# A NEW METHOD OF DYNAMIC RELIABILITY MANAGEMENT FOR CHIP MULTI-PROCESSORS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Alexandre Yasuo Yamamoto

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

July 2014

Fargo, North Dakota

North Dakota State University
Graduate School

**Title**

A NEW METHOD OF DYNAMIC RELIABILITY MANAGEMENT FOR CHIP MULTI-PROCESSORS

**By**

Alexandre Yasuo Yamamoto

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

SUPERVISORY COMMITTEE:

Ivan T. Lima Jr.
Chair

Cristinel Ababei

David Rogers

Majura Selekwa

Approved:

| | |
|---|---|
| 07/14/14 | Scott C. Smith |
| Date | Department Chair |

# ABSTRACT

This work presents a new dynamic reliability management controller which successfully extends the expected lifetime of Chip Multi-Processors (CMPs). This is achieved by migrating tasks within the CMP, effectively reducing core wear and temperature. While this does decrease performance, results obtained show that the performance penalty is below 10% while lifetime expectancy increases are above 30%. The estimation of lifetime is done by using a full system simulator to obtain execution, power and temperature traces, and then feeding this data to the REliability eSTimation (REST) tool. REST uses a Monte Carlo based algorithm to estimate the Mean Time To Failure (MTTF) of the CMP according to aging mechanisms which affect the transistors.

## ACKNOWLEDGEMENTS

I'd like to begin by thanking Dr. Cristinel Ababei for the wonderful opportunity to work with him in research, both as an undergraduate and as a graduate student. He is quite possibly the most hard-working person I've ever known, and was always able to keep me motivated and engaged in research - he has taught me much (both directly and indirectly), and I will forever be grateful for what he has done and keeps doing for me.

Additionally, I'd like to thank Dr. Ivan T. Lima Jr. for taking me under his wing and helping me with all of my troubles (which were quite a few!), and the opportunity to work with him in many areas. He was always willing to overlook my oversized ego and guide me to the correct path, and being one of the smartest persons I know (which, I admit, makes following his train of thought hard at times), research was always challenging and fun.

To my mom, Ana Virgnia de Moraes, and dad, Lauro Koichi Yamamoto, who more often than not supported me in my crazy ideas; wishing back nothing more than my happiness. I will never be able to properly express how grateful I am for all you do for me, and I assure you anything great I've ever done (or, hopefully, may still do) is because of you.

I also cannot give enough thanks for my Amorzinho, Emily Bartz, for existing and being there for me. Her support and concern for me kept me going when I thought I couldn't and has motivated me to do things I would deem impossible before.

To all of my friends, who shared in my smiles and my tears - I do have a truly marvellous list of names, which this acknowledgements section is too small to contain. Just know that although I may not maintain contact as often as I wish (or should), I still consider each and every one of you as my friends, and will be there for anything you need from me, as you were when I needed you.

For the students I had the pleasure to teach while at NDSU, I had a great time with each and every class. Your enthusiasm to learn and your tolerance of my particularly verbose moments made every single class enjoyable, and I can only hope I was able to pass along the knowledge you deserve.

## DEDICATION

To my mother and father, whose support made all my dreams possible

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF SYMBOLS

| | |
|---|---|
| $V$ | Voltage applied to processor core |
| $a$ | TDDB voltage dependency constant |
| $b'$ | TDDB voltage-temperature correlation constant |
| $X$ | TDDB temperature dependency fitting constant |
| $Y$ | TDDB temperature dependency fitting constant |
| $Z$ | TDDB temperature dependency fitting constant |
| $k$ | Boltzmann constant |
| $T$ | Temperature in Kelvin |
| $A$ | NBTI temperature dependency fitting constant |
| $B$ | NBTI temperature dependency fitting constant |
| $C$ | NBTI temperature dependency fitting constant |
| $D$ | NBTI temperature dependency fitting constant |
| $\beta'$ | NBTI temperature dependency shape constant |
| $J$ | Current density |
| $J_{crit}$ | Critical current density for which Electro Migration starts |
| $n$ | Material constant |
| $E_{a_{EM}}$ | Activation energy for Electro Migration |
| $\Gamma(.)$ | Gamma function |
| $\alpha$ | Weibull scale parameter |
| $\beta$ | Weibull shape parameter |
| $tf$ | Time to failure |
| $x$ | Randomly generated failure time sample |
| $\overline{x}$ | Mean time to failure |
| $w$ | Uniformly distributed random variable |
| $N$ | Number of Monte Carlo iterations |

|  |  |
|---|---|
| $F$ | Number of failure mechanisms |
| $S$ | Number of sub-blocks |
| $u$ | Artificial neural network output |
| $f(.)$ | Activation function |
| $b$ | Artificial neuron bias |
| $X_n$ | Artificial neuron input n |
| $w_n$ | Artificial neuron synaptic weight n |
| $\varepsilon$ | Error |
| $U$ | Artificial neural network desired output |
| $\alpha_L$ | Learning rate |

# 1. INTRODUCTION

The continuous miniaturization of devices has enabled many advancements in processing power and portability, but at the same time has introduced many difficulties which did not exist before - reducing the transistor size increases transistor count and thus power density, leading eventually to dark silicon and the necessity for advanced methods of managing power, temperature and reliability of CMOS devices.

It is reasonable to state that managing reliability is of great importance for practical purposes. A good reliability management scheme will also manage temperature, power and other factors as a direct consequence, and is of great value for the average end user; a device with a short useful life and low reliability will be ultimately undesirable, no matter how good it performs. Additionally, aging mechanisms may also cause performance degradation and reduced fault tolerance. Because of this, researchers from both industry and academia recognize that reliability is becoming a primary design concern in current integrated circuits [1, 2].

Among the factors that will contribute to the aging of CMOS devices, we have process variations, different workloads, operation environment and many others. They will also cause varying on-chip temperatures, which will contribute to an overall decreased reliability. Since most of these factors cannot be known at design time, a dynamic method is required. So, to properly address the problem of managing lifetime reliability, two main goals need to be achieved:

1. Estimate lifetime reliability.

2. Development of hardware and software techniques for improving the system lifetime.

Previously, reliability has been addressed by employing fault tolerance for either the communication units (such as buses or the NoC) or the computation unit. While they have their merits, designs that do not consider the system as a whole will lack accuracy and will be sub-optimal at best. Chapter 7 shows this clearly, by evaluating the difference between considering or not the NoC in the reliability design. Since it is a primary communication medium for CMPs with tens, hundreds, and even thousands of cores [3, 4], ignoring it will have major impact in most modern CMPs.

This work proposes a successful new dynamic reliability controller, which accurately estimates the lifetime expected using an Artificial Neural Network (ANN) and reacts to feedback from the Chip Multi-Processor (CMP) in order to achieve noticeable gains in reliability with low performance penalties.

# 2. FIGURES OF RELIABILITY AND REVIEW OF AGING MECHANISMS

Proper control of any aspect of a system will depend on a well designed closed loop control system - as has been done in many dynamic power [5]-[8] and thermal [9, 10] management schemes. This simple control system is shown in Fig. 1.

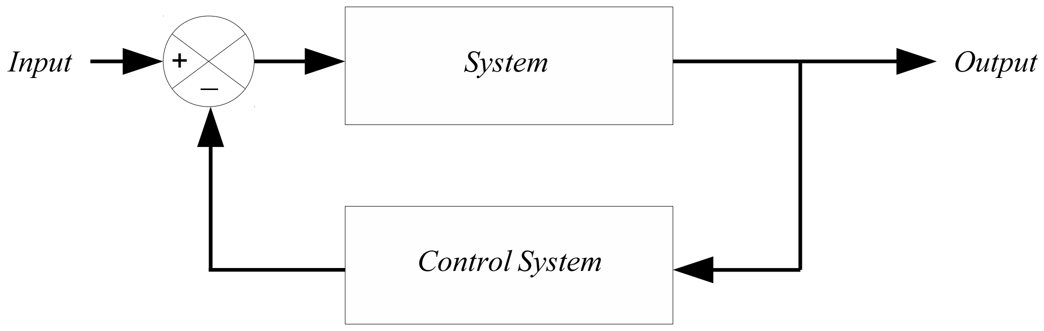


Fig. 1. Generic closed loop control system.

The main issue with controlling the reliability of a system is the question of which metric to use, as there are many options. As examples, availability, data integrity, MTTF, MTTR and others can be mentioned and have been used to quantify reliability.

From the average end user point of view, all that really matters is whether or not a system is working properly; to measure this, one can define the function $R(t)$, which indicates the expected lifetime remaining at any given time $t$. It is a upside-down bathtub curve, as it is inversely proportional to failure rates; Fig. 2 shows an example. We can then obtain the Mean Time To Failure (MTTF) by integrating $R(t)$:

$$MTTF = \int_0^{+\infty} R(t)\, dt. \tag{1}$$

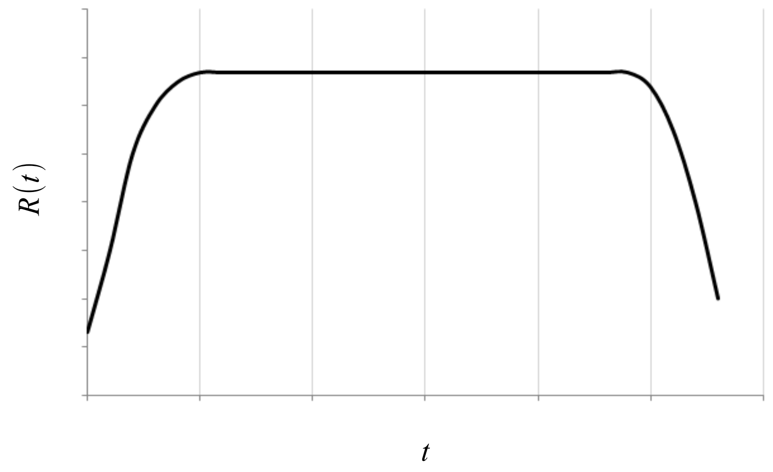


Fig. 2. Expected lifetime remaining R(t).

Thus, if our intent is to maximize the probability that a system will work properly at any given time $t$, our main goal is to maximize the MTTF - which has been one of the most popular measures for reliability.

Significant work has been done to estimate the reliability of single-core processors, multi-core processors [11]-[15] and computer networks [16], while the reliability of NoCs has only recently been studied [17]-[19]. There is also work specific to reliability aware design [20]-[26]; however, most of this work is done with many assumptions to make analytical calculations easier, such as considering uniform device density, identical vulnerability of the devices to failure mechanisms and that the statistical distribution that represents those failure mechanisms can be approximated by an exponential distribution [11]-[14]. Those assumptions may generate significant error in the final figures of lifetime reliability [27], and thus better ways to calculate those figures are needed to provide an accurate estimate for the control system. Additionally, communication and processing units are considered separately, which also cause discrepancies in reliability [28].

To begin to understand how to properly calculate and control the reliability of a system, it is necessary to first understand what are the mechanisms that cause it to age and fail.

## 2.1. Time Dependant Dielectric Breakdown (TDDB)

With the constant drive for lower voltages and higher speeds, the thickness of the gate oxide in CMOS transistors has decreased dramatically over the years. With the reduced dimension comes a reduced threshold to the electric field that causes a breakdown; where the oxide no longer properly insulates the gate terminal. Under normal operating conditions, charges will tunnel through the oxide, eventually becoming trapped in it, as shown in Fig. 3.

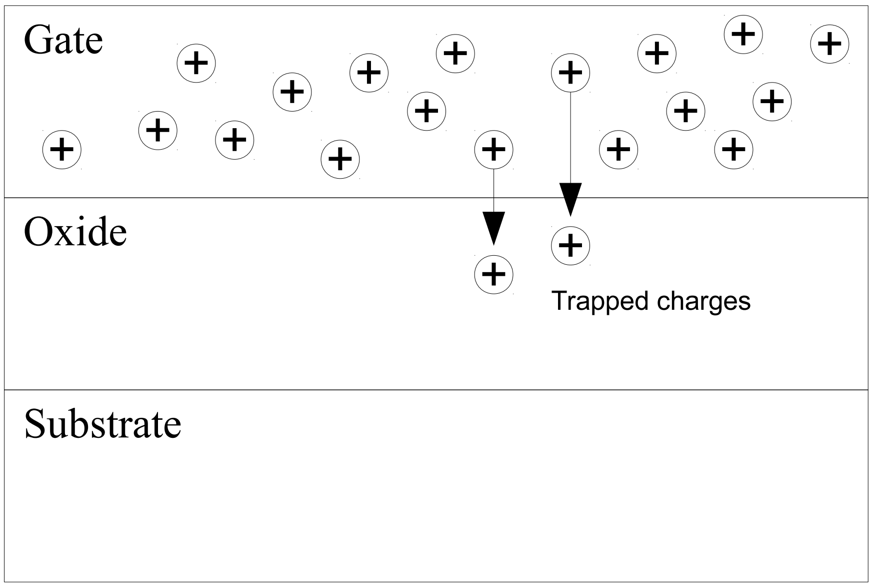


Fig. 3. Charges being trapped in the oxide due to tunneling current.

With time, more and more charges will be trapped in the oxide, which is called the *build-up stage*. When sufficient charges are trapped in the oxide, the electric field will be higher than the threshold of the oxide, causing it to break down and the current to increase, as shown in Fig. 4.

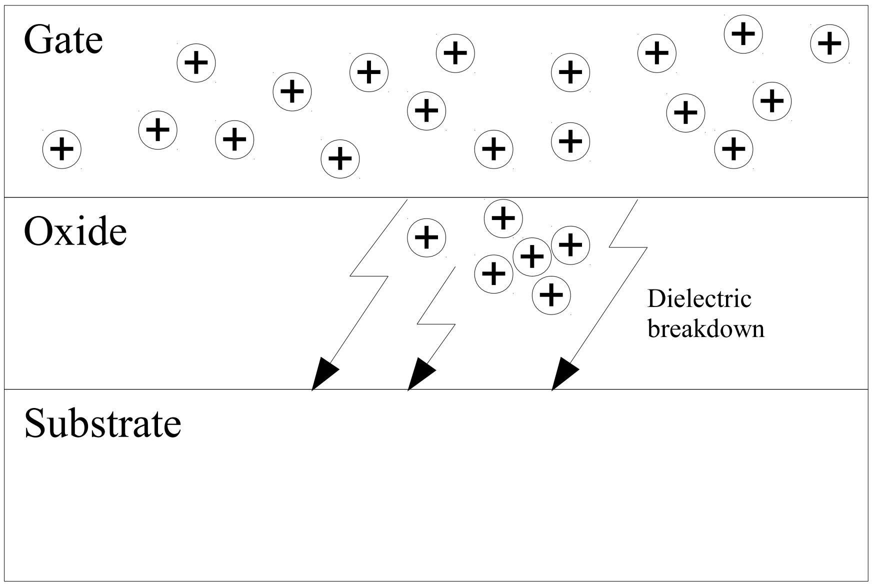


Fig. 4. Trapped charges causing dielectric breakdown.

This breakdown will cause the oxide to conduct large currents, which will heat it up and cause it to conduct even more current. This positive feedback loop will eventually destroy the dielectric, in the stage known as the *runaway stage*. Because of these characteristics, a device can suffer a hard breakdown or various soft breakdowns before the final hard breakdown occurs [29].

