then Moreau *et al.*[20] implemented Systolic NN Accelerator in Programmable logic (SNNAP) on the Zynq SoC and demonstrated an average of as much as 3.8 times speedup and 2.8 times energy savings on average over software execution by using approximate computing techniques. Venkatesan *et al.*[21] proposed constructing an untimed circuit that represents the behavior of an approximate circuit and provided an evaluation methodology to compare the approximate circuit with the original implementation. Sjalander *et al.*[22] considered the design of a tunable cache memory hierarchy that could be achieved by a combination of approximate computing and emerging multi-value devices. Moreover, they reduced the amount of error by storing approximate data in quaternary format and precise data in binary format. Gopalakrishnan *et al.*[23] implemented an automated differential verification which could make formal guarantees of program approximations.

As will already be apparent, much of the research into approximate computing is to identifying suitable applications and to reducing the error resulting from approximate operation. Chippa *et al.*[2] presented a technique for characterizing error resilience in applications that are based on approximate adders. Then, Grigorian *et al.*[24] presented a methodology for performing adaptive error analysis and recovery based on high-level, application-specific metrics or Light-Weight Checks (LWCs). They exploited imprecision tolerance at the application level. The resulting methodology adapts to output quality at runtime, providing a guarantee for the worst-case application-level error. The metrics are integrated with applications to enable compatibility with any approximate acceleration technique. The authors claimed[24] that their approach effectively employs LWCs to minimize the overhead from error analysis and initiates recovery according to need. Mishra *et al.*[25] discussed Intel™'s Approximate Computing Toolkit (iACT), which is targeted at analyzing approximations in applications. Pekhimenko *et al.*[26] proposed a system for applying the idea

of approximate computing to predict the cache miss ratio in order to reduce the overall memory latency by trading-off application accuracy. Their results show that they predicted 62% of Level 1 ($L1$) cache misses resulting in a 12.5% speedup. Additionally, 56% of $L2$ cache misses were predicted, leading to a performance improvement of 69%.

2.2. *Design space exploration*

DSE for MPSoCs, as developed in this paper, is an important field of research, aimed at achieving higher throughput with lower energy consumption. Many optimization algorithms have previously been used by researchers to explore an optimal design space for different reconfigurable architectures. Givargis *et al.*[27] proposed Platune, which is an optimization framework that exploits the concept of parameter independence to differentiate between approximate Pareto curves without performing an exhaustive search over the whole design space. Platune offered many configurable parameters and subsequently pruned the search space by isolating interdependent parameters from independent parameters. However, the level one cache parameters, being dependent, were explored exhaustively. Nonetheless, Platune is not a general framework because only design space exploration for MIPS-based systems is possible. Moreover, performance comparisons with other simpler techniques, i.e., random search or full parameter space exploration, were not presented.[27] Alternatively, Palermo *et al.*[28] demonstrated a discrete particle swarm optimization (DPSO) methodology for MOO of various search-space parameters by using aggregation techniques and formulating PSO for the multi-objective domain. Their main focus was intended to optimize energy consumption, but energy consumption minimization without affecting throughput is actually not possible.[29]

Various optimization techniques have been tried out in the quest for approximate computing. In Ref. 30, Wang *et al.* employed ant-colony optimization for DSE to minimize the completion time of applications, while effectively utilizing computational resources. Their method[30] focused on achieving a trade-off between hardware cost and timing performance. Then, Beltrame *et al.*[31] used design-time analysis strategies to generate multiple mappings for the application for multi-processor platforms. They minimized the number of simulations needed to detect the mappings while providing energy/delay trade-offs. The authors explored two design parameters, i.e., the optimum number of processors and cache sizes. Genetic algorithms are another natural-world optimization algorithm, along with fuzzy logic, NNs and ant-colony. Palesi *et al.*[32] utilized genetic algorithms for DSE of parametrized SoC to find Pareto-optimal configurations.

Frameworks and toolsets have also appeared in recent research into DSE. Calborean *et al.*[33] proposed a new software tool named Framework for Automatic DSE (FADSE). This tool solves single and MOO problems but the tool is not capable of handling large design spaces. Kang *et al.*[10] presented a search and Machine

Learning-based Framework for Fast Multi-core DSE and Optimization (Magellan). Magellan can be used as an automated tool for exploration/optimization. Zaccaria *et al.*[34] presented Multi-cube Explorer, which is an open-source framework for the DSE of MPSoCs. It enables the designer to automatically explore a design space of configurations for a parametrized architecture. It is command-line/script driven and can be integrated with any configurable simulator. Further, Ortego *et al.*[35] developed SuperESCalar Simulator (SECS), which is a microprocessor architectural simulator. It models a full out-of-order pipeline with branch prediction, caches, buses and every other component of a modern processor necessary for accurate simulation.

However, in the literature, to our knowledge, we have not found any technique, whether offline or online, that uses approximate computing itself to explore design space parameters to target minimum energy consumption and higher throughput.

3. Methodology

With the ever increasing complexity of MPSoCs, a variety of design space parameters can be considered to best fit the trade-offs involved to achieve a set of desired objectives. This process is known as DSE involving a MOO problem. It is generally recommended to remove some parameters of least interest from the design space. In our paper, we have scaled down the design space to only include those parameters that are expected to have a significant impact on energy consumption and throughput. Hence, cache size, number of processor cores and clock operating frequency were chosen as the configurable parameters in the design space of MPSoC architecture. The importance of these three parameters is as follows:

(1) Simply increasing the number of cores will not necessarily increase the throughput according to Amdahl's law for speedup.[36] Speedup is dependent on the fraction of code that is parallel and the number of processing elements. For example, synchronization of operations on different processing elements is a sequential overhead, as is any communication of intermediate results between those elements. Therefore, the number of cores for any application needs to be chosen carefully. The set containing the number of cores can be represented as $\{1, 2, 3, \ldots, N\}$, where N is the highest number of cores that the architecture can use while being configured.
(2) Cache size is considered to have a significant impact on cache performance and energy consumption.[37] The L_1 cache sizes considered herein can be represented in the form of 2^k KB, where $k = 0, 1, 2, \ldots, k$.
(3) Energy consumption depends upon the rate of memory accesses.[38,39] Consequently, the frequency at which a program executes significantly affects energy efficiency and throughput. Hence, frequency scaling is always catered in a

reconfigurable architecture. Therefore, the CPU frequency is included in the design space of this paper and a set of operating frequencies for the architecture can be represented in the form of $F = \{f_1, f_2, \ldots, f_i\}$, where i can be any positive integer number.

