Exploring Multi-core Systems with Lifetime Reliability and Power Consumption Trade-offs

Dolly Sapra and Andy D. Pimentel

University of Amsterdam, Amsterdam, Netherlands
{d.sapra,a.d.pimentel}@uva.nl

Abstract. Embedded multicore systems are often built for a specific application, operating a combination of homogeneous and heterogeneous cores. These devices are often deployed for a long term and therefore system lifetime reliability is an important consideration while designing them. In principle, placing extra cores increases the lifetime reliability albeit at the cost of increased power consumption and chip area. We propose a framework to explore platform architectures and their floorplans, highlighting the trade-offs between lifetime reliability and power consumption. The framework is based on a Genetic Algorithm and employs a high level simulator to calculate the Mean Time to Failure (MTTF) of the chip. The simulator runs multiple times, also called Monte Carlo simulation, to take the averages of both failure times and power usage. The high number of simulations required makes the framework compute intensive. We therefore propose two variations of the design space exploration to reduce the number of simulations. Our results show that total number of simulations is reduced by ≈30% and the total GA convergence time by ≈55%, while the resulting floorplan designs are similar in their characteristics across all exploration varieties.

Keywords: Design Space Exploration · Multicore Systems · Lifetime Reliability .

1 Introduction

Modern microchip design technologies are able to incorporate multiple cores of different processor types onto a single chip. The amalgamation of resources in one place can significantly improve the performance of these microchips. However, it is imperative that over long periods of deployment time, some of these cores will start to deteriorate owing to the ageing process and will eventually fail. Core failures pose a significant challenge to system reliability over long-term use of a multi-core System-on-Chip (SoC).

The operational temperature and power consumption of a core together are majorly responsible for its ageing rate [5]. Higher temperatures and higher power consumption cause faster deterioration of a core through various fault mechanisms dependent on these factors [7]. In essence, the workload of the core is

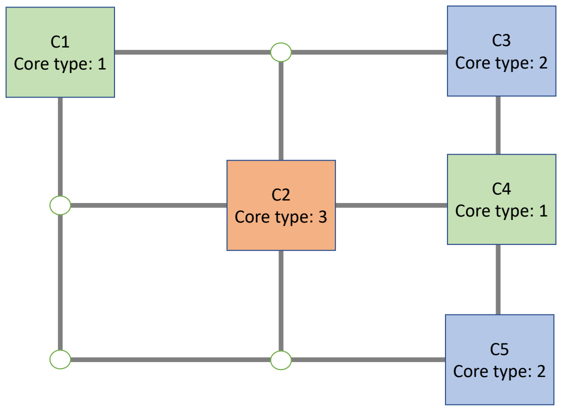

Fig. 1. An example of floorplan created through our framework: 5 cores of 3 unique core types.

the most important factor influencing its ageing process. Heavy computational workloads draw more power to perform the needed operations and at the same time significantly raise the temperature of the core. Moreover, a hot core can cause the neighbouring cores to heat up simultaneously. Therefore, the workload of the whole SoC contributes to the ageing of each core, even though it is expected that each core runs a different workload. This leads to uneven ageing of each resource and thus different failure times for each core. Moreover, the manufacturing variations and random nature of an actual fault occurrence renders it impossible to exactly predict the moment when a core will fail [13].

Embedded multicore platforms are often designed for a specific application and execute periodical tasks. Such systems are regularly deployed for a very long duration and for this reason, system reliability is a crucial factor to consider at the design time. Since the workload of such devices are typically known, it is possible to design the hardware with extra cores. When a core fails, the workload can be redistributed over the remaining cores. As long as task deadlines are met with fewer cores, the SoC can remain in operation. However, placing an extra core comes with added cost of area and power consumption. This then raises the question of how many extra cores would really be sufficient. Every extra core in the system may also consume some power, also known as the leakage power, even when it is not computing anything. Embedded devices are intended to be cheap and power-efficient, so placing every extra core on the chip has to be justified for its extra cost.

In this paper, we propose a framework to explore platform architectures and their floorplan, while attempting to provide a balance between lifetime reliability and power consumption. A floorplan design considers a specific workload and known types of available cores. The framework is based on design space exploration of possible floorplans on a fixed grid size. Figure 1 illustrates one of the design points in our framework's search space. It has five cores of three different types. To evaluate a design point, we use a simulator to predict its Mean Time To Failure (MTTF) and average power consumption. A higher MTTF of a chip represents the fact that the chip will fail after longer duration and hence is predicted to operate for longer duration after initial deployment. In this context, MTTF can be interpreted as an estimate of active lifetime of the chip and thus the reliability of an SoC.

