Application source code modification for processor architecture lifetime improvement

Montassar Ben Saad*

National School of Engineering,
University of Sfax,
Sfax 3038, Tunisia
Email: montassar.bs@gmail.com
*Corresponding author

Ahmed Jedidi

College of Engineering,
Ahlia University, Bahrain
and
National School of Engineering,
University of Sfax, Tunisia
Email: ajedidi@ahlia.edu.bh

Smail Niar

LAMIH,
University of Valenciennes and Hainaut-Cambrésis,
59300 Valenciennes, France
Email: smail.niar@univ-valenciennes.fr

Mohamed Abid

National School of Engineering,
University of Sfax,
Sfax 3038, Tunisia
Email: mohamed.abid_ces@yahoo.fr

Abstract: In the optimal functioning of SoCs, two significant metrics of quality are the most important; life time and reliability. The context of this paper focuses on methods to increase the lifetime of a processor. Two methods are presented; relax point injection (RPI) and code structure adaptation (CSA). In RPI, a specific treatment is incorporated into the application code to prevent a harmful rise in the temperature of the chip. The MTTF of the processor is increased by 33.88% through means of an RPI method. However, the execution time of the application is sometimes increased by the RPI to a higher than 12%. In CSA method, the arrangement of the application code is regulated to improve the lifetime of the processor. The MTTF of the processor is increased up to 28% by CSA technique and the implementation time is maintained.

Keywords: mean time to failure; thermal dissipation; relax point injection; code structure adaptation.

Reference to this paper should be made as follows: Ben Saad, M., Jedidi, A., Niar, S. and Abid, M. (xxxx) 'Application source code modification for processor architecture lifetime improvement', *Int. J. Embedded Systems*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Montssar Ben Saad. is currently PhD student and member in the Laboratory Computer and Embedded Systems University of Sfax, Tunisia. He received his Master degree in Intelligent and Communicating Systems in 2010 at National Engineering School of Sousse, University of Sousse, Tunisia. His current research interests are monitoring, exploration and optimisation of the thermal dissipation and improving lifetime in SoC.

Ahmed Jedidi received his PhD in Computer Engineering from the University of Sfax in 2012. Currently, he is a Chairperson/Assistant Professor in College of Engineering Ahlia University Bahrain. Also he is a member of the laboratory Computer and Embedded Systems (CES Laboratory), Sfax University, Tunisia. His research interests are in embedded systems performance, FPGA and optical network communication.

Smail Niar received his PhD in Computer Engineering from the University of Lille in 1990. Since then, he has been a Professor at the University of Valenciennes where leads the mobile and embedded systems research group at the Laboratory of Automation, Mechanical and Computer Engineering, a joint research unit between CNRS and the university of Valenciennes. He co-supervises the 'intelligent infrastructures and vehicles' task within the International Campus on Safety and Inter-modality in Transportation (CISIT). He is a member of the European Network of Excellence on High Performance and Embedded Architectures and Compilation (HIPEAC) and EuroMicro. He is an IEEE senior member. His research interests are in multi-core architectures, heterogeneous and reconfigurable embedded systems, design space exploration, and reliability issues for embedded and mobile systems in general and in transportation systems (automotive) in particular.

Mohamed Abid is a Professor at the Engineering National School of Sfax (ENIS), University of Sfax, Tunisia. He received his PhD degree from the National Institute of Applied Sciences, Toulouse (France) in 1989 and the Thèse d'état degree from the National School of Engineering of Tunis (Tunisia) in 2000 in the area of computer engineering and microelectronics. He is founding member and Head of the research laboratory 'Computer Embedded System' CESENIS since 2006. He occupied the post of Director of Doctoral School Sciences and Technologies, ENIS-University of Sfax (2009–2013) and the post of Doctoral degree Computer System Engineering, ENIS – University of Sfax (2003–2010, and since 2014). He is author or coauthor of more than 115 publications in journals, than 300 papers in international conferences and joint author of more than 15 book's chapters. He was a supervisor or co-supervisor of 48 PhD defended, some of them are in cooperation.

1 Introduction

In the previous system-on-chip (SoC) generation, there was a greater power density due to the increase in complexity and reduced size of the feature. At the same time, power is becoming the major limitation of the system design. However, the power measurement and management techniques have become a necessity. In this context, to measure the power related in program behaviour, Chunling et al. (2007) presented an infrastructure based on simulation and physical measurement to correlate between instructions-per-cycle and power dissipation for the each piece of code. On the other hand, Ayala et al. (2007) proposed a hardware/software approach to reduce the power of shared register files in embedded VLIW processors. This work is relies on a set of special hardware extensions that are controlled by the compilers of the embedded platforms. The results obtained show that this approach can reduce the power up to 60% without any performance degradation.

The high power density results in the development of temperature hotspots, consequently lead to ageing in a variable manner, and a rise in the malfunction of the chip. Indeed, these hotspots can cause some processor errors. However, the processor errors can be classified in two categories. First, soft errors due to electrical noise or external radiation. These errors do not fundamentally damage the microprocessor and are not viewed as a long-term reliability concern. Second, Hard errors are caused by defects in the silicon or metallisation of the processor package. They will result in permanent processor failure. Then, hard errors directly determine long-term processor reliability. Furthermore, Hard failures can be divided into extrinsic failures and intrinsic failures (Pecht et al., 1999). Indeed, extrinsic failures are caused by process and manufacturing defects. Such as, contaminants on the crystalline silicon surface and surface roughness which can cause adielectric breakdown (Srinivasan et al., 2004). Other extrinsic failures include short circuits and open circuits in the interconnects due to incorrect metallisation. Intrinsic failures, however, depend on the processor's materials, related to the processor wear-out. Some examples of intrinsic failures include time dependent dielectric breakdown (TDDB) in the gate oxides, electromigration and stress migration in the interconnects, and thermal cycling. We will describe these errors in the section 3.2.4 with the RAMP model, which is the only processor models intrinsic failures. However, not only the functioning of the system is impacted through temperature gradients and hotspots, but the processing of the circuit also becomes unpredictable and the lifetime of the chip is also impacted. As a result, efforts are being made to improve the chip lifetime and prevent harm caused by increased temperatures. In fact, thermal monitoring and management, and lifetime improvement techniques are deployed. For example, the dynamic thermal monitoring methods adopted by Intel and AMD, which conduct the corrective actions required to regulate the temperature of the on-chip (McGowen et al., 2006). In many cases, the corrective action power off the system or lower the voltage and frequency of the system, resulting in lower performance efficiency.

In this paper, we propose the techniques of relax point injection (RPI) and code structure adaptation (CSA) to improve the lifetime of the microprocessor. The cost-efficient techniques adopted for the modification of application code and regulations of microprocessor performance are of two types. The first one includes the implantation of a specific treatment by the RPI into a particular location in the application code to prevent damage of the microprocessor by rising in the temperature. The second, the arrangement of the application code is enhanced by CSA to determine the most efficient code which results in processor main time to failure (MTTF) increase. The remaining paper is divided into the following parts. Section 2 describes the work associated. In Section 3, we define the tools of profiling and simulation utilised for processor MTTF and thermal profile calculation and approximation. In Section 4 we will present a relax point injection (RPI) technique to improve processor MTTF and we will analyse the results. And in Section 5 the code structure adapter (CSA) technique will be presented and explained. Additionally, we will propose a simulation process based on the Pilot Application (PApp) to determine the effectiveness of our techniques and the results will be assessed. In the end, a conclusion and a work for the future will be discussed.

2 Related works

Higher temperature makes the processor much more vulnerable to various failure mechanisms such as electromigration, stress migration and dielectric breakdown (Blish and Durrant, 2000). In fact, a 10°C–15°C increase in temperature can reduce the mean time to failure of a device by half (Viswanath et al., 2000). Further, monitoring temperature and reducing hotspots are critical for achieving reliable and efficient operation of complex systems on a chip. In this context, to evaluate and control high temperature in processor, several methods and techniques are deployed. We divided this section into two parts. In the first one, we mentioned some temperature modelling techniques. In the second one, we described different thermal management and reliability improvement techniques.

