Dependable Systems: Final Project

Lorenzo Di Tucci - lorenzo.ditucci@mail.polimi.it

A.Y 2014/2015

1 Introduction

Nowadays, one of the biggest challenges in modern multicore systems is reliability. Elevated operating temperatures and high power densities leads to an acceleration of device wear-out.
If the wear-out failures are not taken into consideration during task allocation and scheduling processes, some processors might age much faster than the others and become the reliability bottleneck for the system, significantly reducing the system's service life.[2]
It's clear that both power and thermal management techniques should be exploited in order to have a low power and reliable system on chip. Many techniques have been proposed: Dynamic Voltage Frequency Scaling (DVFS), for example, act by changing the voltage frequency, lowering the power consumption and reducing the chip average temperature resulting in improving the reliability of the system by reducing hard failures. Some power management policies decrease the average temperature by turning off some idle core improving reliability and diminishing the probability of phenomena like Electromigration (EM) and Time-Dependet-Dieletric-Breakdown (TDDB).
However, all this techniques do not concentrate on deleting the thermal stresses as Thermal Cycle (TC) because they do not consider the deteriorating impacts of TC on the overall system reliability.[3]

As Thermal Cycling (TC) we refer to the wear caused by thermal stress, resulting from mismatched coefficient of thermal expansion for adjacent material layers; run-time temperature variation results in inelastic deformation, eventually leading to failure.[4]
A thermal Cycle occurs when the temperature starts from some initial value, reaches an extreme point, and returns to the starting values. The MTTF (Mean Time To Failure) due to Thermal Cycling, strongly depends on the peak temperature and cycle amplitude. A cycle can be denoted bu a non-adjacent peak/valley pair.
An important problem to be addressed here is the method to count how many cycles there are in a set of temperatures. A proper estimation is complex, this because, double counting the number of cycles (counting them in a naive way) could carry to an underestimation of the MTTF. In contrast, if we count only the neighbouring peak/valleys we would overestimate the MTTF by 2.7x [4]. In our model, we decided to use the Rainflow Counting Algorithm in order to count the number of cycles in a profile of temperatures. We've chosen this method as it's said to be the "most widely used cycle counting method in material science and is commonly accepted as the best method for estimating the damage of a material due to random loading fluctuation.[4]

The model we are presenting here, is a framework to calculate the reliability of a system considering how thermal cycle could affect it. We considered a system level model, using a single core with a variable load. In particular we will be feeding our system with a profile of temperature and we will calculate the effect that thermal cycle has on the overall system.

This model can be used in two different scenarios, that we identify as the static and the dynamic one.The reason they are identified in this way is because the framework can give a live estimate of a working system, or it can be used offline.

1.1 Static Scenario

The first scenario considered, is the static one. In this case we are using the framework offline, and not live on a system. Here, we have all the data we need to estimate the reliability as we know that the system have a periodical trend, with a fixed schedule table that keep looping forever. This because the scenario is completely predictable at design time.

1.2 Dynamic Scenario

The second scenario is dynamic. Here we have no knowledge on how the trend will be as we have no knowledge of the future. The system is working on-line, sampling the data from a file that contains the temperatures sampled live from the system. It is clear that this scenario is not predictable at design time as we do not have all the profile of temperature. Another factor is that the workload can change during time and we need to take into account that in order to provide a good estimation.
In the following sections I will present the framework firstly in the static case and secondly in the dynamic one, it's important to divide the two scenarios as there are some differences in the calculation of the reliability.

