Thermal analysis and modeling of embedded processors

José L. Ayala , Cándido Méndez , Marisa López-Vallejo

Department of Computer Architecture, Complutense University of Madrid, 28040 Madrid, Spain
Department of Electronic Engineering, Politecnica University of Madrid, 28040 Madrid, Spain

A B S T R A C T

This paper presents a complete modeling approach to analyze the thermal behavior of microprocessor-based systems. While most compact modeling approaches require a deep knowledge of the implementation details, our method defines a black box technique which can be applied to different target processors when this detailed information is unknown. The obtained results show high accuracy, applicability and can be easily automated. The proposed methodology has been used to study the impact of code transformations in the thermal behavior of the chip. Finally, the analysis of the thermal effect of the source code modifications can be included in a temperature-aware compiler which minimizes the total temperature of the chip, as well as the temperature gradients, according to these guidelines.

Keywords:
Thermal model
Embedded processor
Temperature

1. Introduction

Continuing advances in semiconductor technology have allowed dramatic performance gains for general-purpose microprocessors and embedded systems. These improvements are due both to increasing clock rates as well as to advanced support for exploiting instruction-level parallelism and memory locality using the additional transistors available in each process generation. However, as a negative consequence, this causes a significant increase in power dissipation, due to the fact that the dynamic power is proportional to both clock frequency and switching capacitance (which increases with the number of devices and components integrated in the chip). Thus, despite continuous attempts to reduce voltages and to design lower power circuits, power dissipation levels have steadily increased with every new microprocessor.

As technology scales, higher power consumption coupled with smaller chip area will result in higher power density, which in turn will lead to higher power temperature on the chip [1,2]. In fact, extrapolating the changes in microprocessor organization and the device miniaturization, one can project future power density to 200 W/cm^2 [3]. This requires extensive efforts on cooling techniques which have shown to be complex and expensive. Early considerations at different design levels of these thermal effects can lead to more accurate design parameter estimation and faster design convergence [4], what at the end is translated into better designs and reduced design time. Moreover, the analysis and management of every design parameter with a strong impact on the thermal behavior can provide an effective way to minimize the total temperature of the chip as well as the temperature gradient. Both metrics degrade the circuit performance and the reliability of the die and the package.

While hardware solutions to temperature management problems are very important, software can also play an important role because it determines the circuit components exercised during the execution and the period of time for which they are used. In particular, compilers and source code transformations determine the data and instruction access patterns of applications, what shapes the power density profile.

In this paper we present a parameterized thermal characterization of a programmable microcontroller and we analyze the effect of different code-level transformations in the thermal map. The use of the thermal model and the analysis tool developed in this work allows characterizing the thermal behavior of the functional units in the microcontroller, as well as the evolution in time of the thermal behavior when different implementations of the same benchmark are executed. Moreover, the analyzed code transformations can be included in a thermal-aware compiler which reduces the gradient of temperature in the chip area of the target architecture.

The contributions of this paper are:

(1) Definition of a black box methodology to characterize the thermal behavior of the chip under different working loads. This methodology can be applied to different target architectures of embedded processors even if the implementation details are unknown.
(2) Analysis of the effect of several source code transformations in the thermal behavior of the chip. The analysis is not only focused on the total temperature of the chip but it also takes into account the gradients of temperature that can appear between the functional units.
(3) Establishment of the basis to build a temperature-aware compiler have been stated.

This paper is composed as follows: Section 2 presents the previous relevant works in this topic, while our proposed methodology is briefly explained in Section 3. Finally, Section 4 covers the conducted experimental works and some conclusions are drawn in Section 5.

2. Related work

In recent years there has been an increasing interest to provide a detailed die temperature distribution [5,6]. In these works, the authors present different detailed full chip thermal models. All these models have detailed temperature distribution across the silicon die and can be solved efficiently. However, it has been reported recently that the results achieved by these models present inaccuracy problems [7,8]. All these works provide an analytical method for studying the thermal distribution in the die of high-performance processors but they require the complete knowledge of the layout details. Moreover, these models are constrained to the processor layout and cannot be easily extended to different target architectures.

There are other approaches which rely on dynamic measures to characterize the thermal behavior of the chip [9]. These techniques, opposite to ours, require not only the complete knowledge of the target architecture but also the capability to modify it. Some other approaches like [10] or [11] also propose techniques based on electrical measures to develop a power consumption model. However, these works have not dealt with the thermal behavior of the chip. Moreover, the thermal effect of the source level optimizations has not been previously addressed, being one of the keys in our work.

In the field of temperature optimization, research on solving this problem has mainly focused on runtime techniques. These approaches use predictive algorithms [12], dynamic profiling [1] or activity migration [13] to reduce the temperature in a chip by dynamically distributing jobs on multiple processors. Our work is different from these in that it is static in its approach and targets the effect of source-level modifications in the thermal behavior. The related work in the area of static thermal management is still scarce and few works can be found targeting the temperature-aware floorplanning of the chip [14] and the loop parallelization for multiprocessors [15]. However, this is the first work which analyzes the effect of code transformations in the thermal behavior of the functional units of the chip, providing an expertise that can be incorporated in a temperature-aware compiler.

3. Methodology

In this paper we propose and present a methodology to parameterize the thermal behavior of the components of a processor-based system, starting with a low-cost microcontroller. Furthermore, this paper uses our thermal parameterization methodology to evaluate the temperature distribution in the microcontroller for a specific application program and the impact of the source code modifications in the temperature behavior.

The reason for selecting a low-cost microcontroller as our target processor is that nowadays this kind of microcontrollers are widely used in many low temperature applications, but the methodology can be also applied to other architectures of embedded processors. Moreover, their reduced instruction set and their simple architecture, with few easy-to-distinguish functional units, facilitates the task of characterizing the whole architecture blocks. For most microcontrollers, the functional blocks under study are: General Purpose RAM, Instruction Memory, Data EEPROM, Register file, ALU, ALU accumulator, Program Counter and Decode and Control unit. Fig. 1 shows these functional blocks and the data communication in a typical low-cost microcontroller (PIC architecture [16]).

One of the issues when performing a thermal characterization of these microcontrollers is the lack of information about the implementation of their functional units and their layout. Another problem is the difficulty of measuring the temperature of the chip. These measures need expensive external equipment to be made. We have avoided these problems devising a black box characterization, which does not require this information, and using power as an intermediate parameter. The power consumption is a variable that can be easily measured in the processor without expensive and complex equipment.