2.1 Thermal modelling

To evaluate thermal variations on chip, the circuit designers insert within chip thermal sensors that acquire temperatures at few selected locations. The acquired temperatures are then used to guide runtime thermal management techniques. In this context (Nowroz et al., 2010), temperature is characterised by signals of real processors and devise thermal sensor allocation techniques, and devise signal reconstruction techniques that fully characterise the thermal status of the processor using the limited number of measurements from the thermal sensors. For modelling a temperature in chip, several methods and techniques are deployed. First, works in Yi-Kan and Sung-Mo (2000), Torki and Ciontu (2002), Rencz et al. (2000), Vladimir et al. (1997), Koval and Farmaga (1994) and Batty et al. (2002) are focus in describing the techniques for modelling localised heating within a chip due to different power densities of various blocks, but none of these tools are easily adapted to architectural exploration. Skadron et al. (2002) proposed a simple model for tracking temperature on a perunit level, but it ignored the effect of lateral heat diffusion. The analytical model in Michaud et al. (2005) is based on an explicit solution to the heat equation. Moreover, some thermal models are intended to be used at the full-system level. Indeed Heath et al. (2006) proposes a system-level temperature emulation suite that uses offline calibration and the online update of per-component utilisation information to calculate the temperature of the systems. Choi et al. (2007) employs computational fluid dynamic (CFD) modelling of rack-mounted servers. Although, both tools are considered the microprocessor yet there is only one component within the system and hence no localised information is obtained, which is essential for architectural studies.

2.2 Thermal management and reliability improvement techniques

2.2.1 Thermal management techniques

Techniques to overcome the thermal problems became necessary, among these, the dynamic thermal management techniques (DTM). DTM measure a microprocessor temperature either directly by the circuit sensors or indirectly with the performance analysers, and these measures will be used to modify the microprocessor configuration parameters in order to maintain its temperature below a given threshold (Kong et al., 2012). In addition, multiple thermal control methods for microprocessors and shows trade-offs between temperature profile, frequency settings, power consumption and implementation complexity are proposed in Zanini et al. (2013). Design-time thermal optimisation techniques for embedded systems are proposed in Liu et al. (2007). This technique can be used in the system design phase. Furthermore, Intel and AMD, the leading microprocessor vendors, had dynamic thermal monitoring techniques that took necessary corrective action to maintain on-chip temperature (Blish and Durrant, 2000). Unfortunately the corrective actions, in most cases, shut down the system or reduced system voltage and frequency, leading to considerable performance degradation. In other hand, software can also play an important role in identifying and eliminating thermal hotspots. This is particularly true for compiler-scheduled very long instruction word (VLIW) data paths. Indeed, Mutyam et al. (2006) has focused on a compiler-based approach to make the thermal profile more balanced in the integer functional units of VLIW architectures. For balanced thermal behaviour and peak temperature minimisation, the author in Mutyam et al. (2006), propose a technique based on load balancing across the integer functional units with or without rotation of functional unit usage. Also, while traditional task scheduling techniques have focused on performance improvement, without regardless to temperature issues, modern techniques, such as proposed in Kumar et al. (2006), managed the temperature through the software-hardware cooperation. Extensive research, like Buyuktosunoglu et al. (2003), Hughes et al. (2001) and Srinivasan and Adve (2003) has gone into techniques that can maximise energy and thermal performance by exploiting architectural features and adaptation capabilities.

2.2.2 Reliability and lifetime improvement techniques

The higher temperature, performance, energy, and lifetime reliability of processor are directly related. In fact, many thermal managing and reliability techniques are highlighted. In this context, a thermal management technique classification, mainly in temperature monitoring and thermal reliability/security is suggested in Kong et al. (2012). The first one, requirement for dynamic thermal management (DTM), included temperature estimation and sensor placement techniques for accurate temperature measurement or estimation. The second one dealt with the problems of temperature-dependent reliability modelling, dynamic reliability management (DRM), and malicious codes that specifically cause overheating. More, Coskun et al. (2009) presented a framework for evaluating the effectiveness of a number of mechanisms of thermal control (job scheduling, job migration, dynamic voltage and frequency scaling) in various combinations, and it presented effective new policies for managing thermal effects. However, the authors in Coskun et al. (2009) show that the techniques that are nearly identical in performance, power, and even peak temperature can differ by a factor of two in an expected processor lifetime, with a performance cost of less than 4%. The authors in Song et al. (2016), study the lifetime reliability consequences of heterogeneous multicore processors. They present the lifetime reliability theoretical models of multicore processors based on Amdahls Law, including compact thermal estimation. Particularly, this work shows that there's a strong correlation between heterogeneous multicore processors and device aging. However the authors in Viswanathan et al. (2009) analysed the impact of reliable overclocking on on-chip temperature. Reliable overclocking, on an average, achieves 35% increase in performance over a non-overclocked system. Although reliable overclocking mechanisms facilitated improved performance, but a major hurdle in realising them was their impact on on-chip temperature. As the systems operated faster, on-chip temperatures quickly reached and exceeded the safe limits. This led to system crash and caused a serious threat to the lifetime reliability of the system. When a thermal throttle was applied, the performance drops by 25%. Other, a non-overclocked system has a longer lifetime, of about 30 years, as its onchip temperature does not exceed 347K. However, a reliably overclocked system has a much shorter lifetime of about 9 years (Viswanathan et al., 2009).

To improve multiprocessor system reliability, Jun et al. (2014) proposed some novel static fault-tolerant scheduling techniques. Indeed, the goal of fault-tolerance is to avoid the failure of the overall system when some of its subsystems fail. In fact, Jun et al. (2014) studied a static scheduling algorithm and propose an ILP formulation for optimally fault-tolerant scheduling. Buyuktosunoglu et al. (2003) demonstrated that simply balancing power consumption and decreasing maximal on-chip temperature are not sufficient to significantly improve the processor lifetime reliability. However, to improve the processor reliability, Buyuktosunoglu et al. (2003) proposes that a processor core should be set to different frequency according to its power consumption. Further, Basoglu et al. (2010) proposes a technique to model NBTI degradation with dynamic changes in temperature, voltage, and frequency. Based on this model, the authors utilise this knowledge of the guard band and a predictive model to absolutely improve processor power consumption and lifetime without impacting the processor performance against negative bias temperature Instability (NBTI) degradation. By this approach the authors improve the lifetime of 8-core processor at the 45 nm technology by two years and saves up to 16% of the dynamic energy consumed. Also, the authors in William et al. (2015) present a lifetime reliability characterisation of many-core processors based on a full-system simulation of integrated microarchitecture, power, thermal, and reliability models. This work present two variance reduction technique for proactive reliability management: proportional dynamic voltage-frequency scaling (DVFS) and coordinated thread swapping. Additionally, dynamic reliability management solutions are often adopted in multi-core systems to mitigate aging and wear-out effects. The authors in Bolchini et al. (2016) analyse the effects on reliability of a set of classical policies, on a multi-core architecture, by systematically varying the related parameters, such as the number of spare cores to be selected, in the whole value space. The reported experimental results of this work show that the peculiarities of both the architecture and the workload have to be taken into account in the selection of the most proper runtime policy.

3 Simulation tools

In recent years, lifetime and reliability become important quality metrics in the high-performance SoCs. In order to estimate and increase the lifetime (MTTF) of microprocessor, we propose a profiling and simulation tools to understand the behaviour of application code, and estimate the thermal profile and MTTF of the microprocessor.

3.1 Profiling tool

Usually, to optimise its application, a programmer is interested to know the evolution and the nature of the code and mapping with different parameters. For that, profiling tools and analysis are extremely important for understanding the behaviour of the program. Valgrind is a suit of a debugging and profiling tool. In our method, we used the profiler callgrind (Nethercote et al., 2006) of Valgrind tools. It records the call history among functions in programs run and the collected data consisting in the number of instructions executed, their relationship to source lines, the caller/callee relationship between functions, and numbers of such calls (Weidendorfer et al., 2004). It can produce a callgraph of fonctions.

3.2 *Thermal and MTTF estimation tools*

In this section, we present our tool to assess and calculate MTTF in SoC. Our tool represents the combination between the power model from the Wattch simulator (Tiwari et al., 2000), the thermal model of the HotSpot simulator (Stan et al., 2003) and the lifetime reliability model from RAMP (Srinivasan et al., 2004) as shown in Figure 1. The target at processor architecture in our simulation tools is the third generation superscalar Alpha 21264 Ev6 microprocessor.