Since temperature will cause the tunnelling current to increase - which will increase the number of trapped charges - it is natural to deduce that the expected MTTF of a device due to TDDB will decrease with temperature increases. This is shown by (2), which models the mechanism. In this equation, $a = 78 \backslash b = -0.081K^{-1} \backslash X = 0.759eV \backslash Y = -66.8eVK \backslash Z = -8.37e^{-4}eV$, $V$ is the supply voltage and $T$ is the temperature in Kelvin as in [11, 30].

$$MTTF_{TDDB} \propto \left(\frac{1}{V}\right)^{a-b'T} e^{\frac{X+\frac{Y}{T}+ZT}{kT}}. \qquad (2)$$

## 2.2. Negative Bias Temperature Instability (NBTI)

When negative voltages are applied to the gate, it will in time experience an increase in threshold voltage and a degradation in transconductance, drain current and mobility [31]. While pFETs are obviously more vulnerable to the effect due a negative voltage being present at the gate during normal operation, nFETs may experience the effect since they can have negative voltages applied to the gate when in the accumulation regime.

These effects will occur due to the creation of interface traps by negative bias and elevated temperatures. The model for the MTTF due to the NBTI mechanism is shown in (3), where $A = 1.6328$, $B = 0.07377$, $C = 0.01$, $D = -0.06852$, $\beta' = 0.3$ and $T$ is the temperature in Kelvin as in [11, 32].

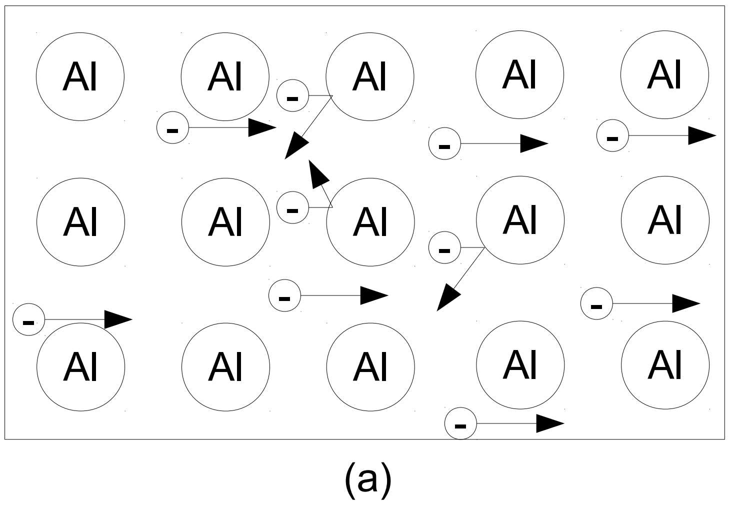
$$MTTF_{NBTI} \propto \left\{ \left[ ln\left(\frac{A}{1+2e^{\frac{B}{kT}}}\right) - ln\left(\frac{A}{1+2e^{\frac{B}{kT}}} - C\right)\right] \frac{T}{e^{\frac{D}{kT}}} \right\}^{\frac{1}{\beta'}}. \tag{3}$$

## 2.3. Electromigration (EM)

When electrons collide with the metal ions in the interconnects, there is a transference of momentum, as shown in Fig. 5(a).

With sufficient momentum, the atoms are going to be permanently displaced, causing the interconnects to increase resistance, short with other interconnects or open completely, as shown in Fig. 5(b). It is easy to see that this effect will be accelerated by current density - which will increase the number of collisions - and by temperature. The model for the MTTF shown in (4), also makes that clear.

$$MTTF_{EM} \propto (J - J_{crit})^{-n} e^{\frac{Ea_{EM}}{kT}}. \tag{4}$$



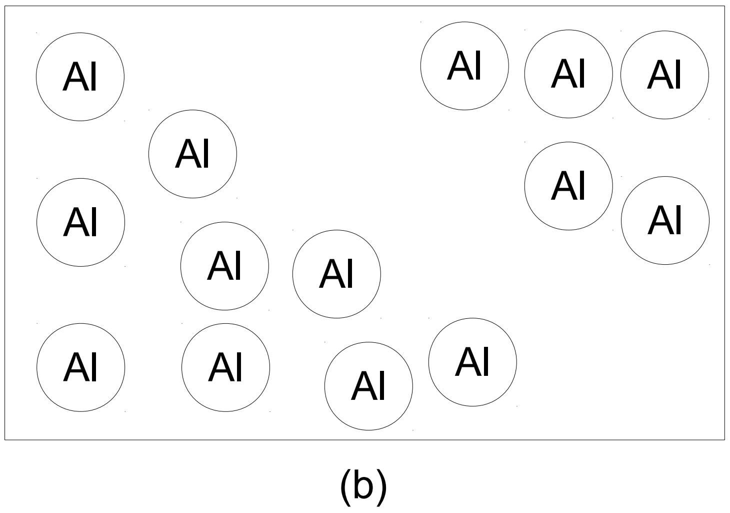


Fig. 5. Electromigration effects.

In this equation, $J$ is the current density, $J_{crit}$ is the critical current density for electromigration, $n$ depends on the material of the interconnect and $E_{a_{EM}}$ is the activation energy for electromigration of said material.

## 2.4. Other Failure Models

Many other failure models that need to be considered may be discovered in the future. While this thesis is making use only of the models for NBTI and TDDB since they have the most effect with regards to temperature and voltage, it is very straightforward to modify the algorithms slightly to consider more failure models, whatever their statistical distribution.

## 2.5. Failure Models and Device Area

One important fact to keep in mind is that all aging models shown above are specific to each and every transistor present in the device. While calculating and joining the MTTF for each transistor will result in the reliability of the device, that would be highly impractical, especially given the number present in modern devices. Thus, it is easier to adjust the equations such that all transistors with the same temperature are accounted for, resulting then in the MTTF for the module they represent. The model equations will then be updated according to (5).

$$MTTF_{AreaFailureModel} \approx \frac{MTTF_{FailureModel}}{Area}. \tag{5}$$

## 2.6. Combining Failure Models

To properly combine the failure models to obtain the MTTF for the whole system, the first problem to be faced is how to combine every aging mechanism based only on their mean values. If the failure models had exponential distributions, combining them to provide a single reliability figure would be straightforward using the SOFR model; however, this is not the case, since they are best described by Weibull or lognormal distributions. This means that the assumption of exponential distributions will lead to inaccuracies.

Additionally, if they were all Weibull distributed, one could apply the Minimum Closure [33] and obtain the new mean by using (6).

$$MTTF_{combined} = \Gamma(1 + \frac{1}{\beta}) \left[ \sum_{i=1}^{n} \left( \frac{\Gamma(1 + \frac{1}{\beta})}{MTTF_i} \right)^{\beta} \right]^{-\frac{1}{\beta}} . \tag{6}$$

This, however, relies on the assumption that the $\beta$ parameters (which is the Weibull shape constant) are equal for all, and also that all failure models are Weibull distributed. Unfortunately this is not the case, since some are Lognormal distributed, and new models may be discovered in the future with other distribution patterns. With that in mind, to calculate the actual value, it is necessary to make use of a Monte Carlo type simulation.

# 3. CALCULATION OF FAILURE RATES USING THE MONTE CARLO METHOD

## 3.1. Temperature Based Failure Rates

As seen on the results presented on the previous chapter, the MTTF of a system will be dependent on the temperature and voltage to which the silicon is subject. However, analytically calculating the expected lifetime of the system is not such a simple task.

For one thing, we cannot assume that the system has uniform temperature. Temperatures can and will vary from core to core, but it will also not be uniform inside the core itself. Additionally, we are assuming the statistical modelling for the failure rates is of such complexity that calculating the mean lifetime of the system is not trivial given the mean lifetimes of the components. This is the case where you have different distributions for different failure models (e.g. Weibull and lognormal). Thus, the mean failure for the cores and for the system need to be calculated through a Monte Carlo simulation.

## 3.2. Serial Failure Model

The first thing to consider is how eventual failures will affect the system. In this work, the serial failure model is considered, where a failure in any of the subcomponents will result in the failure of the system; which means that the system is not fault tolerant. As an example, consider the system shown in Fig. 6: A failure in module 4 will generate a failure in the whole system, independent of whether or not all of the other modules are working. Thus, having the failure times of the individual components, to obtain the failure time of the system it is only necessary to find the minimum value, as presented in (7).

$$tf_{system} = min\{tf_1, tf_2, tf_3, ..., tf_i\}. \tag{7}$$

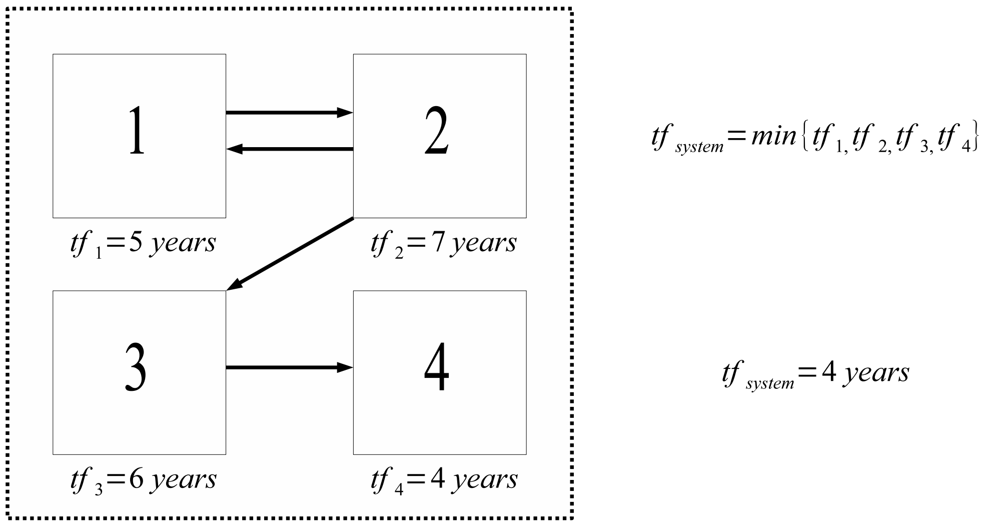


Fig. 6. Serial failure model: System failure occurs at earliest failure.

The method can be adapted, however, to systems where some sort of fault tolerance is present (for example, in systems with redundant cores) by simply considering a failure only when *all* of the redundant modules fail. This will complicate the method somewhat, but calculation of the MTTF is still possible.

## 3.3. Failure Times Generation

In order to use (7) to generate the system failure times and then compute the MTTF, it is necessary to first generate the failure times for each of the modules. In the previous chapter, these failure times will depend on temperature, voltage and each of the aging mechanisms considered, and they are modelled by different distributions such as the Weibull and the lognormal. So to generate each instance of failure, it is necessary to use a random number generator that generates these numbers according to a distribution of choice.

However, generating random numbers according to specific distributions is not trivial. Most random number generators generate uniform distributions in the [0,1] interval; so it is easier to map this uniform distribution to generate the chosen distribution. To do that, we make use of the fact that the CDF of any distribution will fall into the same [0,1] interval. The Weibull CDF is given by (8).

$$F(x) = 1 - e^{-(\frac{x_{sample}}{\alpha})^{\beta}}. \tag{8}$$

Rearranging the terms, we can write the sample from the CDF, as in (9). Thus, by substituting $F(x)$ by the uniformly distributed random variable $w$, we can map the random number generator as shown in Fig. 7 via (10).

$$x_{sample} = \alpha\{-ln[1 - F(x)]\}^{\frac{1}{\beta}}, \tag{9}$$

$$x_{sample} = \alpha\{-ln[1 - w]\}^{\frac{1}{\beta}}. \tag{10}$$

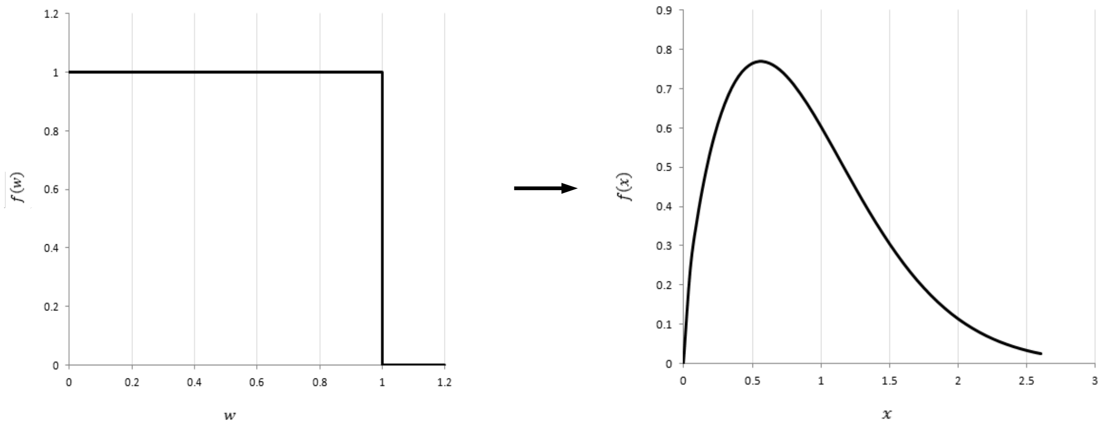


Fig. 7. Mapping from a uniform distribution to a Weibull distribution.

All that is left to do is define the values for the $\alpha$ and $\beta$ parameters. According to [11], we use $\beta = 1.64$ and can then calculate $\alpha$ according to (11), where $\overline{x}$ is the mean. In this case, the MTTF, and $\Gamma(.)$ is the Gamma function.

$$\alpha = \frac{\overline{x}}{\Gamma(1 + \frac{1}{\beta})}. \tag{11}$$

We can then obtain the $generate\_instance_{weibull}()$ function, which is represented in (12). In this equation, $w$ is a uniformly distributed random variable in the interval $[0, 1]$.

$$generate\_instance_{weibull}(MTTF) = \frac{MTTF}{\Gamma(1 + \frac{1}{\beta})}\{-ln[1 - w]\}^{\frac{1}{\beta}}. \tag{12}$$

The same logic can be applied to lognormal distributed failure mechanisms, but in this case the CDF is given by (13):

$$F(x) = \frac{1}{2} + \frac{1}{2}erf\left[\frac{ln(x) - MTTF}{\sqrt{2}\beta}\right]. \tag{13}$$

Rearranging the terms as before, we can have a $generate\_instance_{lognormal}()$ as shown in (14):

$$generate\_instance_{lognormal}(MTTF) = e^{\left[MTTF + \sqrt{2}\beta erf^{-1}(2w-1)\right]}. \tag{14}$$

Therefore, failure times will vary according to the subsystem temperature, voltage and failure mode. For each combination of these factors, it is possible to generate sample failure times that will represent the MTTF calculated analytically using the methods described in the previous chapter. So the Monte Carlo pseudo-algorithm for the calculation of the MTTF for the system is as shown in Fig. 8.

```
for number of samples $j \leftarrow 1$ to $N$ do
    for failure types $l \leftarrow 1$ to $F$ do
        Calculate $MTTF_l$ using Chapter 2 Eqs. ;
        for number of sub-blocks $k \leftarrow 1$ to $S$ do
            $tf_{l_k} \leftarrow generate\_instance(MTTF_l)$ ;
        end
        $tf_l \leftarrow mintf_{l_k}$ ;
    end
    $tf_{system_j} \leftarrow mintf_l$ ;
end
$MTTF_{system} = \frac{\sum_{j=1}^{N} tf_{system_j}}{N}$ ;
```

Fig. 8. Monte Carlo Pseudo-algorithm.

By generating enough samples, the failure times for each of the blocks that compose the system will approximate the desired Weibull distribution. Using this method, we are able to generate separate failure times for all failure models and subsystems, thus being able to calculate the system failure time.

**3.4. Performance, Error and Optimizations**

The Monte Carlo Method is very flexible, and can consider all sorts of statistical models for failures, which can be almost effortlessly included in the system by utilizing the method described. It can also be made as precise as desired, since the precision will only depend on the amount of samples to be simulated, according to (15), where $N_{samples}$ is the total number of samples simulated in the method.

$$\delta MTTF \propto \frac{1}{\sqrt{N_{samples}}}. \tag{15}$$

It is worth noting, however, that it is very slow and computationally intensive. For example, a simulation with $10^8$ samples takes 5 minutes to complete in a i5 2GHz machine.