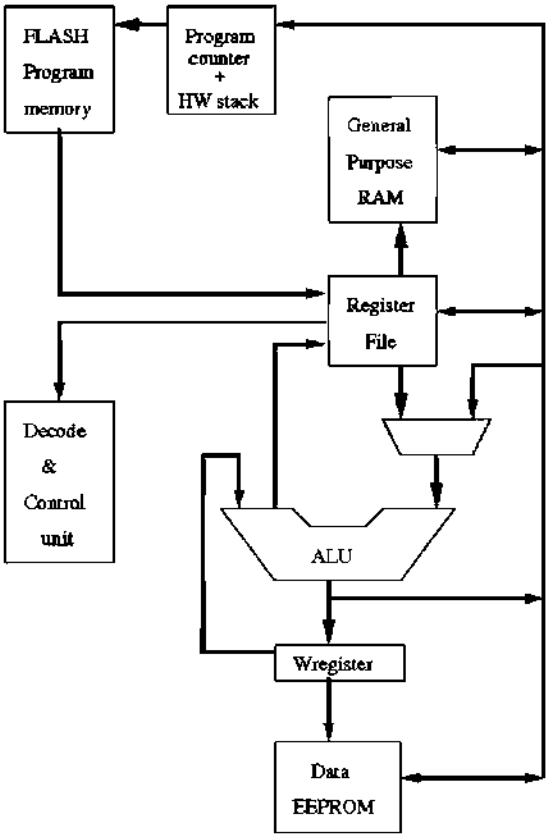

Fig. 1. Analyzed functional blocks in the PIC.

The characterization method we have developed has several advantages. Among others, it is not tied to a specific architecture, and it is easily replicable. Due to this, the methodology can be applied to several target processors without a deep knowledge of their implementation details. Another advantage is that it is fully automated, the characterization can be included in a software tool and the designer is released from this. Also, the thermal model is based on direct measures with high accuracy in the result. Finally, one of the most important advantages is that it presents thermal temporal evolution for all functional units as well as the whole chip.

The use of our characterization methodology has allowed to perform a complete thermal analysis and to evaluate the impact of source code transformations in the thermal behavior. Fig. 2 describes the thermal characterization methodology which is presented in this paper. As this diagram shows, the power characterization of the microcontroller is driven by the current measures performed during the initial stage. The power characterization uses these measures to build a model which predicts the power consumed in the functional units given the utilization of the unit. After that, the estimation of the layout areas allows to define a thermal model which characterizes the architecture. This thermal characterization uses these areas to build a model which predicts the evolution in time of the thermal behavior for every functional unit in the chip. Finally, the thermal characterization achieved is used to perform a thermal analysis of the benchmarks and evaluate the thermal impact of the source code transformations.

The following sections describe briefly the devised methodology for the thermal characterization.

3.1. Power characterization

The first step of our characterization methodology is to obtain the power consumed by every functional unit of the microcontroller by means of electrical measures. Such electrical measures provide an accurate way to obtain the power information of the device.

The power consumption can be obtained just setting a resistor between the power supply and the chip and measuring the current passing through it. Then, using Ohm's law, $P = V \times I$, it is possible to calculate the power consumption of the chip.

To collect these power figures for all the functional units of the microcontroller, we have developed a set of benchmarks that are able to excite the processor in a particular way. Each benchmark is made to stress a specific functional block under diverse workloads. Therefore, a 100% utilization for a functional unit means that every executed instruction makes use of such functional unit. These synthetic benchmarks are composed of atomic operations such as: continuous execution of

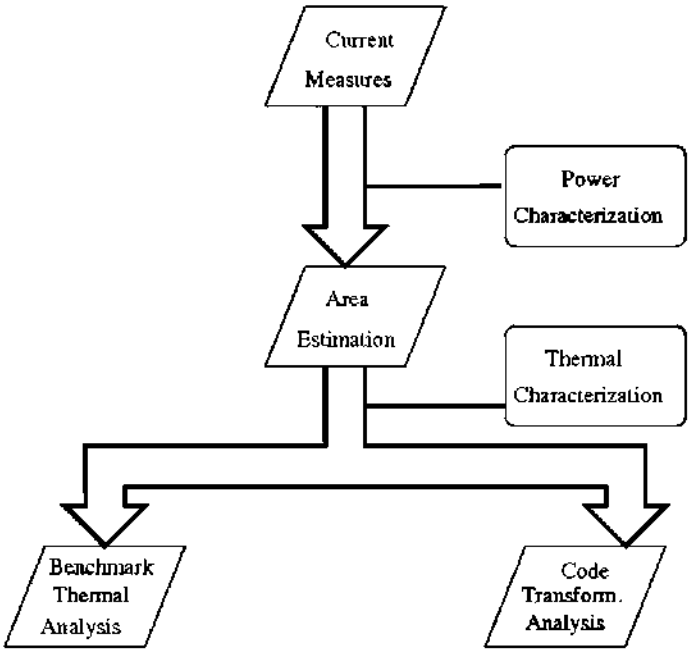

Fig. 2. Characterization methodology.

Table 1
Accuracy of the power model.

Operation	Measured power (mW)	Estimated power (mW)	Error (%)
Multiplication (8-bits integer operands)	6.137	6.268	2.14
Division (16-bits integer operands)	6.175	6.203	0.46
Square root (12-bits integer operand)	6.204	6.192	0.19
HEX to BCD conversion	5.994	6.103	1.82
Log2 calculation	6.451	6.245	3.19

arithmetic operations, continuous execution of memory loads and stores, arithmetic operations with immediate operands, bit-test operations, so on.

In this way, the power measured when executing the benchmark will be proportional to the power consumed by the excited functional unit. We have made experiments with various workloads for the functional units to see how different utilizations affect their power consumption and to refine the power model obtained. Also, the power consumed by the microcontroller during the idle state (P_{idle}) and the energy dissipation due to the communication buses (P_{bus})[1] are measured in order to subtract these values from the functional unit measures.

Once we have gathered power measures from all the benchmarks developed, it is possible to obtain a set of power/utilization equations that let us to estimate the power behavior of the microcontroller for different applications. Therefore, the model can determine the power consumption expected in every functional unit due to its workload.

The accuracy of this model has been analyzed comparing the results given by the model with the electrical measures directly acquired from the resistor (see Table 1). As can be seen, the estimated power consumption does not differ in more than a 3% from the measured power consumption. Moreover, the average error for the devised power model is less than a 2%.

3.2. Thermal parameterization

Once all the power figures have been collected, the next step is to obtain the temperature estimation. In order to do that, we have used a dynamic compact thermal model based on *R*th*C*th-networks [17].