The main contribution of this paper is a novel framework, based on a Genetic Algorithm (GA), to search for appropriate floorplan designs for a fixed-size microchip and a specific workload. GAs are a good choice for search and optimization processes with a vast number of possible design candidates that cannot be explored by exhaustive search. The GA in our framework is constructed as a multi-objective search algorithm for two objectives, namely: MTTF and average power consumption. We want to simultaneously maximize the MTTF and minimize the power consumption. Generally, these two objectives are contradictory to each other. MTTF can be maximized by having many extra cores on the floorplan, which in turn maximizes the power consumption. Keeping this in mind, the GA produces a *Pareto Set* of design points, where one objective cannot be improved without worsening the other objective. It allows the designer to be aware of the floorplan choices available, in terms of which design provides a good trade-off between the lifetime reliability and the power consumption.

Evaluation of a design point within the framework is performed via a simulator, which predicts the MTTF for a floorplan through Monte Carlo simulations using a stochastic fault model. The simulator works at a very high abstraction level to provide quick evaluation of a floorplan design. It estimates the active lifespan of the chip for a specific workload and provides average power consumption. The simulator builds and applies an ageing model to estimate when the cores fail and in what order. With every core failure, the workload is redistributed among the surviving cores, until the workload becomes unschedulable. Since core failure is stochastic in nature, the simulator performs many simulations to predict failures and averages individual Time-To-Failures (TTFs) to estimate the MTTF, and mean power usage. Owing to these large number of simulations required, the GA takes a considerable amount of simulations and time to evaluate all the design points and converge. Later in the paper, we discuss techniques to reduce the total number of simulations performed by the framework. We demonstrate that even with a reduced number of simulations, the framework is able to produce floorplans with similar characteristics in the final *Pareto Set*.

The rest of the paper is organized as follows. We discuss related published works in Section 2. We then present the framework and detail our methodology in Section 3 and describe experimental setup and results of our evaluations and validations in Section 4. Finally, we conclude the paper in Section 5.

2 Related Work

Many simulators have been published in recent times which deep dive into lifetime reliability for multicore SoC. Task allocation and task scheduling are common approaches [18, 8, 9, 7] to improve lifetime reliability of multicore platforms. In these algorithms, tasks (of the workload) are mapped to available resources, and are adjusted during the deployment time to slow down the ageing of the SoC. Usually, with these methodologies the workload is redistributed to avoid any one core from heavy workload and thus failing much before the other cores. Utilization control is another technique proposed by the authors of [12] to maximize lifetime-reliability. In this approach a predictive controller adjusts core frequencies to control the temperatures and thus the ageing process. These frameworks mostly explore the software part of the embedded systems design process and consider specific hardware to be available for which lifetime reliability can be improved. In contrast, our framework specifically explores the hardware design (floorplan) to find the trade-off between lifetime reliability and power consumption.

Moreover, the above frameworks consider only the occurrence of a single core failure. The authors of [4] consider the occurrence of multiple subsequent core failures, thus offering a more precise estimation of the lifetime reliability. One of their strategies is based on spare resources, however, the spare resources are set by the designer without any perception of how many spare cores are sufficient for target lifetime reliability. In comparison, our approach explores the design space to gain insight into the number and the type of requisite cores and their placement on floorplan. To the best of our knowledge, our framework is the first such work to explore multicore platform design while optimizing for both lifetime reliability and power usage.

3 Methodology

In this section, we briefly outline the simulator and then describe the framework based on a Genetic Algorithm. The framework is initialized with information about the workload. All the tasks of the workload and their dependencies are represented in a Directed Acyclic Graph (DAG) format along with the deadline to complete one execution of the workload, see Figure 3 for an example. Additionally, as will be discussed in the next subsection, hardware specific behaviour of these tasks (per core type) is gathered, such as power traces, worst-case execution times and thermal behaviour. The GA based framework converges to produce a *Pareto Set* of unique floorplans, which indicate the trade-offs between lifetime reliability (via MTTF) and the average power consumption for these design points.

3.1 Simulator

The simulator used by the framework is built using multiple tools and models already published and publicly available. The first tool in operation is called

HotSniper [15], which is a low-level cycle accurate thermal simulator. It can simulate the thermal behavior for a given a workload and hardware. Our simulator uses HotSniper to obtain the power traces for each task, which is essentially a periodic reading of the power consumption (of the task) on a particular core type. Further, the input power traces are fed into MatEx [14]. MatEx is a thermal model which uses analytical methods to predict the thermal behavior of a microprocessor. MatEx takes the hardware description and power traces of a workload as inputs to produce a temperature trace. Consequently, the temperature traces obtained from MatEx are used to predict the ageing behavior. The fault mechanism used to predict the ageing behavior in this paper is electromigration, which is the wear-out caused in interconnects due to high temperature. Subsequently, the Black's equation [3] along with a Weibull [11] distribution is used to obtain the fault distribution. This fault distribution is finally used to predict a core failure.

