Thermal Balancing Policy for Multiprocessor Stream Computing Platforms

Fabrizio Mulas, David Atienza, *Member, IEEE,* Andrea Acquaviva, *Member, IEEE,*
Salvatore Carta, *Member, IEEE,* Luca Benini, *Fellow, IEEE,* Giovanni De Micheli, *Fellow, IEEE*

***Abstract*—Die-temperature control to avoid hotspots is increasingly critical in Multiprocessor System-on-Chip (MPSoCs) for stream computing. In this context, thermal balancing policies based on task migration are a promising approach to re-distribute power dissipation and even out temperature gradients. Since stream computing applications require strict quality of service and timing constraints, the real-time performance impact of thermal balancing policies must be carefully evaluated. In this paper we present the design of a lightweight thermal balancing policy, `MiGra`, which bounds on-chip temperature gradients via task migration. The proposed policy exploits run-time temperature as well as workload information of streaming applications to define suitable run-time thermal migration patterns, which minimize the number of deadline misses. Furthermore, we have experimentally assessed the effectiveness of our thermal balancing policy using a complete Field-Programmable Gate Array (FPGA)-based emulation of an actual 3-core MPSoC streaming platform coupled with a thermal simulator. Our results indicate that `MiGra` achieves significantly better thermal balancing than state-of-the-art thermal management solutions, while keeping the number of migrations bounded.**

***Index Terms*—multi-processor architectures, systems-on-chip, thermal balancing, stream computing, task migration.**

I. Introduction

Multiprocessor System-on-Chip (MPSoC) performance in aggressively scaled technologies will be strongly affected by thermal effects. Power densities are increasing due to transistor scaling, which reduces chip surface available for heat dissipation. Moreover, in a MPSoC, the presence of multiple heat sources increases the likelihood of temperature variations over time and chip area rather than just a uniform temperature distribution across the entire die [1]. Overall, it is becoming of critical importance to control temperature and bound the on-chip gradients to preserve circuit performance and reliability in MPSoCs.

This work was partially supported by the Swiss Confederation through the Nano-Tera.ch NTF Project nr. 123618 - CMOSAIC and the Spanish Government Research Grants TIN2005-5619 and TIN2008-00508.

F. Mulas and S. Carta are with DMI - University of Cagliari, 32 Via Ospedale, Cagliari, Italy. E-Mail: fabrizio.mulas@sc.unica.it and salvatore@unica.it

D. Atienza is with ESL - Ecole Polytechnique Fédérale de Lausanne (EPFL), Station 11, EPFL-STI-IEL-ESL, Lausanne, Switzerland; and DACYA - Complutense Univ. of Madrid (UCM), Avda. Complutense s/n, Madrid, Spain. E-Mail: david.atienza@epfl.ch

A. Acquaviva is with DIA - Politecnico Di Torino, Corso Duca degli Abruzzi 24, Torino, Italy. E-Mail: andrea.acquaviva@polito.it

L. Benini is with DEIS - University of Bologna, Viale Risorgimento 2, Bologna, Italy. E-Mail:luca.benini@unibo.it

G. De Micheli is with LSI - EPFL, Station 14, EPFL-IC-ISIM-LSI, Lausanne, Switzerland. E-Mail: giovanni.demicheli@epfl.ch

Thermal-aware policies have been developed to promptly react to hotspots by migrating the activity to cooler cores [17]. However, only recently temperature control and balancing has gained attention in the context of chip multiprocessors [13], [4], [2]. A key finding from this line of research is that thermal balancing does not come as a side effect of energy and load balancing. Thus, thermal management and balancing policies are not the same as traditional power management policies [2], [5].

Task and thread migration have been proposed to prevent thermal runaway and to achieve thermal balancing in general-purpose architectures for high-performance servers [13], [5]. In the case of embedded MPSoC architectures for stream computing (signal processing, multimedia, networking), which are tightly timing constrained, the design restrictions are drastically different. In this context, it is critical to develop policies that are effective in reducing thermal gradients, while at the same time preventing *Quality-of-Service (QoS)* degradation due to task deadline misses caused by task migrations. Moreover, these MPSoCs typically feature non-uniform, non-coherent memory hierarchies, which impose a non-negligible cost for task migration (explicit copies of working context are required). Hence, it is very important to bound the number of migrations for a given allowed temperature oscillation range.

We propose a novel thermal balancing policy, i.e., `MiGra`, for typical embedded stream-computing MPSoCs. This policy exploits task migration and temperature sensors to keep the core temperatures within a predefined range, defined by an upper and a lower threshold. Furthermore, the policy dynamically adapts the absolute values of the temperature thresholds depending on average system temperature conditions. This feature, rather than defining an absolute temperature limit as in hotspot-detection policies [13], [2], [17], allows the policy to keep the temperature gradients controlled even at lower temperatures. In practice, `MiGra` adapts to system load conditions, which affect the average system temperature.

To evaluate the impact of `MiGra` on the QoS of streaming applications, we developed a complete framework with the necessary hardware and software extensions to allow designers to test different thermal-aware *Multiprocessor Operating Systems (MPOS)* implementations running onto emulated real-life multicore stream computing platforms. The framework has been developed on top of a cycle-accurate MPOS emulation framework for MPSoC [14]. To the best of our knowledge, this is the first multiprocessor platform that supports OS and middleware emulation at the same time as it enables a complete run-time validation of closed-loop thermal balancing policies.

Using our emulation framework, we have compared `MiGra` with other state-of-the-art thermal control approaches, as well as with energy and load balancing policies, using a real-life streaming multimedia benchmark, i.e., a Software-Defined FM Radio application. Our experiments show that `MiGra` achieves thermal balancing in stream computing platforms with significantly less QoS degradation and task migration overhead than other thermal control techniques. Indeed, these results highlight the main distinguishing features of the proposed policy, which can be summarized as follows: i) Being explicitly designed to limit temperature oscillations within a given range using sensors, `MiGra` performs task migrations only when needed, avoiding unnecessary impact on QoS; ii) for a given temperature-control capability, `MiGra` provides a much better QoS preservation than state-of-the-art policies by bounding the number of migrations; iii) `MiGra` is capable of very fast adaptation to changing workload conditions thanks to dynamic temperature-thresholds adaptation.

The rest of this paper is organized as follows. In Section II, we overview related work on thermal modeling and management techniques for MPSoC architectures. In Section III we summarize the software/hardware characteristics of MPSoC stream computing platforms. In Section IV we describe the implemented task migration support for these platforms, and the developed thermal emulation flow is presented in Section V. Then, in Section VI we present the proposed thermal balancing policy and, in Section VII, we detail our experimental results and compare with state-of-the-art thermal management strategies. Finally, in Section VIII, we summarize the main conclusions of this work.