There exists a well-known duality [18] between heat transfer and electrical phenomena. Heat flow can be described as a current passing through a thermal resistance, leading to a temperature difference analogous to a voltage. Thermal capacitance is also necessary for modeling transient behavior, to capture the time required for a mass to change temperature

[1] Power consumption in the communication buses is estimated mathematically after the calculation of the switching activity of the data transfer.

and hence to account for the delay before a change in power results in the temperature's steady state. Lumped values of thermal *R*th and *C*th can be computed to represent the heat flow among regions of a chip and from each region to the thermal package. The thermal resistances and capacitances together lead to exponential rise and fall times characterized by thermal *R*th*C*th time constants, analogous to the electrical *RC* time constants. The rationale behind this duality is that current and heat flow are described by exactly the same differential equations for a voltage difference. In the thermal-design community, these equivalent circuits are called compact models [17], or dynamic compact models if they include thermal capacitors. This duality provides the convenient basis for an architecture-level thermal model. For a microarchitectural unit, heat conduction to the thermal package and to neighboring units are the dominant mechanisms that determine the temperature.

With this thermal model, each functional block is modeled as an *R*th*C*th pair, representing the vertical heat transfer. The lateral heat transfer is not considered in the model because embedded single-core processors are not constrained by this phenomenon, on the contrary, the vertical heat transfer is much more intense [19]. Also, this model provides the following formulae to calculate R_{th} and C_{th}:

$$R_{th} = t/(k \times A)$$

and

$$C_{th} = t \times c \times A$$

where k is the thermal conductivity of the material per unit volume,[2] c is the thermal capacitance per unit volume,[3] t is the chip thickness and in our experiments it has been set to 0.5 mm as in [17], and A is the estimated or measured area of the device.

As this model shows, we need some information about the area required by the layout of the blocks. To overcome this problem, we have devised a mathematical model where the estimated area is proportional to the power dissipated by the chip when all the functional units have a 100% utilization. For that, the total power consumption due to the described functional units when they have a 100% utilization (PT) is modeled as:

$$P_T = \sum_i P_i - P_{idle}$$

where P_i corresponds to the power measured by the system when a particular functional unit is stressed (obtained by the previous characterization) and P_{idle} is the power consumed by the system during the idle state (obtained by electrical measures).

Given that the power consumed by the functional unit is proportional to its active area

$$P_i = I^2 \times R_{sup} \times A$$

where R_{sup} is the resistance per unit area, we obtain the relationship between the power consumed by the functional unit with a 100% utilization and the active area of the device. This relationship allows to calculate the area A of the functional unit and, hence, the *R*th and *C*th values are obtained. This characterization achieves an estimation of the layout of the microcontroller where the device areas are estimated but the placement cannot be assured. Any information provided by the manufacturer can be incorporated into the model to improve the results. The proposed model relates the power dissipation with the area on silicon of the functional units. This assumption fails for high-performance architectures, but it has been shown to be accurate in embedded processors with standard pipelines, as the PIC considered in this work [20].

Once the *R*th and *C*th values are known, and the power behavior of the functional units of the chip has been modeled, the thermal evolution can be estimated.

As a result of this process, a relationship between temperature and utilization of each functional unit is obtained. With all this information, the thermal evolution for every functional unit, as well as for the whole chip, can be studied.

3.3. Application of the model

Finally, the characterization presented in the previous sections has been used to analyze the thermal effect of the source modifications in a set of application examples. We have developed various alternative implementations of these benchmarks with several code-level transformations. Doing this, it can be observed how the different code variations affect the temperature distribution inside the chip. Examining the results, the designer can reach conclusions such as which transformations should be selected to reduce the total temperature in the chip, or which transformations achieve the best temperature distribution minimizing the gradients. These conclusions could be then included in a future temperature-aware compiler.

Summarizing, in preceding sections we have presented the methodology followed to accurately characterize the thermal behavior of a low-cost microcontroller. With all the parameters gathered, it is possible to simulate the thermal variances of the chip and its functional blocks. Besides, this simulation could be fully automated, so an easy-to-use thermal simulator can be developed. It has also been clearly shown that this methodology is independent from the architecture of the processor and a different target architecture can be used without a deep knowledge of its implementation details. A problem of this

[2] 100 W/m K for silicon and 400 W/m K for copper at 85 °C.

[3] 1.75×10^6 J/m^3 for silicon and 3.55 ⁄ 10^6 J/m^3 for copper.

Table 2
Power characterization and area estimations.

	Power (mW)	Area (mm^2)
Decoder	3.016	2.317E-01
RAM	0.374	2.878E-02
ALU	0.115	8.850E-03
Accumulator	0.148	1.142E-02
Register file	0.100	7.761E-03
Program counter	0.358	2.751E-02
Flash	0.616	4.737E-02
EEPROM	0.225	1.736E-02
Leakage	0.295	
Static	0.489	
Fetch	1.075	
Bus switch	0.764	

```
for (i=1; i<100; i++)
{
  g();
}
```

```
for (i=1; i<50; i+=2)
{
  g();
  g();
}
```

Fig. 3. Example of loop-unrolling compiler transformation.

characterization method is the difficulty of reaching a lower level of granularity. This is imposed by the simplicity of the microcontroller and the lack of knowledge about the functional units implementation. This problem can be overcome by the selection of a processor with a richer architecture would allow to increase the granularity of this analysis.

4. Experimental work

For our set of experiments we have selected a Microchip PIC16F873, complex enough to validate the thermal characterization methodology proposed here, and can be found in many embedded systems with low-power and low-temperature constraints. This device is a low-cost microcontroller with a Harvard RISC architecture. It includes 4 K × 14 words of FLASH program memory, 192 bytes of RAM and a data EEPROM of 128 bytes. This microcontroller also includes a SRAM Register file, a Program Counter with an eight level deep hardware stack and an 8-bit ALU with an 8-bit accumulator. As can be seen, the architecture resembles the common characteristics of a broad range of embedded processors.

One of the reasons to select this PIC for our studies is that, due to its low-complexity core architecture, a simple selection of the composing functional units can be found. Moreover, the reduced set of instructions minimizes the number of power measures and simplifies the aforementioned thermal characterization process.

The first stages of the thermal characterization process provide the power consumption of every functional unit in the chip with a 100% utilization, and the estimation of their active areas (see Table 2). The thermal characterization methodology can be applied to a different target architecture just changing the values stored in the table, and these values can come from direct power measures or information from the manufacturer.

In order to analyze the thermal behavior of the PIC when applying a source code transformation, two common embedded applications have been chosen: an image processing algorithm and a dijkstra algorithm from the MiBench benchmark suite [21]. These benchmarks stress the processor architecture in the similar way to common embedded applications. The first benchmark is basically an image filtering application which is based on a loop dominated algorithm.[4] Due to the severe memory constraints imposed by the PIC architecture, this algorithm had to be adapted to satisfy all the memory requirements. The second application is a well-known algorithm to solve shortest path problems. In this case, the application is dominated by the data transfers to registers.