However, the nature of the Monte Carlo method is one that allows for intense parallelism, which can be explored to further reduce this time. The speedup is very close to linear. Running the code adapted to parallel GPU computing in a CUDA enabled machine with a K20c GPU yields a computing time of less than a second for the same $10^8$ samples.

# 4. RELIABILITY ESTIMATION FRAMEWORK

The MTTF of a system will depend on many factors, such as floorplan, temperature, voltage and others. To obtain all of these parameters, there's the necessity to use different software for each task, as illustrated in Fig. 9. This Reliability Estimation Framework [28] (or REST, for short) is capable of calculating the MTTF of cores due to select implemented failure models.

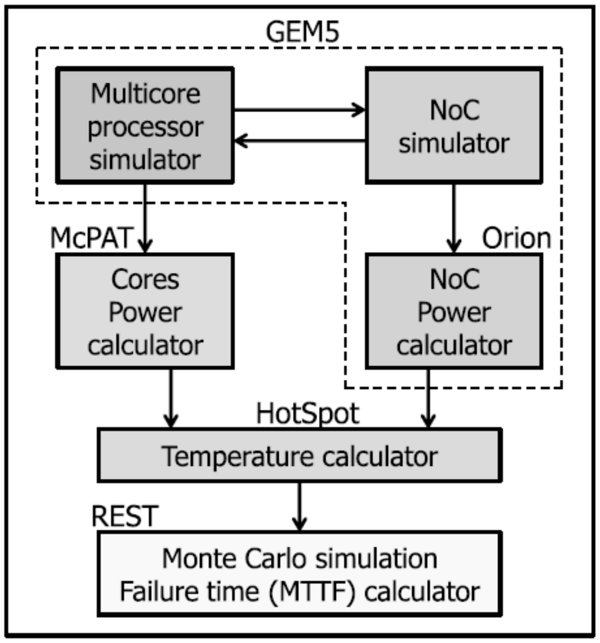


Fig. 9. Block diagram of REST tool framework.

## 4.1. Cycle-accurate Simulator - GEM5

To properly test the proposed DRM technique and to evaluate the lifetime reliability of a system, it is necessary to simulate it completely. The emulator used is GEM5 [44], which is a combination of the M5 full system simulator [45] and GEMS, a modified version of Ruby to support cache coherence protocols and interconnect models via Garnet [46].

GEM5 is a very powerful simulator capable of providing a cycle-accurate full system simulation of many processor designs such as the ALPHA and the ARM [44]. In addition, it supports many different parameters such as the type of intra-chip communication, different cache coherence protocols, the number of processors and many more. It is worth noting that the simulator does have limitations: While technically there is no limit to the number of cores it can emulate, in practice most software will correctly support only a maximum of 4 cores. The specific software being used for this thesis supports up to 64 cores, which is why that is the maximum number of cores used in the results. There is, however, no limit on the cores supported by the proposed algorithm.

One other factor to be considered is that while GEM5 logs all data necessary for the calculation of power and temperature, it does so after finishing the simulation; which means power and temperature will only represent averages throughout the entire runtime. This is not adequate for a system which will require real-time feedback. To circumvent this limitation, the simulation has to be periodically stopped and all desired data dumped and subsequently reset so that the values seen now will be averages only for that segment, as shown in Fig. 10.

The interval between pauses shown in Fig. 10 can be adjusted arbitrarily depending on the level of precision required.

## 4.2. Power Calculations - McPAT

GEM5 will provide the power consumed by each of the NoC routers, but it will not do the same for the power being consumed by the cores themselves. To do that, it is necessary to use McPAT [48], which is a power estimator. It will use all of the information provided by the system simulator and additional information about the fabrication process to provide power data for all of the units in a core for all different cores in the MPSoC, as shown in Fig. 11.

## 4.3. Temperature Calculations - HotSpot

Having all of the powers corresponding to the sub-blocks of each core, it is then possible to calculate the temperature of each of these sub-blocks, as shown in Fig. 12.

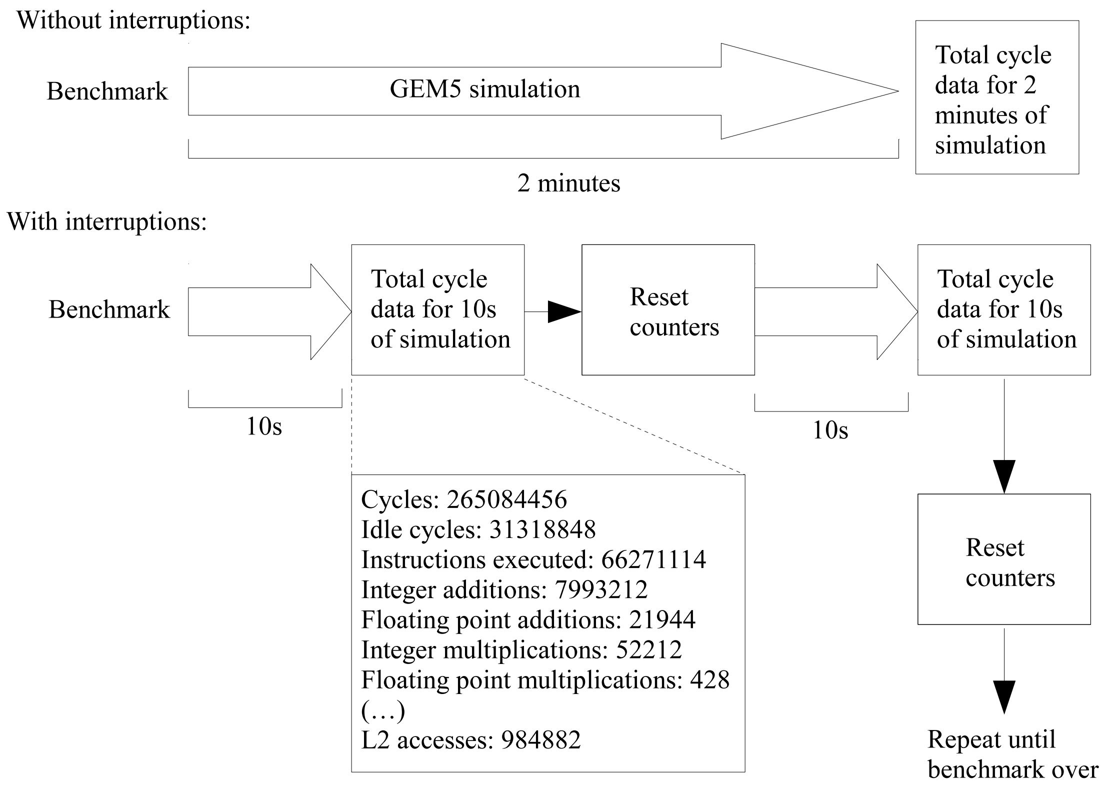
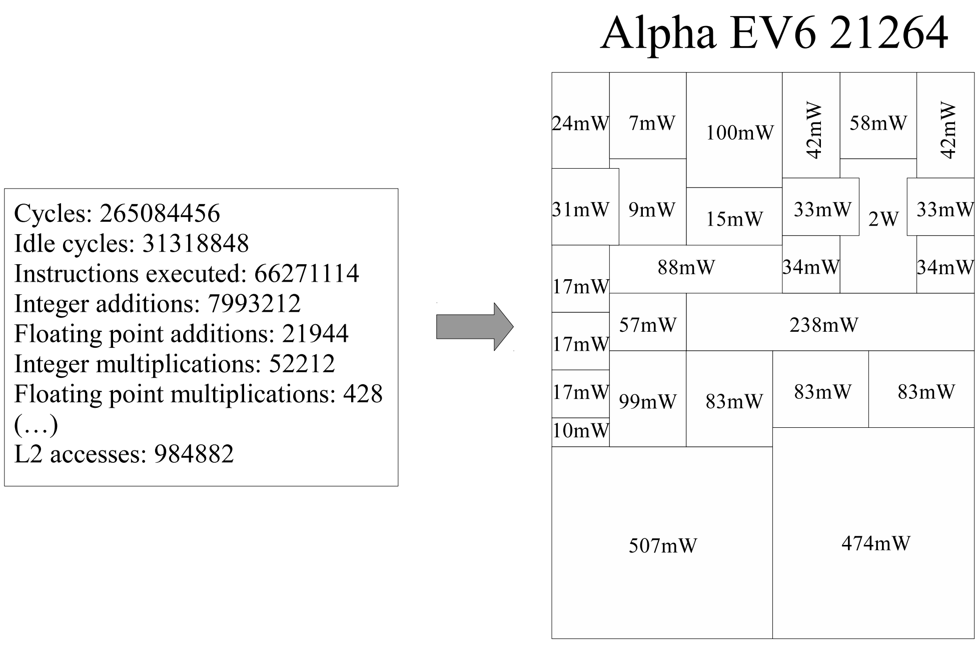

Without interruptions:
Benchmark
GEM5 simulation
Total cycle data for 2 minutes of simulation
2 minutes
With interruptions:
Benchmark
Total cycle data for 10s of simulation
Reset counters
Total cycle data for 10s of simulation
10s
10s
Cycles: 265084456
Idle cycles: 31318848
Instructions executed: 66271114
Integer additions: 7993212
Floating point additions: 21944
Integer multiplications: 52212
Floating point multiplications: 428
(…)
L2 accesses: 984882
Reset counters
Repeat until benchmark over

Fig. 10. Execution of a benchmark in GEM5 with interruptions.

To do this, all of the results obtained with McPat are fed into HotSpot [49], which is a fast and accurate thermal model that uses an equivalent circuit of thermal resistances and capacitances for each of the micro-architecture blocks in order to compute the temperature at each quantized interval of computation.

To correctly compute the temperatures at each core, HotSpot needs the floorplan and the thermal characteristics of the heat extraction device (heatsink). Since the floorplan is divided into each sub-unit of every core, they are treated as being uniform - which means that each will be considered as having a single temperature in the whole area of the sub-unit. This also helps computing MTTF.



```
Cycles: 265084456
Idle cycles: 31318848
Instructions executed: 66271114
Integer additions: 7993212
Floating point additions: 21944
Integer multiplications: 52212
Floating point multiplications: 428
(…)
L2 accesses: 984882
```

Fig. 11. Conversion of timing and cycles data to power by McPAT.

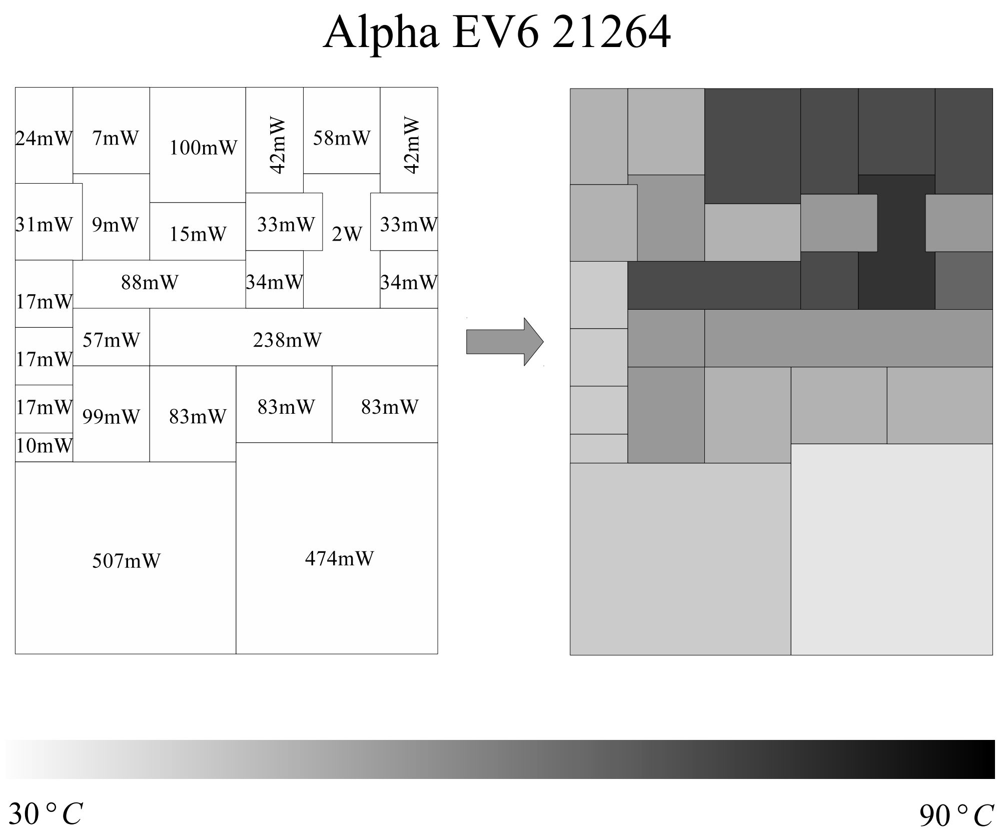


Fig. 12. Conversion of power to temperature by Hotspot.

Additionally, the heatsink cannot be kept constant for CMPs with different core numbers. A CMP with 8 cores will dissipate much more power than one with 2 cores (provided they are the same architecture and fabrication process), so the heatsink must be adjusted accordingly. Since this thesis will compare a few CMPs with different core numbers, the best way to ensure the results will not depend on the heatsink variation is to set the heatsink according to the number of cores. To do so, the heat extraction capabilities were increased according to the number of cores - meaning the heatsink for a system with 8 cores will have better heat extraction than the heatsink for a system with 2 cores.

It is also important to check if the heatsink for a single core (which will be used as a base for all the other systems with more cores) is adequate. Temperatures of the system should never surpass reasonable operating temperatures (usually below $+140\,^{\circ}\mathrm{C}$), but actual maximum temperatures will depend on the specific SoC in use.

## 4.4. Failure Times Generation and Obtaining System MTTF

All of the temperatures and voltages for the systems' sub-blocks are used to calculate its MTTF. Then, using the method shown in Chapter 3, failure times are generated for each, and the minimum failure time is selected as being the failure time for the whole system, as shown in Fig. 13. The process is repeated as desired, obtaining in the end the entire system MTTF.

However, this MTTF obtained should be multiplied by a calibration constant which will be responsible to bring the reliability figure to the actual value in terms of a time unit. This is because there are many variations in the manufacture of the CMP that cannot be accounted for; and need, thus, be determined empirically.

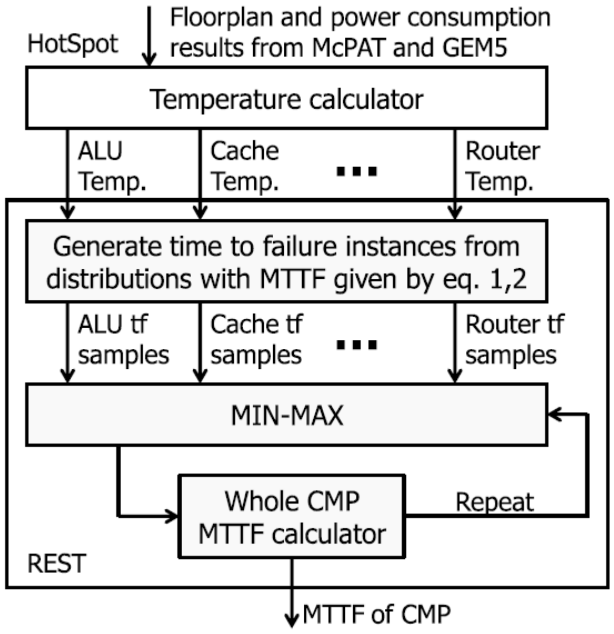

HotSpot
Floorplan and power consumption results from McPAT and GEM5
Temperature calculator
ALU Temp.
Cache Temp.
Router Temp.
Generate time to failure instances from distributions with MTTF given by eq. 1,2
ALU tf samples
Cache tf samples
Router tf samples
MIN-MAX
Whole CMP MTTF calculator
Repeat
REST
MTTF of CMP

Fig. 13. REST system flowchart.