3.2.1 *Alpha 21264 microprocessor*

The Alpha 21264 Ev6 is the third generation superscalar Alpha microprocessor with out of-order and speculative execution. They are used for performance optimisation. The instructions of Alpha EV6 contain integrated and floating point instructions, which can be classified into arithmetic, comparison, bit-level, load and store, conditional move, branch, and conversion classes (Kessler et al., 1998). The Alpha Ev6 also includes a high-bandwidth memory system to quickly deliver data values to the execution core, providing robust performance for many applications, including those without cache locality. Figure 2 shows that the Alpha EV6 functional units are represented by eighteen blocks in the Ev6 floorplan. With his functional units and his dynamic execution techniques, Alpha 21264 is 50% to 200% faster than its predecessor for many applications.

3.2.2 *Power model of Wattch simulator*

Wattch () is an accurate, architecture level power tool embedded within the SimpleScalar simulator. SimpleScalar is a set of tools that model a virtual computer system with CPU, Cache and Memory Hierarchy. The power model keeps track of which units per cycle and records the total energy consumed for an application. Indeed Wattch calculates instantaneous power at every cycle, and outputs the total power accumulated over a simulated period of time and the average power. Wattch uses a modified version of simplescalar sim-outorder to collect results (Tiwari et al., 2000). Wattch has three power information type cc1, cc2, cc3. They are built from the architectural functional unit power information.

In our work we use a third one cc3 to calculate the Alpha 21264 microprocessor power for each application. Or, cc3 is non-ideal conditional clocking. It shows models power leakage by assuming that an idle unit consumes only 10% of its maximum power for a cycle in which it is inactive (Tiwari et al., 2000). It is the most real. The instantaneous power traces provided by the Wattch power tool is used to calculate the temperature in HotSpot simulator.

3.2.3 *Thermal model of HotSpot simulator*

HotSpot is an architecture level simulator, fast enough to allow for the simulation of long dynamic temperature traces on the order of seconds, it's designed to calculate temperature profiles which are accurate for the experiments at the architecture level. It is one of the thermal simulators widely used in the computer architecture community. It is based on an equivalent circuit of thermal resistances and capacitances that correspond to microarchitecture blocks and essential aspects of the thermal package (Stan et al., 2003).

Basically, HotSpot aims to evaluate the application thermal effect on the microprocessor. In that, it can be integrated with a power/performance architectural simulators, like Wattch and SimpleScalar, to obtain the power information related to each application. To convert power into temperature of each functional unit HotSpot proceeds in two stages. In the first one, the conversion is performed until the temperature becomes stable. In the second one, the temperature increases by continuously converting the power into temperature based on the stable temperature.

Figure 1 MTTF and thermal simulation tool (see online version for colours)

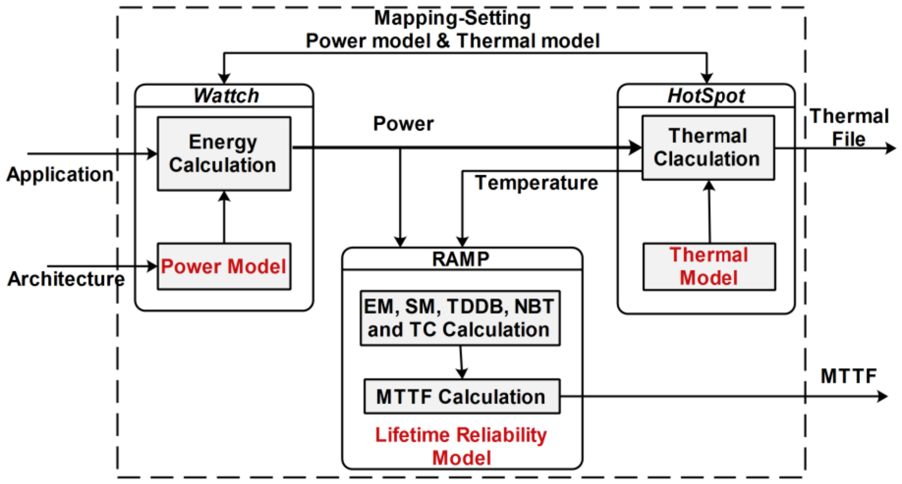
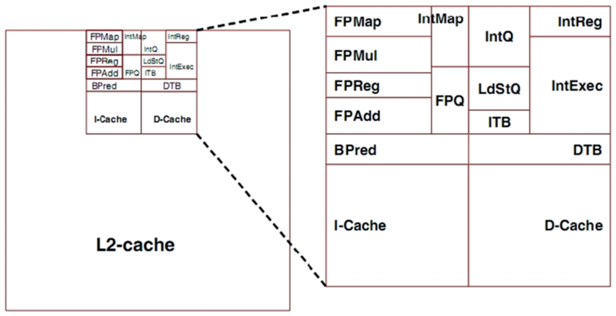


Figure 2 Alpha 21264 Ev6 processor floorplan (see online version for colours)



3.2.4 Reliability model of RAMP

Reliability-aware microprocessors (RAMP), developed by researchers at the IBM T.J. Watson Research Center and the University of Illinois Urbana-Champaign, is the first application-aware architecture-level methodology that use the analytic models for important failure mechanisms to evaluate the microprocessor lifetime and reliability (Srinivasan et al., 2004). It is based on the current temperature, utilisation, and power profile of microprocessor, provided by the power and thermal simulators, to calculate its lifetime. The MTTF represents a microprocessors' expected average lifetime and it is based on five processor failure mechanisms:

- Electromigration (EM): Occurs in aluminum and copper interconnects due to the mass transport of conductor metal atoms in the interconnects. Electromigration has an exponential dependence on temperature. The MTTF associated at this mechanism is described by the Black model.

$$MTTF = A\left(J - J_{crit}\right)^{-n} e^{\left(\frac{Ea}{KT}\right)} \tag{1}$$

  where A is a constant, J is the current density in the interconnect, Jcrit is the critical current density required for electromigration, Ea is the activation energy for electromigration, K is Boltzmann's constant, T is absolute temperature in Kelvin, and n is an empirical constant (Srinivasan et al., 2004).

- Stress migration (SM): Similar to electromigration, stress migration is a phenomenon where the metal atoms in the interconnects migrate. It is caused by mechanical stress due to differing thermal expansion rates of different materials in the device.

$$MTTF = A\left|T_0 - T\right|^{-n} e^{\left(\frac{Ea}{KT}\right)} \tag{2}$$

  where T_0 is the stress free temperature of the metal (metal deposition temperature), and T is the operating temperature (Srinivasan et al., 2004).

- Time-dependent dielectric breakdown (TDDB): is the gate dielectric's gradual breakdown which leads to transistor failure.

$$MTTF = \left(\frac{1}{V}\right)^{(a+bT)} e^{\left(\frac{X+\frac{Y}{T}+Z}{kT}\right)} \tag{3}$$

  where a, b, X, Y and Z are fitting parameters, and k is Boltzmann's constant (Srinivasan et al., 2004).

- Negative-bias temperature instability (NBTI): an electrochemical reaction that upwards transistor threshold voltage which in return leads to processor failure because of timing constraint violations.

$$MTTF = \left[\left(\ln\left(\frac{A}{1+2e^{\left(\frac{B}{KT}\right)}}\right) - \ln\left(\frac{A}{1+2e^{\left(\frac{B}{KT}\right)}} - C\right)\right) \times \left(\frac{t}{e^{-\left(\frac{D}{KT}\right)}}\right)\right]^{\frac{1}{\beta}} \tag{4}$$

  where A, B, C, D, and β are fitting parameters, and k the Boltzmann's constant (Srinivasan et al., 2004).

- Thermal and cycling (TC): it is permanent damage accumulated every time there is a cycle in temperature, eventually leading to failure. There are two types of TC; large cycles which occur at a low frequency because of changes like powering up and down, and small cycles which occur at a high frequency because of changes in work-load.

$$MTTF = \frac{A}{\left(T - T_{ambient}\right)^q} \tag{5}$$

  where T is the average temperature of the structure, Tambient is the ambient temperature and q is the Coffin-Manson exponent (Srinivasan et al., 2004).