II. Related Work

In this section we first review the latest thermal modeling approaches in the literature. Then, we overview state-of-the-art thermal management policies and highlight the main research contributions of this work.

Background on Thermal Modeling and Emulation: Regarding thermal modeling, as analytical formulas are not sufficient to prevent temperature induced problems, accurate thermal-aware simulation and emulation frameworks have been recently developed at different levels of abstraction. [1] presents a thermal/power model for super-scalar architectures. Also, [20] outlines a simulation model to analyze thermal gradients across embedded cores. Then, [21] explores high-level methods to model performance and power efficiency for multicore processors under thermal constraints. Nevertheless, none of the previous works can assess the effectiveness of thermal balancing policies in real-life applications at multi-megahertz speeds, which is required to observe the thermal transients of the final MPSoC platforms. To the best of our knowledge, this work is the first one that can effectively simulate closed-loop thermal management policies by integrating a software framework for thermal balancing and task migration at the MPOS level with an FPGA-based thermal emulation platform.

Background on Thermal Management Policies: Several recent approaches focus on the design of thermal management policies. First, static methods for thermal and reliability management exist, which are based on thermal characterization at design time for task scheduling and predefined fetch toggling [10], [1]. Also, [9] combines load balancing with low power scheduling at the compiler level to reduce peak temperature in *Very Long Instruction Word (VLIW)* processors. In addition, [11] introduces the inclusion of temperature as a constraint in the co-synthesis and task allocation process for platform-based system design. However, all these techniques are based on static or design-time analysis for thermal optimization, which are not able to correctly adjust to the run-time behavior of embedded streaming platforms. Hence, these static techniques can incur many deadline misses and do not respect the real-time constraints of these platforms.

Regarding run-time mechanisms, [5] and [17] propose adaptive mechanisms for thermal management, but they use techniques of a primarily power-aware nature, focusing on micro-architectural hotspots rather than mitigating thermal gradients. In this regard, [19] investigates both power- and thermal-aware techniques for task allocation and scheduling. This work shows that thermal-aware approaches outperform power-aware schemes in terms of maximal and average temperature reductions. Also, [18] studies the thermal behavior of low-power MPSoCs, and concludes that for such low-power architectures, no thermal issues presently exist and power should be the main optimization focus. However, this analysis is only applicable to very low-power embedded architectures, which have a very limited processing power, not sufficient to fulfill the requirements of the MPSoC stream processing architectures that we cover in this work. Then, [12] proposes a hybrid (design/run-time) method that coordinates clock gating and software thermal management techniques, but it does not consider task migration, as we effectively exploit in this work to achieve thermal balancing for stream computing.

Task and thread migration techniques have been recently suggested in multicore platforms. [13] and [16] describe techniques for thread assignment and migration using performance counter-based information or compile-time precharacterization. Also, thermal prediction methods using history tables [3] and recursive least squares [4] have been proposed for MPSoCs with moderate workload dynamism. However, all these run-time techniques target multi-threaded architectures with a cache coherent memory hierarchy, which implies that the assumed performance cost of thread migration and misprediction effects are not adapted to MPSoC stream platforms. Conversely, in this work we target specifically embedded stream platforms with a non-uniform memory hierarchy, and we propose accordingly a policy that minimizes the number of deadline misses and expensive task migrations, outperforming existing state-of-the-art thermal management policies.

Main contribution of this work: The main contribution of this work is the development of a thermal balancing policy with minimum QoS impact. Thermal balancing aims at reducing temperature gradients and average on-chip temperature even before the panic temperature is reached, thus improving reliability. Traditional run-time thermal management techniques, such as `Stop&go`, act only when a panic temperature is reached, thus they are not able to reduce temperature gradients, because in presence of hotspots there could be only one core very hot while others are cold. Moreover, `Stop&go` imposes large temporal gradients as the main counter-measure is to shut-off the processor when its temperature overcomes a panic threshold. Conversely, our policy (`MiGra`) acts proactively, as it is triggered also in normal conditions, when the temperature is lower than the panic. Upon activation, it migrates tasks to processors to flatten the temperature. While this improves reliability, a potential performance problem can arise, since balancing is achieved through task migrations. Thus, we have quantified the overhead imposed by migrations in a realistic emulation environment and a QoS-sensitive application, thus proving the effectiveness of the proposed policy to achieve better thermal balancing and less migration overhead than the previously mentioned state-of-the-art run-time thermal control and thermal balancing strategies. This result is obtained by `MiGra`'s capability to exploit temperature sensors to detect both large positive and negative deviations from the current average chip temperature. Moreover, the lightweight migration support implementation allows to bound migration costs.

III. Stream Computing Platforms

Stream computing platforms are distributed memory architectures where each core has its own local memory for storing code and data. A shared memory is also present, typically off-chip, for allocating large buffers. In fact, stream applications are very representative of the type of execution requirements of many multimedia MPSoCs nowadays, which possess quite demanding computation needs at the same time as soft-real time requirements [25], [27], [19]. In typical stream computing platforms, the considered target MPSoC exploits shared memory to implement the communication between tasks. In homogeneous platforms, as the 3-core streaming MPSoC we are targeting in this work (see Section VII-A), all cores are identical and run user-level tasks. However, from the software support viewpoint, we implemented a master-slave configuration where one core runs the centralized thermal balancing policy.

In stream computing architectures, each core runs from the private memory its own instance of a customized version of a light operating system, which is optimized for fast inter-processor communication. Then, to support MPOSes in stream computing, dedicated hardware must be designed to support OS execution and communication among processes running on different processing cores. This includes: i) inter-processor interrupt controller; ii) semaphore memory through hardware mutexes; iii) address translator as the memories of each core have non-overlapping address ranges; iv) frequency and voltage scaling support, which is included to effectively reduce power consumption as the workload of the MPSoC changes over time. Therefore, the MPOS can dynamically set the frequency of the cores at run-time.

From the software viewpoint, streaming applications are composed of multiple tasks communicating data and synchronizing with each other using a message passing paradigm.