# 5. ESTIMATING CORE RELIABILITY USING NEURAL NETWORKS

In order to be able to effectively maintain the system reliability above a desired threshold, it is important to be able to calculate it quickly and with minimal effort. The Monte Carlo method is very precise, but is computationally intensive and slow. Therefore, a quick method is required to be able to correctly react to the changes in core temperature while minimally affecting performance.

Artificial Neural Networks (ANNs) are perfect for this task as they involve very few operations and can be trained at design time to represent any function or data set. They are also able to approximate and generalize results for inputs which they were not trained for, within reason. One of their most interesting characteristics, however, is that their training can be reinforced or completely redone if the need to do so arises, giving them a very high degree in flexibility.

## 5.1. Review of Artificial Neural Networks

ANNs were inspired by the capability and functionality of the biological neuron, and they try to copy its behaviour as close as possible. The inputs are treated as receiving neural synapses. There is an activation function and an output synapse, which may or may not connect to other neurons. By using methods of reinforced learning, the ANN can be made to represent a function which is not known in advance or that is too complex to be calculated analytically.

The diagram for the neural cell is presented in Fig. 14. The output $u$ can be calculated directly (16).

$$u = f(b + \sum_{i=1}^{n} X_n w_n). \tag{16}$$

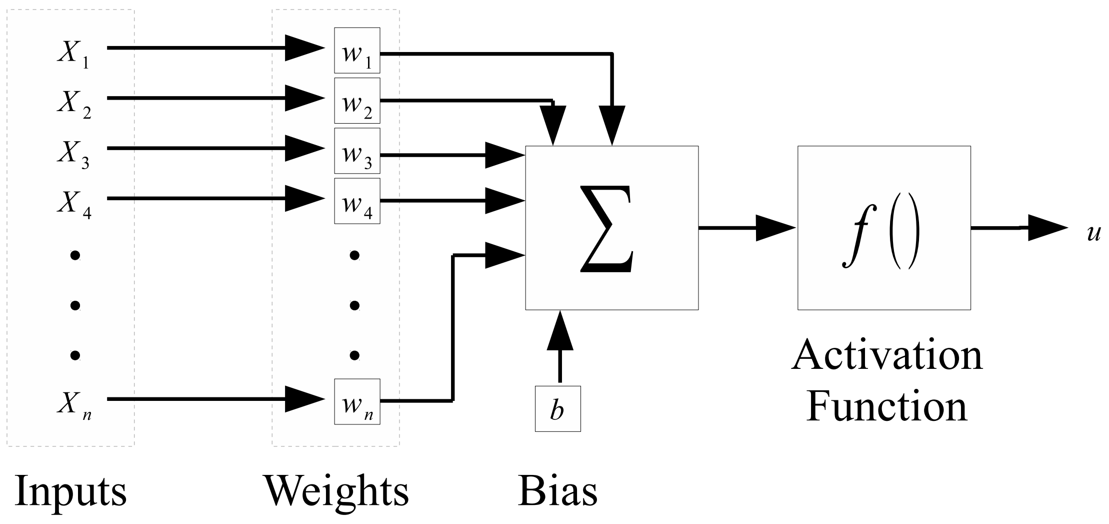


Fig. 14. Diagram of a Neural Cell.

Since the synaptic weights are essentially constants when the neuron is not in training, the datasets and functions the neuron will be able to represent will depend almost exclusively of the activation function $f(u)$. As an example, if the activation function is linear; such as $f(u) = u$, the neuron will excel in representing linear functions of the inputs, but will have unacceptable error when representing, for example, a second-order function.

Since it is difficult to choose an adequate activation function, as often information about the function or dataset to be represented is not known, the solution is to add more neurons. This is done by adding hidden layers, which have this name because they do not have synaptic outputs, but only synaptic connections to other neurons. A representation of a ANN with hidden layers is shown in Fig. 15. This more complex network is capable of representing complex functions and datasets without needing to preselect an adequate activation function. Usually, the activation function for the hidden layer neurons is the SigTan function, and the activation function for the output layer is the linear function.

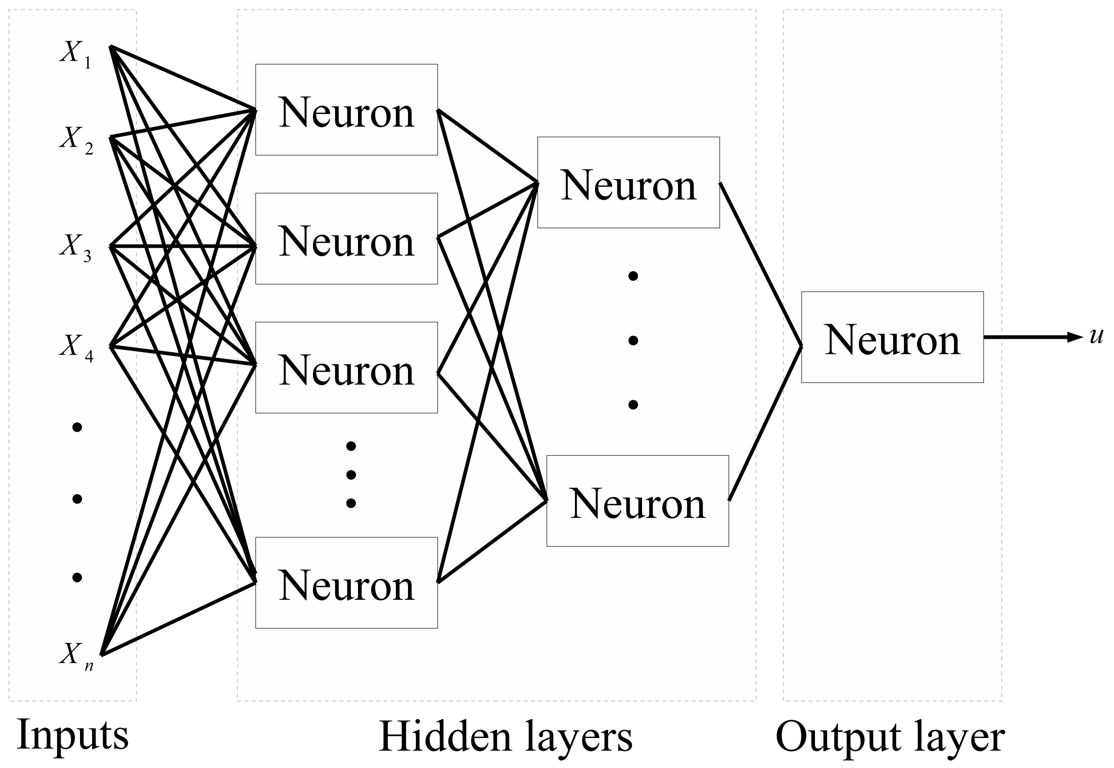


Fig. 15. Diagram of an MLP.

The output of neuron when it is created is very much like a variable that was not initialized properly; it is random in nature and not of much use. Therefore it is always necessary to subject it to a learning process before it can accurately represent a dataset or function. The method used in this thesis is called backpropagation, where errors detected in the output of the neuron are propagated back to their origins, reducing the synaptic weight of the inputs related to the error.

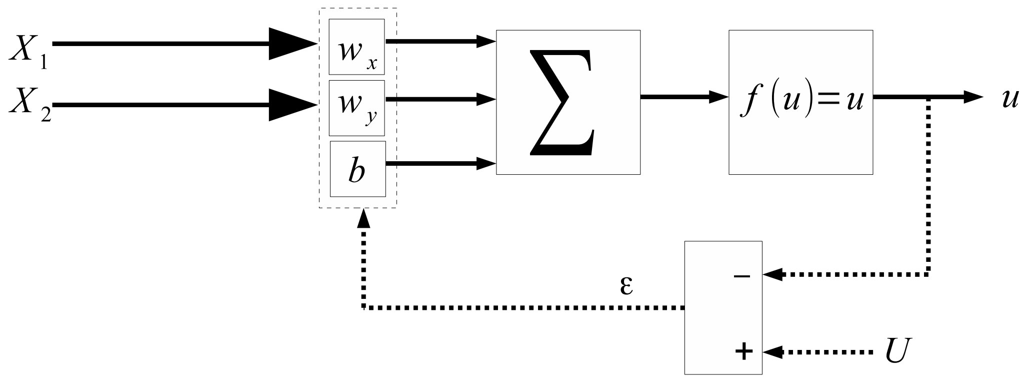


Fig. 16. Example neuron with back propagation training path.

As an example, suppose the single neuron of Fig. 14 has two inputs, $X_1$ and $X_2$. This will result in the ANN shown in Fig. 16. For every pair $(X_1, X_2)$, there exists a target output $U$. To train the neuron using backpropagation, every time a pair $(X_1, X_2)$ generates an output $U + \varepsilon$, that error $\varepsilon$ is propagated backwards towards the inputs changing the synaptic weights. Therefore, for every training input pair where there is an output error, the synaptic weights and bias are updated according to (17) and (18).

$$w_{i_{new}} = w_{i_{old}} + \varepsilon \alpha_L X_i. \tag{17}$$

$$b_{new} = b_{old} + \varepsilon \alpha_L. \tag{18}$$

In these equations, $\alpha_L$ is the learning rate; it determines the magnitude of the synaptic weight change. Raising it will lower training times, but will increase the chance that the training will not converge (i.e., reach the optimal point); lowering will then increase training times, while increasing chance of convergence. When an ANN has done this for every training input once, it is said it endured an epoch of training. This will be repeated until an arbitrary stopping point, which can be the number of epochs, the output MSE or any other. The pseudo-algorithm for learning is presented in Fig. 17.

```
initialize ANN with random weights;
while $MSE\ above\ tolerance$ do
    while $not\ seen\ all\ elements\ in\ training\ epoch$ do
        calculate output via $output = f(bias + \sum_{i=1}^{n} Input_n Weight_n)$ ;
        calculate error $\varepsilon = desired - output$ ;
        update weights $w_{i_{new}} = w_{i_{old}} + \varepsilon\alpha_L x_i$ ;
        update bias $b_{new} = b_{old} + \varepsilon\alpha_L$ ;
    end
    calculate MSE of the epoch ;
end
```

Fig. 17. Backpropagation learning pseudo-algorithm for ANN.

For ANNs with one or more hidden layers, the methodology for backpropagation training is the same, but with the added difficulty that the activation function of the neurons in the hidden layer is non-linear. There are many methods for weight update in this case. The algorithm used in this thesis is a second order algorithm shown in Fig. 18.

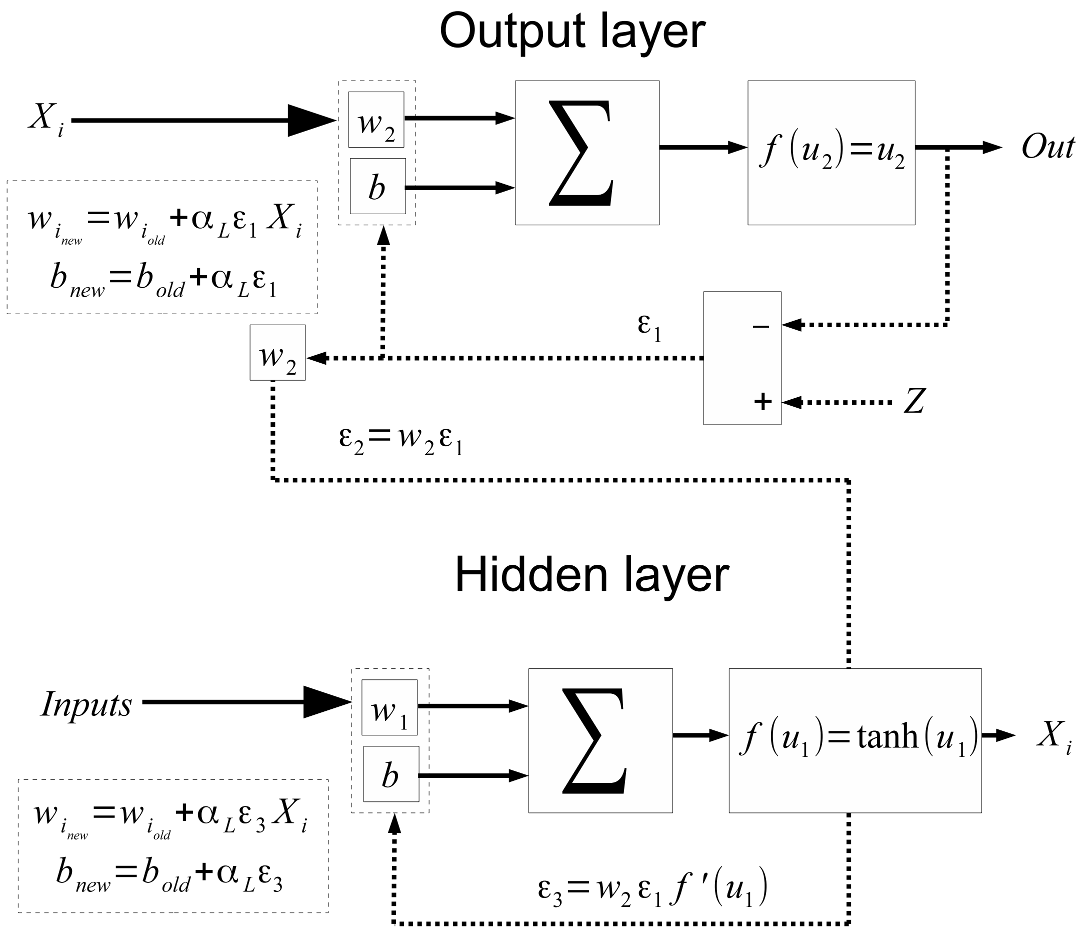

Output layer
$X_i$
$w_2$
$b$
$\sum$
$f(u_2)=u_2$
$Out$
$w_{i_{new}} = w_{i_{old}} + \alpha_L \varepsilon_1 X_i$
$b_{new} = b_{old} + \alpha_L \varepsilon_1$
$\varepsilon_1$
$-$
$+$
$Z$
$w_2$
$\varepsilon_2 = w_2 \varepsilon_1$
Hidden layer
$Inputs$
$w_1$
$b$
$\sum$
$f(u_1) = \tanh(u_1)$
$X_i$
$w_{i_{new}} = w_{i_{old}} + \alpha_L \varepsilon_3 X_i$
$b_{new} = b_{old} + \alpha_L \varepsilon_3$
$\varepsilon_3 = w_2 \varepsilon_1 f'(u_1)$

Fig. 18. Backpropagation of error in ANN with one hidden layer.

### 5.2. Artificial Neural Network Topology and Training

The number of neurons and the geometry of their synaptic connections is called their topology. This aspect will have an impact on all of the ANN characteristics, such as generalization capacity, error and training time. There are many techniques for topology optimization [42, 43], and obtaining the best topology for each application is a complex problem. For the task of estimating either core or system MTTF, it is necessary that the selected topology have a low error and be fast; there is little need for good generalization since data for training the neural network can be generated with as much precision as needed, and within the boundaries of the possible system input parameters such as temperatures and voltages. This means that any temperature that can be reported by the system will have been seen by the ANN during training.

Because of these requirements, it is interesting to keep both the number of neurons and layers at a minimum so that computational overhead can be low enough to not interfere significantly with the system performance or reliability. Determining the minimum number of hidden layers was done using trial and error, beginning with the minimum number; one hidden layer and one output layer, as shown in Fig. 19.