The MTTF of the microprocessor, named here MTTFp, is the inverse of the total failure rate of the microprocessor named λ. It is the sum of the failure rates of the individual structures due to individual failure mechanisms.

$$MTTF_p = \frac{1}{\lambda} = \frac{1}{\sum_{i=1}^{j} \sum_{l=1}^{k} \lambda_{il}} \tag{6}$$

where λ_{il} the failure rate of the i^{th} structure due to the l^{th} failure mechanism (Srinivasan et al., 2004).

3.3 Simulation frameworks

To configure the three simulators, we have to set the different models with the Alpha 21264 EV6 configuration parameters. In fact, we have to initialise Lifetime Reliability Model with the different EV6 floorplan functional unit to calculate failures in each one. Hotspot uses the Alpha 21264 Ev6 floorplan that is different from Wattch and it has eighteen functional units in floorplan. In Wattch some functional units have just the same single value and some units include the operations of integer and floating point.

Indeed, we use the power transferable formula in Wattch to correspond the same floorplan in Hotspot as shown in Table 1.

Table 1 Power formula in hotspot

Alpha EV6 functional blocks (HotSpot)	*Functional blocks formula (Wattch)*
L2	0.72 ∗ dcache2_power/cycle_count
L2_left	0.14 ∗ dcache2_power/cycle_count
L2_right	0.14 ∗ dcache2_power/cycle_count
Icache	Icache_power/cycle_count
Dcache	Dcache_power/cycle_count
IntExec	Ialu_power/cycle_count
DTB	Dtlb_power/cycle_count
Bpred	Bpred_power/cycle_count
FPMul	0.52 ∗ falu_power/cycle_count
FPAdd	0.48 ∗ falu_power/cycle_count
IntQ	0.55 ∗ window_power/cycle_count
FPQ	0.45 ∗ window_power/cycle_count
FPMap	0.53 ∗ rename_power/cycle_count
IntMap	0.47 ∗ rename_power/cycle_count
LdStQ	lsq_power/cycle_count
IntReg	0.59 ∗ regfile_power/cycle_count
FPReg	0.41 ∗ regfile_power/cycle_count
ITB	Itlb_power/cycle count

The power model in Wattch and the RAMP model are built by cycle. For every cycle, RAMP and Wattch will collect and compute failure mechanisms and power consumption for each Alpha Ev6 functional units. Moreover, the input of Hotspot consists of the unit power of functional unit and the increasing temperature of the cycle is very low. That's why to synchronise the models values exchange; we decided to collect all the values of each block every 10,000 clock cycles.

4 Relax point injection

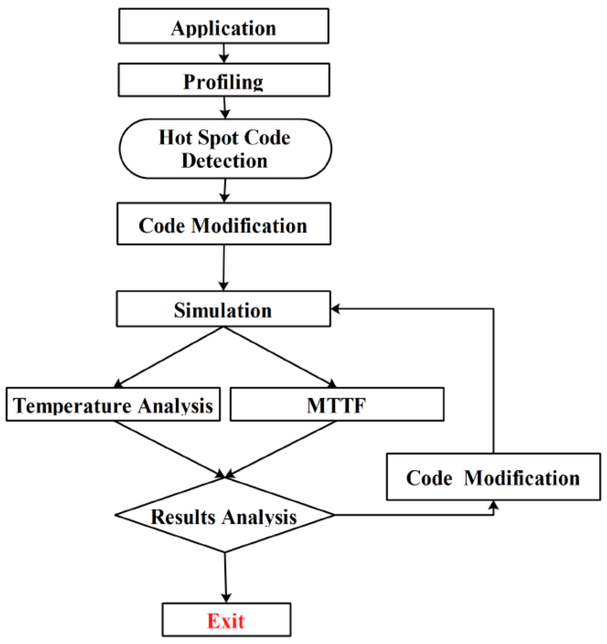
The consequences of drastic temperature increase not only deteriorate the system efficiency, but also shorter the life time of the processor. Keeping this in view, there were various procedures developed for the maintenance of onchip temperature. To solve this problem, we used a software solution to increase processor lifetime and to maintain its temperature. This solution does not have an adverse effect on the processor's efficiency. Yet it bargains between the performance and MTTF by an application code.

4.1 Methodology

Our technique goal is to modify (precisely adjust) the application code to improve the microprocessor lifetime. The flow chart in Figure 3 displays the fundamental concept of this process.

Initially, the selected application is profiled with the aim of requiring some code information's and searching a hot spot code. This part of the code is the section which contains a large number of instructions and it is also the cause of temperature increase in the system. Following the first step, a hot spot code is then altered for an increment in the MTTF. The alterations lead to the simulation of a new code, which is then tested for observation. Finally, if the aims don't meet, the hot spot code is further altered success. It is observed that the modification procedure follows various paths. In a particular section of the code, a loop that increases a counter is introduced using relax point injection (RPI). The temperature of the processor and its resources is directly influenced by RPI which affects its MTTF. One step that is crucial to this process is the determination of the place of insertion of RPI in the code. In order to ensure proper insertion, the data gathered from profiling and the quantity of instructions and their execution time are noted.

Figure 3 Profiling and modifying the application code to improve the processor lifetime (see online version for colours)



The hot spot code affects the CPU by increasing its temperature and thereby influencing the MTTF. It is observed that the processor undergoes a temperature increase more rapidly than a decrease. In order to avoid that, the RPI insertion should be carried out prior to each target code in the entire application code.

The total rate of application instruction divided by 3 is called RPIsize and a characteristic feature of RPI. It can be calculated using the following equation:

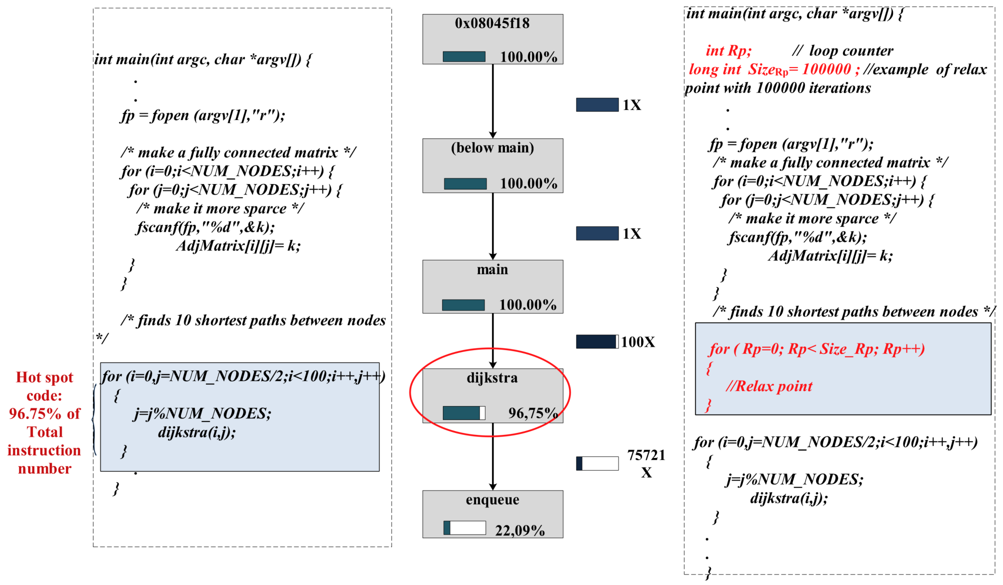
$$RPIsize = \frac{TNI \times InsR}{3} \qquad (7)$$

The total number of the application instruction is denoted by TNI, and the rate of instruction to be added is represented by InsR. The division by 3 is carried out due to the fact that every loop iteration contains three instructions. The example given in Figure 4 can be used to further elaborate this technique. An outline is created of the benchmarks of Dijkstra Mibench (Guthaus et al., 2001) with a callgrind profiler (Nethercote et al., 2006). After this, the part of code having the greatest application instruction number is identified. The dijkstra (i, j) function, described in Figure 4(a), was termed in the loop, and in Figure 4(b) is constituted by 96.75% of the overall instructions on the application. RPI inserted in the target is following this step. The RPI in the given example was found to have an RPIsize 10,000 loop iteration, as described in Figure 4(c).