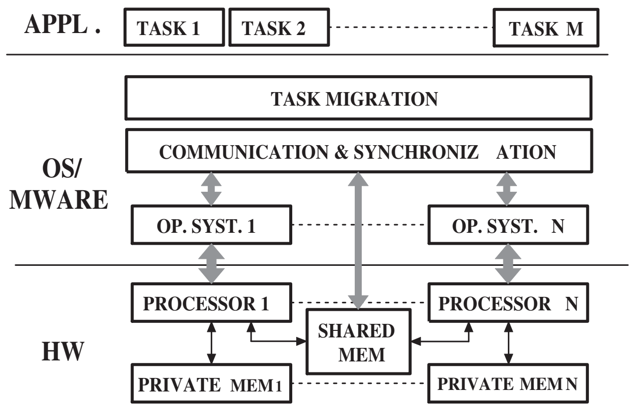

Fig. 1. Scheme of the software abstraction layer

Then, tasks are spread on the various cores, depending on resource availability, to exploit the architecture parallelism. As such, communication takes place using the inter-processor buffers located in private memories or shared memory. In our case we exploit the shared memory for storing message queues. Indeed, streaming applications follow a data-flow oriented paradigm, where tasks continuously process frames arriving in the input queue and make available frames on the output queue for the next processing stage (cf. Section VII-B).

The programming model adopted in stream computing assumes that each task is represented using the process abstraction. This means that each task has its own private address space. Hence, task communication is carried on using a dedicated shared memory area controlled by a distributed MPOS. The overall software abstraction layer is described in Figure 1. It is based on three main components: (i) Stand alone OS for each processor running in private memory; (ii) lightweight middleware layer providing data sharing/synchronization and communication services; (iii) task migration support layer for distributed MPOS control.

Finally, a frequent communication scheme in stream computing is message passing through mailboxes. Thus, this is the paradigm we have adopted in our baseline MPSoC stream computing architecture. We developed a lightweight message passing scheme able to exploit scratch-pad memories or physical shared memories to implement ingoing mailboxes for each processor core. For our experiments, we defined a library of system calls that each process can use to perform blocking write and read of messages on data buffers.

IV. Task Migration Support

To enable task migration, we implemented two different migration strategies, which differ in the way the memory is managed by the middleware. Our MPOS framework is based on a customized version of uClinux [22], which is a light operating system that we have extended for very fast inter-processor communication and run-time task migration. uClinux includes a Linux 2.x kernel release intended for cores without *Memory Management Unit (MMU)*, as well as a collection of user applications and libraries, which makes it very suitable for fast multiprocessor synchronization with limited overhead. Furthermore, we have integrated into uClinux

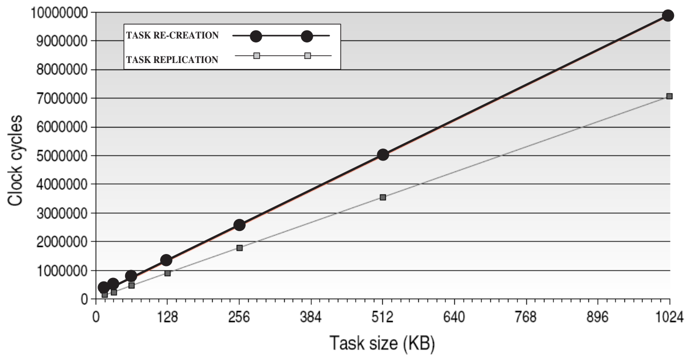

Fig. 2. Migration cost as a function of task size for task-replication and task-recreation.

additional support for communication, synchronization and task migration using shared memory on a distributed MPOS middleware layer running on top of each MPSoC processor. Moreover, to emulate the combined effect of frequency scaling policies with task migration, hardware programmable dividers have been placed in the output of the clock generators to obtain a configurable speed setting support in our FPGA-based MPSoC emulation platform (cf. Section V). Therefore, the MPOS can set the frequency of all the cores at run-time by accessing the memory locations where the dividers are mapped.

Then, migration is allowed only at predefined checkpoints provided to the user through a library of functions together with message passing primitives, and a *master daemon* runs in one of the cores and takes care of dispatching tasks on the processors. A first version, based on a *task-recreation* strategy, kills the process on the processor of origin and recreates it from scratch on the target processor. This strategy only works in OSes supporting dynamic loading, such as uClinux. Task recreation is based on the execution of *fork-exec* system calls that takes care of allocating the memory space required for the incoming task, which is an option in stream computing, as the input dynamically changes with each input stream. Thus, we implemented an alternative migration strategy where a replica of each task is present in each local OS, called *task-replication*. Only one processor at a time can run one replica of the task. While in one processor the task is executed normally, in the other processors it is in a queue of suspended tasks, but a memory area is reserved for each replica in the local memory of each core. Hence, even if this latter strategy leads to a partial waste of local memory for migratable tasks, it is much faster, since it cuts down on memory allocation time with respect to a task recreation strategy.

During execution, when a task reaches a user-defined checkpoint, it checks for migration requests performed by the master daemon. If the migration is taken, the task is either suspended or killed (depending on the strategy); thus, it is left waiting to be deallocated and restored on another processor by the migration middleware. When the processor of origin decides to migrate a task, a dedicated shared memory space is used as a buffer for the task context transfer.

A quantification of the memory overhead due to task replication and recreation is shown in Figure 2. In this figure, the cost is shown in terms of processor cycles needed to perform a migration as a function of the task size. In both cases, part of the migration overhead is due to the amount of data transferred through the shared memory. For the task recreation technique, there is another source of overhead due to the additional time required to re-load the program code from the file system; thus, the offset that appears between the two curves. Moreover, the task recreation curve has a larger slope due to a larger amount of memory transfers, which leads to an increasing contention on the bus. Finally, we have experimentally measured the variation of the energy consumption cost due to migration, which indicates a maximum value of 10.344 mJ for a 1024 KB task size and a minimum one of 9.495 mJ for a value of 64 KB task size (both values are for a single migration cost). Thus, our migration approach produces a very limited energy migration overhead for different task sizes for both types of migration techniques. The analyzed overheads due to task migration for both execution time and energy consumption are included in the MPOS level to take the migration decisions, as explained in Section VI-A.

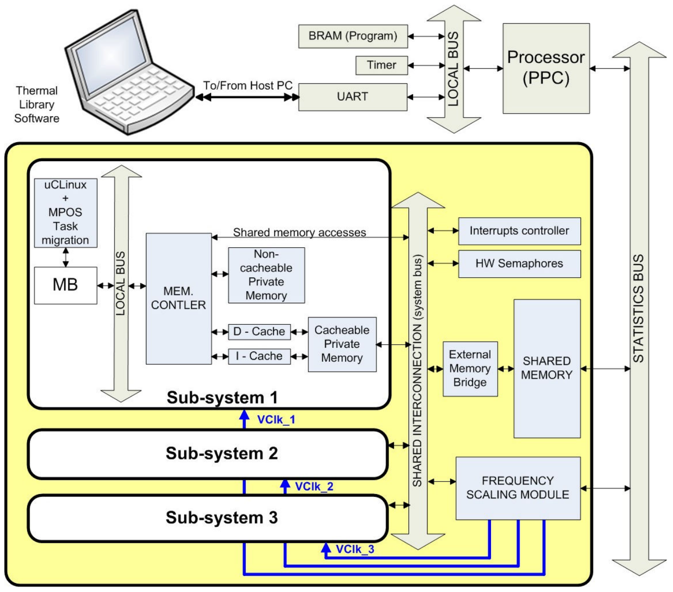

Fig. 3. Overview of the MPSoC thermal emulation framework for stream computing platforms

V. Thermal Emulation Flow of Stream Computing Platforms

To explore the effects of thermal management strategies on MPSoC thermal balancing, we need to evaluate the different strategies for realistic MPSoC-MPOS architectures. For this, we need to extract detailed statistics of hardware components, operating system and middleware operations for simulated time intervals long enough to be meaningful for thermal analysis. This cannot be easily achieved by software simulators. In this work, we leverage a complete FPGA-based thermal emulation infrastructure [23], extended in the directions detailed below. An overview of the extended framework is presented in Figure 3.