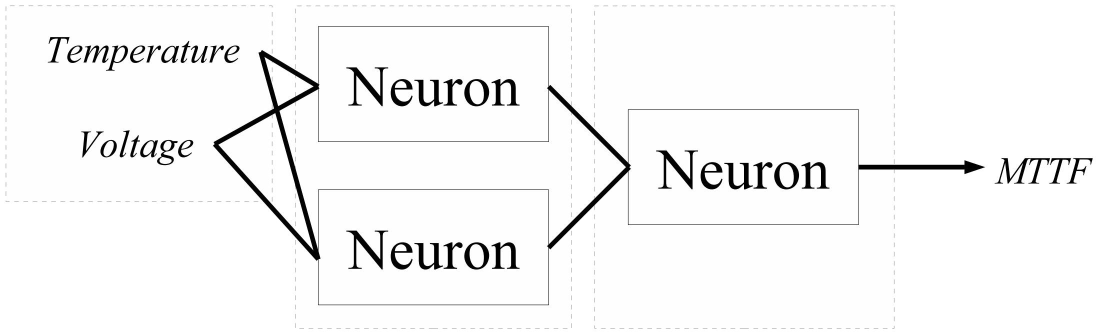


Fig. 19. Final design of the ANN to estimate core MTTF.

The initial number of neurons was chosen arbitrarily. After this first topology design is ready, it is necessary to train the ANN to verify whether or not its performance is satisfactory. Should it not be, the number of neurons can be increased, and, ultimately, if it is still not performing adequately, the number of layers can then be increased.

In order to train the ANN, the REST framework is used to generate the input and target training vectors. The input vector is composed of all possible combinations each core can have regarding temperature and voltage. It is possible to include other parameters that would affect reliability (such as load, air temperature, cooling capacity and others) but the addition of these parameters will come at a cost: Not only the training time and input data set will grow exponentially, but it may also require the ANN to have more layers and neurons. In case the input data set due to added parameters becomes too big to be practical, some data can be left out to force generalization by the ANN.

## 5.3. Training Results and Implementation

While the REST framework can calculate the estimated MTTF of a core or a system based on the individual temperatures of each subunit, the ANN has the disadvantage of relying on the reported temperatures of sensors scattered on the chip. In practice, devices will not have dozens of temperature sensors - but only a select few in strategic places. This has to be considered in both using and training the ANN, since the training epochs will need to represent every possibility of what these thermal sensors will report.

To do that, REST generates a training epoch with the MTTF being calculated via the Monte Carlo method for all possible temperatures in the operating range, and it also generates what would be the temperature seen by the thermal sensor. Additionally, the ANN was trained to the systems being used in this paper; specifically, the ALPHA 64 cores. However, the ANN is flexible enough to be used for most similar CMPs, if the correct training is given. The training algorithm for MATLAB is shown in Appendix A.

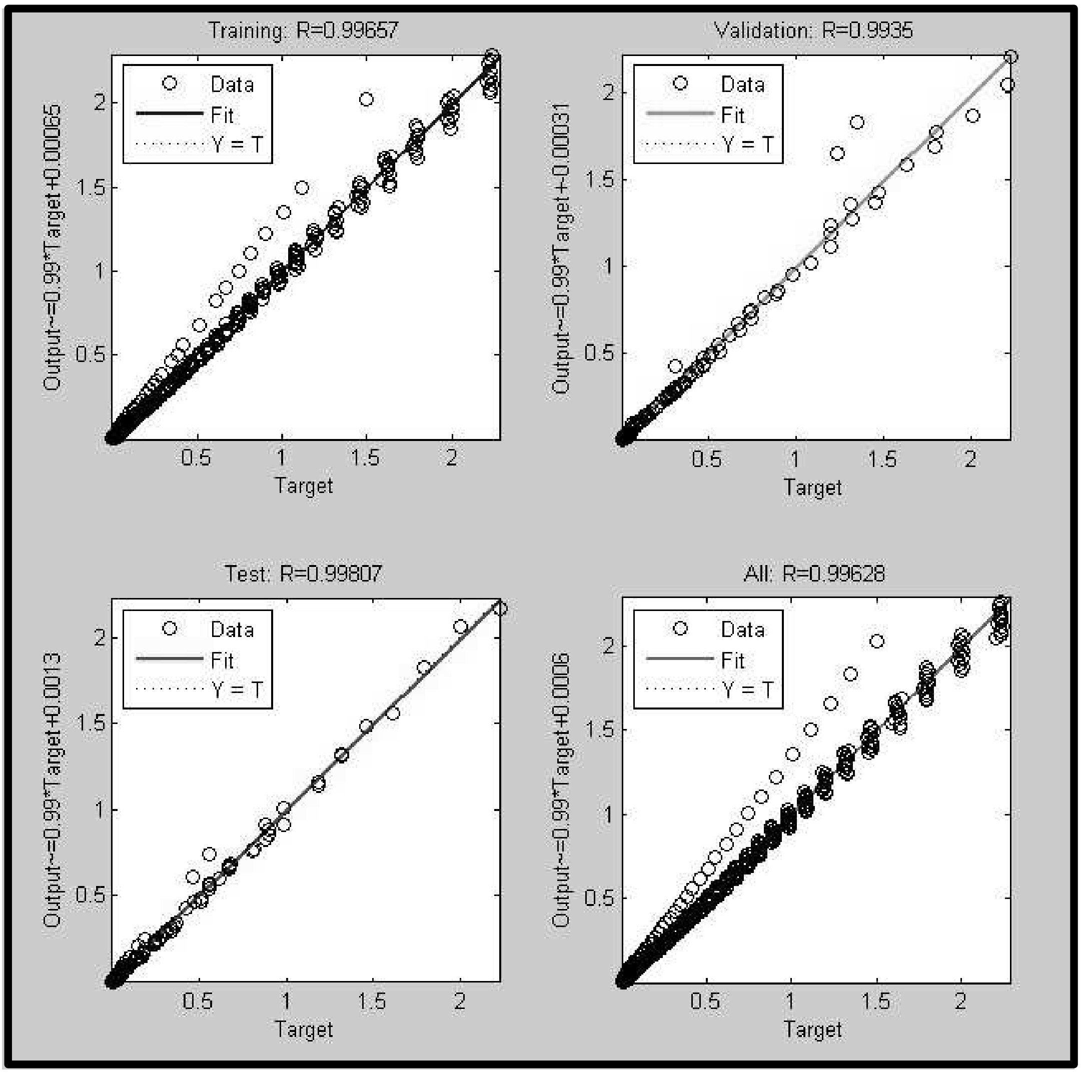
Passing the training epochs through the ANN presented in Fig. 19, we obtain the metrics shown in Fig. 20.

Since the ANN was able to reproduce the results obtained by the REST framework with very low MSE, it is an accurate representation of the MTTF for each of the cores. After the ANN is trained, all that is needed is to implement the trained model into the migration controller. This is done by simply transferring the input weights into the migration controller and calculating the output as shown by (19).

$$MTTF_{neural} = \sum_{i=1}^{2} \left[ w_{i_{output}} \left( \frac{2}{1 + e^{-2 \sum\limits_{j=1}^{N} (w_j Input_j + b_j)}} - 1 \right) + b_{i_{output}} \right]. \qquad (19)$$

In (19), $MTTF_{neural}$ represents the output of the network, being the estimated MTTF for the core. The $w_{output}$ parameters are the synaptic weights of the output layer, and the $w$ parameters are the synaptic weights of the hidden (input) layer. Similarly, $b_{output}$ and $b_j$ represent the bias weights of the output layer and hidden layer, respectively. $N$ represents the number of inputs, which in this case represents the number of temperature and voltage sensors.

The neural network can easily be expanded to include more feedback metrics from the SoC (such as workload and other performance counters) so as to improve MTTF estimation. Additionally, the learning technique explained could be implemented to be done on-line and programmed to respond to different factors, creating an adaptable reliability mnagement scheme.



$$MSE \approx 0.1\,\%$$

Fig. 20. Performance metrics of the ANN training for MTTF estimation.

# 6. TASK MIGRATION BASED DRM

To properly control any system, the most efficient way is to get some form of feedback from either the variables you want to control or at the very least from correlated ones. Thus, since our goal is to control the MPSoC MTTF, it is necessary to calculate the MTTF of the system and feed it into a control system that will vary the inputs. The generic structure for such a controller is shown in Fig. 21.

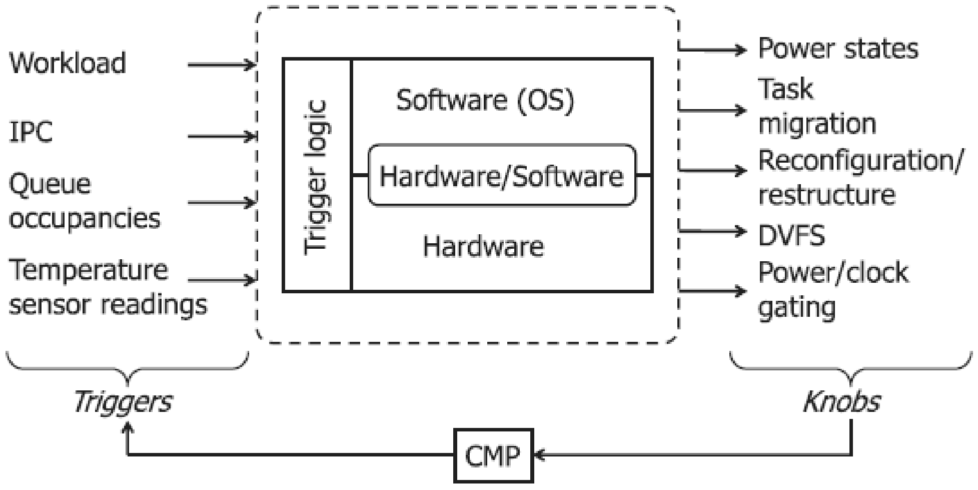


Fig. 21. Generic diagram for a reliability controller.

Acquiring data from the CMP, the reliability controller can then make decisions and adjust the input parameters (the ”knobs”) as necessary.

## 6.1. Processes Effect on Thermal Profiles

One important thing to consider is that different tasks will have different thermal effects on a given core. Depending on memory accesses, arithmetic operations performed and many other factors, hotspots may be generated in different parts of the core, as the example noted in Fig. 22. This example shows the temperatures of different units in the floorplan of an Alpha EV6 21264. Task 1 contains mostly floating point operations, and Task 2 contains mainly integer operations. These hotspots and the overall different temperatures generated will cause the cores to wear out unevenly; and, since the first core to fail will cause the entire system to fail, it is important for maximizing reliability that all cores wear out uniformly.

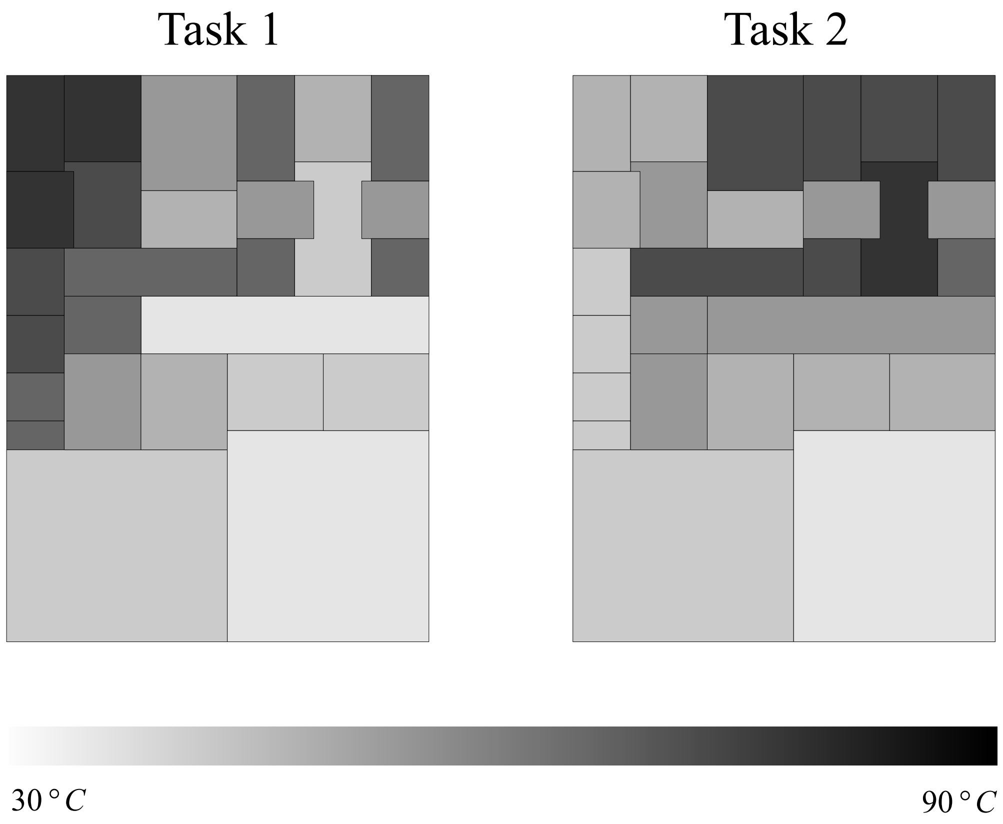


Fig. 22. Different tasks having different hotspots.

## 6.2. Managing Reliability

Managing overall temperature will certainly help increase reliability, but will do nothing to achieve uniform wear. Thus, an alternative is to assign tasks based on the processor wear; This is an interesting concept, but it will come with a performance penalty, since tasks that need to communicate with each other will most likely not be in the optimal position to reduce traffic on the NoC.

Because of this, it is easier to manage the system based on the user requirement. If the user has no need for system reliability, tasks are free to be allocated with performance in mind. If, however, a higher reliability figure is desired, the system can then relocate these tasks, promoting wear levelling and lower hotspots. This provides greater flexibility, and causes performance penalties only when it is necessary for the user. Since the controller will need to act on the tasks at any given moment, the best choice is to be able to remap tasks to a different core. Fig. 23 shows such a controller, which takes as inputs the desired user reliability and parameters from the CMP that make it able to calculate said reliability.

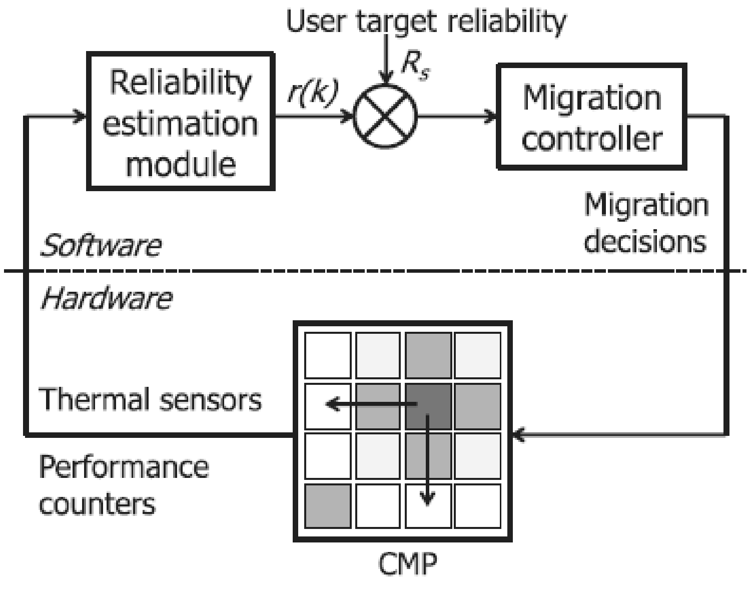


Fig. 23. Task migration controller with reliability estimation module.