3.1. *DSE objective*

Energy consumption has always been a key concern in reconfigurable architectures.[40] However, it is known that a reduction in energy consumption without affecting throughput is impossible. With a decrease in energy consumption, the throughput decreases. Therefore, it is necessary to strike a balance between these two objectives, i.e., energy and throughout. As mentioned in Sec. 1, in the past, DSE has consisted of offline techniques that in general take a significant amount of time to complete evaluation for a single configuration. Hence, DSE has always been a time-consuming process. On the other hand, reconfigurable processor architectures require design parameters at runtime in order to evaluate the impact of these parameters at an early stage. Therefore, an online technique to assist modern energy aware reconfiguration engines at runtime is required. The DSE objective can be formally represented as follows:

$$F(\mathbf{X}) = \begin{cases} \underset{\mathbf{x}}{\text{minimize}} & E(\mathbf{X})\,, \\ \underset{\mathbf{x}}{\text{maximize}} & T(\mathbf{X})\,, \end{cases} \tag{1}$$

$$\text{where} \quad \mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x_D}\}\,.$$

In Eq. (1), $E(\mathbf{X})$ and $T(\mathbf{X})$ are the functions representing energy consumption and throughput, respectively. $\mathbf{X}$ represents the configuration being evaluated, consisting of design variables, i.e., the number of cores, the cache size and the operating frequency. D is the total number of design variables, i.e., $D = 3$ in AC-DSE. Therefore, x_1 is the number of cores, x_2 is the cache size and x_3 is the operating frequency.

3.2. *AC-DSE in more detail*

AC-DSE obtains design space solutions through a variety of optimization algorithms, which have been evaluated by standard benchmark tools such as SPLASH-2.[41] The optimization algorithms and benchmarking tools are further escribed in Secs. 3.3 and 3.4. The design space solutions act as training material for an NN. Sample input to the NN is shown in the appendix (Tables A1, A2, and A3). Figure 1 shows the composition of the NN of the AC-DSE in which the trained NN is

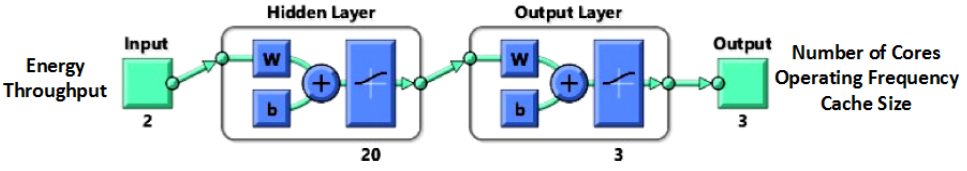


Fig. 1. Proposed NN.

Table 1. NN parameters.

Network type	Feed-forward back propagation
Input parameters	2
Output parameters	3
Training algorithm	Bayesian regularization
Performance function	MSE
Number of neurons (hidden layer)	20
Network layers	2
Transfer function	Logistic

required to provide the optimum design space parameters for a desired energy consumption and throughput.

The operating configuration of the NN appears in Table 1. Input parameters to the NNs are the desired energy consumption and throughput and the three output parameters are the design space parameters. Empirically, it was seen that the input data best fit a log function. Therefore, the transfer function used is the logistic regression. From trial tests, these parameters appear to be the most suitable parameters for the given data sets and for training the network. Moreover, these parameters train the network well in the sense that they prevent an over-fit or an under-fit.

3.3. *Optimization algorithms*

DSE mostly considers MOO problems, which imply concurrent optimization of conflicting objectives. Hence, these problems mostly have various optimal solutions rather than one single solution. As previously mentioned in Sec. 1, the solution set is known as a Pareto set and the respective solutions are known as pareto-optimal solutions. In AC-DSE, the goal is to use as many as possible of these optimal solutions generated by various established optimization algorithms as a knowledge-base to train an NN.

This section now presents the optimization algorithms considered for exploration of the design space.

(a) **NSGA-II:** This algorithm is an evolutionary algorithm that is based on the Darwinian theory of survival of the fittest.[42] It uses nondominated sorting. The genetic operators, selection, cross over and mutation, select individuals and cause the formation of new populations.

(b) **MOPSO:** PSO is a meta-heuristic that mimics the behavior of birds searching for food.[43] The concept of social interaction is applied in this algorithm for problem solving. The number of particles constitutes a swarm moving around the search space to find the optimal solution. Each particle in the swarm has a particular position and velocity. The particles exchange this information with each other to arrive at the best solution.

(c) **MOSA:** Simulated annealing is motivated by the annealing process of heated solids. In this algorithm, annealing temperature plays a vital role in the acceptance criterion.[44] The temperature is gradually reduced to get the desired results. A new configuration is constructed in this algorithm by imposing a random displacement.

(d) **APRS:** This algorithm has a dynamic window size.[45] The size of window is reduced as the number of iterations increases and also depends upon the goodness of design point found in the current window. The windows are centered on the current pareto solutions and the new configurations are randomly selected within the windows.

(e) **SEMO:** This algorithm is a population-based multi-objective evolutionary algorithm.[46] It consists of a variable size population and stores all nondominated solutions. A genetic mutation is applied on a parent drawn from a randomly chosen population according to some probability distribution. The child is included in the population if no other population dominates it and its objective function value is not already present.

(f) **FEMO:** This algorithm is an improved version of SEMO.[47] A fair selection strategy is implemented in this algorithm by counting the number of times an individual has undergone mutation. It guarantees the individual that has produced the least number of children is selected for mutation; ties are broken randomly.

(g) **GEMO:** This algorithm is an extended version of FEMO.[48] In it, the offspring of the most recently successful mutant is selected and all the search effort is allocated to it.

3.4. *Benchmark applications*

Benchmarks are the standard tools for the performance evaluation of a microprocessor. For evaluation of the design space, a number of benchmark applications were used from SPLASH-2,[41] the benchmarking suite. Refer to Table 2 for a run-down of these applications.

3.5. *Learning algorithms*

For training the NN, Bayesian regularization acted a learning algorithm. This algorithm gives optimal regularization parameters in an automated fashion.[53] In Bayesian regularization, the weight and bias values are updated according

Table 2. Benchmark applications from SPLASH-2.