With each core failure, the tasks are scheduled again on remaining cores, until another core fails. This process is repeated until the tasks cannot be scheduled anymore on the remaining cores. It is noteworthy that even with all of the behaviour and fault models, the prediction of an actual fault time is stochastic in nature. Hence, the simulator uses random Monte Carlo simulations, with each simulation resulting in different moments at which the cores fail. One such simulation predicts a Time To Failure (TTF) of the microchip and the power. Through the Monte Carlo simulations, the simulator produces a mean of TTFs in multiple simulations, i.e., an MTTF and average power consumption. The multiple runs by the simulator ensures a closer approximation of the actual mean values of these two objectives is obtained.

Algorithm 1 depicts the Monte Carlo simulation to evaluate a batch of individuals (i.e., candidate floorplan designs) in the GA. We refer to this standard simulator as *stdSim*. Every design point has a simulation budget, i.e. the number of simulations performed by the simulator for individual floorplan evaluation. In the standard simulator, every design point has exactly the same simulation budget. While this approach is useful to evaluate individual floorplans, it becomes compute intensive for the GA. This is because of a large population and multiple iterations of the algorithm, leading to an extremely large number of simulations to be performed.

3.2 Genetic Algorithm

Genetic Algorithms are iterative population based algorithms where a better population evolves over subsequent iterations [10]. The algorithm always has a constant number of design points in its population, though the individual floorplans keep changing through the iterations. The population size is required to be large enough so that enough diversity is maintained among the candidates in the population. If a floorplan is dropped from the population (when it is not performing as well as others), then it is replaced by another floorplan. During each iteration, some individuals from the population of floorplans are altered using genetic mutation operators.

Algorithm 1 Monte Carlo simulator (*stdSim*)

```
Require: A list of $k$ simulation points $\mathcal{S}$
Require: The simulation budget $n$
1: function MCS($\mathcal{S}, n$)
2:     Initialize $\bar{X}$          ▷ the objectives vector, of size $k$, for each simulation point
3:     for each simulation point $s_i \in S$ do
4:         simulate $s_i$ by running the simulator for $n$ times
5:         update $x_i \in \bar{X}$    ▷ update MTTF and mean power for $i^{th}$ simulation point
6:     end for
7:     return $\bar{X}$
8: end function
```

Search Space: The search space refers to all possible floorplans with their configurations and constraints that the framework can evaluate. Random floorplans are sampled from this search space to initialize the population. In this framework, the search space is formulated on a fixed grid size and available types of cores. The example shown in Figure 1 has a grid size of $3x3$ and three unique types of cores. Additionally, the physical size of the grid is required by the simulator to correctly generate thermal behaviour based on distance from adjacent cores. Moreover, the simulator requires to run every task (from the task graph) on each of the available core type individually, to collect power traces and subsequently thermal behaviors. In addition to power traces, the framework also needs to know the worst case execution times of the tasks on each core. This is necessary to check the schedulability of the workload on active cores on the microchip at any point in the simulation.

Mutations: The mutation operators are used by the GA after every iteration to explore the search space. In a single mutation, only one floorplan from the population undergo alterations. The aim of the mutation is to both explore and exploit the search space. Small impact mutations, such as changing the type of one of the cores on the floorplan explores a similar floorplan in the next iteration. Big impact mutations where a core is added, removed or moved to a new position attempt to exploit the search space by creating very different floorplans for the next generation. Together, these mutation operators are responsible for traversing the large design space of floorplans in an efficient manner. Figure 2 illustrates three of the mutation operations from our framework.

All these operator maintain the constraints of the search space. For instance, a core is never moved outside the available grid area and they ensure that the constraint on the minimum number of cores is always respected.

Selection and Replacement: One of the most important features of the GA is that every subsequent population attempts to be better than the previous iteration. This is achieved through selection and replacement policies designed to retain the good floorplans at every step. Since the GA in our framework