FPGA emulation is exploited to model the hardware components of the MPSoC platform at multi-megahertz speeds. The hardware architecture consists of a variable number of soft-cores (currently three cores, as required by the modeled MPSoC, shown in Figure 5) that are emulated on a Virtex-II Pro v2vp30 FPGA [24]). Then, the first extension of our framework with respect to [23] is that each core runs a customized version of uClinux OS [22] including the additional support described in Section III for global communication, synchronization and task migration. Thus, the MPOS assigns tasks to processing cores with a global system view, applies locally an OS-based DVFS scheme per core [5], and implements different thermal-aware task migration policies.

The second extension with respect to the thermal emulation framework presented in [23] is the addition of a specialized thermal monitoring subsystem, such that the run-time temperature of the emulated stream computing platform can be observed at the MPOS level. This new monitoring subsystem is based on hardware sniffers, a virtual clock management peripheral and a dedicated non-intrusive subsystem, which implements the extraction of statistics through a serial port. These statistics are provided to a software thermal simulation library for bulk silicon chip systems [23], which resides in a host workstation, and calculates the temperature of each cell according to the floorplan of the emulated MPSoC and the frequency/voltage of each Microblaze (MB) soft-core processor. Then, the temperatures coming out from the simulator provide a real-time temperature information visible by the running uClinux in each processor through emulated memory-mapped temperature sensors, which are updated by the thermal monitoring subsystem as configurable regular updates. In our experiments we have fixed this updating interval to 10 ms to guarantee very accurate thermal monitoring (see Section VII). Finally, thanks to a handshake mechanism between the thermal model and the MPOS middleware to synchronize the upload/download of temperatures, our extended framework implements a closed-loop thermal monitoring system, which enables exploring the impact of task migration and scheduling on system temperature balancing at multi-megahertz speed, and the observation of the real thermal transients of MPSoC stream platforms.

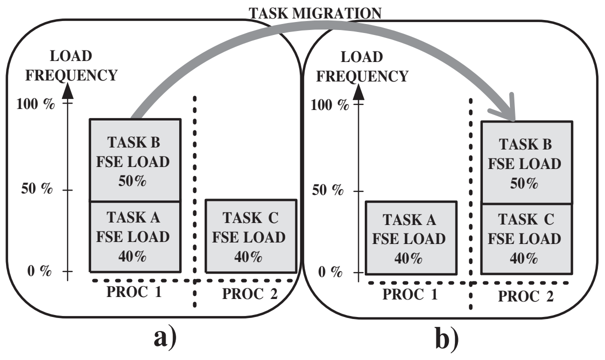

Fig. 4. Simple thermal balancing example between two cores

VI. Thermal Balancing for Stream Computing

In general, thermal balancing does not come as a side effect of energy balancing. In Figure 4.a, a typical situation where a two-core system running three tasks (A, B, C) is energy-balanced (but thermally-unbalanced) is shown. Both processors can independently set their frequency and voltage to reduce energy/power dissipation to the minimum required by the current load. Tasks are characterized by their *Full-Speed-Equivalent (FSE)* load, which is the load imposed by a task when the core runs at maximum frequency. Core 1 runs tasks A and B, having FSE of 50% and 40% respectively; core 2 runs task C that has a FSE of 40%. In this case core 1 can ideally scale its frequency to 90% of its maximum value, while core 2 can scale it to 40%. No better tasks mapping exists that further reduces energy/power dissipation. In this situation, due to the different power consumed, the temperature of core 1 will be higher than the temperature of core 2. Therefore, a thermally balanced condition can be achieved by periodically migrating task B from the first core to the second core [19] (as represented in Figure 4.b), obtaining, on average, an equalized workload on the two cores (i.e., 40% +50%/2 = 65%). If the temperature variations caused by migrations are slower than the migration period, a temperature close to the average workload (i.e, 65%) will be achieved on both cores. Although this is a simplified case, it outlines that the main challenge of a thermal balancing algorithm is the selection of the task sets to migrate between cores, such that the overall temperature is balanced, while keeping migration costs bounded.

A. `MiGra`*: Thermal Balancing Algorithm*

The thermal balancing strategy we propose in this paper, `MiGra`, is inspired by [14]. To prevent impact on QoS caused by migration, `MiGra` is based on run-time estimation of migration costs to filter migration requests driven by temperature differences between cores. Thus, `MiGra` considers performance and energy migration costs caused by the underlying migration infrastructure (cf. Section IV). Moreover, in our implementation, `MiGra` lies on top of a *Dynamic Voltage and Frequency scaling (DVFS)* policy [5]. Thus, the power consumption of a task can be roughly estimated at run-time by assuming that it is proportional to its load (cf. Figure 2).

`MiGra` implements a strategy that tries to bound the temperature of each processor around the current average temperature, as well as minimizing the overhead in terms of number of migrated tasks and amount of data transferred between the cores due to migrations. Therefore, a maximum distance of the temperature of each processor from the current average temperature is defined by `MiGra`, identifying a range of permissible temperatures for each single processor between an upper and a lower threshold. These thresholds are dynamically adapted at run-time according to the current workload. `MiGra` also control thermal runaway by stopping the core that reaches a temperature above a predefined panic threshold. Nonetheless, this extreme situation should never occur in realistic streaming applications, and `MiGra`'s regular operation always keeps its upper threshold below this panic one, by trying to minimize temperature gradients. Each time the temperature of a processor reaches the upper threshold around the average temperature of the MPSoC platform, MiGra triggers a migration to move away a set of tasks from the hot processor to another processor having a temperature below the current average temperature. On the other side, each time the temperature of a processor reaches the lower threshold, a migration is triggered so that a set of tasks are moved to that processor from a hotter processor to reduce the overall MPSoC average temperature.