Benchmarks	Description
FMM	Implementation of a parallel adaptive fast multi-pole method to simulate the N-body problem.[49]
OCEAN	Simulation of large-scale ocean movements based on eddy and boundary currents.[50]
BARNES	Implementation of the Barnes–Hut method to simulate the N-body problem.[51]
Water spatial	Evaluation of the forces and potential that occur over time in a system of water molecules using the predictor–corrector method.
Water N-squared	Imposes a three-dimensional (3D) spatial data structure on the cubical domain to solve the molecular dynamics N-body problem.
Cholesky	Implementation of blocked Cholesky factorization on a sparse matrix.[52]
LU	Factorization of a dense matrix into the product of a lower triangular and an upper triangular matrix.[50]

to Levenberg–Marquardt,[54] which is an algorithm that trains NNs 10 to 100 times faster, as compared to the commonly used gradient-descent, back-propagation method. After minimizing the squared errors and weights combination, the algorithm discovers the optimal combination to make a network that generalizes well. Moreover, while employing Bayesian regularization, training of the network until it reaches convergence is very important. When the network has converged, the sum squared error, the sum squared weights and the effective number of parameters should reach constant values. A better generalized performance is given by Bayesian regularization when used as a training function because it uses all the data and does not require separating the validation data out of the training set.

3.6. *Simulation setup*

The overall workflow of the simulation is summarized in Fig. 2. In AC-DSE, the goal is to use optimal solutions generated by various genetic algorithms as a knowledge-base to train an NN. After setting up a simulator to construct an NN, various genetic algorithms operating on SPLASH-2 benchmarks were employed to generate the training data for the NN, shown as input to the NN in Fig. 2. Once the NN is trained, it is possible to enter an arbitrary desired energy constraint and throughput configuration into the NN. The trained NN is then able to output a design space solution that specifies the number of cores, operating frequency and cache size (the output hypothesis of Fig. 2). Thus, once the NN is trained, it can be used repeatedly and will return different design space solutions in response to different desired input configurations. Furthermore, there is no restriction to inputting a configuration from the appendix, as the input configuration is not constrained by the training set. Though in Sec. 4, input from the Appendix (Tables A1, A2, and A3) test set (not the training set) was used to check the operation and accuracy of the NN, a trained NN operates independently of its input configuration.[55] Clearly, however, the NN is restricted to this application, as it will only accept as input an energy constraint with throughput

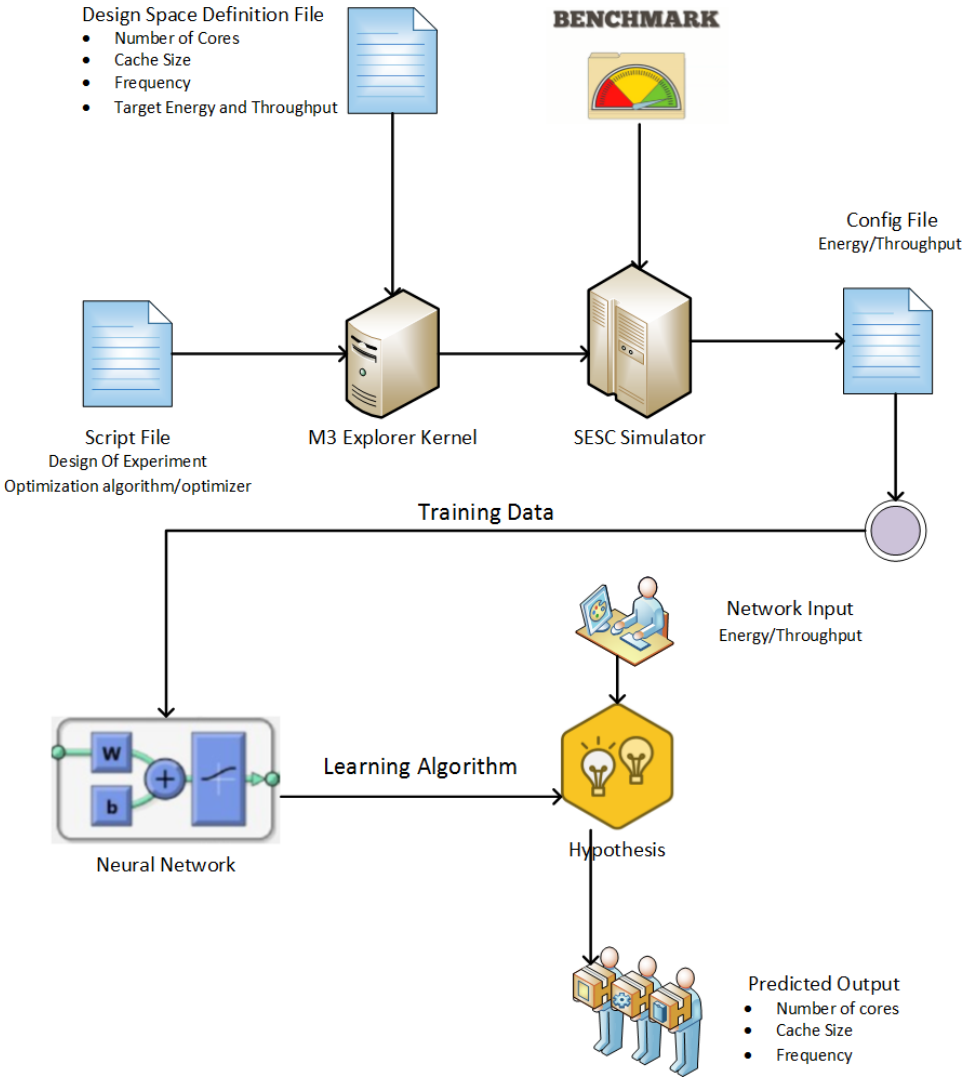


Fig. 2. Overall workflow.

configuration and will only output a solution specifying the same three parameters, namely, the number of cores, operating frequency and cache size.

To explore the design space parameters Multi-cube/M3 Explorer, a DSE framework, was integrated with the SESC simulator,[35] which is an event-driven microprocessor architecture simulator. The targeted operating system was Ubuntu 13.10 (kernel 3.1.1, on an Intel Core i5, with 16 GB RAM, running at 3.2 GHz). Multi-cube explorer requires a script file in which the optimization algorithm (see Sec. 3.5) is specified along with the experimental design. A design space definition file is also

required by Multi-cube explorer (M3Explorer)[56] in which design space parameters and objective parameters are specified. M3Explorer generates optimal design solutions that meet AC-DSE's objectives in a Pareto set. SPLASH-2 benchmark applications (see Sec. 3.4) are run on the SESC simulator by taking virtual hardware parameters generated as a solution set by M3Explorer. SESC simulates benchmarks and results in an output configuration file having energy and number of cycles consumed by the application. The design space parameters with their respective energy consumption and throughput (see the appendix (Tables A1, A2, and A3)) train an NN with the parameters shown in Table 1. Finally, as previously mentioned, the NN generates a hypothesis, i.e., the number of cores, cache size and operating frequency required.