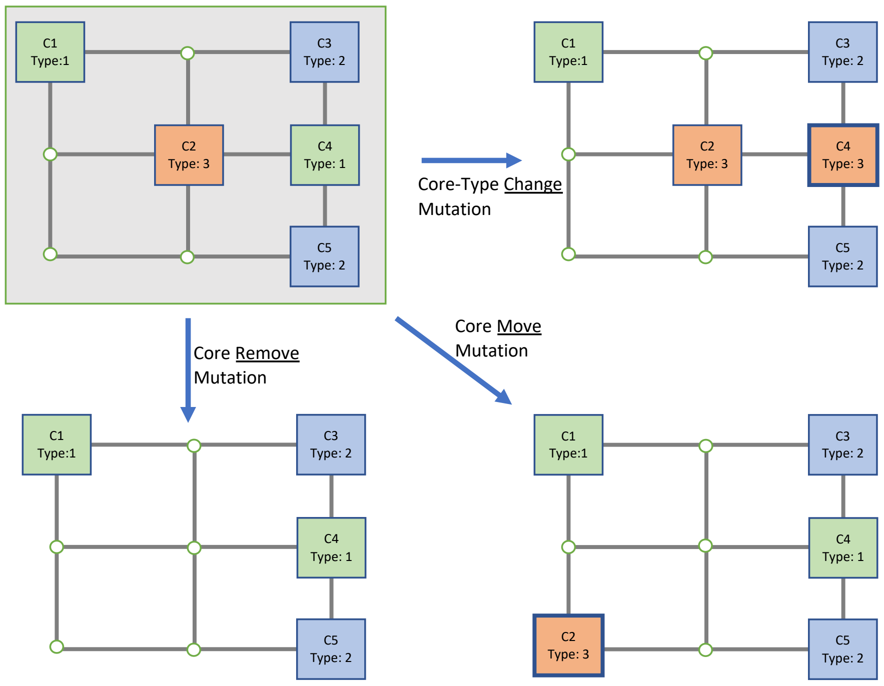

Fig. 2. Exploring the floorplans through mutations.

is a multi-objective search, the best floorplans are selected via the NSGA-2 algorithm [6]. NSGA-2 ensures that equal importance is given to both MTTF and power consumption objectives. These best floorplans are saved in the next iteration, to ensure that the best candidates found so far do not get lost during successive iterations. In addition, a tournament selector strategy is employed to select the candidates to be replaced via the mutation operations. Wherein a group of design points and the best among these is elected as the best to win the tournament. The best candidate is always selected and the worst candidate is always removed from the population in this strategy.

Algorithm: We combine all the concepts discussed in this section to outline the algorithm for our framework. Algorithm 2 illustrates the complete algorithm, the goal is to find floorplans, for a given workload, with lifetime reliability and power as main objectives. The GA starts with an initial population, which is an arbitrary group of design points from the search space. Out of the current population, a new generation will be formed through a mutation operator. To allow convergence towards a more optimal solution, the best floorplans of the parental population and the offspring are selected and are used to determine the next generation. This process of selection, mutation and replacement is repeated a finite number of times.

Algorithm 2 Genetic Algorithm

```
Require: Task graph $DAG$
Require: Task information per core type
 1: function GA($DAG$)
 2:     Initialize population with design points $\mathcal{D}$
 3:     for each iteration of GA do
 4:         Simulate($\mathcal{D}, n$)                  ▷ $n$ is maximum simulation budget
 5:         Select best candidates                      ▷ Through NSGA-II algorithm
 6:         Select parents via tournaments    ▷ For candidates in the next generation
 7:         for each parent $p_i$ do
 8:             Mutate $p_i$                            ▷ Traverse the search space
 9:         end for
10:         Replace population $\mathcal{D}$ with best candidates and new mutated offsprings
11:     end for
12:     Create $ParetoSet$
13:     return $ParetoSet$
14: end function
```

Output: After all the iterations are complete, a *Pareto Set* is selected from the population based on evaluated objectives. All the designs in the Pareto set are considered to be equally adequate to be marked as the best model. In this scenario, the final selection lies in the hands of the system designer, and may also be based on higher priority placed on one of the objectives.

3.3 Speeding up the GA

As mentioned earlier, the simulator works by running multiple times and taking averages of failure times and power usage to estimate MTTF and average power consumption of the chip. Since this leads to a very high number of simulations, we propose two methodologies to reduce the simulation budget of some design points.

The first variation of the simulator, called *10FSim*, runs for only 10% of the total available budget. The simulator keeps track of the number of core failures leading to the microchip failure. By looking at the average number of core failures in 10% of the simulations, we can adjust the remaining simulation budget to be used. When the average number of core failures is $< F_{ThresholdLow}$ (lower threshold for number of failures), there is little scope of getting design points with a very long MTTF and the initial number of cores are almost sufficient for the device (i.e. extra cores are not available for reliability). Therefore, we reduce the total simulation budget to new simulation budget, which is SB_{Low}% of the original budget. On the other hand, when the average number of core failures is $> F_{ThresholdHigh}$ (higher threshold for number of failures), there are already too many extra cores available and MTTF is going to be higher than other design points. So, in this scenario, the simulation budget is reduced to $SB_{High}\%$ of the original budget. All other design points where number of core failures is between the lower and the higher threshold values are simulated with full budget. These will be the points which will illustrate the trade-offs between power consumption and MTTF appropriately. Algorithm 3 outlines the algorithm for *10FSim* simulator.