To reduce the amount of computations needed to select the tasks to move, MiGra implements an algorithm that moves tasks only between two processors at a time. Hence, the processor that triggers the migration (a hot one) will only select one target processor (a cold one) to balance the workload between them. Moreover, MiGra must minimize thermal gradients without increasing overall energy dissipation when tasks are migrated, as well as minimizing performance overhead in the final MPSoC. As a result, the thermal balancing algorithm implemented in MiGra consists of two phases.

In the first phase, the candidate processors (source and target) are selected, while in the second phase the task sets to be exchanged are defined. During the first phase, if all the following three conditions are verified, the dst processing core becomes a candidate to exchange workload with the src processing core:

- If the temperature of the source core is beyond the average temperature (t_{mean}), the destination core has to be below: $(t_{src} - t_{mean}) * (t_{dst} - t_{mean}) < 0$
- The frequency of the source core must be higher than the average if the one of the destination core is below: $(f_{src} - f_{mean}) * (f_{dst} - f_{mean}) < 0$
- The total overall power dissipated by the two cores (source and destination) after the migration has to be lower than the total power dissipated by the two core before the migration: $(f_{src} * v^2{}_{src} + f_{dst} * v^2{}_{dst})_{before\_migr} \geq (f_{src} v^2{}_{src} + f_{dst} * v^2{}_{dst})_{after\_migr}$

The first condition is about temperature and assures that the migration achieves reduction in average system temperature. However, the temperature condition cannot ensure that the candidate destination processor is currently highly-loaded, but just that its temperature transient is still to be stabilized (and most likely still growing). In fact, the temperature is not a good workload monitor if it is evaluated independently. Thus, in order to avoid an additional allocation of workload to a potentially overloaded core, we also need to evaluate its current frequency, which is proportional to the allocated workload. Hence, the migration is allowed only if the frequency of the candidate destination core, which represents its current workload, is lower than the mean frequency of all the cores in the system. As a result, we avoid that additional workload is allocated to a core that is currently highly-loaded, but its temperature is still low. Finally, the third condition of MiGra compares the total power of the source and destination cores before and after migration, making sure that the new overall power consumption on the MPSoC does not increase. In fact, while the previous conditions ensure that temperatures are stabilized (constraint 1) and no oscillations are caused by workload re-allocations (condition 2), this third condition indicates that thermal balancing is performed only if the new task allocation is not worse, from a power consumption perspective.

The result of this phase can be either one or multiple destination candidates for a certain source processor. Also, no pairs of candidates may exist, which occurs in case of perfect thermal balancing (i.e., all cores are at the same temperature). Thus, MiGra does not perform any migration and the rest of the algorithm is skipped.

Next, in the second phase of the thermal balancing algorithm of MiGra, the selection of the number of tasks and the final selection of the target processor is performed (in case several potential destination cores have been found for a specific source core in the first phase). This final selection of the destination processor and tasks depends on the evaluation of the migration costs (performance, energy and temperature increase estimation). As a result, our cost function is the product of the amount of data moved due to the migration by the frequency of migrations. Then, to estimate the appropriate migration frequency, given a certain temperature difference between two processors, the benefit of triggering a new migration is proportional to the difference between the current temperature of the target processor in the migration and the average on-chip temperature. Thus, the selected target processor of a migration (tgt_{sel}) is the processor with the minimum cost, according to the following cost function:

$$tgt_{sel} = arg \min_{tgt} \left\{ \frac{\sum_i^I (C^{src}{}_i) + \sum_j^J (C^{tgt}{}_j)}{(t_{tgt} - t_{mean})^2} \right\} \quad (1)$$

Where $C^{src}{}_i$ is the amount of data to move for the $i-th$ of I tasks running on the source processor, and $C^{tgt}{}_j$ is the amount of data to move for the $j-th$ of J tasks running on the tgt processor.

In the current implementation of MiGra, in order to reduce the run-time overhead of the aforementioned selection, we have included an additional optimization phase. It selects the set of tasks to be migrated according to the observation that the temperature-balancing benefit of migrating a task decreases together with its workload. Therefore, the larger the workload required by a task is, the more advantageous it is to migrate that task to balance the temperature in a processor. This approximation shows very good results and allows us to limit drastically the number of tasks to be considered for migration at run-time (only the 5-10 tasks requiring the highest loads in each processor are used in our experiments). Moreover, an exhaustive search comparing the migration cost of all possible combinations of tasks and candidate processors found in the first phase is not practical in real systems.

Finally, although in this work we specifically target the use of MiGra for MPSoC stream computing platforms, our thermal balancing algorithm does not make any specific assumption about the application domain itself. Therefore, it can be applied to any general-purpose application after a suitable pre-characterization phase of the task migration costs (as described in Section IV). Nonetheless, MiGra is not suited for hard real-time platforms at present (e.g., [6]), since it does not provide any guarantees about avoidance of deadline misses.

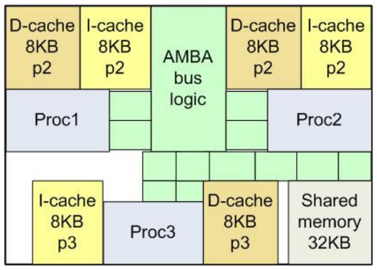

Fig. 5. Emulated 3-core MPSoC streaming architecture

VII. Thermal Balancing Policy Validation

We have assessed the benefits of `MiGra` for thermal balancing on the emulation framework using as case study an industrial 3-core MPSoC running a multi-task streaming application. Therefore, in the next sections we first describe the concrete instance of used MPSoC architecture, as well as the power figures and two different packaging models considered (Section VII-A). Then, we present the other state-of-the-art thermal management strategies evaluated in comparison with `MiGra` (Section VII-C). Lastly, we present the analysis of the thermal balancing capabilities of the different thermal management approaches with respect to temperature standard deviation, deadline misses and performance overhead. To this end, we have performed two sets of experiments. First, we have analyzed the behavior of `MiGra` and other basic temperature-limit control (`Stop&go`, see Section VII-C) and thermal balancing approaches when applied to stream MPSoC platforms with different thermal packages. This first set of experiments illustrates that thermal balancing cannot be achieved as a side effect of energy balancing policies or a standard thermal control policy, which is meant to react only when the chip reaches a panic temperature (i.e., a temperature where the system cannot operate without seriously compromising system reliability). Second, we have conducted exhaustive experiments to define the limits of `MiGra` and state-of-the-art thermal control approaches to minimize spatial thermal variations at run-time in highly variant (i.e., high-performance) stream MPSoCs, from the thermal gradient viewpoint.