4.2 Experimentation results

The assessment of the RPI technique through MTTF is the aim of this experiment. To achieve this goal, one loop counter incrementing is inserted and its size has increased by three times (*InsR = 5% then 10% then 20%)* in all of the mibench benchmarks i.e. *Basicmath, Qsort, Adpcm_encode, Adpcm_decode, Dijkstra, FFT and Rijndeal* to obtain various forms of these (*version InsR 5%. InsR 10% and InsR 20%*). We analysed the variations of MTTF, Maximum Temperature, CPI and Cycle Number metrics compared to the RPIsize for each benchmarks as shown in the Figure 5 and Table 2.

Figure 4 The Dijkstra Code example, (a) profiling and target code detection (b) kcachegrind call graph (c) relaxing point injection in Dijkstra code (see online version for colours)



```
int main(int argc, char *argv[]) {
      .
      .
   fp = fopen (argv[1],"r");

   /* make a fully connected matrix */
   for (i=0;i<NUM_NODES;i++) {
    for (j=0;j<NUM_NODES;j++) {
      /* make it more sparce */
      fscanf(fp,"%d",&k);
           AdjMatrix[i][j]= k;
    }
   }

   /* finds 10 shortest paths between nodes
*/

 for (i=0,j=NUM_NODES/2;i<100;i++,j++)
   {
      j=j%NUM_NODES;
         dijkstra(i,j);
   }
      .
   }
```

```
int main(int argc, char *argv[]) {

   int Rp;        // loop counter
 long int SizeRp= 100000 ; //example of relax
point with 100000 iterations
      .
      .
   fp = fopen (argv[1],"r");
    /* make a fully connected matrix */
    for (i=0;i<NUM_NODES;i++) {
     for (j=0;j<NUM_NODES;j++) {
       /* make it more sparce */
       fscanf(fp,"%d",&k);
            AdjMatrix[i][j]= k;
     }
    }
    /* finds 10 shortest paths between nodes */
   for ( Rp=0; Rp< Size_Rp; Rp++)
   {
      //Relax point
   }

 for (i=0,j=NUM_NODES/2;i<100;i++,j++)
   {
      j=j%NUM_NODES;
         dijkstra(i,j);
   }
   .
   .
   }
```

(a) (b) (c)

Figure 5 MTTF, CPI and maximum temperature variation rate (see online version for colours)

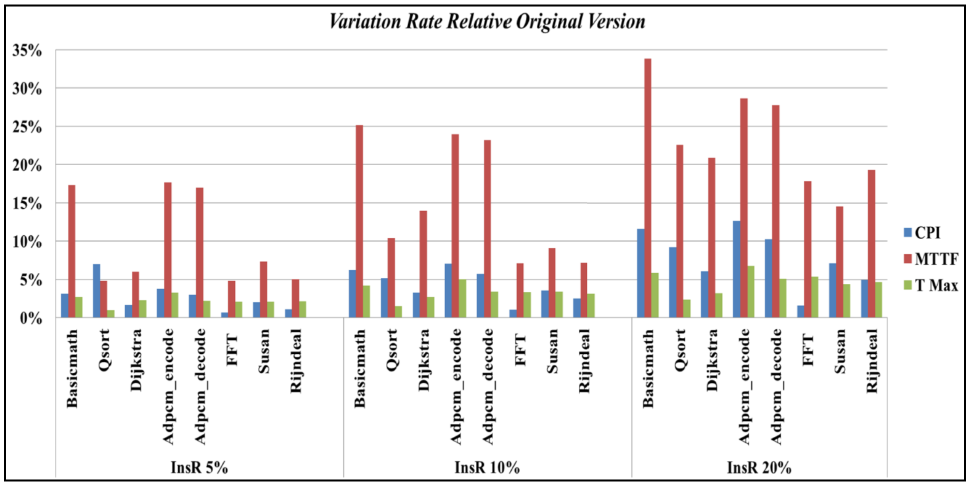


Table 2 MTTF, temperature, CPI, RPIsize and cycle numbers variations

Benchmarks		*Basicmath*	*Qsort*	*Dijkstra*	*Adpcm_encode*	*Adpcm_decode*	*FFT*	*Susan*	*Rijndeal*
Instructions number	VO	6,374,690,678	515,237,860	255,620,666	605,872,292	498,397,968	915,640,054	395,124,652	391,487,888
RPIsize	RPI_5%	106,244,845	8,587,298	4,260,345	10,097,872	8,306,633	15260667	6585410	6524798
	RPI_10%	212,489,690	17,174,596	8,520,689	20,195,744	16,613,266	30521335	13170821	13049596
	RPI_20%	424,979,380	34,349,191	17,041,378	40,391,487	33,226,532	61042670	26341643	26099192
MTTF	VO	12.1	26.06	17.32	18.77	30.3	9.8	28.9	24.6
	RPI_5%	17.33%	4.83%	5.99 %	17.66%	17.01%	4.80%	7.30%	5%
	RPI_10%	25.19%	10.44%	13.98%	24.00%	23.18%	7.10%	9.10%	7.20%
	RPI_20%	33.88%	22.59%	20.87%	28.67%	27.77%	17.80%	14.50%	19.30%
CPI	VO	1.0253	0.7057	0.5264	0.8042	0.7172	0.7717	0.5784	0.4874
	RPI_5%	–3.14%	–5.13%	–1.67%	–3.77%	–3.00%	–0.70%	–2.04%	–1.11%
	RPI_10%	–6.23%	–6.99%	–3.29%	–7.08%	–5.69%	–1.00%	–3.53%	–2.50%
	RPI_20%	–11.60%	–9.22%	–6.08%	–12.63%	–10.28%	–1.58%	–7.14%	–4.97%
Temperature	VO	86.3	70.62	76.24	67.7	70.5	92.2	72.33	70.55
	RPI_5%	–2.73%	–0.94%	–2.28%	–3.29%	–2.20%	–2.06	–2.10%	–2.12%
	RPI_10%	–4.15%	–1.54%	–2.68%	–5.00%	–3.44%	–3.33	–3.40%	–3.12%
	RPI_20%	–5.83%	–2.38%	–3.18%	–6.77%	–5.09%	–5.39	–4.40%	–4.68%
Cycle Nbr	VO	6,502,184,491	365,640,223	134,547,295	487,121,323	357,351,343	652,089,198	225,500,827	190,828,182
	RPI_5%	2.01%	2.33%	3.09%	2.85%	2.76%	2.10%	2.15%	2.11%
	RPI_10%	3.74%	4.79%	6.28%	5.70%	5.52%	3.46%	3.45	3.30%
	RPI_20%	7.19%	9.71%	12.66%	11.40%	11.04%	532%.	4.32%	4.66%

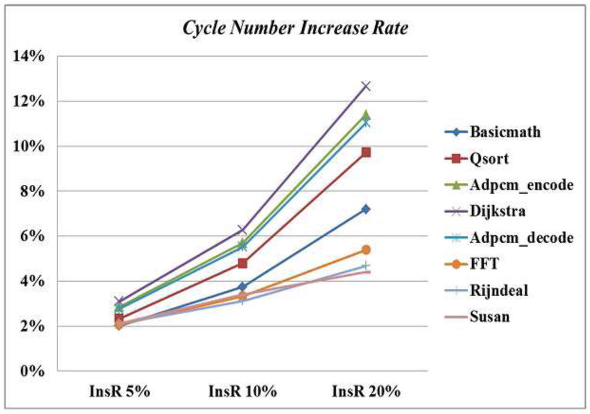
These variations confirm our idea that when we insert relax point we increase MTTF in all the benchmarks.

As matter of fact, MTTF increases with the size of the relax point. For example the Basicmath MTTF are increased by 17.33%, 25.19% and 33.88% for relax point size increased by 5%, 10% and 20% respectively. As shown in Figure 5, we note that the variation of MTTF has a direct relation to CPI rate and Maximum Temperature decrease, but if we take the example of Adcpm_encode benchmarks, when the CPI rate evolves from 3.77% to 12.63%, MTTF increased from 17.66% to 28.67% for relax point size 5% and 20%, respectively. Also, temperature dissipation evolves from 3.29% to 6.77% for the same example. When we decrease the CPI and Maximum Temperature values we increase MTTF. Consequently, we note that the execution time for each benchmark increases as shown, in Figure 6. Indeed, when we insert a third type of relax point (InsR 20%) in Dijkstra, a 12.66% increase was seen in the cycle number. In fact, these evolutions are normal because our idea consist to add instructions in the application despite these instructions are loops with an empty body.