Periodically, the migration controller will poll the CMP for performance characteristics and, if necessary, remap tasks to more reliable cores in order to keep system reliability above the set target. There are many possible ways to choose which tasks will be remapped (and to which core), but in order to keep performance penalties at a minimum, we chose to simply remap the task contained in the core with the lowest MTTF to the core with the highest MTTF. This process is shown in Fig. 24.

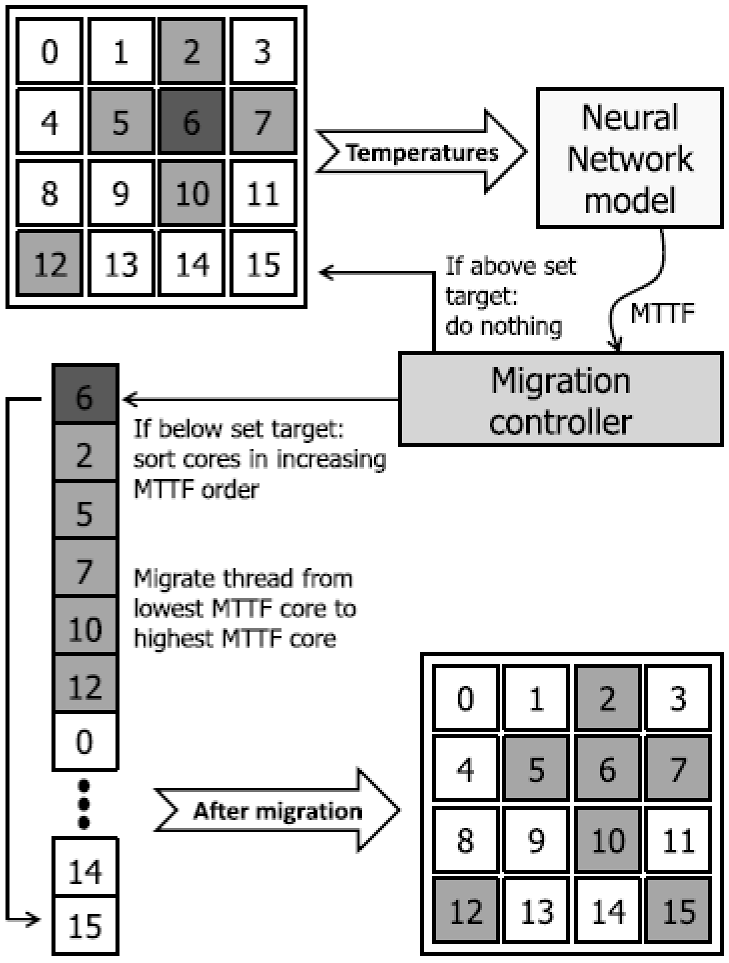

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Temperatures
Neural Network model
MTTF
If above set target: do nothing
Migration controller
If below set target: sort cores in increasing MTTF order
6
2
5
7
10
12
0
14
15
Migrate thread from lowest MTTF core to highest MTTF core
After migration

Fig. 24. Task migration controller flow.

This migration process will occur every time the system reliability is below the desired target. Because of that, both reliability and polling/migration intervals must be set correctly to avoid severe performance penalties. Too short polling intervals will cause severe performance penalties because of excessive migration (if below target) and computation of each core MTTF, and too long intervals will cause the system to violate reliability targets very often.

An alternative would be to make these intervals dynamic; shortening them when system reliability gets lower and increasing them when they get higher. For this work, the polling interval was set as a constant.

## 6.3. DRM Implementation

The proposed DRM can be implemented both in hardware and in software. For testing and flexibility, we decided to implement the migration controller in software; this is very flexible in that minor adaptations are needed in order to implement it in existing systems, and it can easily be updated when new techniques are developed. It will, however, not be ultimately as efficient as a hardware implementation, since the software will need to make use of a core to compute the reliability estimates.

# 7. REST AND DRM RESULTS

The REST framework serves two purposes: to generate reliability data specific for an MPSoC floorplan so as to train the corresponding ANN and to be able to simulate reliability figures for testing of the proposed DRM scheme.

## 7.1. Testbed System and Benchmarks

For testing of the proposed DRM scheme, it is fairly challenging to predict what the user behaviour will be; the best we can do is to assume a default set of workloads and a system that tries to represent the average user. Because of the availability of support and floorplan data on the ALPHA 26264 processor, it was the processor of choice for the simulations. The ALPHA core used on the testbed system is shown in Table 1.

Table 1. ALPHA core MPSoC parameters.

| Parameter | Value |
|---|---|
| Core (Frequency, VDD) | Alpha EV6 21264 (1 GHz, 2 V) |
| Branch predictor | 2-bit counter |
| Reorder buffer | 80-entries |
| L1 ICache | 32KB |
| L1 DCache | 64KB |
| L2 | 2MB |
| Network | 2D regular mesh, 1 router per core |
| Link bandwidth | 32 bits |
| Routing algorithm | XY |
| Number of virtual channels (VCs) | 2 |

To properly test the system under load, we have chosen to utilize the PARSEC set of benchmarks. This set of benchmarks provides a decent load in the system and is repeatable and not too time-consuming; most benchmarks used are finished before 24 hours of real-time simulation.

In addition, to further test the proposed technique, the DRM was also used in a quad-core Intel i7 processor, which is shown in Table 2. The benchmark running in the i7 core is the open-source *y-cruncher*, which computes $\pi$ with arbitrary precision and is multi-threaded.

Table 2. Intel i5 system parameters.

| Parameter | Value |
|---|---|
| Core (Frequency, VDD) | Intel i5 3200 (3GHz, 1.2V) |
| Operating System | Ubuntu 13 |
| Kernel | Modified 22.4.1 |
| RAM Size | 16GB |
| Chipset | X58 |

## 7.2. Using the REST Framework to Evaluate NoC Floorplan Design Impact on Reliability

One very important design consideration to be made is the floorplan of the core. In respect to reliability, the floorplan is important because it will change the location of hotspots, which may reduce or improve reliability. Fig. 25 illustrates this, by showing two cores. Core 1 has all arithmetic units clustered together, which generates a massive hotspot which will reduce lifetime dramatically (since MTTF will exponentially decrease with temperature). Core 2, however, processes the same tasks with a much higher reliability. This is because spacing out very powerful intensive logic units with units that do not produce as much power will make the latter behave as heatsinks, effectively evening out the temperature across the core.

With that in mind, we then use REST to verify what is the impact of the location of the router in the core. They cannot be moved to many positions in the core, having to mainly stick to the edges. Thus, we evaluate reliability when assigning the router to two positions; in the side of the core, or on top of it, as shown in Fig. 26.

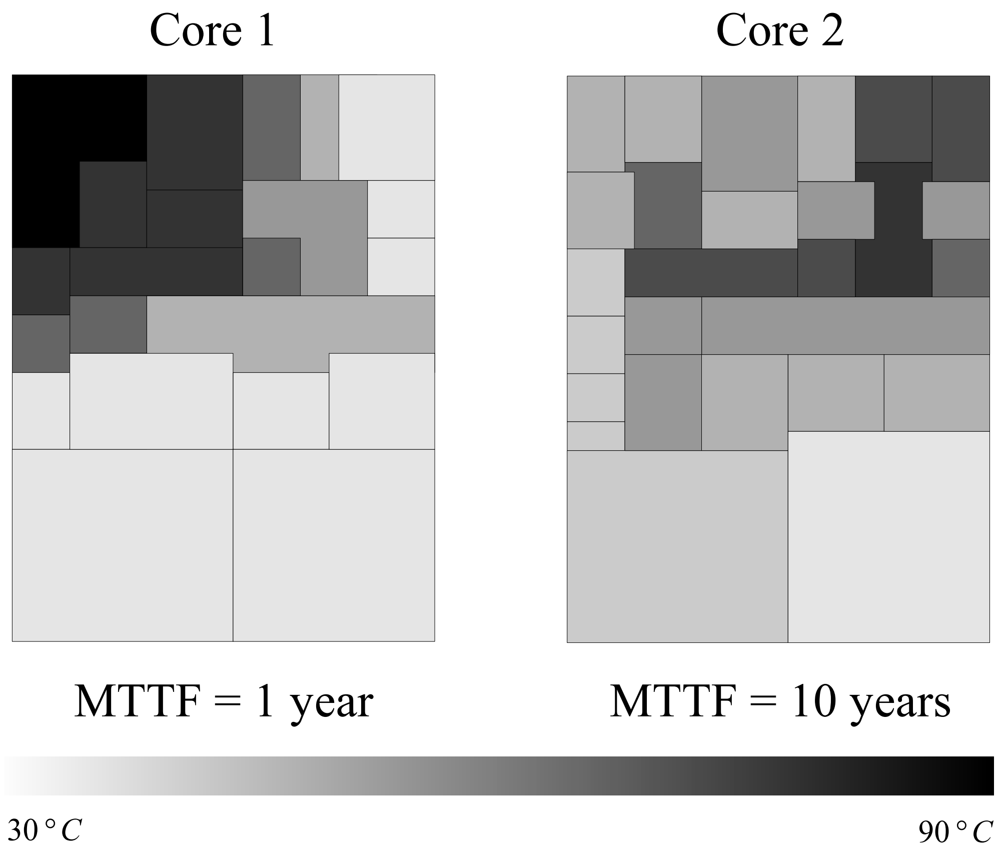


Fig. 25. Hotspots in two cores with different floorplans.

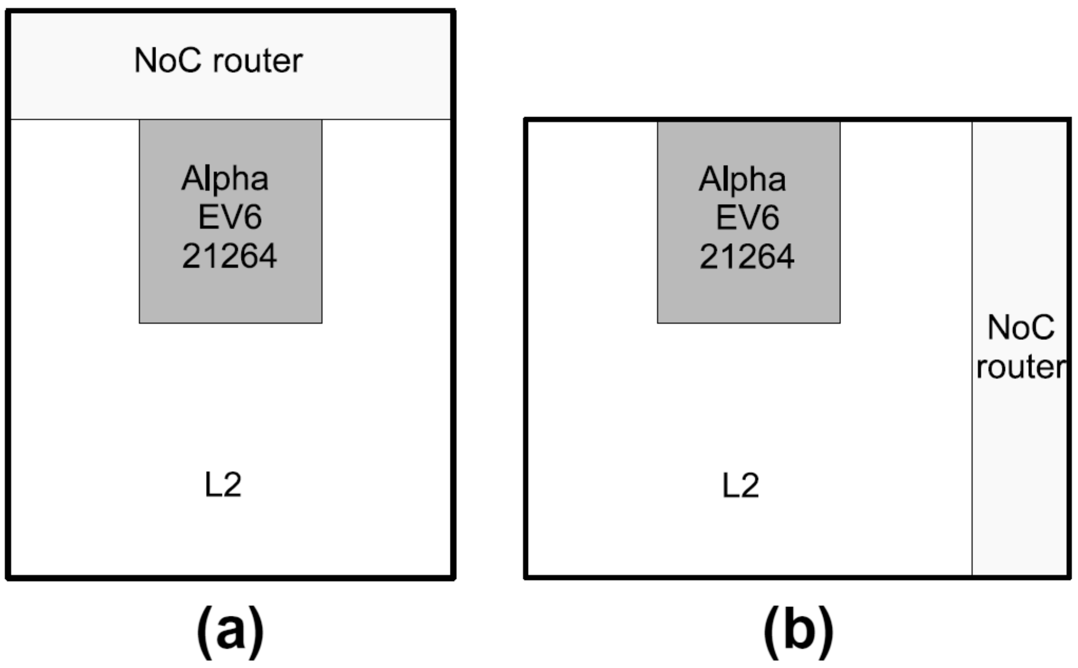


Fig. 26. Two different locations for the NoC router.

The area occupied by the router will vary depending on the number of ports and buffer size, but we chose one that is 20% of the area of the core, based on the discussions and designs in [53, 54]. To verify the difference between the two router locations, several PARSEC benchmarks were run (Blackscholes, Facesim, Canneal and Swaption) in a MPSoC composed of 4, 16 and 64 ALPHA EV6 cores. To properly ensure that all cores would run threads, all benchmarks were run with 64 threads (for all MPSoCs, even those with less than 64 cores). The results are shown in Fig. 27.

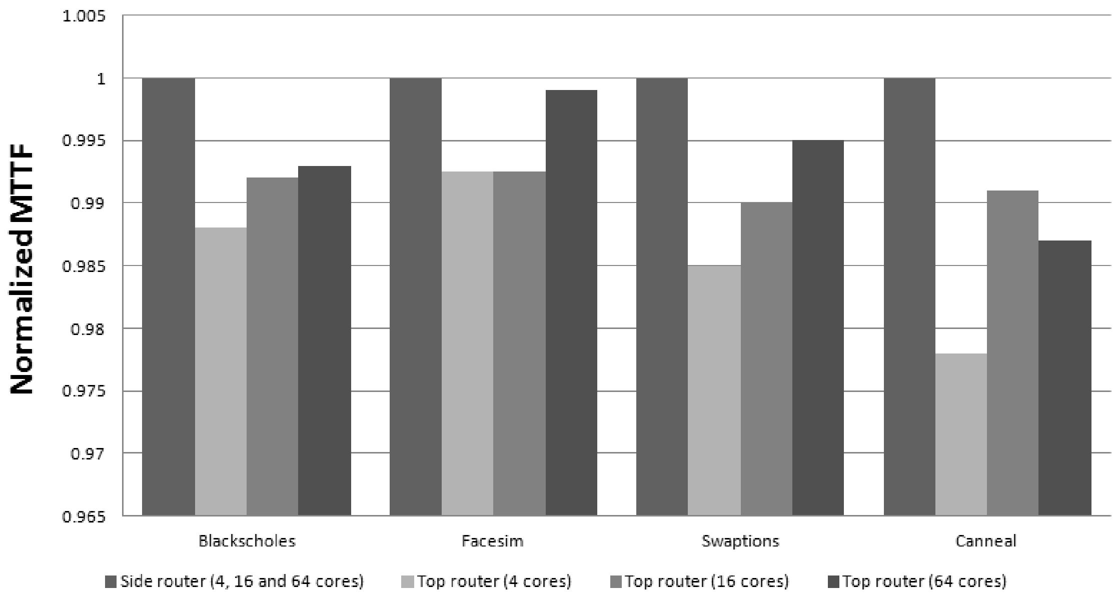


Fig. 27. Relative comparison of MTTF due to router location.

All of the results shown are normalized to the lifetime of the case where the router is by the side of the core (Fig. 26(b)). We can easily see that the difference between both configurations is very small, usually not exceeding 2%. This may be because the benchmarks used did not have sufficient communication to stress the NoC, and thus would not generate enough heat to create local hotspots. It can be noted however that all the cases where the router was on top (Fig. 26(a)) - which means it was right next to the processing unit; performed worse, even though by a small margin. This suggests that placing the router as far away as possible from the core will help reliability by reducing local hotspots.

## 7.3. Evaluating NoC Presence Impact on Reliability

Previous reliability models did not account for the NoC, obtaining a optimistic figure for the reliability of the system. To evaluate how accurate the assumption that the NoC impact on reliability is negligible, REST was used to estimate the reliability figures of the same system used for determining NoC floorplan impact in reliability. The results are shown in Fig. 28. The router position for all of the simulations was that of Fig. 26(b).