Algorithm 3 10F simulator (*10FSim*)

```
Require: A list of $k$ simulation points $\mathcal{D}$
Require: The simulation budget $n$
 1: function 10FS($\mathcal{D}, n$)
 2:     Initialize $\bar{X}$          ▷ the objectives vector, of size $k$, for each simulation point
 3:     for $n/10$ simulation points $d_i \in D$ do          ▷ 10% of the simulation budget
 4:         simulate $d_i$ by running the simulator for $n$ times
 5:         update $x_i \in \bar{X}$   ▷ update MTTF and mean power for $i^{th}$ simulation point
 6:         Update $n_i$ number of core failures
 7:     end for
 8:     Update average number of core failures $N_f$
 9:     if $N_f < F_{ThresholdLow}$ then
10:         Update $n = n * SB_{Low}\% - n/10$
11:     end if
12:     if $N_f > F_{ThresholdHigh}$ then
13:         Update $n = n * SB_{High}\% - n/10$
14:     end if
15:     Simulate for $n$ times
16:                                                         ▷ Repeat lines 4 to 6
17:     return $\bar{X}$
18: end function
```

The second variation of the simulator, *UtilSim*, works in a similar manner to the *10FSim* variation. Instead of looking at number of core failures at simulation time, it statically analyzes the (expected) core utilization after scheduling the workload on available cores, i.e. before the simulation starts. When the core utilization is $> Util_{ThresholdLow}$, then most of the cores are busy and even one core failure has a huge impact on schedulability of the microchip. Similar to the first case of *10FSim* simulator, the total number of simulations is reduced to $SB_{Low}\%$ of the original budget. When the utilization is $< Util_{ThresholdHigh}$, the total number of simulations is reduced to $SB_{High}\%$ of the original simulation budget. The rest of the design points are simulated with the whole budget. We outine the algorithm of this *UtilSim* simulator variation in Algorithm 4.

Algorithm 4 Resource Utilization based simulator (*UtilSim*)

```
Require: A list of $k$ simulation points $\mathcal{D}$
Require: The simulation budget $n$
 1: function UTILS($\mathcal{D}, n$)
 2:     Initialize $\bar{X}$        ▷ the objectives vector, of size $k$, for each simulation point
 3:     Check core utilization $U$        ▷ After scheduling the tasks
 4:     if $U > Util_{ThresholdLow}$ then
 5:         Update $n = n * SB_{Low}\%$
 6:     end if
 7:     if $U < Util_{ThresholdHigh}$ then
 8:         Update $n = n * SB_{High}\%$
 9:     end if
10:     for $n$ simulation points $d_i \in D$ do
11:         simulate $d_i$ by running the simulator for $n$ times
12:         update $x_i \in \bar{X}$    ▷ update MTTF and mean power for $i^{th}$ simulation point
13:     end for
14:     return $\bar{X}$
15: end function
```

4 Experiments

In this section, we evaluate all the algorithms in the framework and outline our experimental setup. We have used the Java based Jenetics library [1] for the GA and the simulator is written in python. Our experiments run on Apple M1 pro platform.

4.1 Setup

We perform our experiments with a synthetic application (as the workload itself is of less importance for our proof of concept), represented in DAG format, and is illustrated in Figure 3. All the tasks (T1-T6) were executed in the HotSniper [15] simulator to obtain the power traces for 3 types of cores. Additionally, we use the Cecile Coordination Compiler [16] to estimate worst case execution time on each of the core type. Scheduling of the tasks on cores is done via HEFT algorithm [17] for a given deadline (but different algorithms can be used in the simulator).

The search space for the floorplans is restricted to a grid of size $3x3$, which means that the search space constitutes of 4^9 design points and as such intractable for an exhaustive search (given the fact that each design point requires a number of simulations). The GA explores around three thousand design points to produce a near-optimal *Pareto Set*.

The parameters of the algorithms are summarized in Table 1. Pareto size refers to the range of design points that are desired in the final *Pareto Set*. Mutation probability of each mutation type refers to the probability with which a selected parent will undergo the respective mutation to create offsprings for the next generation.