5 Code structure adaptation

RPI can be described as a software solution. Its advantages include increasing the MTTF of the system and lowering the processor's highest temperature without extra costs. It also has disadvantages, such as the increase in the execution time of the application. A new technique is suggested, in this section, based on the modification of source code structure, so that these disadvantages con be overcome.

Figure 6 Cycle number increasing (see online version for colours)



5.1 Methodology

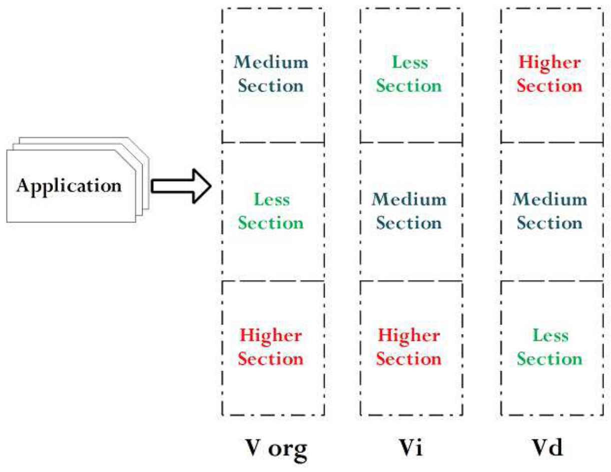
Based on the RPI results, we note that the processor temperature and MTTF are associated to the execution process. Indeed, if the processor executes basic instructions (like an empty loop) before the hot spot code, its temperature decreases and its MTTF improves. This observation has pushed us to find a new technique that reorganises the application code to improve the processor

MTTF. In fact, modification of the application code structure to increase the MTTF of the processor is the major theme of this technique. First, based on code statistics, we deconstructed the application code in independent sections. And, we classifed these sections by their instruction number, then, we adapted their order among one another in order to obtain an optimum MTTF code structure for the processor. 'Adapt' means changing the position of independent code sections between themselves, without adding a new treatment, contrary to RPI technique. The method discussed is called 'code structure adaptation' (CSA).

Unlike RPI technique, this technique does not alter the execution time of the application. Four benchmarks are selected to assess the CSA: Basicmath, FFT, Dijkstra and Qsort. These bookmarks are deconstructed by the CSA into three categories: less, medium and higher. The instruction number of every target section is the basis for this classification. Higher in the hot spot section with a bigger instruction number, less define the section with a smallest instruction number and medium for the medium one. Following this step, the structure of the benchmark code is rearranged and three versions of the application are achieved:

- *Vorg* version: is the original application version, it must be started with the medium section, then follow by the less section and finished with the higher one.
- *Vd* version: is the modified application version where we put the higher section of code at the top of application code, then follow by the medium section and finally the less one.
- *Vi* version: is another modified version of the application, where we put the less section of code at the top of application code, then follow by the medium section and finally the higher one. These can be seen in Figure 7.

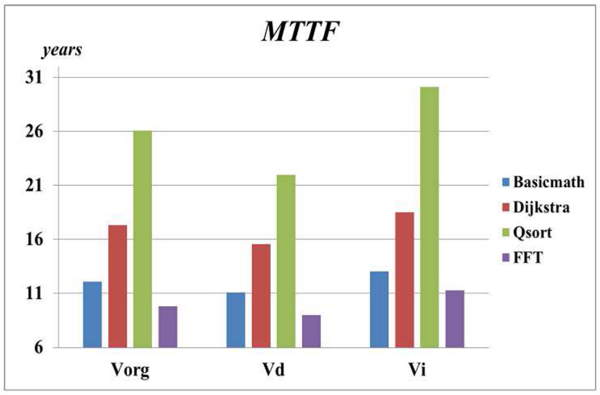
Figure 7 Application Code Decomposition (see online version for colours)



5.2 *Experimentation and discuss results*

In this step we focus on MTTF values relative to each version of the benchmarks and we select the optimal code structure. The experimentation goal is to analysis the effect of the hot spot section position in the application code on the processor MTTF. For the Vi version, hot spot section is put in the last position of code unlike Vd version. We noted that the Vi version for each benchmark gave the highest processor MTTF as shown in Figure 8. As an example, in Qsort benchmark, we-have more than 14 % Increase of MTTF between Vorg and Vi, and more than 35 % between Vd and Vi.

Figure 8 Alpha EV6 MTTF for each application versions (see online version for colours)



Indeed, if the execution process started with the hot spot instructions, the processor lifetime will be severely affected. In contrast, the version Vi make the system more safe. In this way, we can deduce, by putting the simplest treatment in the top of application code and the most complex one at the end, the processor will be safer. Indeed, if the processor executes the higher section in first position, its temperature has rapidly increased. Moreover, knowing that the processor undergoes a temperature increase more rapidly than a decrease, the effect in the Vd version of medium and less section will be more important on the processor, because they will be executed depending on the temperature which is already high because of the hot spot section. In addition, the CSA doesn't change neither the performance of the processor nor the cycle numbers. Which is normal because using CSA we don't add any new instructions in the application code; we only reorganised its structure. Unfortunately, the identification of the independent code sections limited by the CSA technique. Therefore, structural adaptation is not apparent. The problem can be further elaborated through discussing the example of matrix multiplication. The code can be broken down into three categories: initialisation, multiplication and results. However, the code has a disadvantage. It cannot be altered due to the requirement of upper section result by every section.

Srinivasan et al. (2005) describes in detail the dynamic reliability management (DRM) technique. With DRM, the designers can meet reliability targets by using processor adaptation to reduce processor temperature, current density, voltage, and/or frequency as needed at runtime. Indeed, the authors studied an oracular control algorithm for processor adaptations and evaluating its performance on processors with different reliability-qualification cost. Particularly, for an application running on a processor with a specific reliability-qualification cost, it selected the configuration that gave maximum performance and met the required MTTF target. So we deduct that the effectiveness of this method depends essentially on the processor architecture and the application domain. Indeed the authors use the maximum constant temperature the processor can run at while meeting the target MTTF, and the results have shown that for different processor configurations with a temperature <100°C, the performance can be reduced from 0 to 30%. Its results differ from one application to another. However, our methods (RPI and CSA) are a software techniques. Specifically, we modify the application's source code to increase the MTTF while maintaining a fixed processor configuration.

Like mentioned before, we used the Alpha 21264 microprocessor in the simulation and its operating temperature is 100°C. in this case, we increased the MTTF by more than 35% without any loss of performance in CSA method. Also, we obtained average 12% MTTF increase with RPI method. Based on these results, our methods offer an interesting solution for the developer to increase the reliability and the lifetime of the processor with a lower cost and fixed configuration.

5.3 Pilot application process

5.3.1 Simulation process

In this section we propose a simulation process to overcome of the CSA limitation. During this simulation process, a full application takes the place of the section of code and a family of independent benchmarks replace a full application in CSA technique.The main objective of this part of work is to analyse the effect of the execution behaviour of the serial applications on the MTTF. Then, we test and validate the efficiency of the CSA method. For that, we use a Pilot Application (PApp) in this execution process. The formation of a (PApp) is the main purpose. The principles of the PApp must contain:

1 the PApp must be built by an independent application

2 the origin family of the applications must be the same

3 the size and thermal profile of the applications must vary (maximum temperature).