2 The Framework

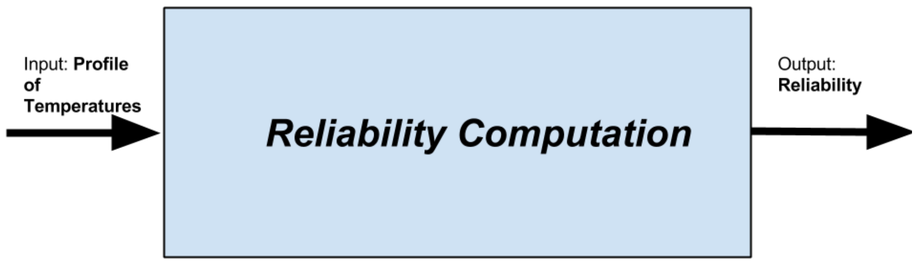


Figure 1: Framework - Overview

The Framework presented here is based on a study of the existing literature, in particular for the static scenario I found particularly fitting what is described here [7]. For what concern the dynamic scenario, I had to change part of the formulas as described in this paper [1].
The Frameworks take as input a profile of temperatures, representing the workload of the core and outputs the reliability of the system. Note that in this particular case, the system we are considering is composed of only one core, but it's trivial to adapt the framework to support systems with more than one core. The workload of the system can be generated with tools as for example HotSpot.
The computation of the result is done by three different modules that works in a cascade-style computation. In particular, the modules that identifies the three phases of the framework are the following:

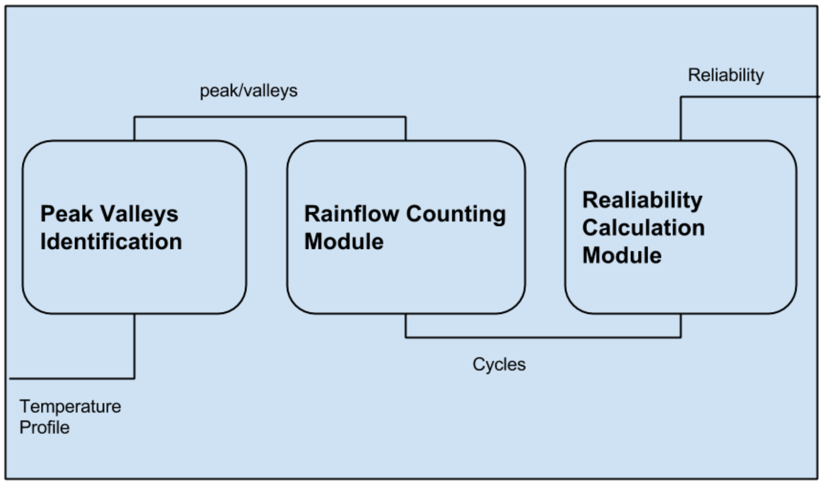


Figure 2: Framework - Zoom

- Peak/Valleys identification: This module take care of, given a trend of temperature as input, filter them identifing all the peak and valleys.
- Rainflow Counting Module : Here is applied the Rainflow Counting, algorithm used to count and identify the Temperature Cycles in the input given.
- Reliability Calculation Module : it takes the output of the previous module and calculates the final value of the reliability of the system

2.1 Static Scenario

2.1.1 Peak/Valley Identification

The module simply filter all the data that is given as input (the profile of temperature) and outputs only that values that represent a peak or a valley. This is the first step of the framework and it is very easy to understand.
The algorithm works using a 2 element window and calculates the derivative in order to understand the change of trend. When there is a change in the trend, the result of the derivative changes sign and we know that we have identified a peak or a valley.
In particular, the program consider the current value (the one that has been read) the previous one and the trend. The trend has three possible values and its current values depends on the previous iteration of the algorithm. It can be 0 if in the previous iteration we found out that the values were the same, 1 if we are searching for a valley and 2 if we are searching for a peak. The program simply compare this three values and, if the current value is greater that the previous one and we are searching for a valley or the previous values were equal,

then this means that we have found a valley in the previous value.The previous value is returned and the trend is updated to 2 as now we are searching for a peak. If, from the other hand, the current value is less than the previous one and we are searching for a peak (trend = 2) or the previous values were equals, then it means that we have found a peak in the previous value, and we return it after updating the trend to 1 (we will search for a valley). If none of the previous conditions are correct, the program checks if the values are the same, and if this is true, it just updates the trend to 0 and returns -2 that simply means that no peak nor valley is found. Note that -2 is returned also in case none of the previous case is correct.
Here is the pseudo-code of the algorithm.

Algorithm 1 find next peak valley(prevValue, value, trend)

```
1: procedure
2:     if $value > prevValue \ \& \ \& \ (trend == 0||trend == 1)$ then
3:         $trend \leftarrow 2$
4:         return prevValue
5:     if $value < prevValue \ \& \ \& \ (trend == 2||trend == 0)$ then
6:         $trend \leftarrow 1$
7:         return prevValue
8:     if $value == prevValue$ then
9:         $trend \leftarrow 0$
10:    return -2 //means no peak/valley found
```

Here we are using a 2-value window. Obviously, it is possible to achieve better results, for example trying to reduce noise. In order to do that, we could use a larger window in order to understand what is the general trend, however, this is not the goal of this framework and we decided that a 2 value window was enough.