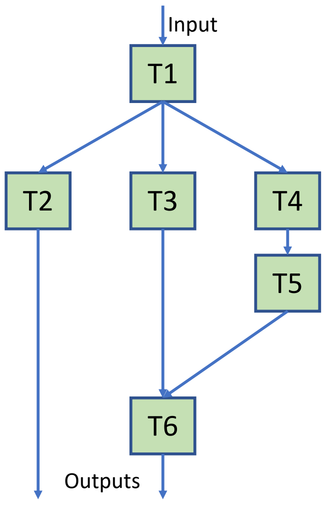

Fig. 3. The task graph (DAG) of the application used in the experiments.

The parameters for Simulators, *stdSim*,*10FSim* and *UtilSim* were emperically determined after some initial experiments and are summarized in Table 2. Simulation budgets were reduced to 50% in floorplans with fewer cores failures handling capability (i.e. high core utilization). In addition, simulation budgets were drastically reduced to 33.33% when a floorplan had large slack to handle core failures (and low core utilization).

Table 1. Framework Setup details

Parameter	**Value**
Number of GA iterations	100
Population size	50
Pareto size	Range(10,15)
Floorplan grid size	3x3
Number of Core types	3
Scheduling policy	HEFT
Mutation probability	
Core-Add mutation	0.2
Core-Remove mutation	0.2
Core-Move mutation	0.3
Core-ChangeType mutation	0.3

Table 2. Simulator Setup details

Parameter	**Value**
Initial simulation budget per simulation	100
10FSim parameters	
$F_{ThresholdLow}$	2
$F_{ThresholdHigh}$	5
UtilSim parameters	
$Util_{ThresholdLow}$	0.5
$Util_{ThresholdHigh}$	0.25
Simulation Budgets Modification factors	
$SB_{High}\%$	33.33%
$SB_{Low}\%$	50%

4.2 Results

Firstly, we performed the experiment with the standard simulator with each design point having a simulation budget of 100 runs. In this scenario, every design point was given equal importance and their MTTF and average power consumption was calculated using the full budget. Figure 4 shows the result of a GA run. All the design points that are explored by the GA are represented in this graph via their average power consumption and MTTF as estimated through the simulator.

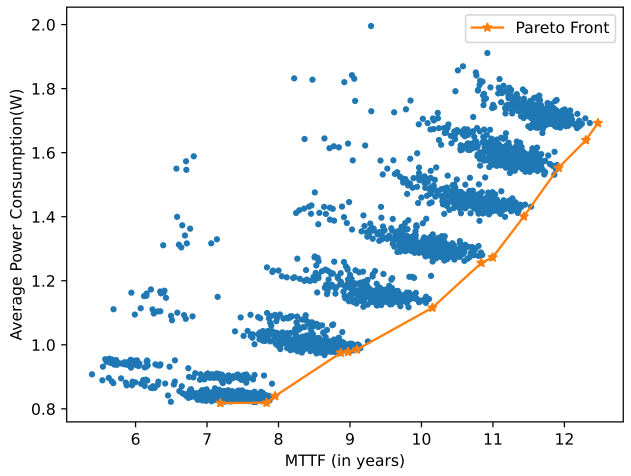

Fig. 4. MTTF and average power consumption for all the design points explored through the GA. The orange line highlights the point in the *Pareto Set*.

Figure 4 illustrates the *Pareto Set* (in Orange) found by the GA at the convergence. The cluster of points corresponds to a specific number of cores on the floorplan. The points on the *Pareto Set* can provide the system designer with informed choices on design points with trade-off between lifetime reliability and power consumption. Figure 5 illustrates the floorplans of some of the points on the pareto front. The top two floorplans refer to the two extreme points on the Pareto front. The floorplan with the lowest power consumption has only 3 cores with approximately 7 years of estimated MTTF. On the other hand, the floorplan with the highest MTTF (>12 years) uses 9 cores, but doubles the power consumption. Please note that the spare cores in a system may also consume some power (e.g., leakage power) and heat up from adjacent cores, therefore idle cores also slowly age along with rest of the cores on the floorplan [2]. The other two floorplans were picked randomly from middle of the *Pareto Set*, highlighting the variety of design choices available to the system designer.