5.3.2 Simulation results

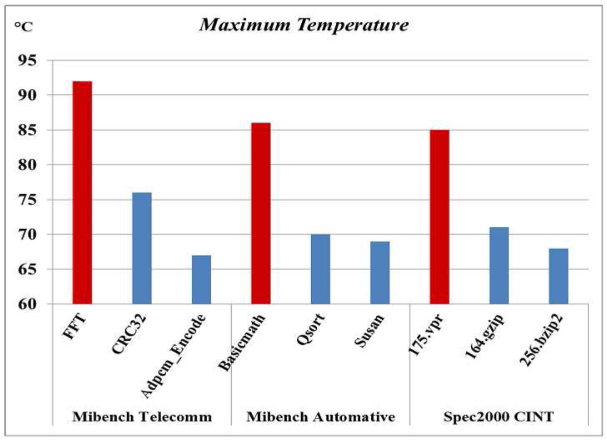
The key criterion in the PAdd realisation is the maximum temperature of processor relative for each benchmark. Indeed, we categorise the applications on the basis of temperature influence (maximum temperature) as shown in Figure 9. The benchmark with a higher temperature is considered a 'hot spot application'. For each application family, the execution order of the applications is altered in every PApp form. And in the end, a number of PApp versions are executed and the processor MTTF is calculated for each of these. We build 18 PApp for three families, as shown in Table 3:

1 Mibench:
- Automative :Basicmath, Qsort and Susan
- Telecomm: FFT, CRC32 and Adpcm encode.

2 SPEC2000 CINT: 175.vpr, 164.gzip and 256.bzip2.

Knowing that the processor MTTF is strongly related to its temperature, in this experiment step we will be interested in the influence of position of 'hot spot application' into the execution process on the processor lifetime. For that, first, for each application family: Basicmath, FFT and 175.vpr, present a 'hot spot application'. Indeed the maximum temperature generated by each one is respectively 86°C, 92°C and 85°C. Changing the position covered by each of them in their respective families can modify their influence on the processor temperature and consequently MTTF.

Figure 9 Alpha Ev6 maximum temperature for each benchmarks family (see online version for colours)



Second, we elaborate the execution process versions (PApp) presented in Table 3. For each PApp the position of the hot spot benchmark is changing, for example in version 4 it is executed in the last position unlike to version 1. After the simulations step, Figure 10 presents the Alpha Ev6 processor MTTF for each version of PApp for each benchmarks family. Based on these results, we note a significant difference of MTTF between these PApp versions. It varies between 8% and 28%. Also we notice that the version 4 of each benchmarks family have a higher MTTF. In fact, the processor is more safe with a version 4 than version 1.

Certainly these results confirm the effectiveness of the CSA technique. If the processor executes the hot spot treatments in the last of the execution process, the hot spot influence over its lifetime is consequently decreased. And we can conclude that, during its execution, if the application increases the temperature gradually (version 4) until these maximum values, these make a processor safer.

Figure 10 Alpha Ev6 MTTF of each Pilots Application (see online version for colours)

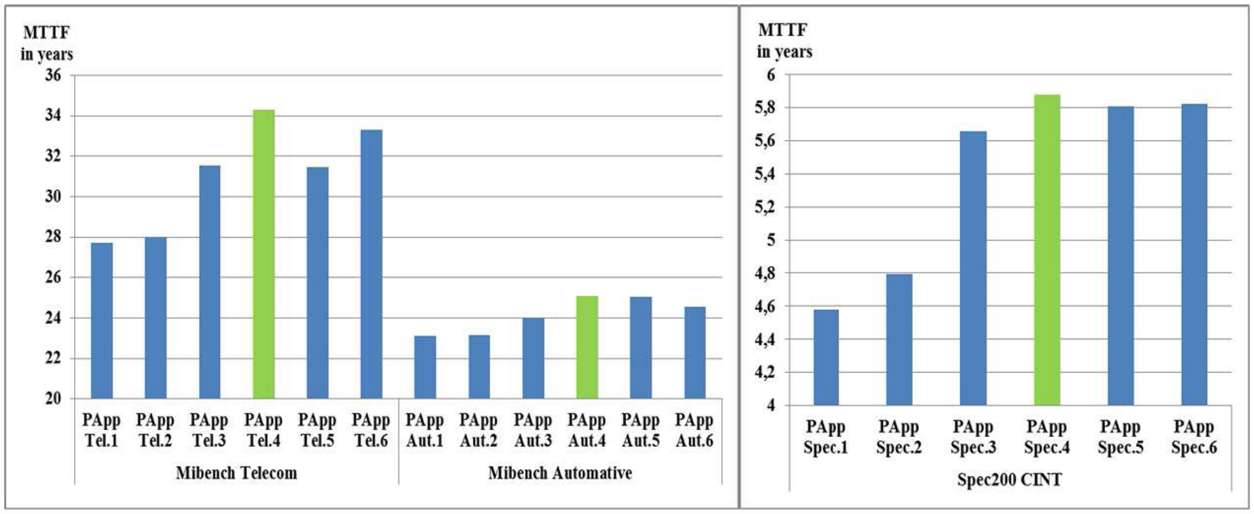


Table 3 PApp versions for each benchmarks family

Benchmarks			*T Max*	*PApp Version*	*PApp Name*
Mibench	*Telecomm*	FFT	92°C	FFT-CRC32-Adpcm_encode	*PApp Tel.1*
				FFT-Adpcm_encode-CRC32	*PApp Tel.2*
		CRC32	76°C	Adpcm_encode -FFT-CRC	*PApp Tel.3*
				Adpcm_encode-CRC32-FFT	*PApp Tel.4*
		Adpcm_encode	67°C	CRC-FFT-Adpcm_encode	*PApp Tel.5*
				CRC-Adpcm_encode-FFT	*PApp Tel.6*
	Automative	Basicmath	86°C	Basicmath-Qsort-Susan	*PApp Aut.1*
				Basicmath-Susan-Qsort	*PApp Aut.2*
		Qsort	70°C	Susan-Basicmath-Qsort	*PApp Aut.3*
				Susan-Qsort-Basicmath	*PApp Aut.4*
		Susan	69°C	Qsort-Susan-Basicmath	*PApp Aut.5*
				Qsort-Basicmath-Susan	*PApp Aut.6*
Spec2000		175.vpr	85°C	175.vpr-164.gzip-256.bzip2	*PApp Spec.1*
				175.vpr-256.bzip2-164.gzip	*PApp Spec.2*
		164.gzip	71°C	256.bzip2-175.vpr-164.gzip	*PApp Spec.3*
				256.bzip2-164.gzip-175.vpr	*PApp Spec.4*
		256.bzip2	68°C	164.gzip-175.vpr-256.bzip2	*PApp Spec.5*
				164.gzip-256.bzip2-175.vpr	*PApp Spec.6*

Finally, the application designers can use the data from our technique CSA, confirmed by the simulation process PApp, to produce efficient applications that does not degrade the performance and does not reduce the processor lifetime.

6 Conclusions

The increased complexity and the decreased feature sizes have caused a very high power density in the last generation of system-on-chip (SoC). This situation generates temperature hotspots, which in turn may lead to a non-uniform ageing and an acceleration of chip failure. In fact, temperature gradients and hotspots not only affect the performance of the system, but also lead to unreliable circuit operation and affect the chip lifetime. In order to ensure the proper functioning of a chip and increase its lifetime, it is essential to regulate the temperature and lower hotspots. In this context, the content of this paper proposed two techniques for MTTF increase.

The first technique, namely relax point injection (RPI), consists in adding a particular treatment in the application code to protect the chip from a fatal temperature increase. With an RPI technique, we increase a processor MTTF by 33.88%. Unfortunately, in some cases, RPI increases the execution time of the application more than 12%.

The second technique, namely code structure adaptation (CSA), consists to readjust the structure of application code to increase processor lifetime. CSA raises processor MTTF by 35%, and keeps the application execution time unchanged. But, the limit of CSA method is to find the independent sections of code. So, it is not evident to modify its structure. For that, to improve this technique, we proposed a simulation process based on the Pilot Application (PApp). This represents a set of application of the same family, with different temperature profiles, who executed in series on the same processor. Each application replaced a section of code in CSA. And we deducted that, to make a processor safer, it must execute the application which a higher temperature in the last position. With this process, we improved the processor MTTF by 28%. Our techniques can be of great help to the application designer because it allows them to take into account the thermal profile during the development of their application. We can apply these MTTF optimisation methods on the multicore architectures in further studies. And we can integrate these techniques in other thermal management method.

References

Ayala, L.J., Lopez-Vallejo, M., Atienza, D., Raghavan, P., Catthoor, F. and Verkest, D. (2007) ‘Energy-aware compilation and hardware design for VLIW embedded systems’, *Int. J. of Embedded Systems*, Vol. 3, Nos. 1/2, pp.73–82.

Basoglu, M., Orshanshy, M. and Erez, M. (2010) ‘NBTIaware DVFS: a new approach to saving energy and increasing processor lifetime’, *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pp.253–258.

Batty, W. et al. (2002) ‘Global coupled EM-electrical thermal simulation and experimental validation for a spatial power combining MMIC array’, IE*EE Transactions on Microwave Theory and Techniques*, Vol. 50, No. 12, pp.2820–2833.