2.1.2 Rainflow Counting Module

The output of the first module, a profile of peak and valleys, is the input of the Rainflow Counting module. This module applies the Rainflow Counting Algorithm and outputs, as a result, the thermal cycles identified. This module is very important as Thermal Cycling highly depends on the number of cycle identified by this algorithm.
The rainflow-counting algorithm is used in the analysis of fatigue data in order to reduce a spectrum of varying stress into a set of simple stress reversals. It is very important, as it allows the application of Miner's Rule, key formula to calculate the MTTF. The algorithm was developed by Tatsuo Endo and M. Matsuishi in 1968, however, in this model we are implementing the newer version of Downing and Socie.[5].

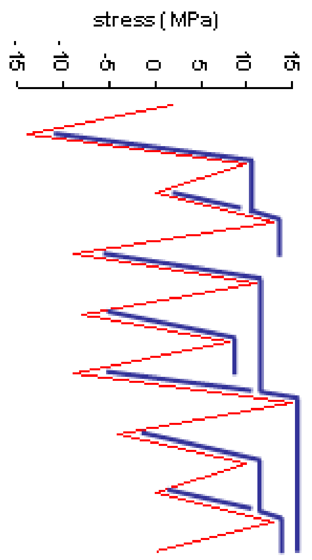
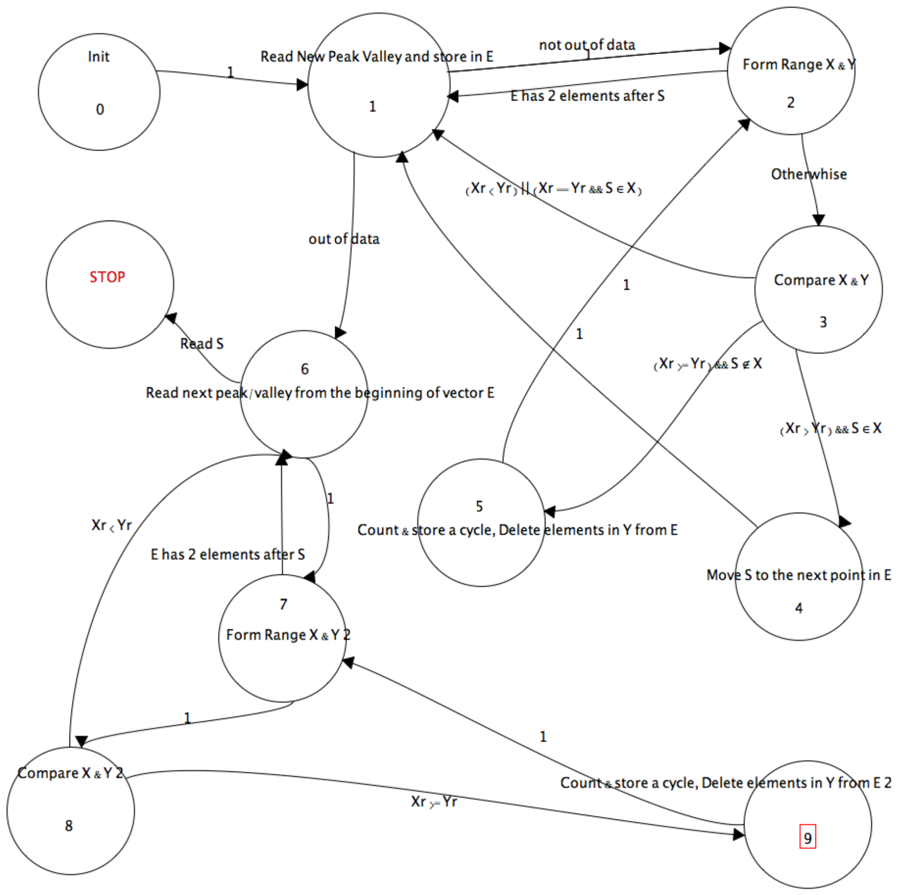