We have developed four different implementations of the image filtering benchmark and the dijkstra application according to different source-level transformations. These implementations are executed by the processor and the thermal information is retrieved, providing the impact of the code modifications in the temperature of the chip.

We have analyzed four different implementations of every benchmark:

(1) *Baseline implementation:* Original implementation of the application adapted to be executed by the PIC. This case responds to a single-processor single-stage algorithm.

[4] As many other multimedia applications.

```
int x[100];

int *x;
...
x = (int *)malloc(100*sizeof(int));
...
free (x);
```

Fig. 4. Example of dynamic memory implementation.

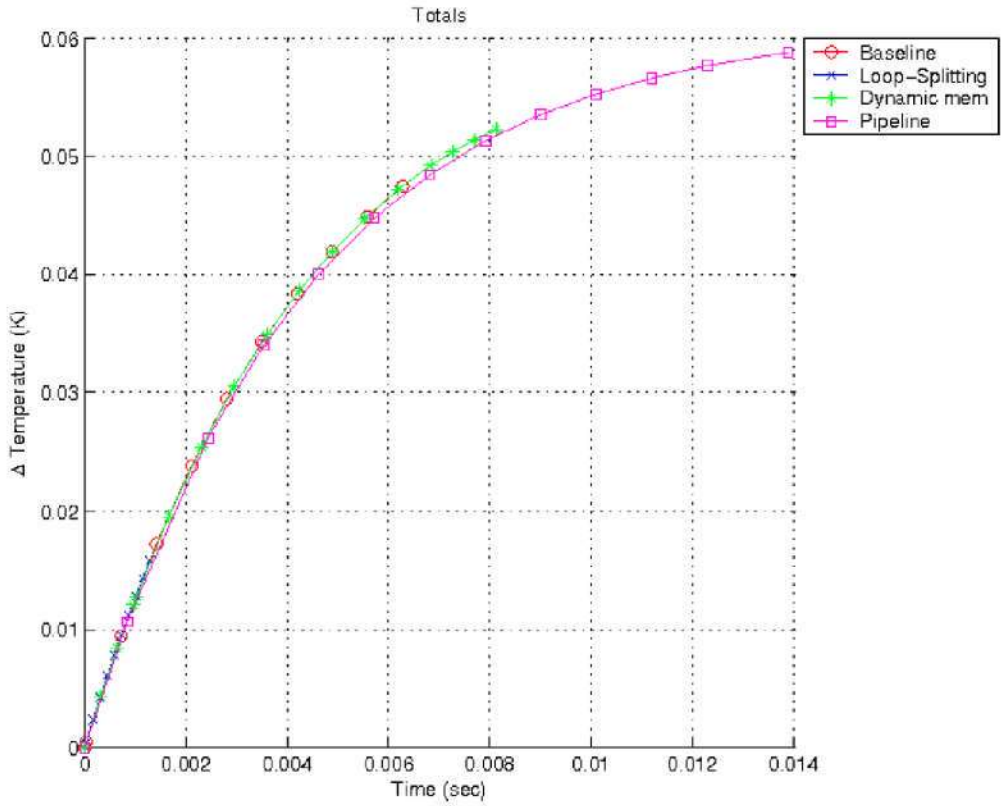

Fig. 5. Evolution in time of the total temperature in the chip (image processing algorithm).

(2) *Loop-unrolled (split) implementation:* Modified code after a loop-unrolling transformation applied to the main loop of the original processing algorithm. The goal of this optimization is to increase the program's speed by reducing the "end of loop" test on each iteration. In this way, loops are re-written as a sequence of independent statements which eliminate the overhead of the loop controller (as appears in Fig. 3).
(3) *Dynamic memory implementation:* Modified code in which the static declaration of variables and data has been moved to dynamic memory and managed in order to decrease the memory requirements. Fig. 4 shows an example of this technique in a simple code.
(4) *Pipelined implementation:* Segmented version of the code in which the image processing algorithm and the dijkstra application are split into multiple stages which communicate along a pipeline. In our results, the temperature variations for just one of the stages are presented.

The thermal characterization methodology has been used to perform a global (total temperature) and local (temperature gradient) experimental work. The global and local thermal analysis of the previous implementations of the image filtering algorithm and the dijkstra application will provide a complete knowledge of the thermal impact of the source-level transformations. A complete set of design guidelines can be incorporated in a temperature-aware compiler.

4.1. Global analysis

The impact of the code modifications in the temperature of the whole chip has been analyzed. This analysis provides a general view on how the temperature evolution of the chip is depending on how the benchmark is implemented. This kind

of global heating has a negative effect on propagation delays, signal integrity and power consumption. Therefore, one of the first goals in our research is to evaluate such global heating in the chip considering the implementation details of the executed benchmarks.

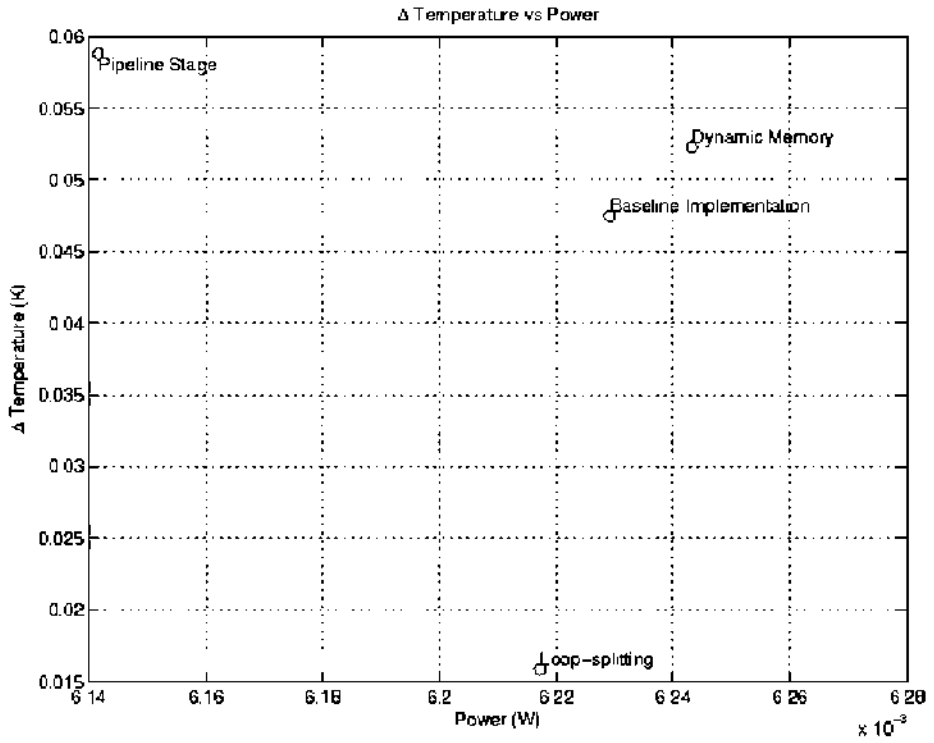

Fig. 6. Power consumption vs. Total temperature in the chip (image processing algorithm).