4. Findings

This section describes the verification of the detailed evaluation of the approximations returned by NN and correct operation of the AC-DSE system.

4.1. *Assessing the NN performance*

To analyze the performance of the NN, several experiments were performed. This section presents the results of those experiments for the NN parameters of Table 1. It should be noted that 70% of the data given in the appendix (Tables A1, A2, and A3) were used to train the NN and the remaining 30%, the test set, was used to test the trained NN.

Figure 3 shows the regression plots obtained after training the NN. These three plots represent the training, testing and complete data. The dashed line in each plot represents the perfect regression. The solid line represents the best fit linear regression line between the outputs and targets of the NN. The value of R indicates the relationship between network outputs and targets. $R = 1$ indicates that there is an exact linear relationship between outputs and targets, whereas, $R = 0$ would show that there is no linear relationship between outputs and targets. In our case, the value of R is 0.97 and 0.98 for training and testing, respectively.

The error has been shown in Fig. 4. The solid line represents zero error and it can be seen that most of the error lies in the range of -3 to 3. Thus, Fig. 4 confirms that the training data were sufficient for the trained NN to be able to predict the approximated solutions within the error range of 3% for over 90% of the samples, which met our original specification for the AC-DSE system. Given that each of the algorithms employed to generate the training data is a well-established way of generating an optimized solution (refer back to Sec. 3.3), the implication of the error statistics is that, after the training procedure, the output hypotheses of the trained NN can be relied upon. Reducing the number of training algorithms would reduce the accuracy of the results and vice versa increasing the number of training algorithms holds out the possibility of improving accuracy further.

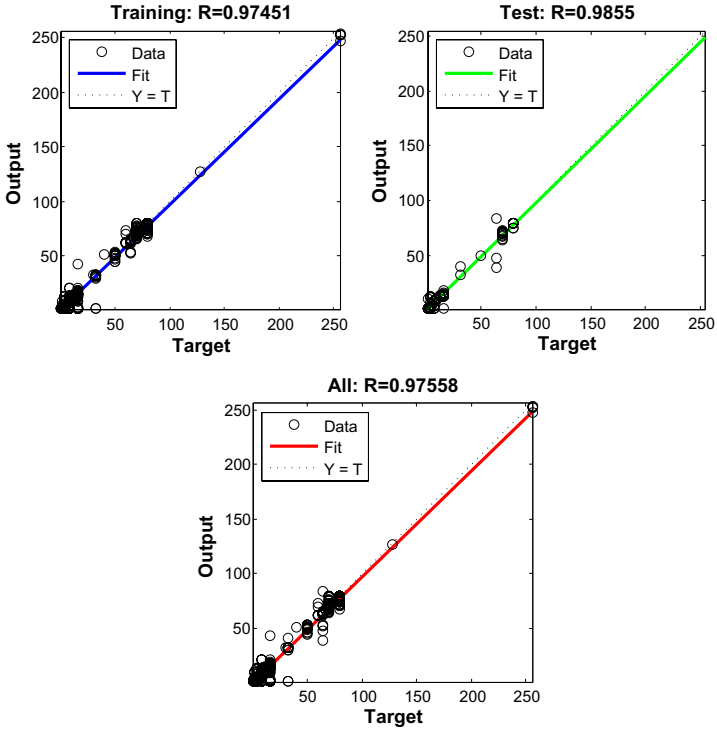


Fig. 3. Regression plots.

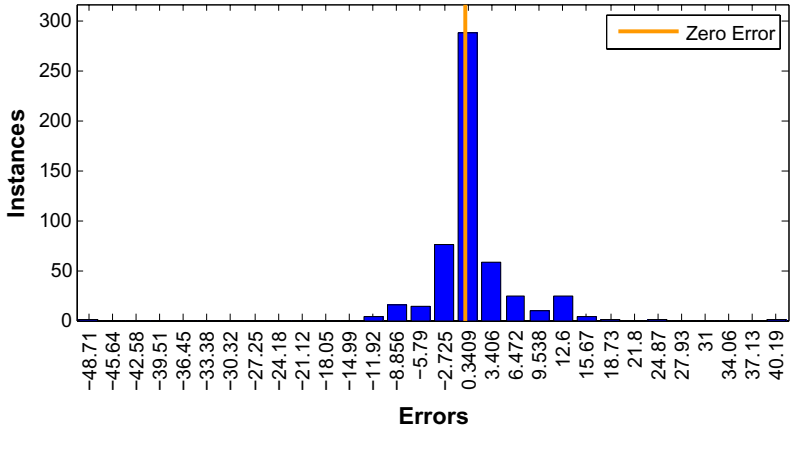
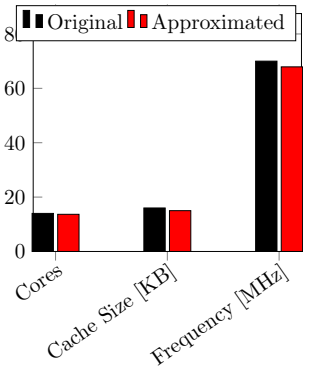
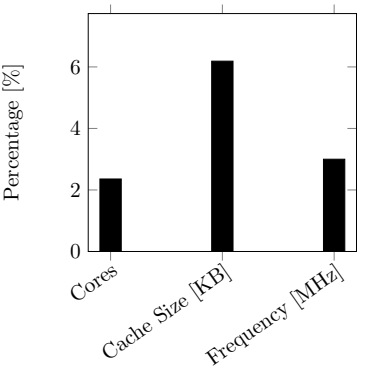


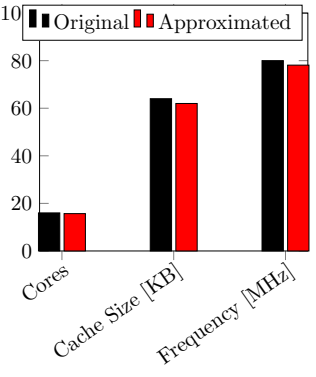
Fig. 4. Trained network errors.