Figure 3: Example of Rainflow Counting

The Naive version of the Algorithm, works in 8 steps:[6]

1. Reduce the time history to a sequence of peaks and Valleys
2. Imagine that the time history is a template for a rigid sheet (pagoda roof)
3. Turn the sheet clockwise 90 degrees (earliest time to the top).
4. Each tensile peak is imagined as a source of water that "drips" down the pagoda.
5. Count the number of half-cycles by looking for terminations in the flow occurring when either:
    - It reaches the end of the time history;
    - It merges with a flow that started at an earlier tensile peak; or
    - It flows when an opposite tensile peak has greater magnitude.
6. Repeat step 5 for compressive valleys.
7. Assign a magnitude to each half-cycle equal to the stress difference between its start and termination.
8. Pair up half-cycles of identical magnitude (but opposite sense) to count the number of complete cycles. Typically, there are some residual half-cycles.

In the framework, we are implementing, as said before, the version of the Rainflow Counting by Downing and Socie [5]. An idea of how it works can be seen by the following Finite State Machine.



Init
0
1
Read New Peak Valley and store in E
1
not out of data
1
Form Range X & Y
2
E has 2 elements after S
Otherwhise
(Xr < Yr) || (Xr == Yr && S ∈ X)
out of data
STOP
1
1
Compare X & Y
3
Read S
6
Read next peak/valley from the beginning of vector E
(Xr >= Yr) && S ∉ X
(Xr > Yr) && S ∈ X
1
5
Count & store a cycle, Delete elements in Y from E
Xr < Yr
E has 2 elements after S
Move S to the next point in E
4
7
Form Range X & Y 2
1
1
Compare X & Y 2
8
Count & store a cycle, Delete elements in Y from E 2
Xr >= Yr
9

Figure 4: FSM: Rainflow Counting Algorithm

Where E is the vector where all the peak and valleys read are stored, S is the head of the previous vector,X and Y contains the value in position i, $i-1$ and $i-3$, finally X_r and Y_r are the range of the values considered.
At the beginning, after the initial configuration (state 0) the algorithm try to read a new peak (or valley) from state 1 and, if it actually reads a value, then move to state 2 (if not, we are out of data and moves to 6). In this state it first checks if there are 2 values after the head (S) (and in case there are not, come back to state 1), and secondly form the two ranges X and Y by simply putting the element of vector E in position i and $i-1$ into X and the elements in position $i-1$ and $i-2$ into Y. After doing this, it moves to state number 3. This state compare the ranges of X and Y and, if $(X_r < Yr)||(X_r == Yr \ \& \ \& \ S \in X)$ holds it moves to state 1, if $(X_r > Y_r) \ \& \ \& \ S \in X$ holds, it moves to state 4 where it moves the head to the next point in vector E and come back to state 1. Otherwise if $(X_r \geq Y_r) \ \& \ \& \ S \notin X$ holds, it switch to 5, count a new cycle delete the element in Y from the vector E and move to state 2.
When there are no more peak and valleys to read, the algorithm moves to state 6 where it starts reading the new peak/valley from the beginning of E. It will eventually stop when it will finish the computation after reading again the head. From state 7, the algorithm will behave like before with the only difference that a cycle will be counted only if $X_r \geq Y_r$, otherwise it will keep reading from E till it will find the head again.

This is Downing and Socie's implementation of Rainflow Counting Algorithm. It differs from the naive version as there is no needing to reorder the data, as once we are out of data we start again from the head to search for new temperature cycles. In this way we could find all the cycles by doing only one pass.

2.1.3 Reliability Calculation Model

Once we have identified all the temperature cycles (the output of the previous module) , we can calculate the reliability of our system. The formulas we used to calculate the values are the following. The calculation of the reliability starts with the calculation [7], for each Thermal Cycle, of the number of cycles to failure. This, is calculated using Coffin-Mansion's Rule:

$$N_{TC}(i) = A_{TC}(\delta T_i - T_{Th})^{-b} * e^{\frac{E_a}{KT_{max(i)}}} \tag{1}$$

Where

- $N_{TC}(i)$: number of cycles to failure due to the i_{th} thermal cycle
- A_{TC}: Empirically Determined constant
- δT_i: Amplitude of the i_{th} thermal cycle

- T_{Th}: Amplitude at which elastic deformation begins
- b: Coffin-manson exponent constant
- E_a: Activation Energy
- $T_{max}(i)$: Maximum temperature in the i_{th} thermal cycle

Note that in this paper [10] is stated that "the Coffin-Manson model can not tell the difference between temperature cycle and thermal shock" and thermal shock has more damage than Temperature Cycle. This means that using Coffin-Manson model we are overestimating the reliability of the system.