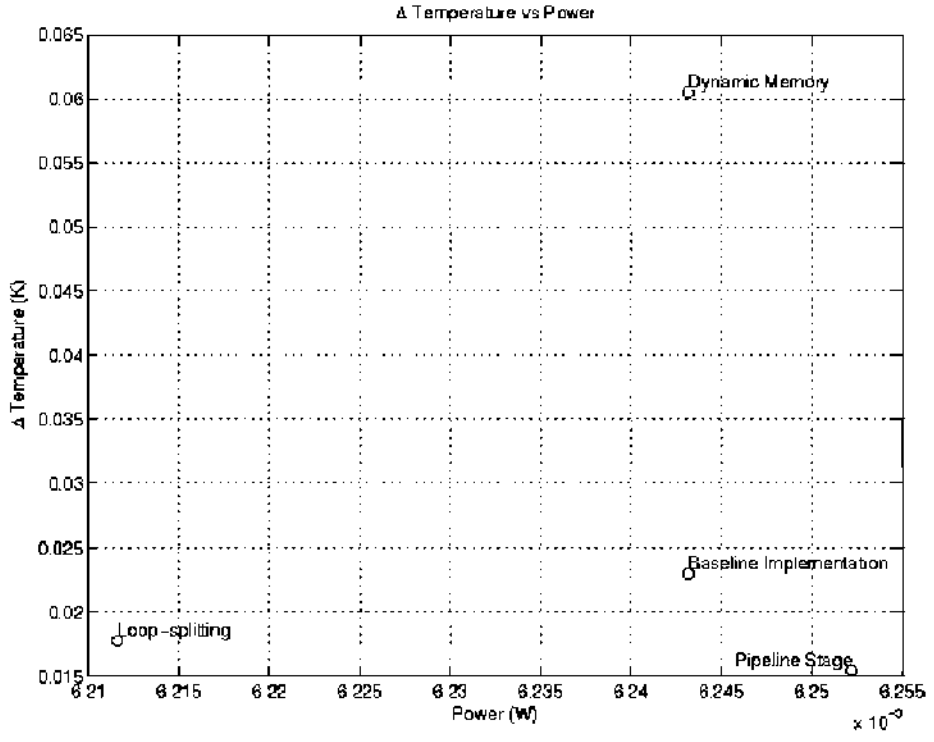

Fig. 7. Power consumption vs. Total temperature in the chip (dijkstra application).

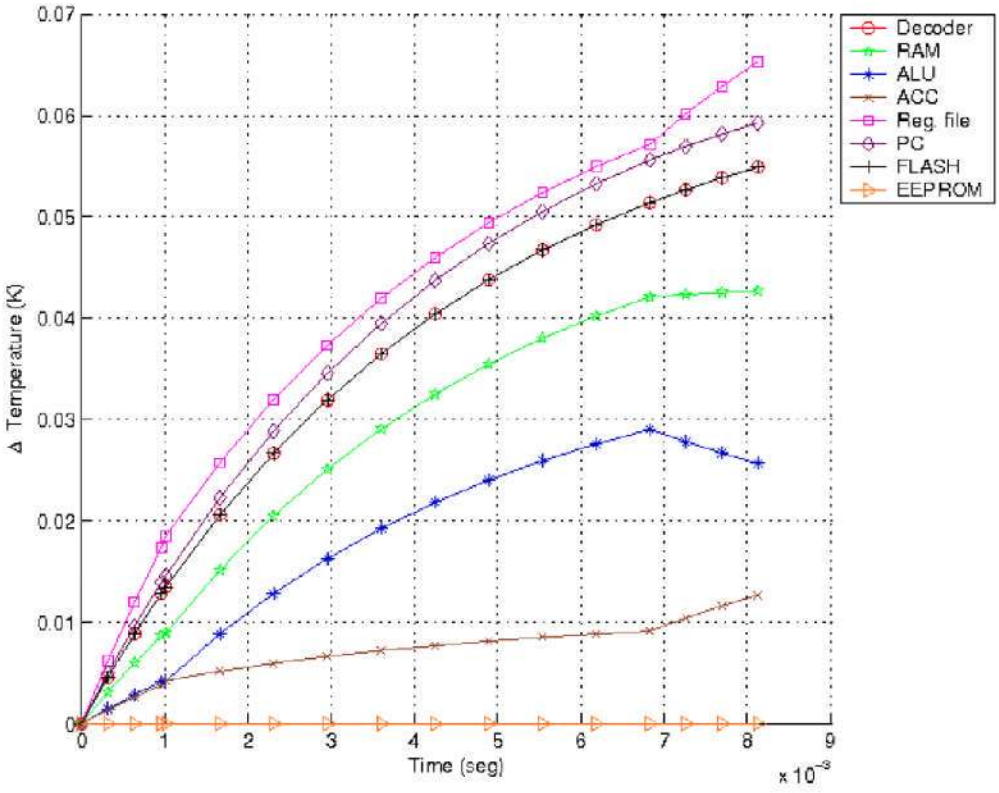

Fig. 8. Local thermal evolution for the dynamic memory implementation (image processing algorithm).

Fig. 5 shows the evolution in time of the total temperature in chip when the four different implementations of the image processing algorithm are executed. It can be observed that there are few differences in the way that the temperature behaves in time for all the implementations. The results obtained for the dijkstra application show a similar behavior. After this analysis, we can conclude that the studied code transformations do not have a representative impact in the global behavior of the temperature in the chip.

However, it is clear that there are absolute temperature differences between implementations. Such differences are caused by the execution time required to finish the applications. For example, the loop-splitted implementation is executed seven times faster than the pipeline implementation. This speed up in the execution causes that the chip does not reach the high temperatures suffered during the execution of other benchmark implementations. Therefore, the loop-splitted implementation can be considered a low-temperature transformation.

However, as will be shown later, there are source code transformations for which the reduction of the execution time and the temperature of the chip has the cost of an increase in the power consumed by the application. Therefore, there is a trade-off for the power consumed and the temperature in the chip that the designer (or the compiler) has to take into account.