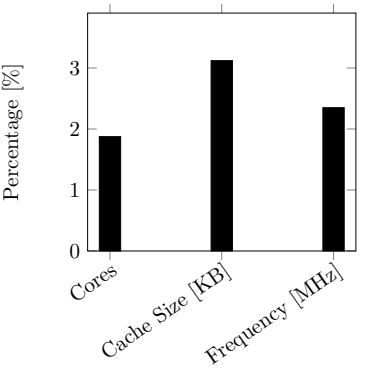
(a) Energy = 0.642 J and throughput = 0.812



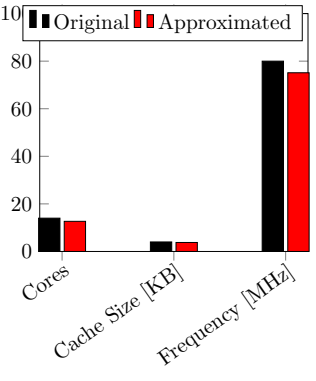
(b) Error percentage



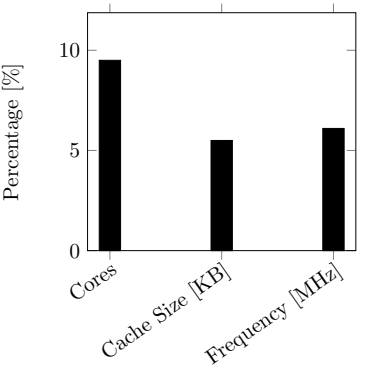
(c) Energy = 0.724 J and throughput = 0.993



(d) Error percentage



(e) Energy = 0.662 J and throughput = 0.926



(f) Error percentage

Fig. 5. NN verification tests.

4.2. *Verifying the behavior of AC-DSE*

To verify the correct behavior of the AC-DSE system, the trained NN was checked by entering desired energy constraints and throughput from the test set. To demonstrate this verification procedure, Fig. 5 shows the results of entering three different energy constraint and throughput configurations, which were arbitrarily selected from the *test* data set. The outputs from the NN, also reported in Fig. 5, show the approximated design/search space solutions, namely the number of cores, operating frequency and cache size. Additionally given in Fig. 5 are the original inputs arrived by using the optimized algorithms.

From the accompanying error percentage bar charts, it will be seen that the error from using the approximated solutions in all cases was never greater than 10%. Also note that the NN solutions were returned instantaneously to the user.

Figures 6–8 show input/output relationships in terms of the three input parameters, i.e., number of cores (i.e., Fig. 6), CPU operating frequency (i.e., Fig. 7) and Cache size (i.e., Fig. 8), each against the two input parameters, i.e., energy constraint and throughput. The data are also compared against the original dataset, the pareto-optimal solutions output by the optimization algorithms of Sec. 3.3. In most of the cases, the predicted output overlaps completely or partially the input dataset. Thus, Figs. 6–8 demonstrate that, by using the trained NN (with the configuration parameters in Table 1), one can jointly satisfy a desired throughput given

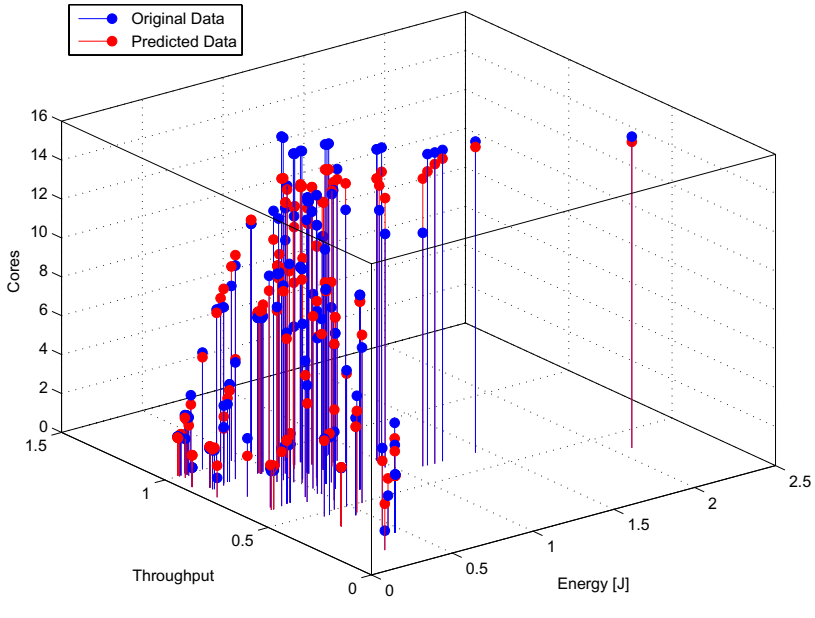


Fig. 6. NN output comparison for number of cores.

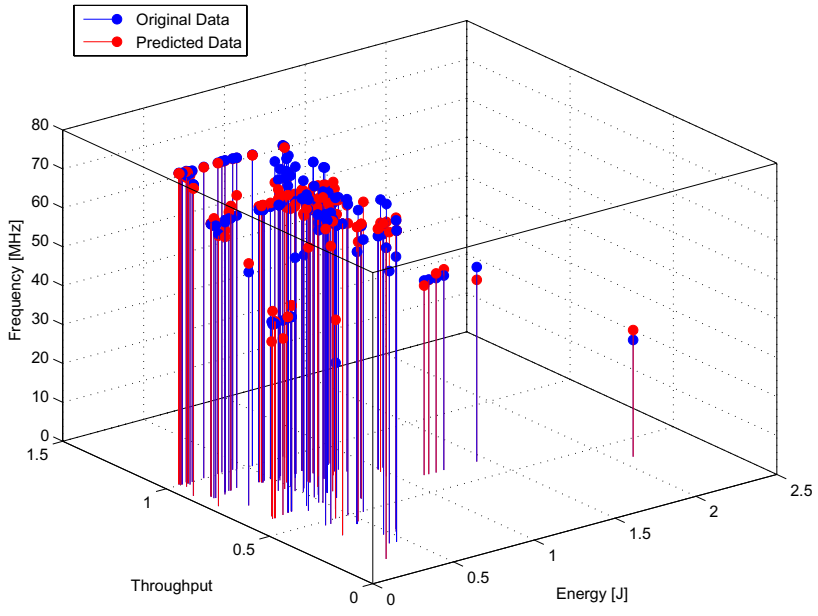


Fig. 7. NN output comparison for operating frequency.