Once we have the number of cycle to failure of each thermal cycle, we need to calculate the effective number of cycles to failure for the entire system N_{TC}:

$$N_{TC} = \frac{m}{\sum_{i=1}^{m} \frac{1}{N_{TC}(i)}} \tag{2}$$

At this point we are ready to calculate the reliability of the system. As we are considering a static case, where all the temperatures are known at design time, we can calculate the Mean Time To Failure (MTTF) of the system. Note that we can do that because using the formula of the N_{TC} we are calculating the number of cycle to failure that we will project for the entire system.
The MTTF can be calculated using Miner's Rule as follows:

$$MTTF = \frac{N_{TC} \sum_{i=1}^{m} t_i}{m} \tag{3}$$

Where
t_i is the time for the i_{th} thermal cycle, m the number of thermal cycles calculated with the Rainflow Counting Algorithm and N_{TC} the effective number of cycles calculated using the previous formula.
Note that the previous formula comes from the Weibull distribution:

$$R(t) = e^{-(\frac{\tau}{\alpha(T)})^{\beta}} \tag{4}$$

2.2 Dynamic Scenario

It's also possible to use this framework in a dynamic scenario, with just a little changes. In this case, the profile of temperature is not known at design time and the workload can change during time.
The three modules of the Framework in this case work as three different threads, still in a cascade-style computation but in this scenario, the programs runs live and take the input data directly from a run-time updated file. The file is periodically sampled in order to check if new values are available. Note that in the file, is stored, at run-time, new data sampled directly from the system we are

considering.

2.2.1 Peak/Valley Identification

The framework starts as in the static scenario, in fact as the first module works by considering a 2 value window (in particular it consider the last read value and the previous one) it can works also on data that is on the fly. It will just keep reading periodically for new values and, when it founds some, it will process them identifying which of them are peaks, valley or none of the previous.

2.2.2 Rainflow Counting Module

The second module in the meantime keep checking periodically for new peaks/valleys identified by the first module and as soon as it finds out that there are new values, it process them applying the Rainflow Counting Algorithm. Note that in this dynamic case we reimplemented the Rainflow Counting Method by Downing and Socie [5] by not starting again from the beginning once the algorithm runs out of data. This is because, this scenario is not periodic and theoretically infinite so, probably, we will never be able to start over from the beginning of vector E.
It could seems that in this way we loose a considerably amount of data. Actually the amount of data we loose is around 1% as it's observable from the table below that show the amount of temperature cycles calculated by the complete version of the algorithm (that can be used in the static scenario) and the reimplemented version.

Peak/Valleys Numb.	Static Cycles	Dynamic Cycles
10 000	5000	4995
100 000	50 000	49 994
1 000 000	500 000	499 996

2.2.3 Reliability Calculation Model

The model for the third module is slightly different. Here it is not possible to calculate a average N_{TC} as the scenario is not periodic.In the static case the workload period is way smaller than the life of the system so I can estimate the average N_{TC} and use it in the Weibull formula than can be rewritten as the MTTF formula. All this things are not valid in the dynamic case. In this case I need to use a different formulation for the reliability, in particular we will be using the formula presented here [1]:

$$R(t) = e^{-\left(\sum_{j=1}^{i} \frac{\tau_j}{\alpha_j(T)}\right)^{\beta}} \tag{5}$$

where τ_i is the duration of each period of time with temperature T_j up to time t.
In our scenario the formula has been rearranged as follow:

$$R(t) = e^{-(\sum_{i=1}^{m} \frac{\delta t_i}{N_{TC}(i)})^{\beta}} \tag{6}$$

with m number of cycles identified, δt_i duration of the $i_t h$ cycle and $N_{TC}(i)$ number of cycles to failure due to the i_{th} thermal cycle.
This is the only change needed in our framework to work in the dynamic case. Periodically, the third module wakes up and check for new values from the second one. Once there are new values, it updates the value of the reliability.
The program keep running until a final value is given. At that points all threads terminate and the third module outputs the real reliability value.