Blish, R. and Durrant, N. (2000) *Semiconductor Device Reliability Failure Models*, International SEMATECH, Tech. Rep.

Bolchini, C., Cassano, L. and Miele, A. (2016) ‘Lifetime-aware load distribution policies in multi-core systems: An in depth analysis’, *2016 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Dresden, pp.804–809.

Buyuktosunoglu, A. et al. (2003) ‘Energy efficient coadaptive instruction fetch and issue’, in *Proc. of the 30th Annual Intl. Symp. on Comp. Architecture*.

Choi, J., Kim, Y., Sivasubramaniam, A., Srebric, J., Wang, Q. and Lee, J. (2007) ‘Modeling and managing thermal profiles of rack-mounted servers with thermostat’, in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.

Chunling, H., Jimenez, D.A. and Kremer, U. (2007) ‘An evaluation infrastructure for power and energy optimisations’, *Int. J. of Embedded Systems*, Vol. 3, Nos. 1/2 pp.31–42.

Coskun, A.K., Strong, R., Tullsen, D.M. and Rosing, T.S. (2009) ‘Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors’, *SIGMETRICS Perform. Eval. Rev.*, Vol. 37, No. 1, pp.169–180.

Guthaus, M.R., Ringenberg, J.S., Austin, D.E.T.M., Mudge, T. and Brown, R.B. (2001) ‘Mibench: Afree, commercially Representative embedded benchmark suite’, in *WWC – 4.2001 IEEE International Workshop*, pp.3–14.

Heath, T., Centeno, A.P., George, P., Ramos, L. and Jaluria, Y. (2006) ‘Mercury and freon: temperature emulation and management for server systems’, in *Proceedings of the Twelfth Conference on Architectural Support for Programming Languages and Operating Systems*, pp.106-116.

Hughes, C.J., Srinivasan, J. and Adve, S.V. (2001) ‘Saving energy with architectural and frequency adaptations for multimedia applications’, in *Proc. of the 34th Annual Intl. Symp. on Microarchitecture*.

Jun, Z., Sha, E.H-M., Zhuge, Q., Yi, J. and Wu, K. (2014) ‘Efficient fault-tolerant scheduling on multiprocessor systems via replication and deallocation’, *Int. J. of Embedded Systems*, Vol. 6, Nos. 2/3, pp.216–224.

Kessler, R.E., McLellan, E.J. and Webb, D.A. (1998) ‘The Alpha 21264 microprocessor architecture’, *Proceedings International Conference on Computer Design*, *ICCD98*.

Kong, J., Chung, S.W. and Skadron, K. (2012) ‘Recent thermal management techniques for microprocessors’, *ACM Comput. Surv*., Vol. 44, No. 3, pp.13:1–13:42.

Koval, V. and Farmaga. I.W. (1994) ‘MONSTR: a complete thermal simulator of electronic systems’, in *Proceedings of the ACM/IEEE 31st Design Automation Conference*.

Kumar, A., Shang, L., Peh, L. and Jha, N.K. (2006) ‘HybDTM: a coordinated hardware-software approach for dynamic thermal management’, in *Proceedings of the 43rd Annual Design Automation Conference (DAC06)*, pp.548–553.

Liu, Y., Yang, H., Dick, R.P. and Wang, H. (2007) ‘Thermal vs energy optimization for dvfsenabled processors in embedded systems’, in *Quality Electronic Design, 2007. ISQED ’07. 8th International Symposium*, pp.204–209.

McGowen, R., Poirier, C., Bostak, C., Ignowski, J., Millican, M., Parks, W. and Naffziger, S. (2006) ‘Power and temperature control on a 90-nm Itanium family processor’, *IEEE Journal of Solid-State Circuits*, Vol. 41, No. 1, pp.229–237.

Michaud, P., Sazeides, Y., Seznec, A., Constantinou, T. and Fetis, D. (2005) *An Analytical Model of Temperature in Microprocessors*, Technical Report PI-1760/RR-5744, IRISA/INRIA, Bibliography 153.

Mutyam, M., Li, F., Vijaykrishnan, N., Kandemir, M.T. and Irwin, M.J. (2006) ‘Compiler-directed thermal management for VLIW functional units’, in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES06)*, pp.163–172.

Nethercote, N., Walsh, R. and Fitzhardinge, J. (2006) ‘Building workload characterization tools with Valgrind’, Invited tutorial, *IEEE International Symposium on Workload Characterization (IISWC 2006)*, San Jose, California, USA, October.

Nowroz, A.N., Cochran, R. and Reda, S. (2010) ‘Thermal monitoring of real processors: techniques for sensor allocation and full characterization’, *Design Automation ACM/IEEE Conference (DAC), 47th*.

Pecht, M.G. et al. (1999) *Guidebook for Managing Silicon Chip Reliabilty*, CRC Press, Boca Raton, FL.

Rencz, M., Szekely, V., Poppe, A. and Courtois, B. (2000) ‘Friendly tools for the thermal simulation of power packages’, in *Proceedings of the International Workshop On Integrated Power Packaging*, pp.51–54.

Skadron, K., Abdelzaher, T. and Stan, M.R. (2002) 'Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management', in *Proceedings of the Eighth IEEE International Symposium on High-Performance Computer Architecture*, pp.17–28.

Song, J.W., Mukhopadhyay, S. and Yalamanchili, S. (2016) 'Amdahl's law for lifetime reliability scaling in heterogeneous multicore processors', *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Barcelona, pp.594–605.

Srinivasan, J. and Adve, S.V. (2003) 'Predictive dynamic thermal management for multimedia applications', in *Proc. of the 2003 Intl Conf. on Supercomputing*.

Srinivasan, J., Adve, S.V., Bose, P. and River, J.A. (2004) 'The case for lifetime reliability- aware microprocessors', *Proceedings of 31st International Symposium on Computer Architecture (ISCA 04)*.

Srinivasan, J., Adve, S.V., Bose, P. and Rivers, J.A. (2005) 'Lifetime reliability: toward an architectural solution', *IEEE Micro*, May, Vol. 25, No. 3, pp.70–80.

Stan, M.R., Skadron, K., Barcella, M., Huang, W., Sankaranarayanan, K. and Velusamy, S. (2003) 'Hotspot: a dynamic compact thermal model at the processor-architecture level', *Elsevier, Micro Electronics Journal: Circuits and Systems*, Vol. 34, No. 12, pp.1153–1165.

Tiwari, V., Brooks, D. and Martonosi, M. (2000) Wattch: A framework for architectural-level power analysis and optimizations', in *27th Annual ACM/IEEE International Symposium on Computer Architecture*, pp.83–94.

Torki, K. and Ciontu, F. (2002) 'IC thermal map from digital and thermal simulations', in *Proceedings of the 2002 International Workshop on THERMal Investigations of ICs and Systems (THERMINIC)*, pp.303–308.

Viswanath, R. et al. (2000) 'Thermal performance challenges from silicon to systems', *Intel Technology Journal*, Vol. 4, No. 3, pp.1–16.

Viswanathan, S., Ramesh, P.K. and Somani, A.K. (2009) 'Managing the impact of on-chip temperature on the lifetime reliability of reliably overclocked systems', *The Second Intern Confer on Dependability*.

Vladimir, S., Poppe, A., Pahi, A., Csendes, A. and Hajas, G. (1997) 'Electro-thermal and logi-thermal simulation of VLSI designs', *IEEE Transactions on VLSI Systems*, Vol. 5, No. 3, pp.258–269.

Weidendorfer, J., Kowarschik, M. and Trinitis, C. (2004) 'A tool suite for simulation based analysis of memory access behavior', *Proceedings of the 4th International Conference on Computational Science*, Krakow, Poland.

William, S., Saibal, M. and Sudhakar, Y. (2015) 'Architectural reliability: lifetime reliability characterization and management of many-core processors', *IEEE Computer Architecture Letters*, Vol. 14, No. 2, pp.103–106.

Yi-Kan, C. and Sung-Mo, K. (2000) 'A temperature-aware simulation environment for reliable ULSI chip design', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 10, pp.1211–1220.

Zanini, F., Atienza, D., Jones, C.N., Benini, L. and De Micheli, G. (2013) 'Online thermal control methods for multiprocessor systems', *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 18, No. 1, pp.6:1–6:26.