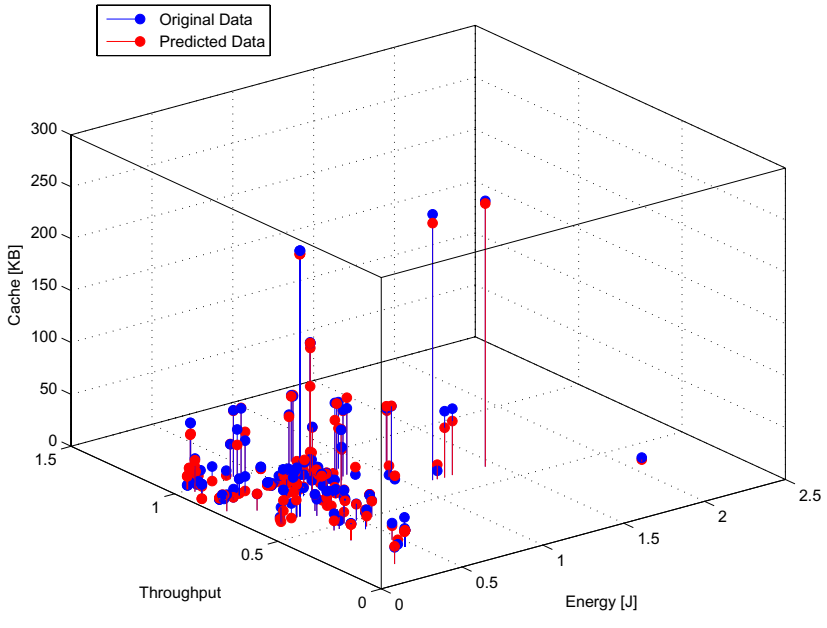


Fig. 8. NN output comparison for cache size.

an energy consumption constraint and at the same time, it comes close to an optimal solution.

This shows the effectiveness of the proposed approach, i.e., without using the computationally expensive algorithms, similar results can be achieved by using the approximate computing approach in the process of DSE for multi-core architectures.

5. Conclusion

This paper presented AC-DSE, which is an online technique for DSE of MPSoCs by means of approximate computing. AC-DSE provided a trade-off between energy consumption and throughput by the selection of efficient number of cores, cache size and operating frequency. It does this, after training on the inputs provided by existing optimization algorithms. In AC-DSE, design space solutions obtained from various optimization algorithms, i.e., NSGA-II, MOSA, MOPSO, APRS, SEMO, FEMO and GEMO, through standard benchmark applications and simulation tools were used. These solutions acted to train an NN.

NN results were found to only deviate at a maximum by 12% from the original inputs from optimization algorithms, though the NN's results, once training was complete, took only a small fraction of the time using the optimization algorithms. By subsequently and repeatedly using the trained NN, further solutions can be found to meet desired energy and throughput goals in real-time, thus compensating for the original training time.

In future work, AC-DSE will be used to explore more design space parameters with other optimization algorithms providing inputs. However, one can already see the effectiveness of the AC-DSE approach in terms of the limited loss of accuracy in return for real-time operation.

Acknowledgment

This work is supported by the National ICT R&D Fund, Pakistan, through Grant Number: ICTRDF/TR&D/2012/65.

Appendix

Table A.1. Sample input part 1.

No. of cores	Cache size [KB]	Frequency [MHz]	Energy [J]	Throughput
16	16	70	0.632	0.866
16	64	60	0.989	0.749
3	16	80	0.449	0.236
13	16	80	0.515	0.852
14	16	70	0.614	0.786
16	64	70	0.835	0.868
2	4	80	0.088	1.007
5	16	70	0.445	0.424
16	64	70	0.852	0.873
16	4	30	2.091	0.375
1	16	80	0.263	0.141
12	16	80	0.523	0.826
14	4	80	0.662	0.926
2	4	70	0.407	0.238
1	16	80	0.047	0.905
12	256	50	1.121	0.628
1	16	80	0.047	0.904
3	32	50	0.276	0.625
3	16	80	0.135	1
3	256	50	0.297	0.625
3	32	50	0.276	0.626
3	16	80	0.134	1.002
10	8	70	0.679	0.587
12	4	80	0.699	0.785
13	4	80	0.704	0.839
8	4	70	0.424	0.881
10	8	70	0.594	0.685
8	4	70	0.588	0.635
13	4	80	0.631	0.91
8	4	70	0.417	0.877
10	8	70	0.536	0.874
8	4	70	0.429	0.872
12	4	80	0.529	0.996
10	8	70	0.536	0.873
8	4	70	0.428	0.873
10	8	70	0.679	0.587
12	2	70	0.964	0.688

Table A.2. Sample input part 2.

No. of cores	Cache size [KB]	Frequency [MHz]	Energy [J]	Throughput
5	16	70	0.555	0.322
2	16	50	0.16	0.61
3	2	70	0.181	0.857
10	64	70	0.553	0.879
2	4	70	0.115	0.85
10	32	80	0.443	1.005
4	64	70	0.227	0.874
3	32	50	0.276	0.625
3	16	80	0.135	1
3	256	50	0.297	0.625
3	4	80	0.134	0.996
2	2	70	0.111	0.868
3	32	50	0.276	0.626
3	16	80	0.134	1.002
13	2	70	0.978	0.727
8	2	70	0.424	0.879
10	8	70	0.594	0.685
13	2	70	0.85	0.79
10	8	70	0.488	0.877
8	2	70	0.45	0.877
10	128	70	0.686	0.88
10	8	70	0.536	0.874
10	4	70	0.535	0.872
8	2	70	0.431	0.868
10	128	70	0.679	0.875
10	8	70	0.536	0.873
10	4	70	0.536	0.871
8	2	70	0.43	0.869
10	128	70	0.68	0.875
10	8	70	0.679	0.587
12	4	80	0.702	0.781
13	4	80	0.704	0.839
8	4	70	0.424	0.881
10	8	70	0.594	0.685
8	4	70	0.588	0.635
13	4	80	0.631	0.91
8	4	70	0.417	0.877
10	8	70	0.536	0.874
8	4	70	0.429	0.872
3	4	80	0.134	0.998
2	2	70	0.111	0.869
14	16	70	0.612	0.788
14	32	70	0.765	0.79
13	16	70	0.608	0.739
12	16	70	0.6	0.694
10	16	80	0.504	0.677
11	16	80	0.508	0.733
9	16	80	0.492	0.626
8	16	80	0.488	0.562
14	16	70	0.615	0.784
8	16	70	0.572	0.493

Table A.3. Sample input part 3.