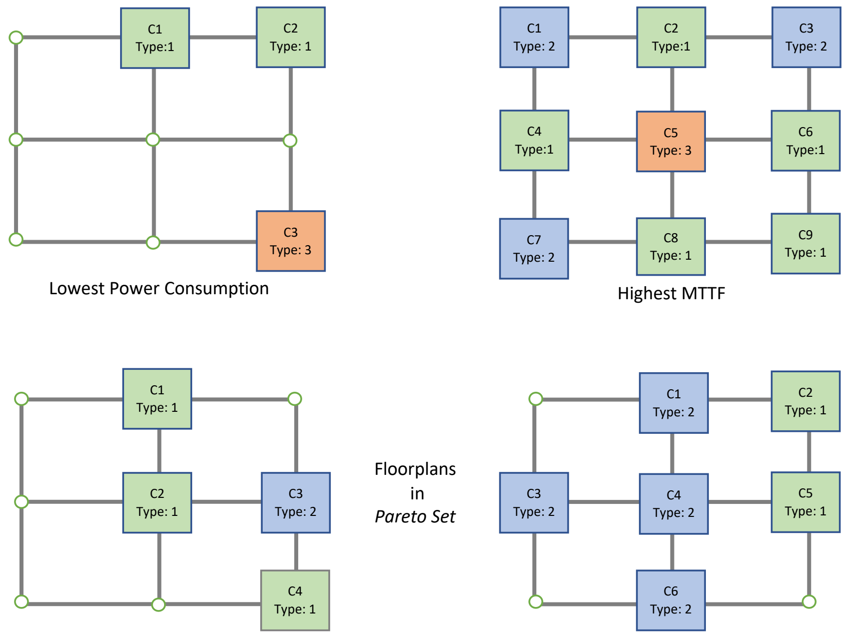

Fig. 5. Some of the floorplans in the *Pareto Set*. Top two floorplans are expected to have lowest power consumption and highest MTTF respectively. Bottom two floorplans are randomly picked from the *Pareto Set*.

Further experiments were performed with the simulator varieties *10FSim* and *UtilSim*. The comparison between their *Pareto Sets* is illustrated in Figure 6. As is evident from the graph, the design points on the *Pareto Sets* are very similar in their characteristics. However, there is a noticeable difference in the points lying in the high MTTF region. It is important to note that only the standard simulator used its whole simulation budget for designs in this area, the other two varieties used only 33.3% of the original simulation budget. It is possible that they over-estimate the active lifetime of a floorplan with fewer simulations. However, for faster design space exploration, an error of a few months with an estimated MTTF of more than 12 years might be an acceptable trade-off.

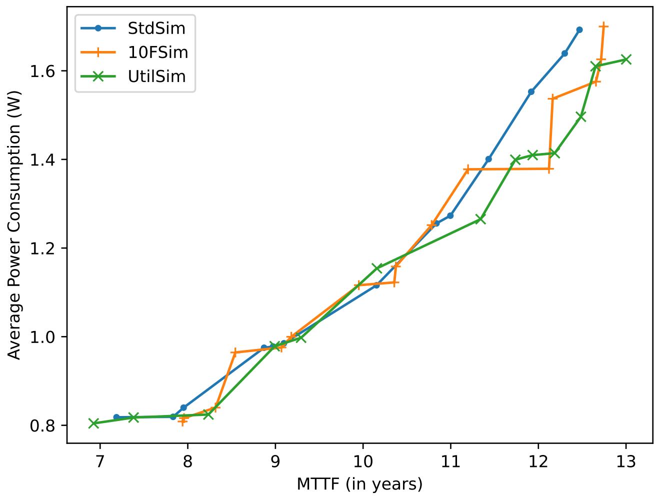

Fig. 6. Pareto fronts achieved with different simulators varieties in our framework.

The total number of simulations done by the standard simulator was $\approx 3.2 * 10^5$. The *10FSim* and *UtilSim* simulators, on the other hand, performed $\approx 2.1 * 10^5$ and $\approx 2 * 10^5$ simulations in total, respectively. As also depicted in Figure 7, by using 10FSim and UtilSim, the total number of simulations are reduced by ≈30%. On an Apple M1 pro, the standard GA took 216 minutes to finish the exploration. With the *10FSim* and *UtilSim* simulators, the GA finished in 97 and 94 minutes, respectively. The total GA convergence time was thus reduced by ≈55% with the modified simulator varieties.

5 Conclusion

Modelling and exploring embedded multicore systems is a time-consuming and a complex task. In this paper, we presented a framework to explore the design space of platform architectures and their floorplans, with the contradictory objectives

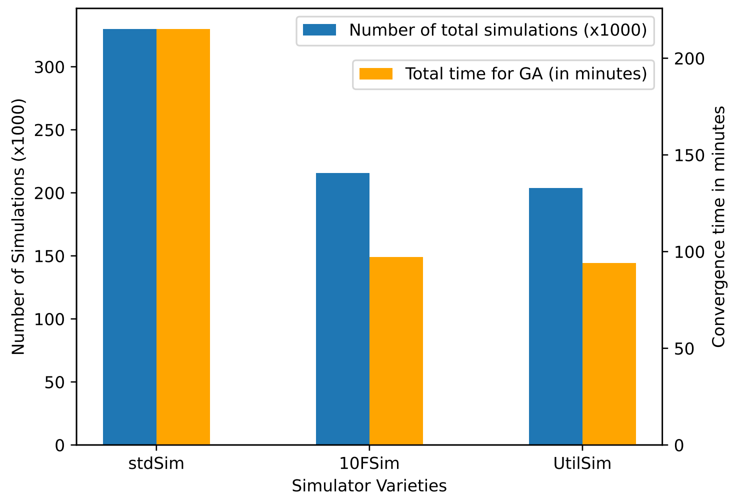

Fig. 7. Comparison between different simulators used by our framework. By using 10FSim and UtilSim, total number of simulations are reduced by ≈30% and total GA convergence time by ≈55%