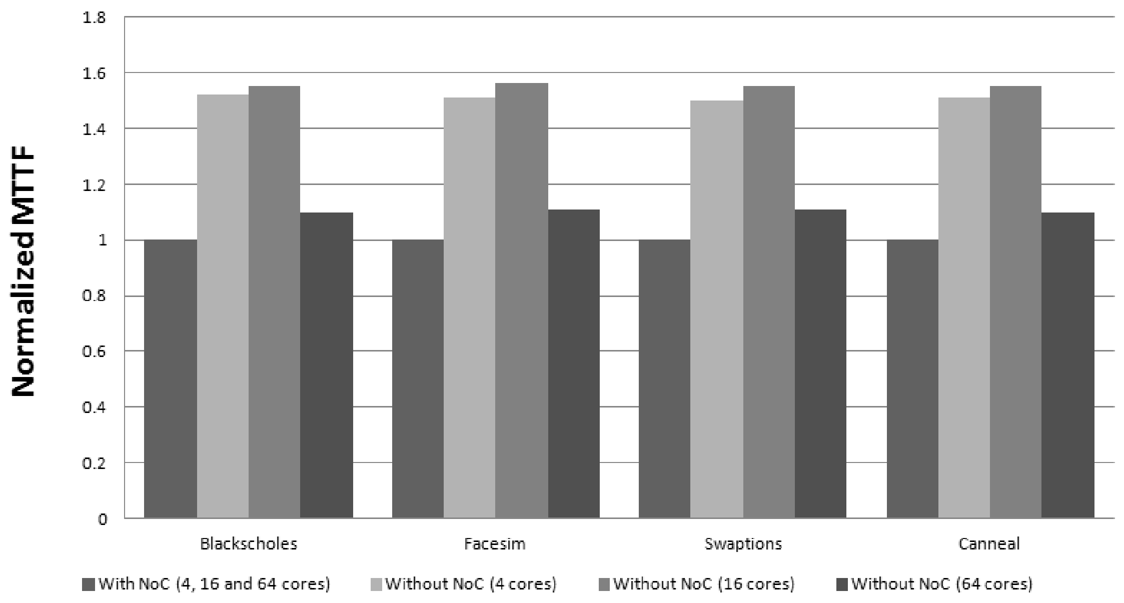


Fig. 28. Relative comparison of MTTF when the NoC is considered or not.

It is easy to see that the difference in the reliability when NoCs are not considered is very high, reaching close to 60%. This is because the production of power by the router is not negligible; previous study has shown that processors and routers alone can generate temperatures of up to $77.9\,°\mathrm{C}$ and $68.6\,°\mathrm{C}$, respectively, while when considered as one unit, the peak can reach up to $104.7\,°\mathrm{C}$ [55].

This indicates that reliability schemes that do not consider processors and network together may not be optimal.

## 7.4. Proposed DRM Scheme Results in GEM5 Simulation

Using the DRM scheme and the migration controller proposed in the previous chapter, we ran simulations for the same PARSEC benchmarks used previously. The CMP is composed of 16 ALPHA EV6 cores, arranged in a 4x4 grid with a mesh interconnected NoC. Routers are positioned as in Fig. 26(b), and the configuration parameters remain those listed in Table 1. All benchmarks were set to be executed using 16 threads, assuring every core would be busy. The result for the run of *blackscholes* is shown in Fig. 29.

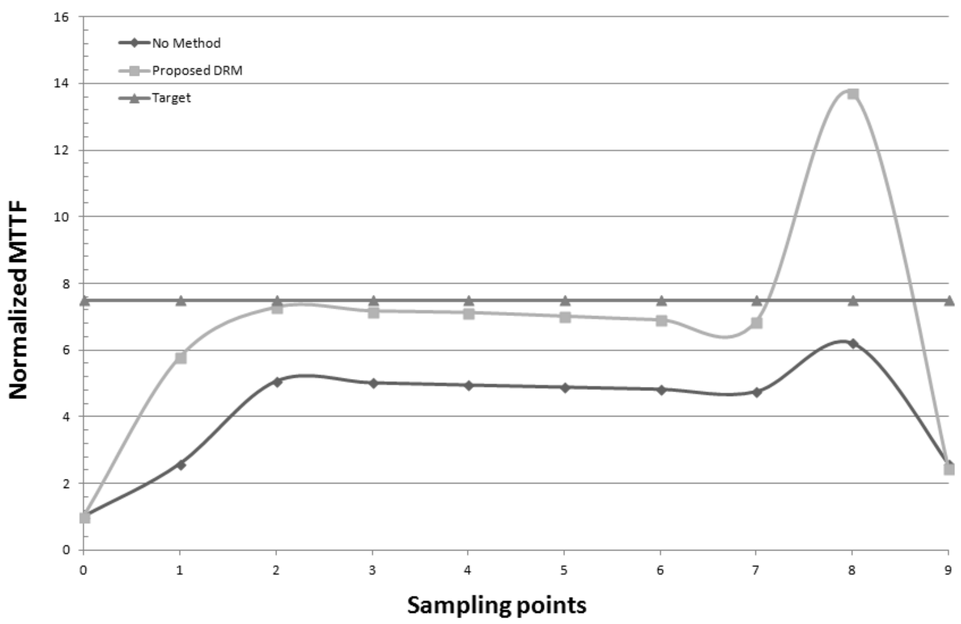


Fig. 29. MTTF results for GEM5 simulation of the PARSEC benchmark *blackscholes*.

The results represent the ROI for the benchmark, where all of the intensive calculations are done. No action is made on the initialization, for the most part because reliability will always be above the set target (unless the set target is unrealistic). This happens because during initialization, there's no more than three cores active. Each of the data points represent the time where reliability is evaluated, and a decision is made of whether to migrate tasks or not; thus, each data point represents the average reliability figure since the last intervention.

For this benchmark, the normalized reliability target is set at 7.5. It should be noted that the set reliability target is too high, and (with the exception of one point) cannot be achieved only with task migration. However, reliability figures are still better than without any DRM by $50.12\%$ on average. The benchmark spends $568\ ms$ of system time in ROI without any DRM scheme. With the proposed method, this time increases to $608\ ms$, a $6.51\%$ performance penalty. This happens because tasks spend extra time to migrate from core to core to even out the temperatures across the CMP; since the target is almost never met, migration will occur frequently, which reflects in performance.

When the reliability target is reasonably set as to allow migration to only occur every other time, the performance penalty will be reduced. This is shown in the results from the *swaptions* benchmark, in Fig. 30.

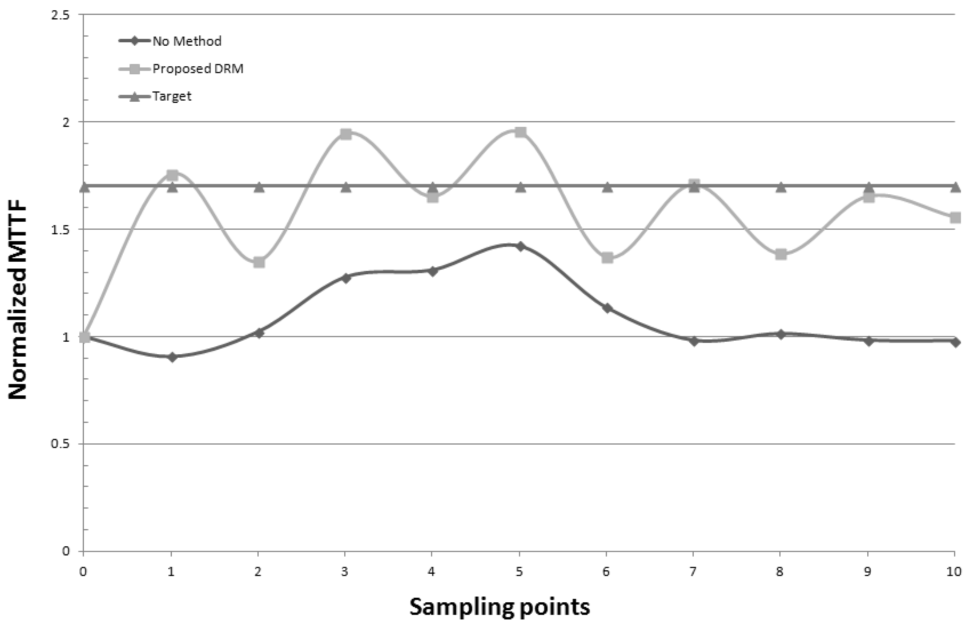


Fig. 30. MTTF results for GEM5 simulation of the PARSEC benchmark *swaptions*.

In this case, the normalized reliability is set to 1.7, a lower target that is obtainable with the task migration controller. It can also be seen that because of the simple feedback path of the controller, reliability will quickly oscillate around the set reliability target when it is met. This could further be reduced with controlling techniques, a suggestion that is left for future work. Since not many task migrations will be executed, the running time of the ROI increases to $804\ ms$ with the proposed DRM method, from an ROI time of $781\ ms$ without any method; a performance penalty of only 2.94%. The results for two other benchmarks, *facesim* and *canneal*, are shown in Fig. 31 and Fig. 32, respectively.

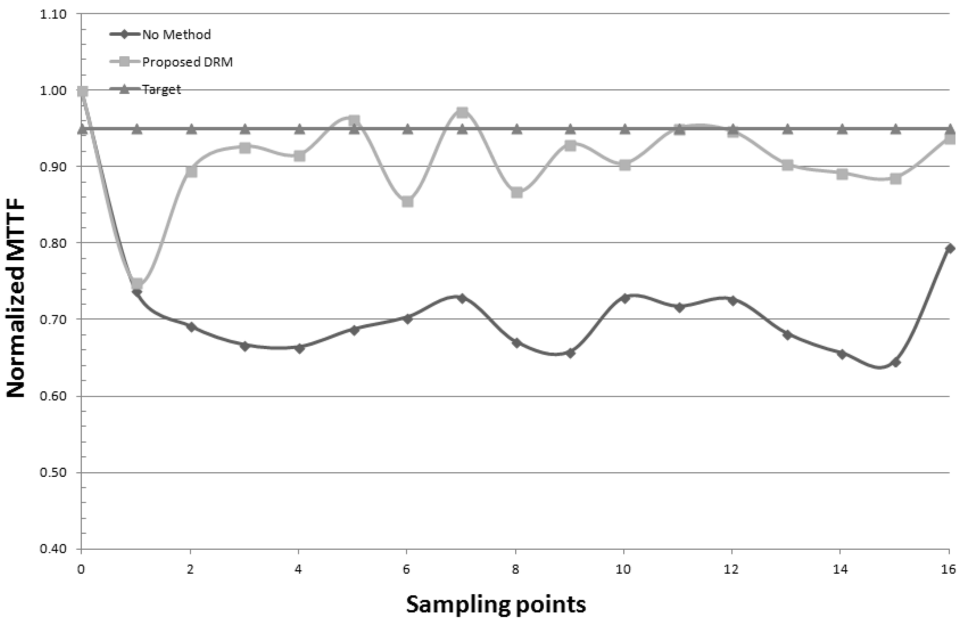


Fig. 31. MTTF results for GEM5 simulation of the PARSEC benchmark *facesim*.

Since the reliability target is again almost never met, it is expected that the performance penalty is going to be higher. For *facesim*, ROI execution time is $2448\ ms$ with the method against $2319\ ms$ without it, representing a 7.28% penalty; target reliability was set at 0.95. For *canneal*, ROI execution time is $524 ms$ with the method against $480 ms$ without it, representing a 9.16% penalty; target reliability was set at 2.25. All of the performance characteristics of the proposed DRM method are summed up in Table 3.

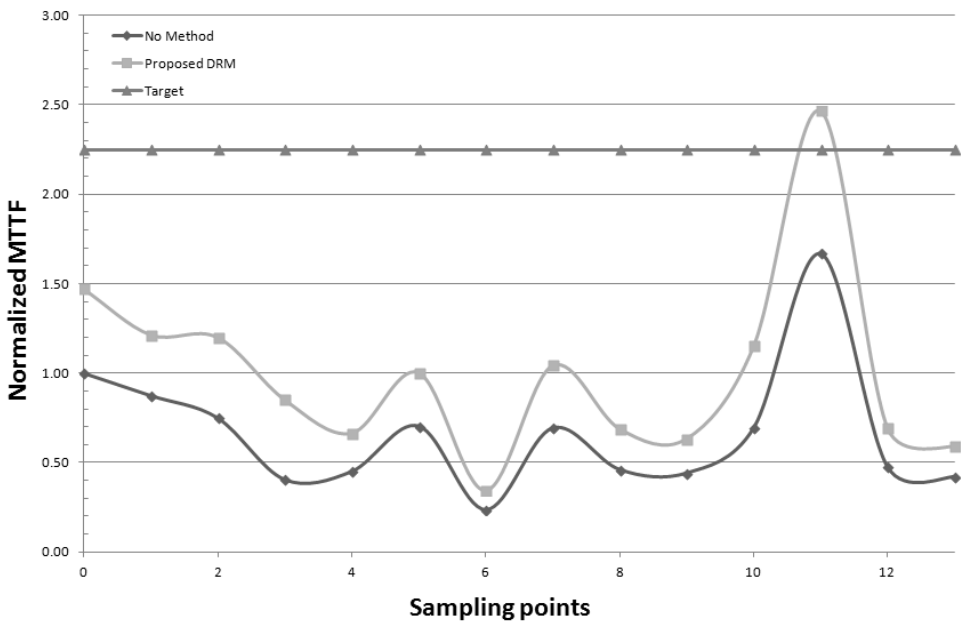


Fig. 32. MTTF results for GEM5 simulation of the PARSEC benchmark *canneal*.

Table 3. Benchmarks performance metrics.

| Benchmark [52] | Avg. MTTF improv.[%] | Perf. Penalty [%] | ROI Exec. time (no DRM) [ms] | ROI Exec. time (DRM) [ms] |
|---|---|---|---|---|
| Blackscholes | 45.24 | 2.94 | 781 | 804 |
| Swaptions | 50.12 | 6.51 | 568 | 605 |
| Facesim | 32.7 | 7.28 | 2319 | 2448 |
| Canneal | 52.2 | 9.16 | 480 | 524 |

## 7.5. Proposed DRM Scheme Results in a Current Consumer CMP

Because the proposed DRM was implemented in software, it does not require any extra hardware to be deployed (as long as the CMP has temperature and voltage sensors, which is true for most). To demonstrate this, it is possible to run the DRM scheme in a common multi-core computer such as the one mentioned in Table 2. The results of the ROI of the simulation are shown in Fig. 33. The *y-cruncher* benchmark is run with 8 threads, and the reliability target is set at 1.575.

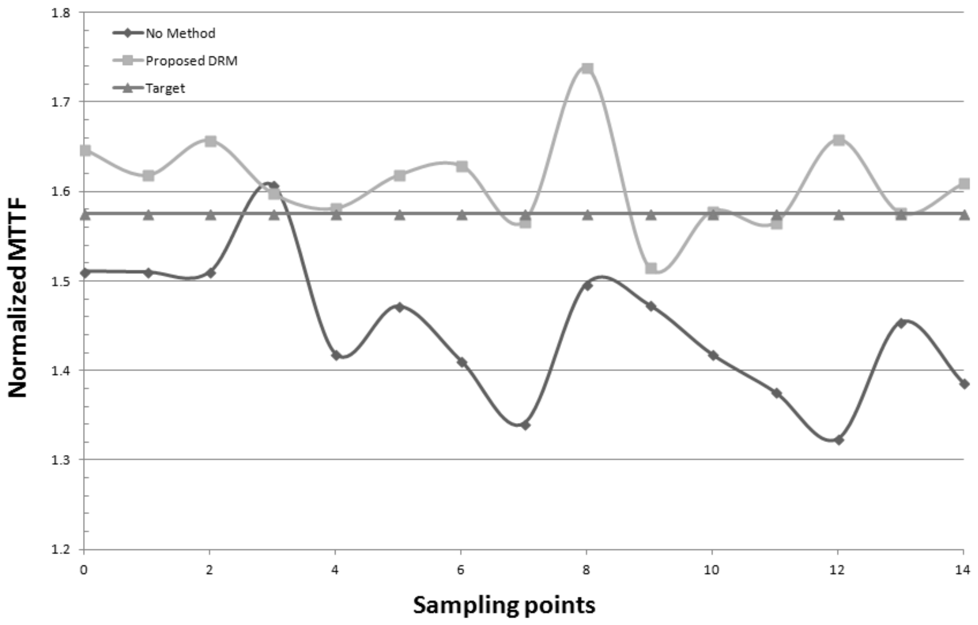


Fig. 33. MTTF results for computation of pi using *y-cruncher*.