Finally, in all the experiments, DVFS is always active and works separately in each processor (i.e., local DVFS [5]), and independently from the applied thermal balancing policy. In particular, in our 3-core MPSoC case study, the implemented DVFS scheme chooses the final frequency and voltage of each processor between ten different values in the range 100 MHz and 532 MHz, such that it tries to reduce the power consumption of the core by minimizing its idle time.

A. Stream MPSoC Case Study and Packaging Options

We focus on a homogeneous architecture, as presented in Figure 5. In particular, we consider a system based on three 32-bit RISC processors without MMU support to access cacheable private memories, and a single non-cacheable shared memory. It follows the structure envisioned for non-cache-coherent MPSoCs [25], [26]. In Table I, we summarize the values used for the components of our emulated MPSoC. The power values have been derived from industrial power models for a 90nm CMOS technology. On the software side, each core runs its own instance of the uClinux OS [22] in the private memory (see Section III for more details about the MPOS software infrastructure).

TABLE I
Power of components in 0.09 μm CMOS

	Max. Power@500 MHz
RISC32-streaming (Conf1)	0.5W (Max)
RISC32-ARM11 (Conf2)	0.27W (Max)
DCache 8kB/2way	43mW
ICache 8kB/DM	11mW
Memory 32kB	15mW

We considered two different packaging solutions. The first package shows temperature variations of around 10 degrees in few seconds [27], while the second packaging option shows similar thermal variations in less than a second. In Table II we enumerate the main thermal properties of these two different packaging options. Regarding package-to-air resistance, since the amount of heat that can be removed by natural convection in MPSoCs strongly depends on the environment (e.g., placement of the chip on the PCB), we have tuned these figures according to the experimental figures measured in our industrial 3-core case study [27], according to the final MPSoC working conditions indicated by our industrial partners.

TABLE II
Thermal properties of the different packages

silicon thermal conductivity	$150 \cdot \left(\frac{300}{T}\right)^{4/3} W/mK$
silicon specific heat	$1.945e-12 J/um^3K$
silicon thickness	$300um$
copper thermal conductivity	$400W/mK$
copper specific heat	$3.55e-12 J/um^3K$
copper thickness	$1000um$
package-to-air conduct. (low-cost)	$12K/W$
package-to-air conduct. (high-cost)	$1K/W$

B. Benchmark Application Description

We ported to our emulation framework different multi-task variations of the *Software FM Defined Radio (SDR)* benchmark, which is representative of a large class of streaming multimedia applications. The application model follows the Streamit application benchmarks [8], used as baseline for the implementation of our parallel SDR versions. This class of applications is characterized by tasks communicating by means of FIFO queues, as depicted in Figure 6, where tasks are graphically represented as blocks. As this figure shows, the output data of the tasks of the SDR application is stored in different buffers or queues ($Q_{x,y}$) and consumed at the required frame rate. Thus, a deadline miss occurs when the consumer (periodically) attempts to read a frame from the final buffer and it is empty.

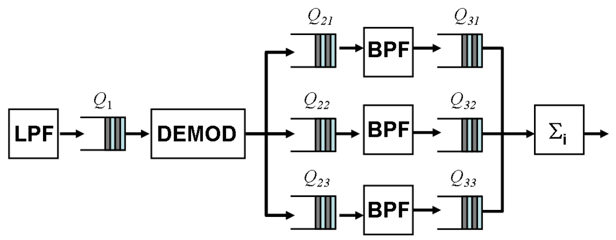

Fig. 6. SDR case study (six tasks version)

We performed two sets of experiments. In the first set we used a very dynamic workload made of multiple instances of the SDR application, using versions divided in three or six tasks (as in Figure 6). The input data to the SDR application represents samples of the digitalized PCM radio signal to be processed in order to produce an equalized base-band audio signal. In the first step, the radio signal passes through a *Low-Pass-Filter* (LPF) to cut frequencies over the radio bandwidth. Then, it is demodulated by the *demodulator* (DEMOD) to shift the signal at the baseband and produce the audio signal. The audio signal is then equalized with a number of *Band-Pass-Filters* (BPF) implemented with a parallel structure. Finally, the *consumer* (Σ) collects the data provided by each BPF and makes the sum with different weights (gains) in order to produce the final output. Communication among tasks is done using message queues.

C. Evaluated State-of-the-Art Thermal Control Policies

MiGra has been compared with the following state-of-the-art thermal management policies:

Energy-Balancing: This policy maps the SDR tasks to balance the energy consumption [17] among cores. Energy is computed from the frequency and voltage imposed by the running tasks, which are dynamically adjusted using DVFS [5].

Stop&Go: This policy prevents thermal runaway by shutting down a core when it reaches a panic temperature threshold. In its original version [13], the core execution is resumed after a predefined timeout. However, we modified this policy to fairly compare it with our thermal balancing algorithm, MiGra, by using the upper threshold of our algorithm as the panic threshold, and our lower threshold defines when to switch the core on instead of a fixed timing out value, which would be unable to adapt to very dynamic working conditions.

Rotation: This policy tries to achieve thermal balancing by performing migrations between cores in a rotatory fashion, at regular intervals. Thus, at the beginning of a task migration interval (i), a set of tasks in $core_j$ is migrated to $core_{(j+1)modN}$.

Temperature-Based (TB): This policy considers the migration of tasks between cores according to the temperature differences between each pair of processing cores in regular intervals, namely, the set of tasks running on the hottest core is swapped with the set on the coldest core, the set of tasks on the second hottest core is swapped with the one on the second coldest core, etc. Thus, at the beginning of each task migration process, the cores are ordered by temperature. Then, the set of tasks executed on $core_j$ is swapped with the set running on $core_{N-j-1}$.

TABLE III
SDR APPLICATION MAPPING

Core / freq.	*Task*	*Load* [%]
Core 1 (533 MHz)	BPF1	36,7
	DEMOD	28,3
Core 2 (266 MHz)	BPF2	60,9
	Σ	6,2
Core 3 (266 MHz)	BPF3	60,9
	LPF	18,8

Temperature-Based Threshold-limited (TB-Th): This policy is an enhancement of the previous TB policy, which was originally aimed to reduce peak temperature rather than thermal gradients. Therefore, we have introduced an additional minimum temperature threshold, which tries to minimize the number of unnecessary migrations of the original TB approach between cores when the worst temperature of the MPSoC has not reached a critical point. The minimum threshold has been carefully selected off-line to find the best option for each working condition of our sets of experiments.

In the following sections we assess the performance of MiGra with respect to the previously described policies in different workload conditions and for different types of packaging solutions in stream computing platforms.