3 Example

In this section we will present an example in the static case. We will give to the algorithm a dataset of 256 temperatures and we will report each step, module by module.

The input of the framework is represented in Figure 5. The picture represent the complete profile of temperature given as input. The temperature is in Kelving degrees while the time is expressed in hours. In particular, here the data has been sampled with a period of 5 seconds. Clearly, there is the possibility to change the step by simply changing the temperature when writing the input data into the input file.

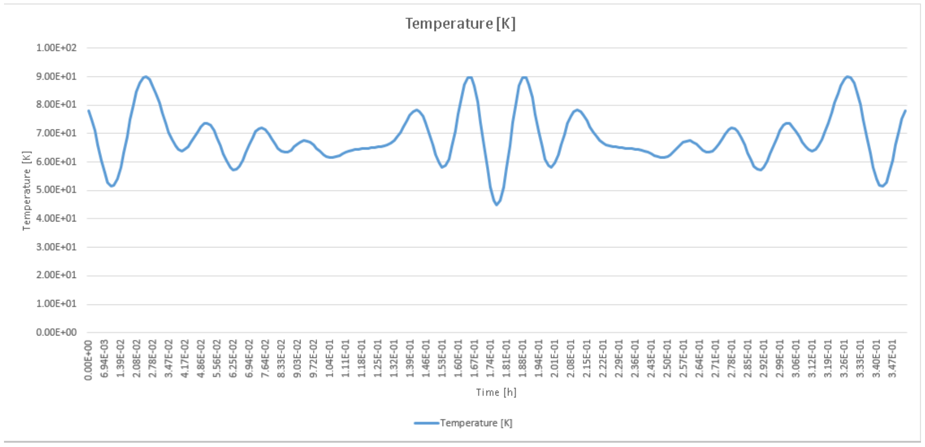


Figure 5: Rainflow Example: raw data

This data is processed and filtered from module one, whose task is to identify the peak and the valleys. For this particular input , the peak and valleys identified have been drawn on the following chart:

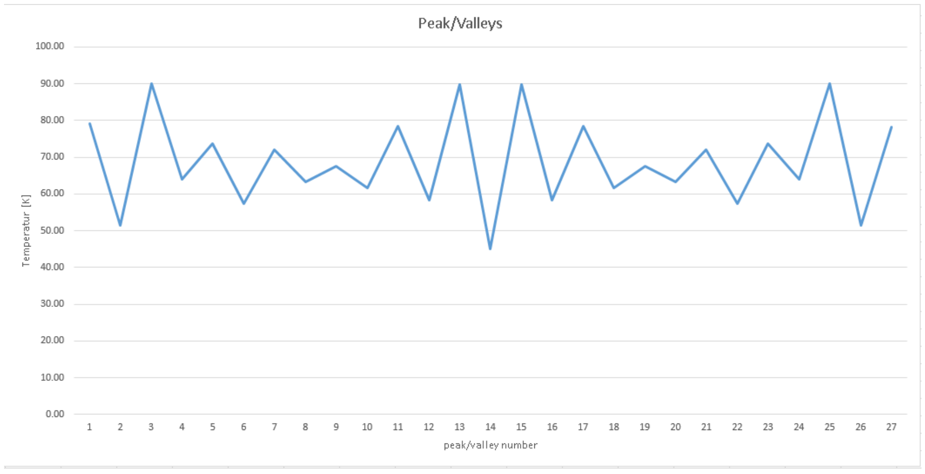


Figure 6: Rainflow Example: Peak and Valleys

It is noticeable how the amount of data is highly reduced (almost to the 10%) by the filtering module. The input dataset is a profile of 256 temperatures, but actually only 27 values are identified as peak or valleys. This reduce the amount of data that the Rainflow Counting module need to process.
This 27 values are then given as input to the second module that identifies 13 cycles, as is observable from the following table where the Range is in Kelvin degrees and the time in Hours:

Number	Range[K]	Time[h]
1	9.6	0.0417
2	4.1	0.0861
3	10.4	0.0764
4	20.2	0.0764
5	32.7	0.0264
6	4.1	0.261
7	10.4	0.25
8	20.2	0.201
9	9.6	0.303
10	32.4	0.19
11	1	0.354
12	0	0.344
13	0.302368	0.19