The first important thing to mention is that, differently from the other graphs, the reliability figures will not match for the beginning of the ROI, because the DRM method will already be acting before the benchmark reaches that point. Additionally, the reliability gains, although present, are now for this case 11.25% on average; where the performance penalty was under 5%.

Thus, the proposed DRM method also works for current CMPs, albeit not as efficiently.

# 8. FUTURE WORK

While the proposed dynamic reliability management scheme is successful in maintaining a given system reliability under certain limits, there are a few limitations to the system.

## 8.1. Setting Arbitrary Reliability Goals

The first and most important limitation is the setting for the desired reliability - there's only so much that can be done by only migrating tasks being executed in the MPSoC. While it's entirely plausible that reliability can be improved by 50% or 100% (as shown by the results in Chapter 6), setting the desired reliability to values such as 10 times more lifetime may very well be impractical and will need different DRM techniques such as DVFS.

## 8.2. Wear Levelling

Task remapping can be improved further with the addition of Wear Levelling techniques. In this way, all cores are worn out evenly, maximizing the probability $R(t)$ that the system will keep working correctly. This is because all cores will have been used the same, and will thus have the same probability of failure.

## 8.3. Control Theory Techniques

It is easy to notice from the results in Chapter 6 that reliability does not stabilize at the desired value; but instead keeps fluctuating around it. This is because a very simple feedback system was implemented. By modelling the system in question and using results from control theory, it is possible to design the system such that the actual reliability will have a much smoother behaviour and maintain the desired value.

## 8.4. Adaptable DRM

As explained in Chapter 5, it is possible to implement a reinforced learning algorithm in the ANN responsible for the reliability estimation of the cores. By adding more performance metrics as inputs to the ANN and providing an on-line method of training for the ANN, it is possible to create an adaptable dynamic reliability management scheme that accounts for many parameters that can only be known at the time of use; such as usage patterns, wear, climate data, process and manufacture variations and so on. This would be a very powerful tool for optimal solutions based on the user, as well as being able to compensate for minor variations in fabrication and manufacture.

# 9. BIBLIOGRAPHY

[1] S. Borkar. “Designing reliable systems from unreliable components: the challenges of transistor variability and degradation”. *IEEE Micro*, 25 (6) 2005 1016.

[2] A. DeHon, H.M. Quinn, N.P. Carter. “Vision for cross-layer optimization to address the dual challenges of energy and reliability”. *ACM/IEEE Design Automation and Test in Europe Conf.*, 2010.

[3] S. Borkar. “Thousand core chips: a technology perspective”. *ACM/IEEE Design Automation Conf. (DAC)*, 2007.

[4] The International Technology Roadmap for Semiconductors (ITRS) 2011. http://www.itrs.net/Links/2011ITRS/2011Chapters/2011SysDrivers.pdf.

[5] T. Simunic, S.P. Boyd, P.W. Glynn. “Managing power consumption in networks on chips”. *IEEE Trans. Very Large Scale Integr. Syst. (TVLSI)*, 12 (1) (2004) 96107.

[6] Y. Wang, K. Ma, X. Wang. “Temperature-constrained power control for chip multiprocessors with online model estimation”. *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, 2009.

[7] Y. Wang, Q. Xie, A. Ammari, M. Pedram. “Deriving a near-optimal power management policy using model-free reinforcement learning and Bayesian classification”. *ACM/IEEE Design Automation Conf. (DAC)*, 2011.

[8] P. Bogdan, S. Jain, R. Tornero, R. Marculescu. “An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads”. *ACM/IEEE Int. Symp. on Networks-on-Chip (NOCS)*, 2012.

[9] J. Donald and M. Martonosi. “Techniques for multicore thermal management: classification and new exploration”. *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, 2006.

[10] A.K. Coskun, T.S. Rosing, K. Gross. “Utilizing predictors for efficient thermal management in multiprocessor SoCs”. *IEEE Trans. CAD Integr. Circuits Syst. (TCAD)*, 28 (10) (2009) 15031516.

[11] Jayanth Srinivasan. *Lifetime reliability aware microprocessors*. Ph.D. Thesis, University of Illinois at Urbana-Champaign, 2006.

[12] J. Shin, V. Zyuban, Z. Hu, J. Rivers, P. Bose. “A framework for architecture-level lifetime reliability modeling”. *IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2007.

[13] A.K. Coskun, T.S. Rosing, K. Mihic, G.D. Micheli, Y. Leblebici. “Analysis and optimization of MPSoC reliability”. *J. Low Power Electr.*, 2 (1) (2006) 5669.

[14] Z. Gu, C. Zhu, L. Shang, R.P. Dick. “Application-specific MPSoC reliability optimization”. *IEEE Trans. Very Large Scale Integr. Syst. (TVLSI)*, 16 (5) (2008) 2008.

[15] Y. Xiang, T. Chantem, R. Dick, X.S. Hu, L. Shang. “System level reliability modeling for MPSoCs”. *IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES + ISSS)*, 2010.

[16] J.P.G. Sterbenz, D. Hutchison, E. Cetinkaya, A. Jabbar, J.P. Rohrer, M. Scholler, P. Smith. “Resilience and survivability in communication networks: strategies, principles, survey of disciplines”. *Comput. Networks*, (2010).

[17] A. Dalirsani, M. Hosseinabady, Z. Navabi. “An analytical model for reliability evaluation of NoC architectures”. *EEE Int. on-Line Testing Symposium (IOLTS)*, 2007.

[18] H. Elmiligi, A.A. Morgan, M.W. El-Kharashi, F. Gebali. “A reliability-aware design methodology for networks-on-chip applications”. *IEEE Int. Conf. on Design and Technology of Integrated Systems in Nanoscale Era*, 2009.

[19] C. Ababei, H. Sajjadi Kia, O.P. Yadav, J. Hu. “Energy and reliability oriented mapping for regular networks-on-chip”. *ACM/IEEE Int. Symposium on Networks-on-Chip (NOCS)*, 2011.

[20] T. Austin, D. Blaauw, T. Mudge, K. Flautner. “Making typical silicon matter with Razor”. *IEEE Comput.*, 37 (3) (2004) 5765.

[21] Z. Lu, J. Lach, M.R. Stan, K. Skadron. “Improved thermal management with reliability banking”. *IEEE Micro*, 25 (6) (2005) 4049.

[22] E. Karl, D. Blaauw, D. Sylvester, T. Mudge. “Multi-mechanism reliability modeling and management in dynamic systems”. *IEEE Trans. Very Large Scale Integr. Syst. (TVLSI)*, 16 (4) (2008).

[23] S. Feng, S. Gupta, A. Ansari, S. Mahlke. “Maestro: orchestrating lifetime reliability in chip multiprocessors”. *Int. Conf. on High-Performance Embedded Architectures and Compilers (HiPEAC)*, 2010.

[24] A. Tiwari and J. Torrellas. “Facelift: hiding and slowing down aging in multicores”. *ACM/IEEE Int. Symp. on Microarchitecture (MICRO)*, 2008.

[25] C. Zhuo, D. Sylvester, D. Blaauw. “Process variation and temperature-aware reliability management”. *ACM/IEEE Design Automation and Test in Europe Conf.*, 2010.

[26] A.K. Coskun, R.D. Strong, D.M. Tullsen, T.S. Rosing. “Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors”. *SIGMETRICS/Perform.*, (2009).

[27] *Failure Mechanisms and Models for Semiconductor Devices*. JEDEC Publication JEP122E, 2009.

[28] A.Y. Yamamoto and C. Ababei. “Unified system level reliability evaluation methodology for multiprocessor systems-on-chip”. *IEEE Int. Green Computing Conference (IGCC)*, 2012.

[29] J.H. Stathis. “Reliability limits for the gate insulator in CMOS technology”. *IBM J.Res. Develop.*, 46 (2002) 265286.

[30] E. Wu, J. Sune, W. Lai, E. Nowak, J. McKenna, A. Vayshenker, D. Harmon. “Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides”. *Solid-State Electron.*, 46 (11) (2002) 17871798.

[31] D.K. Schroder and J.A. Babcock. “Negative bias temperature instability: road to cross in deep submicron silicon semiconductor manufacturing”. *J. Appl. Phys.* 94 (1) (2003) 118.

[32] S. Zafar, B. Lee, J. Stathis, A. Callegar, T. Ning. “A model for negative bias temperature instability (NBTI) in oxide and high k pFETs”. *Int. Symposium on VLSI Technology*, 2004.

[33] C. Bauckhage, K. Kersting, B. Rastegarpanah. “The Weibull as a Model of Shortest Path Distributions in Random Networks”. *Eleventh Workshop on Mining and Learning with Graphs*, Chicago, Illinois, 2013.

[34] Y. Zhang and A. Srivastava. “Adaptive and autonomous thermal tracking for high performance computing systems”. *ACM/IEEE Design Automation Conf. (DAC)*, 2010.

[35] P. Kumar and D. Atienza. “Run-time adaptable on-chip thermal triggers”. *ACM/ IEEE Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2011.

[36] R. Jayaseelan and T. Mitra. “Dynamic thermal management via architectural adaptation”. *ACM/IEEE Design Automation Conf. (DAC)*, 2009.

[37] Y. Ge, Q. Qiu, Q. Wu. “A multi-agent framework for thermal aware task migration in many-core systems”. *IEEE Trans. Very Large Scale Integr. Syst. (TVLSI)*, 20 (10) (2012) 17581771.

[38] J. Sun, A.K. Kodi, A. Louri, J.M. Wang. “NBTI aware workload balancing in multicore systems”. *IEEE Int. Symp. on Quality Electronic Design (ISQED)*, 2009.

[39] J.H. Stathis. “Reliability limits for the gate insulator in CMOS technology”. *IBM J. Res. Develop.* 46 (2002) 265286.

[40] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, S. Borkar. “A 5-GHz mesh interconnect for a teraflops processor”. *IEEE Micro*, 27 (5) (2007) 5161.

[41] B. Li, L.-S. Peh, P. Patra. “Impact of process and temperature variations on network-on-chip design exploration”. *IEEE Int. Symposium on Networkson- Chip (NOCS)*, 2008.

[42] Mohammed Attik, Laurent Bougrain, and Frdric Alexandre. “Neural Network Topology Optimization”. *ICANN 2, volume 3697 of Lecture Notes in Computer Science*, Springer, (2005) 53-58

[43] M. Cuellar, M. Delgado, M. Jimnez. “Topology Optimization and Training of Recurrent Neural Networks with Pareto-Based Multi-objective Algorithms: A Experimental Study”. *IWANN, volume 4507 of Lecture Notes in Computer Science*, Springer, (2007) 359-366

[44] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewall, M. Shoaib, N. Vaish, M.D.

Hill, D.A. Wood. “The gem5 simulator”. *ACM SIGARCH Comput. Architect. News Arch.*, (2011)

[45] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, S.K. Reinhardt. “The M5 simulator: modeling networked systems”. *IEEE Micro*, 26 (4) (2006) 5260.

[46] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood. “Multifacets general execution-driven multiprocessor simulator (GEMS) toolset”. *Comput. Architect. News (CAN)*,(2005).

[47] N. Agarwal, L.-S. Peh, N. Jha. “GARNET: A Detailed Interconnection Network Model Inside a Full-system Simulation Framework”. *CE-P08-001, Princeton University*, 2008.

[48] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi. “McPAT: an integrated power, area, timing modeling framework for multicore and manycore architectures”. *IEEE/ACM Int. Symposium on Microarchitecture (MICRO)*, 2009.

[49] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M.R. Stan. “HotSpot: a compact thermal modeling method for CMOS VLSI systems”. *IEEE Trans. Very Large Scale Integr. Syst. (TVLSI)*, 14 (5) (2006).

[50] X. Li, J. Qin, J.B. Bernstein. “Compact modeling of MOSFET wearout mechanisms for circuit-reliability simulation”. *IEEE Trans. Dev. Mater. Reliab.*, 8 (1) (2008) 98121.

[51] O. Khan, S. Kundu. “A self-adaptive system architecture to address transistor aging”. *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, 2009

[52] M. Gebhart, J. Hestness, E. Fatehi, P. Gratz, S.W. Keckler. “Running PARSection 2.1 on M5”. *Technical Report TR-09-32, The University of Texas at Austin*, 2009.

[53] S.R. Vangal et al. "An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS". *IEEE J. Solid-State Circuits*, 3 (1) (2007) 2941.

[54] S. Bell et al. "TILE64  processor: a 64-Core SoC with mesh interconnect". *IEEE SSCC*, 2008.

[55] L. Shang, L.-S. Peh, A. Kumar, N.K. Jha. "Thermal modeling, characterization and management of on-chip networks". *Int. Symp. Microarchitecture*, 2004.

[56] L. Yang, R.P. Dick, P.A. Dinda, G. Memik, X. Chen. "HAPPE: human and application driven frequency scaling for processor power efficiency". *IEEE Trans. Mobile Comput.*, (2013).

[57] C.-L. Chou, R. Marculescu. "Designing heterogeneous embedded network-on-chip platforms with users in mind". *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD)*, (2010).

# APPENDIX A. TWO LAYER ARTIFICIAL NEURAL NETWORK TRAINING ALGORITHM FOR MATLAB

```
% Neural Network Trainer

% Trains a one hidden layer neural network with variable number of
% neurons using a second-order backpropagation method.

% First, load all the training data provided. File should have
% inputs and targets
load TrainingData.mat;

% Number of hidden layer neurons
neurons = 2;

% Number of input variables
inputvars = 2;

% Number of output variables
outputs = 1;

% Weight and bias are initialized randomly
W1 = rand(neurons, inputvars);
W2 = rand(outputs, neurons);
b1 = rand(neurons, outputs);
b2 = rand(outputs);

iterations = 1; % Start at iteration 1
limit_itera = 10000;  % Interations limit

% Learning rate
alfa = 0.001;
```

```
% Initialize MSE higher than tolerance
MSE = 100;

% Initialize tolerance
tolerance = .015;

% Size of training vector
total_training_vector = 10000;

while (iterations < limit_itera) & (MSE > tolerance)

 e = 0; % Error

 W1L = zeros(neurons,2);
 W2L = zeros(1, neurons); % New weight/bias matrices
 b1L = zeros(neurons,1);
 b2L = 0;

 for i = 1:total_training_vector
 % Calculate output
 y0 = [ input(i,1);
 input(i,2);];
 y1 = tanh(W1*y0+b1);
 y2 = W2*y1+b2;

 delta2 = (y2-target(i));  % Compare to target (desired) vector

 % Backpropagate
 for j = 1:neurons
 F1(j, j) = (sech(W1(j, :)*y0+b1(j))^2);
 end
```

```
delta1 = F1*W2'*delta2;

% New weight deltas
W2L = W2L - alfa*delta2*(y1');
b2L = b2L - alfa*delta2;
W1L = W1L - alfa*delta1*(y0');
b1L = b1L - alfa*delta1;

% Error squared
e = e + (y2-target(i))^2;

vMSE(iterations+1) = e/total_training_vector;

end

% Update weights and bias
W1 = W1 + W1L;
W2 = W2 + W2L;
b1 = b1 + b1L;
b2 = b2 + b2L;
e = e/total_training_vector;
MSE = e;
iterations = iterations + 1;

% Cross validation
if vMSE(iteracoes) > vMSE(iteracoes -1)
break;
end

end
% Neural Network is trained!
```