No. of cores	Cache size [KB]	Frequency [MHz]	Energy [J]	Throughput
2	4	80	0.088	1.005
16	64	80	0.727	1.004
5	16	70	0.445	0.424
14	16	70	0.642	0.812
8	16	70	0.477	0.635
9	8	70	0.541	0.687
5	16	70	0.238	0.873
2	32	50	0.175	0.61
12	4	80	0.529	0.996
10	8	70	0.536	0.873
8	4	70	0.428	0.873
16	16	70	0.63	0.868
3	16	80	0.449	0.236
13	16	70	0.607	0.741
16	64	70	0.831	0.874
13	8	80	0.595	0.84
4	8	80	0.533	0.304
16	64	80	0.724	0.993
7	16	80	0.483	0.498
6	16	80	0.472	0.44
4	16	80	0.458	0.306
3	16	80	0.449	0.236
15	64	70	0.833	0.818
16	256	50	1.44	0.623
16	64	50	1.189	0.623
2	4	80	0.088	1.007
14	8	70	0.696	0.81
14	16	70	0.644	0.809
13	16	70	0.613	0.793
10	16	80	0.46	0.786
11	16	80	0.502	0.79
9	16	80	0.419	0.785
8	16	80	0.403	0.726
7	16	80	0.402	0.635
6	16	80	0.368	0.599
4	16	80	0.312	0.472
3	16	80	0.295	0.377
14	64	70	0.803	0.813
16	64	50	1.24	0.626
6	16	80	0.237	1.003
8	32	80	0.355	1.008
4	16	80	0.16	0.999
3	8	80	0.129	0.985
1	16	70	0.056	0.791
8	8	80	0.327	1.005
8	64	80	0.374	1.008
2	32	80	0.097	0.977
2	64	80	0.1	1
2	8	80	0.09	0.999
2	64	80	0.1	1.002
2	8	80	0.09	1

References

1. J. Han and M. Orshansky, Approximate computing: An emerging paradigm for energy-efficient design, *18th IEEE European Test Symp.* (IEEE, Avignon, France, 2013), pp. 1–6.
2. V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, Analysis and characterization of inherent application resilience for approximate computing, *50th Annual Design Automation Conf.* (IEEE, Austin, TX, USA, 2013), pp. 1–9.
3. L. Ferro, L. Pierre, Z. B. H. Amor, J. Lachaize and V. Lefftz, Runtime verification of typical requirements for a space critical SoC platform, *Formal Methods for Industrial Critical Systems* (Springer Verlag, Berlin, 2011), pp. 21–36.
4. Simics Full System Simulator is software supplied by Virtutech AB of Wind Rivers Systems at www.virtutech.com/.
5. A. Patel, F. Afram and K. Ghose, Marss-x86: A QEMU-based micro-architectural and systems simulator for x86 multicore processors, *1st Int. Qemu Users Forum* (Grenoble, France, 2011), pp. 29–30.
6. L. Ren, X. Yu, C. W. Fletcher, M. Van Dijk and S. Devadas, Design space exploration and optimization of path oblivious RAM in secure processors, *ACM SIGARCH Comput. Archit. News* **41** (2013) 571–582.
7. J. Fowers, G. Brown, J. Wernsing and G. Stitt, A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors, *ACM Tran. Archit. Code Optim.* **9** (2013) 25.
8. G. Keramidas *et al.*, Embedded reconfigurable computing: The ERA approach, *11th IEEE Int. Conf. Industrial Informatics* (IEEE, Bochum, Germany, 2013), pp. 827–832.
9. K. Deb, Multi-objective optimization, *Search Methodologies* (Springer Verlag, Berlin, 2014), pp. 403–449.
10. G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria and C. Silvano, A correlation-based design space exploration methodology for multi-processor systems-on-chip, *47th Design Automation Conf.* (IEEE, Anaheim, CA, USA, 2010), pp. 120–125.
11. M. Y. Qadri and S. J. Sangwine, *Multicore Technology: Architecture, Reconfiguration, and Modeling* (CRC Press, Boca Raton, FL, 2013).
12. G. Zhu, L. M. Davis, T. Chan and S. Perreau, Trade-offs in energy consumption and throughput for a simple two-relay network, *IEEE Australian Communications Theory Workshop* (IEEE, Melbourne, VIC, Australia, 2011), pp. 37–42.
13. M. Kang, S. K. Gonugondla, M.-S. Keel and N. R. Shanbhag, An energy-efficient memory-based high-throughput VLSI architecture for convolutional networks, *IEEE International Conference on Acoustics, Speech and Signal Processing* (IEEE, Brisbane, QLD, Australia, 2015), pp. 1037–1041.
14. N. Goswami, B. Cao and T. Li, Power-performance co-optimization of throughput core architecture using resistive memory, *IEEE 19th Int. Symp. High Performance Computer Architecture* (IEEE, Shenzhen, China, 2013), pp. 342–353.
15. S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, Quality programmable vector processors for approximate computing, *46th Annual IEEE/ACM Int. Symp. Microarchitecture* (IEEE, Davis, CA, USA, 2013), pp. 1–12.
16. A. Wang and S. Naffziger, *Adaptive Techniques for Dynamic Processor Optimization: Theory and Practice* (Springer Science & Business Media, New York, NY, USA, 2008).
17. K. Vanherpen, J. Denil, P. De Meulenaere and H. Vangheluwe, Design-space exploration in model-driven engineering — An initial pattern catalogue, Technical Report, McGill University (2014), SOCS-TR-2014.4.
18. C.-H. Huang, Y. Li and L. Dolecek, ACOCO: Adaptive coding for approximate computing on faulty memories, *IEEE Trans. Commun.* **63** (2015) 4615–4628.