of increasing lifetime reliability (through estimated MTTF) and average power consumption. The GA based algorithm in our framework returns the *Pareto Set* from the population upon convergence. The design points in the *Pareto Set* exhibit the trade-off between two objectives and one cannot be considered better over the other w.r.t both the objectives.

Furthermore, we proposed variations of the exploration methodology to reduce the total number of simulations needed for a faster convergence of the main algorithm. These variations were able to reduce the total number of simulations performed by ≈30% and the total GA convergence time by ≈55%, with similar floorplans in their respective *Pareto Sets*.

In future, we aim to extend our framework by exploring Dynamic Voltage and Frequency Scaling (DVFS) options for different cores in our framework. By slowing down or increasing the core frequencies, power profile of the core changes, thus changing their MTTF and power usage.

Acknowledgements. This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No. 871259 for project ADMORPH.

References

1. Jenetics library (2023), https://jenetics.io/

2. Abbas, H.M.: An investigation into ageing-resilient processor design. Ph.D. thesis, University of Southampton (2018)
3. Black, J.R.: Electromigration failure modes in aluminum metallization for semiconductor devices. Proceedings of the IEEE **57**(9) (1969)
4. Bolchini, C., Cassano, L., Miele, A.: Lifetime-aware load distribution policies in multi-core systems: An in-depth analysis. In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE) (2016)
5. Coskun, A.K., Rosing, T.S., Leblebici, Y., De Micheli, G.: A simulation methodology for reliability analysis in multi-core socs. In: Proceedings of the 16th ACM Great Lakes symposium on VLSI (2006)
6. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: International conference on parallel problem solving from nature. Springer (2000)
7. Feng, S., Gupta, S., Ansari, A., Mahlke, S.: Maestro: Orchestrating lifetime reliability in chip multiprocessors. In: High Performance Embedded Architectures and Compilers: 5th International Conference, HiPEAC 2010, Pisa, Italy, January 25-27, 2010. Proceedings 5. Springer (2010)
8. Huang, L., Yuan, F., Xu, Q.: Lifetime reliability-aware task allocation and scheduling for mpsoc platforms. In: 2009 Design, Automation & Test in Europe Conference & Exhibition. IEEE (2009)
9. Kathpal, C., Garg, R.: Reliability-aware green scheduling algorithm in cloud computing. In: Ambient Communications and Computer Systems: RACCCS-2018. pp. 421–431. Springer (2019)
10. Katoch, S., Chauhan, S.S., Kumar, V.: A review on genetic algorithm: past, present, and future. Multimedia Tools and Applications **80** (2021)
11. Lai, C.D., Murthy, D., Xie, M.: Weibull distributions and their applications. In: Springer Handbooks. Springer (2006)
12. Ma, Y., Chantem, T., Dick, R.P., Hu, X.S.: Improving system-level lifetime reliability of multicore soft real-time systems. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **25**(6) (2017)
13. Narayanan, V., Xie, Y.: Reliability concerns in embedded system designs. Computer **39**(1) (2006)
14. Pagani, S., Chen, J.J., Shafique, M., Henkel, J.: Matex: Efficient transient and peak temperature computation for compact thermal models. In: 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE (2015)
15. Pathania, A., Henkel, J.: Hotsniper: Sniper-based toolchain for many-core thermal simulations in open systems. IEEE Embedded Systems Letters **11**(2) (2018)
16. Roeder, J., Rouxel, B., Altmeyer, S., Grelck, C.: Towards energy-, time-and security-aware multi-core coordination. In: Coordination Models and Languages: 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15–19, 2020, Proceedings 22. Springer (2020)
17. Topcuoglu, H., Hariri, S., Wu, M.Y.: Task scheduling algorithms for heterogeneous processors. In: Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99). IEEE (1999)
18. Zhou, J., Sun, J., Zhou, X., Wei, T., Chen, M., Hu, S., Hu, X.S.: Resource management for improving soft-error and lifetime reliability of real-time mpsocs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**(12) (2018)