Fig. 6 shows the power consumed and the temperature reached in the chip during the execution of the four different implementations. As can be seen, the loop-split implementation is able to reduce the total temperature of the chip by a 72% without any impact on the power consumption. However, other benchmark versions, as the pipelined implementation, decrease the power consumed by the application but increase the temperature. Therefore, the temperature-aware compilation is not always followed by the low-power execution.

Fig. 7 shows the results of the same analysis for the dijkstra application. The total temperature in the chip is again reduced by the application of the loop-splitting transformation, but the effect is less dramatic since the dijsktra algorithm is not strongly loop dominated. However, in this algorithm the pipelined implementation achieves an increase in the power consumed and a decrease in the total temperature of the chip, opposite to the results shown for the image processing algorithm. The use of unbalanced pipeline stages in the dijkstra application, as well as the impact of the memory transactions used to communicate the pipeline stages, explain this result.

4.2. Local analysis

When executing an application, not all the functional units are used at the same time, and probably workloads are different between them. This leads to different temperatures in different points of the chip, what is translated into temperature

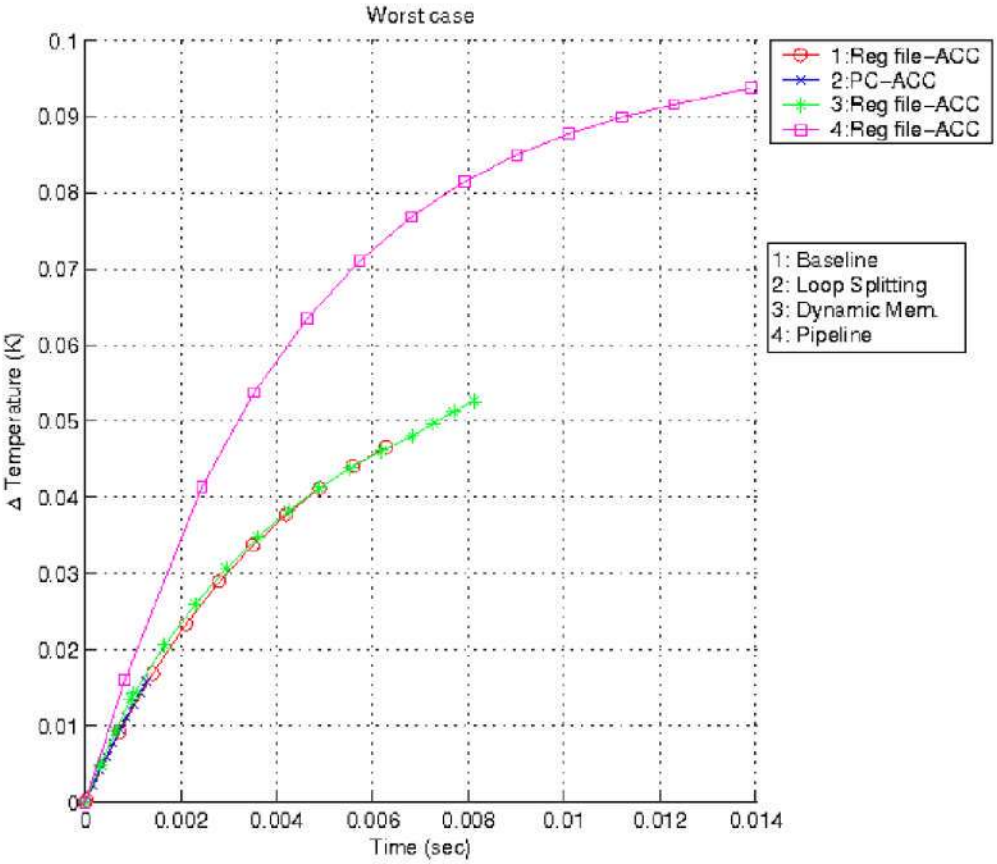

Fig. 9. Maximum expected temperature gradients (image processing algorithm).

gradients. If only the global warming of the chip is observed, these temperature gradients can be probably overlooked and cause several problems to the chip. For example, high temperature gradients can cause problems such as electromigration or thermal diffusion on functional units. Quick variations of the gradients could also cause another kind of technological damages. Therefore, analyzing what functional units are warmer during the execution of an application and the consequent temperature gradients, it is possible to establish techniques and rules that could be included into a temperature-aware compiler to make temperature inside the chip more uniform.

Fig. 8 presents the thermal evolution of all the functional units considered in the microcontroller when executing the dynamic memory implementation of the image processing benchmark. The figure represents the increment in temperature (relative value) with respect to the ambient temperature. As can be seen in the figure, Register File, Program Counter, FLASH instruction memory and Decoder unit show the highest temperature values. This behavior can be explained, on the one hand, because FLASH, Program Counter, and the Register file are used in every cycle when fetching a new instruction. On the other hand, the Decoder unit is always used before executing a new instruction. The zero value of the EEPROM unit shows that it is not used during the execution, so it maintains its initial temperature.

To study the effects of the code transformations on the thermal evolution of different functional units and their impact on the chip, we have considered two types of thermal gradients: First, we have studied the maximum gradient that can appear in the chip, that is, the worst case gradient. Then, we have analyzed a typical gradient.

4.2.1. Worst case analysis

The maximum expected temperature gradient in the chip is the temperature difference between its hottest and its coolest units. Observing this gradient, it is possible to find the worst thermal stress into the chip and to apply the source code modifications in a way that this effect is minimized. Fig. 9 shows maximum gradients for the four implementations of the image processing algorithm. In this figure, for every implementation of the algorithm (baseline, loop splitting, dynamic memory and pipeline) it has been plotted the maximum temperature gradient between two functional units (Register File vs. Accumulator, Program Counter vs. Accumulator, Register File vs. Accumulator and Register File vs. Accumulator, respectively). It can be seen that in the pipelined implementation there is an important temperature gradient between the Register File and the Accumulator of the ALU. In the rest of the implementations, their maximum gradients are much lower. The reason for