Of this 13 cycles, 10 have been identified during the first pass of the al-

gorithm while, the three remaining, have been identified during its last pass. This means that the Rainflow Algorithm implemented into the dynamic scenario would identify only 10 cycles.
Once we have all this results, the framework is ready to perform the latest step: the calculation of the MTTF.
In this example we tuned the parameters as follows:

- $A_{TC} = 11$
- $T_{Th} = 0$
- $b = 9$
- $E_a = 0.5$
- $k = 8.6173324 * 10^{-5}$

It's very difficult to find values for this parameters. As said in this paper [10] the most difficult parameter to determine is E_a, because is specifically related to certain failure mechanisms and failure modes, however it can be determined by test.
The parameters we used here have been taken from this paper [8] and [9]. We've chosen to tune the Coffin-Mansion exponent to 9 as in this paper [9] is said to tune this parameters between 6 and 9 in case of material with "brittle fracture (e.g. Si and dielectrics: SiO2, Si3N4) ". Note that we should really focus on the choice of this parameter as it can considerably change the final result, leading to wrong data.
The T_{Th} represents the portion of the temperature range in the elastic region. If this value is much smaller than the entire temperature cycle range (δT_i) it can be dropped without significant error being introduced and, in this paper is said to be an usual practice. For this reason its value has been set to 0.
The value of the k is the Boltzman constant while the value of the activation energy E_a is taken from this other paper [8]. Note that in this last paper, they set the coefficient of proportionality A to 1 because they say is not significant if you are interested in the relative improvement.

As said before, is not easy to find the parameters of the Coffin-Manson model, however , several estimate have been proposed in the literature [11]. In the work of this paper [11] is presented an extensive statistical evaluation of the existing parameter estimate as well as a new estimation method that can be used to find new values for this parameters.

Given all this parameters, the final output of the framework is:
$MTTF = 136301$ hours, equivalent to 15.5 years.

4 Conclusions

In this project we presented a framework to estimate the reliability of a system (that for simplicity has been considered having only one core) considering a variable load and how the Thermal Cycles affect its wear-out. The Framework can works both in a dynamic and a static scenario and can be easily adapted to work on more complex systems, composed of more than a single core.

References

[1] Cristiana Bolchini, Matteo Carminati, Marco Gribaudo, Antonio Miele - Dipartimento di Elettronica, Informazione e Bioingegneria -Politecnico di Milano, Italy *A Lightweight and Open-source Framework for the Lifetime Estimation of Multicore Systems*

[2] Lin Huang, Feng Yuan and Qiang Xu - CUhk Reliable computing laboratory, The Chinese University of Hong Kong, Shatinm N.T., Hong Kong *Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms.*

[3] Mehdi Kamal, Arman Iranfar, Ali Afzali-Kusha, Massoud Pedram - University of Tehran, Iran *A Thermal Stress-Aware Algorithm for Power and Temperature Management of MPSoCs - 2015*

[4] Yun Xiang, Thidapat Chantem, Robert P. Dick, X. Sharon Hu, Li Shang - University of Michigan, University of Notre Dame, University of Colorado *System-Level Reliability Modeling for MPSoCs*

[5] S.D. Downing, D.F.Socie *Simple Rainflow Counting Algorithm, 1982*

[6] https://en.wikipedia.org

[7] Anup Das, Rishad A. Shafik, Geoff V. Merrett, Bashir M. Al-Hashimi, Akash Kumar, Bharadwaj Veeravalli - University of Singapore and University of Southampton *Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems*

[8] Ivan Ukhov, Min Bao, Petru Eles, Zebo Peng - Linkoping University- Sweden *Steady-State Dynamic Temperature Analysis and Reliability Optimization for Embedded Multiprocessor Systems*

[9] *Failure Mechanisms and Models for Semiconductor Devices - JEP122F - November 2010*

[10] Helen Cui, RF Micro Devices *Accelerated Temperature Cycle Test and Coffin-Manson Model for Electronic Packaging*

[11] Marco Antonio Meggiolaro,Jaime Tupiassu, Pinho de Castr - Mechanical Engineering Department - Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil *A Critical Evaluation of Fatigue Crack Initiation Parameter Estimates*