D. Experimental Results: Exploration with Different Packaging Solutions

We compare MiGra, Stop&Go and the energy balancing task-migration policy implemented in many MPOSes, using a low- and high-cost thermal package. DVFS was also always active in the MPOS to adjust the power dissipated by each core to the required workload.

D.1) Thermal Balancing in Low-Cost Packaging MPSoCs: In the case of low-cost packaging, we observed that after a first execution phase (12.5 sec), the temperatures of the three cores stabilizes. However, it is not balanced and approximately 10^0C difference exists between the hottest (core 1) and the coolest core (core 3). This thermal state is due to the application of DVFS to each core. Moreover, although core 2 and 3 have the same frequency, their temperatures differ because of the different heat spreading capabilities due to their position in the floorplan (see Figure 5). Thus, in our experiments, we trigger our task-migration-based policy (MiGra) to achieve thermal balancing after this initial phase.

When MiGra is applied, each time a core reaches the upper threshold (set to three degrees more than the average temperature), a migration is triggered, one task is moved to a colder core, and the temperature becomes balanced for all cores within 1 second of execution of the SDR application. This demonstrates the effectiveness of our policy to balance temperature.Our results indicate that the hottest core temperature passes the upper threshold while balancing the temperature only for a very limited time (less than 400 ms).

A quantitative evaluation and comparison between our thermal-balancing policy (MiGra), Stop&Go and energy balancing algorithms is provided in the following experiments for the same packaging configuration. Figure 7 shows the temperature standard deviation for the three policies as a function of the threshold values. The X-axis indicates the distance of upper and lower threshold from the mean temperature. As this figure shows, the temperature deviation increases with the threshold. Thus, our policy is more effective in reducing temperature deviation than other techniques because it acts on both hot and cold cores. In particular, the manually-tuned Stop&Go does not improve the temperature of the cold cores. Furthermore, if the original Stop&Go is used [13], [5], considering the highest-supported temperature for the low-cost package as panic threshold, higher temperature swings are observed, which leads to a worst standard deviation value (3.70 ${}^{\circ}$ K more) with respect to those shown in Figure 7.

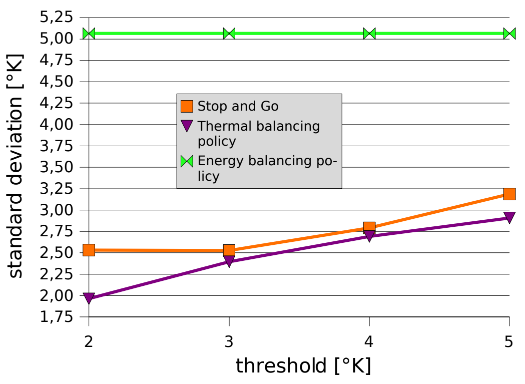

Fig. 7. Temp. standard deviation in low-cost embedded SoCs from the mean on-chip temperature (337 ${}^{\circ}$ K)

Then, Figure 8 shows the number of deadline misses as a function of the threshold values. As shown, our policy leads to few deadline misses while Stop&Go suffers a higher value of missed frames. Deadline misses may be caused by frozen tasks during migration; hence, inter-processor queues are depleted during migration, and if the queue of the last stage gets empty a deadline miss occurs. However, as Figure 8 illustrates, migration is lightweight and fast enough to limit this drawback. In fact, missed frames appear only for the minimum threshold considered in our experiments. Furthermore, we observed that the average queue level does not change because of migration; thus, a queue size handling thermal balancing can always be found and the SDR application can sustain thermal balancing without QoS impact, i.e., the minimum queue size to sustain migration in our experiments was 11 frames.

D.2) Thermal Balancing in High-Cost Packaging MPSoCs: To stress our policy when temperature variations are faster, we repeated the previous set of experiments using the alternative packaging value for high-performance systems (see Section VII-A), where temperature variations are 6$\times$ faster than the previous model. Hence, the 3-core case study experiences gradients of more than ten degrees, i.e, the coolest core typically operate at 56 ${}^{\circ}$ C and the hottest one can reach 67 ${}^{\circ}$ C.

Figure 9 shows the standard deviation of the temperature for the three tested policies. The energy balancing policy achieves very poor results and the modified Stop&Go policy behaves better in terms of temperature deviation, but it causes a large amount of deadline misses (Figure 10). Moreover, using the original version of Stop&Go [5] with the highest-

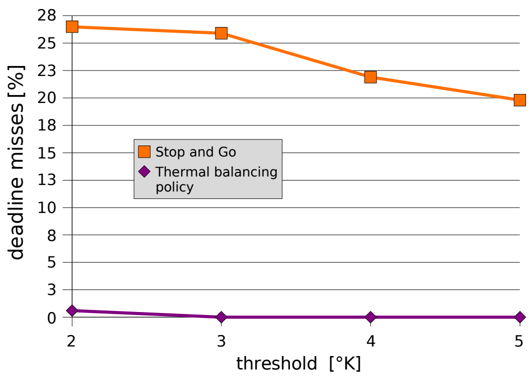

Fig. 8. Deadline misses for the embedded mobile system

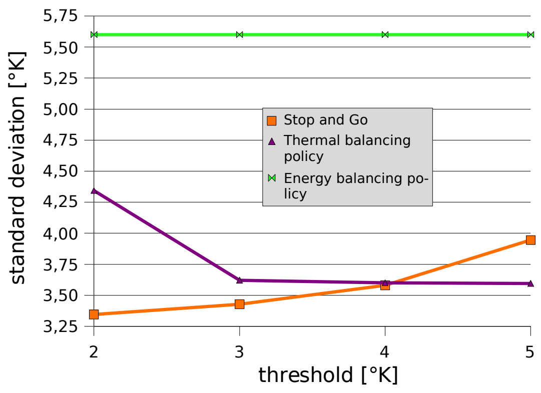

Fig. 9. Standard deviation in high-performance SoCs from the mean on-chip temperature (314 ${}^{\circ}$ K)

supported temperature of the high-performance package as panic threshold, a worst standard deviation value of 4.48 ${}^{\circ}$ K more is observed with respect to Figure 9.

On the contrary, although our algorithm makes temperature oscillate more than the modified Stop&Go (but significantly less than the original Stop&Go), it always causes very few deadline misses (less than 4%). Moreover, our algorithm starts behaving significantly better than Stop&Go when the threshold increases, as less migrations are triggered. Also, we observed that Stop&Go causes less deadline misses with the fast thermal model than with the slow one, due to the faster speed the lower threshold is reached after shutdown. From these experiments, we can conclude that pure software techniques cannot handle fast temperature variations, and a hardware-software co-design approach is needed.