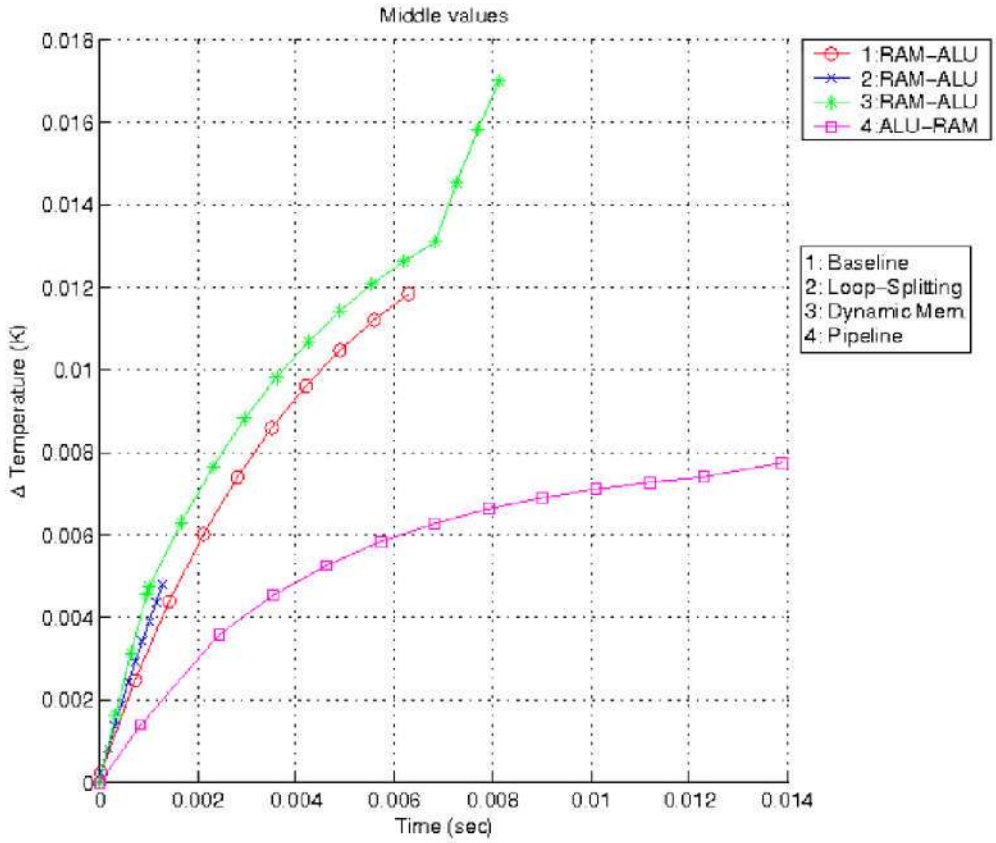

Fig. 10. Typical expected temperature gradients (image processing algorithm).

these differences is that the transfers between different memory spaces are massive in the pipeline implementation. These transfers are made using indirect addressing, what makes use of some special registers located into the Register file. This massive use of registers causes a higher heat-up of the Register file block.

Therefore, the pipelined implementation can be applied to a reduced portion of the source code (local optimization) and, in this way, the effect of the temperature gradient is minimized as being restricted to a shorter execution time. Also, an alternative source code transformation that cools down these units can be employed in parallel with the pipeline code modification.

4.2.2. Typical case analysis

It is also interesting to analyze the thermal gradient existing between functional units that do not have extreme temperatures. This will be the situation of most of the blocks during the execution of a benchmark and needs to be properly characterized. Fig. 10 shows the typical expected temperature gradients for the four implementations of the image processing algorithm. In this case, the dynamic memory implementation presents the highest temperature gradient due to the amount of memory transactions, highly increased with this transformation.

Apart from the pipeline implementation, this figure presents other interesting results. The gradient profile of the dynamic memory implementation shows one quick gradient variation. This variation coincides with the operations of memory release. During these operations, the ALU has little use in favor of the memory transferences. Therefore, this analysis can be taken into account in order to propose a scheduling of source code transformations that homogenizes the thermal behavior in the functional units of the chip.

An example of this homogenization policy for the image processing algorithm is shown in Fig. 11. As can be seen, the combination of the loop-splitting transformation with the use of dynamic memory is able to achieve a better response in terms of thermal gradients. The use of the loop-splitting transformation produces a heating in the RAM of the system during the initial time which reduces the transient in the temperature when the memory begins to be stressed. The sharp temperature transitions are the main factor in the chip damage and must be avoided.

Similar results in terms of thermal gradients have been found for the dijkstra application. The combination of the loop-splitting transformation with the use of dynamic memory is able to reduce the thermal differences between the processing units in the chip. Moreover, as Fig. 12 shows, the use of the loop-splitting technique can reduce the impact of the dynamic

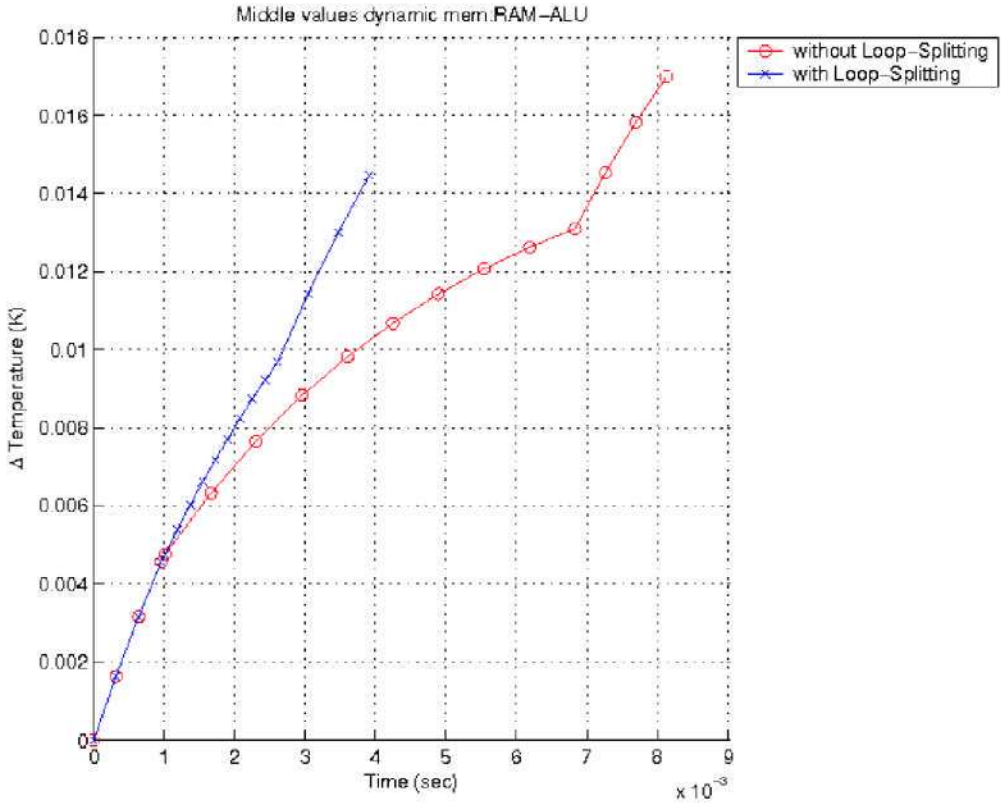

Fig. 11. Typical expected temperature gradients before and after homogenization policy (image processing algorithm).