19. H. Mamaghanian, G. Ansaloni, M. M. S. Aly, D. Atienza Alonso and P. Vandergheynst, Hardware–software inexactness in noise-aware design of low-power body sensor nodes, *Designing with Uncertainty-Opportunities & Challenges* (INFOSCIENCE, York, UK, 2014), pp. 1–3.
20. T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze and M. Oskin, SNNAP: Approximate computing on programmable SoCs via neural acceleration, *IEEE 21st Int. Symp. High Performance Computer Architecture* (IEEE, Burlingame, CA, USA, 2015), pp. 603–614.
21. R. Venkatesan, A. Agarwal, K. Roy and A. Raghunathan, MACACO: Modeling and analysis of circuits for approximate computing, *IEEE Int. Conf. Computer-Aided Design* (IEEE, San Jose, CA, USA, 2011), pp. 667–673.
22. M. Själander, N. S. Nilsson and S. Kaxiras, A tunable cache for approximate computing, *IEEE/ACM Int. Symp. Nanoscale Architectures* (2014), pp. 88–89.
23. G. Gopalakrishnan, A. Haran, S. K. Lahiri and Z. Rakamaric, Automated differential program verification for approximate computing, Technical Report, Microsoft Research (2015).
24. B. Grigorian and G. Reinman, Dynamically adaptive and reliable approximate computing using light-weight error analysis, *NASA/ESA Conf. Adaptive Hardware and Systems* (IEEE, Leicester, UK, 2014), pp. 248–255.
25. A. K. Mishra, R. Barik and S. Paul, iACT: A software–hardware framework for understanding the scope of approximate computing, *Workshop on Approximate Computing Across the System Stack* (ACM, Salt Lake City, Utah, USA, 2014), pp. 1–6.
26. G. Pekhimenko, D. Koutra and K. Qian, Approximate computing: Application analysis and hardware design, Technical Report, Carnegie-Mellon University (2010).
27. T. Givargis and F. Vahid, Platune: A tuning framework for system-on-a-chip platforms, *IEEE Trans. Comput.-Aided Des. Integr. Circuit. Syst.* **21** (2002) 1317–1327.
28. G. Palermo, C. Silvano and V. Zaccaria, Discrete particle swarm optimization for multi-objective design space exploration, *11th IEEE EUROMICRO Conf. Digital System Design Architectures, Methods and Tools* (IEEE, Parma, Italy, 2008), pp. 641–644.
29. M. Lukasiewycz, M. Glaß, C. Haubelt and J. Teich, Efficient symbolic multi-objective design space exploration, *IEEE Asia and South Pacific Design Automation Conf.* (IEEE, Seoul, South Korea, 2008), pp. 691–696.
30. G. Wang, W. Gong, B. DeRenzi and R. Kastner, Design space exploration using time and resource duality with the ant colony optimization, *43rd Annual Design Automation Conf.* (IEEE, San Francisco, CA, USA, 2006), pp. 451–454.
31. G. Beltrame, L. Fossati and D. Sciuto, Decision-theoretic design space exploration of multiprocessor platforms, *IEEE Trans. Comput.-Aided Des. Integr. Circuit. Syst.* **29** (2010) 1083–1095.
32. M. Palesi and T. Givargis, Multi-objective design space exploration using genetic algorithms, *Tenth Int. Symp. Hardware/Software Codesign* (IEEE, Estes Park, CO, USA, 2002), pp. 67–72.
33. H. Calborean and L. Vinţan, Framework for automatic design space exploration of computer systems, *Acta Univ. Cibin. Tech. Ser.* **1** (2011) 1–7.
34. V. Zaccaria, G. Palermo, F. Castro, C. Silvano and G. Mariani, Multicube explorer: An open source framework for design space exploration of chip multi-processors, *23rd Int. Conf. Architecture of Computing Systems* (IEEE, Hannover, Germany, 2010), pp. 1–7.
35. P. M. Ortego and P. Sack, SESC: SuperESCalar simulator, *17th Euro Micro Conf. Real-Time Systems* (IEEE, Madrid, Spain, 2004), pp. 1–4.

36. G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, *ACM Spring Joint Computer Conf.* (ACM, Atlantic City, New Jersey, 1967), pp. 483–485.
37. J. Bui, C. Xu and S. Gurumurthi, Understanding performance issues on both single core and multi-core architecture, *Computer Organization* (ACM, Charlottesville, VA, USA, 2007), pp. 1–8.
38. R. Kotla, A. Devgan, S. Ghiasi, T. Keller and F. Rawson, Characterizing the impact of different memory-intensity levels, *IEEE Int. Workshop on Workload Characterization* (IEEE, Austin, TX, USA, 2004), pp. 3–10.
39. A. Weissel and F. Bellosa, Process cruise control: Event-driven clock scaling for dynamic power management, *Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems* (ACM, Grenoble, France, 2002), pp. 238–246.
40. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach* (Morgan Kaufmann, Amsterdam, 2011).
41. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, The SPLASH-2 programs: Characterization and methodological considerations, *ACM SIGARCH Comput. Archit. News* **23** (1995) 24–36.
42. K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. Evol. Comput.* **6** (2002) 182–197.
43. C. A. C. Coello and M. S. Lechuga, MOPSO: A proposal for multiple objective particle swarm optimization, *IEEE Congress on Evolutionary Computation* (IEEE, Honolulu, HI, USA, 2002), pp. 1051–1056.
44. E. Ulungu, J. Teghem, P. Fortemps and D. Tuyttens, MOSA method: A tool for solving multiobjective combinatorial optimization problems, *J. Multicriteria Decis. Anal.* **8** (1999) 221–236.
45. C. Silvano, W. Fornaciari and E. Villar, *Multi-Objective Design Space Exploration of Multiprocessor SoC Architectures* (Springer Verlag, Berlin, 2014).
46. O. Giel, Expected runtimes of a simple multi-objective evolutionary algorithm, *IEEE Congress on Evolutionary Computation* (IEEE, Canberra, ACT, Australia, 2003), pp. 1918–1925.
47. M. Laumanns, L. Thiele, E. Zitzler, E. Welzl and K. Deb, *Running Time Analysis of Multi-Objective Evolutionary Algorithms on a Simple Discrete Optimization Problem* (Springer Verlag, Berlin, 2002).
48. M. Laumanns, L. Thiele and E. Zitzler, Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions, *IEEE Trans. Evol. Comput.* **8** (2004) 170–182.
49. J. P. Singh, C. Holt, J. L. Hennessy and A. Gupta, A parallel adaptive fast multipole method, *ACM/IEEE Conf. Supercomputing* (IEEE, Portland, OR, USA, 1993), pp. 54–65.
50. S. C. Woo, J. P. Singh and J. L. Hennessy, The performance advantages of integrating block data transfer in cache-coherent multiprocessors, *6th Int. Conf. Architectural Support for Programming Languages and Operating Systems* (ACM, San Jose, California, USA, 1994), pp. 219–229.
51. J. P. Singh, J. L. Hennessy and A. Gupta, Implications of hierarchical N-body methods for multiprocessor architectures, *ACM Trans. Comput. Syst.* **13** (1995) 141–202.
52. E. Rothberg and A. Gupta, An efficient block-oriented approach to parallel sparse Cholesky factorization, *SIAM J. Sci. Comput.* **15** (1994) 1413–1439.
53. F. Burden and D. Winkler, Bayesian regularization of neural networks, in *Artificial Neural Networks. Methods in Molecular Biology*, ed. J. Livingest, Vol. 458 (Humana Press, 2008), pp. 23–42.

54. J. J. Moré, The Levenberg–Marquardt algorithm: Implementation and theory, *Numerical Analysis* (Springer, Dundee, UK, 1978), pp. 105–116.
55. S. Haykin, *Neural Networks and Learning Machines* (Pearson, New York, 2008).
56. H. Calborean and L. Vinţan, An automatic design space exploration framework for multicore architecture optimizations, *9th RoEduNet IEEE Int. Conf.* (IEEE, Sibiu, Romania, 2010), pp. 202–207.