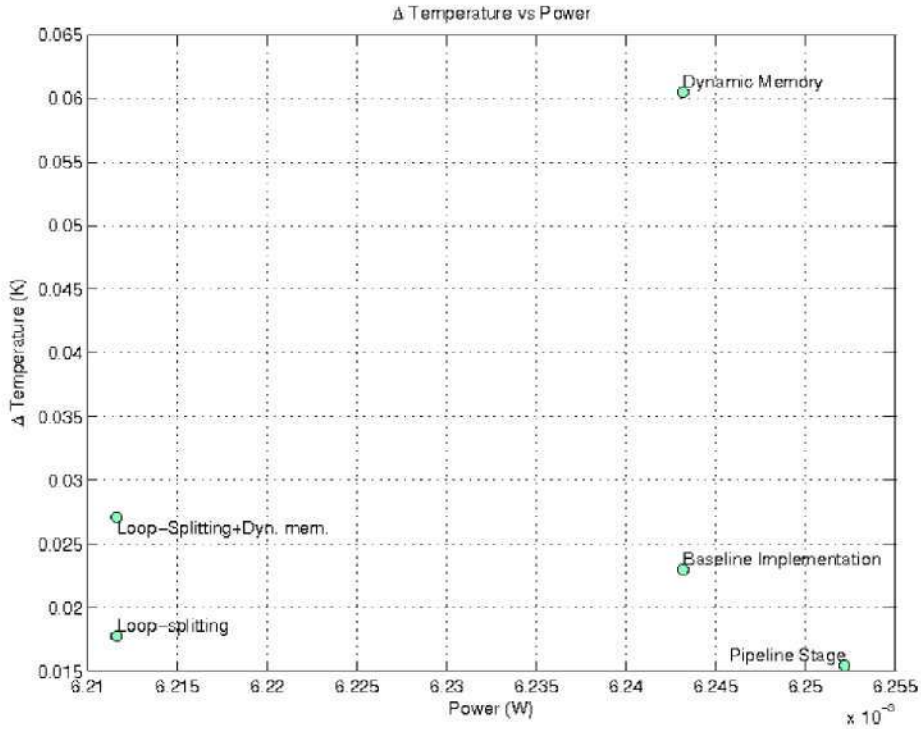

Fig. 12. Power consumption vs. Total temperature in the chip (dijkstra application).

memory transformation in the total temperature of the chip and power consumed by the application. Therefore, the negative impact in terms of power and temperature of the applications implemented with dynamic memory routines can be reduced by the subsequent application of the loop-splitting transformation.

Our ongoing work in this field is focused on increasing the set of source code transformations with a characterized thermal behavior and looking for an analytical way to apply these transformations in the thermal homogenization problem.

5. Conclusions

The thermal analysis in embedded systems is nowadays one of the main problems that the designers of embedded systems have to solve before thermal-aware system can be devised. Moreover, the way the application code is written, and the way the compilers manage this code, has a strong impact in the thermal behavior of the system.

This paper has presented an efficient way to characterize a processor-based system from a thermal point of view. The proposed methodology is target-independent and does not require a deep knowledge of the implementation details. The proposed methodology has been applied to the thermal characterization of several source code transformations. The analysis of this effect is a precious information to be considered for achieving the temperature-aware compilation.

Acknowledgement

This work is supported by the Spanish Government Research Grant TIN2008-508.

References

[1] Brooks D, Martonosi M. Dynamic thermal management for high-performance microprocessors. In: International symposium on high-performance computer architecture; 2001.
[2] Donald J, Martonosi M. Temperature-aware design issues for smt and cmp architectures. In: Workshop on complexity-effective design; 2004.
[3] <http://www.hpl.hp.com/research/dca/smartcooling/>.
[4] Huang W, Stan MR, Skadron K, Sankaranarayanan K, Ghosh S, Velusamy S. Compact thermal modeling for temperature-aware design. In: Design automation conference; 2004.
[5] Su H, Liu F, Devgan A, Acar E, Nassif S. Full leakage estimation considering power supply and temperature variations. In: International symposium on low power electronics and design; 2003.
[6] Li P, Pileggi L, Ashegi M, Chandra R. Efficient full-chip thermal modeling and analysis. In: International conference on computer-aided design; 2004.
[7] Huang W, Humenay E, Skadron K, Stan M. The need for a full-chip and package thermal model for thermally optimized IC designs. In: International symposium on low power electronics and design; 2005.
[8] Huang W, Stan M, Skadron K. Parameterized physical compact thermal modeling. IEEE Trans Compon Packag Manufact Technol 2005;28(4):615–22.
[9] Lopez-Buedo S, Garrido J, Boemo EI. Dynamically inserting, operating, and eliminating thermal sensors of FPGA-based systems. IEEE Trans Compon Packag Technol 2002;25(4):561–6.
[10] Julien N, Laurent J, Senn E, Martin E. Power consumption modeling and characterization of the ti c6201. IEEE Micro 2003;23(5):40–9.
[11] Senn E, Laurent J, Julien N, Martin E. Softexplorer: Estimation, characterization, and optimization of the power and energy consumption at the algorithmic level. In: International workshop on power and timing modeling, optimization and simulation; 2004.
[12] Srinivasan J, Adve SV. Predictive dynamic thermal management for multimedia applications. In: International conference on supercomputing; 2003.
[13] Heo S, Barr K, Asanovic K. Reducing power density through activity migration. In: International symposium on low power electronics and design; 2003.
[14] Sankaranarayanan K, Velusamy S, Stan M, Skadron K. A case for thermal-aware floorplanning at the microarchitectural level. J Instruct Level Parallel 2005;7.
[15] Narayanan SHK, Chen G, Kandemir M, Xie Y. Temperature-sensitive loop parallelization for chip multiprocessors. In: International conference on computer design; 2005.
[16] PIC16F873 Datasheet. Microchip; 1997.
[17] Skadron K, Sankaranarayanan K, Velusamy S, Tarjan D, Stan M, Huang W. Temperature-aware microarchitecture: modeling and implementation. ACM Trans Archit Code Optim 2004;1(1):94–125.
[18] Krum A. The CRC handbook of thermal engineering. CRC Press; 2000 [Ch: Thermal management].
[19] Skadron K, Abdelzaher T, Stan MR. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In: International symposium on high-performance computer architecture; 2002.
[20] Moyer B. Low-power design for embedded processors. Proc IEEE 2001;89(11):1576–87.
[21] <http://www.eecs.umich.edu/mibench/>.