Finally, Figure 11 depicts the average number of migrations per second performed by our thermal balancing policy (`MiGra`) for both mobile embedded and high-performance systems. As expected, the number of migrations is higher for high-performance systems. However, as each migration implies a transfer of 64 Kbytes of data (the minimum memory space allocated by the OS), the required three migrations per second are equivalent to $64 * 3 = 192$ Kbytes per second, which means that our task migration policy implies only a negligible overhead in system performance (1% overall).

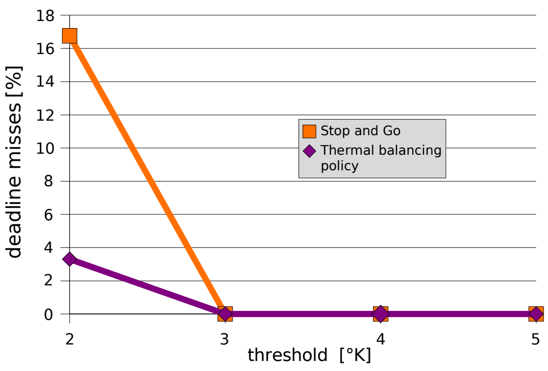

Fig. 10. Deadline misses for high-performance systems

E. Experimental Results: Limits of Thermal Balancing Techniques for High-Performance MPSoCs

In this set of experiments we perform evaluation of the limits of `MiGra` and state-of-the-art task migration policies, i.e., Rotation, TB and TB-Th (see Section VII-B for more details). In all the cases, local DVFS is also active and applied, when possible, in addition to each particular task migration scheme. To stress the reacting capabilities of all these schemes, in this set of experiments we have used the high-performance packaging option, which exhibits faster vertical on-chip heat flow dissipation to the environment than spreading horizontally to other parts of the chip. Thus, even more dynamic and faster thermal imbalance situations occur, because the different parts of the system heat and cool down faster, as shown in our previous set of experiments.

Then, we have evaluated and compared the behavior of the task migration algorithms under three different workloads, made of multiple instances of the SDR case study, which was divided in three internal subtasks for more accurate control of the final workload conditions. In the first workload setup we analyze the behavior of the different task migration policies in the context of a steady-state thermal situation, where there is essentially no thermal imbalance. Thus, the workload of each task was adjusted to make deterministic the replication of load ratio among cores for the tested thermal balancing policies, using a 65% workload approximately for each processor. To this end, we partitioned the SDR case study in three tasks having very similar processor workload requirements. Therefore, in this situation, the processors tend to run at the same frequency. Next, in the second workload setup we performed an uneven partitioned of the workload between the three internal tasks that compose each SDR application. Thus, the processors need to run at different frequencies and with variable number of memory and I/O operations, which results in a clear overall system thermal imbalance. In particular, we used 55%-85%-30% workload at 35 frames/second for cores 1, 2 and 3, respectively. Finally, in the third workload setup, we assess the capabilities of `MiGra` to adapt to very dynamic workloads by varying the frame rate of the SDR case study, and compare this behavior against an offline-tuned version of the TB-Th migration policy. Thus, in this final setup, we obtained a workload of 46%-74%-26%, 55%-85%-30% and 58%-95%-33% for the frame rate interval using 30, 35 and 40 frames/sec, respectively.

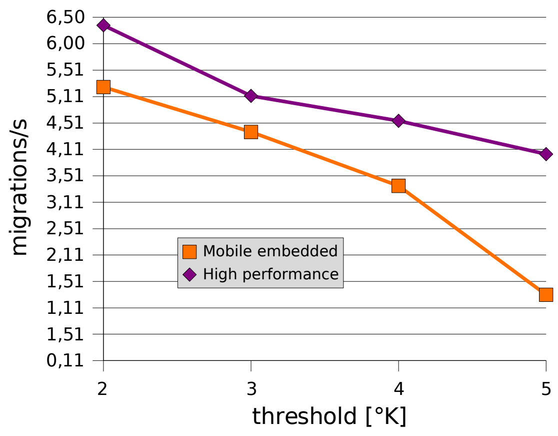

Fig. 11. Migrations/sec of `MiGra` for both types of packages

For each setup we performed various experiments while exploring different values of internal configuration parameters of each policy, namely, for `MiGra` we changed the threshold ranges, for Rotation and TB we modified task migration timeout values, and for TB-Th we varied its minimum temperature unbalance threshold to force the migrations.

E.1) Setup I: Steady-State Thermal Context: Table IV summarizes the experimental results obtained for the first workload setup, where the temperatures of the three cores are already in a steady-state situation. As this table shows, the Rotation and TB policies are not effective, because they try to swap tasks between the different cores without knowledge of the overall temperature gradient across the chip. As a consequence, in highly-demanding working conditions (with small timeouts to apply task migration), both policies show a significant decrease in QoS of the target 3-core platform (i.e, 27% of deadline misses for Rotation and 13% for TB), as they generate a large number of migrations. Conversely, `MiGra` and TB-Th avoid migrations completely, since `MiGra` is able to observe that the standard deviation of the temperature of the cores is within the allowed temperature oscillation range, and also TB-Th does not react because we have manually set up the minimum migration detonation threshold to values that are never reached by any processor.

TABLE IV
EXPERIMENTAL RESULTS FOR SETUP I: TEMPERATURES BALANCED (STEADY-STATE THERMAL CONDITION)

	`MiGra`		Rotation		TB		TB-Th	
Timeout (ms)	10	10	10	20	10	20	10	10
Threshold ($^{\circ}$C)	2	1					316°K	318°K
Standard deviation	0	0	0.18	0	0.16	0	0	0
Deadline misses (%)	0	0	27.64	0	13.12	0	0	0
Migrations. / sec	0	0	30.47	15	20.48	10.00	0	0

E.2) Setup II: Unbalanced Thermal Gradients at Regular Intervals: Table V depicts the experimental results obtained in the context of the second workload setup, where the 3-core MPSoC platform under study experiences thermal gradients, but in regular intervals, due to the unbalanced partitioning of the tasks (but regular overall streaming computation workload). As this figure shows, `MiGra` requires only a linear increase in the number of migrations when we sweep the required threshold of average temperature between the cores from four and one degree around the average temperature of the platform. Moreover, it can be observed that the standard deviation gradually increases, as the policy starts getting closer to the critical threshold or reachable thermal balance limit for the studied 3-core MPSoC (i.e., less than one degree oscillation beyond/below the average temperature), which is due to the unavoidable cost of migrating a certain task between two cores. Nonetheless, even in the smallest range of requested thermal balancing, `MiGra` never experiences deadline misses, as it computes the global benefits of each migration in the overall thermal balance of the MPSoC.

Then, if we compare the results of `MiGra` with the other task migration policies, Table V shows that Rotation has always worst standard deviation and requires many more migrations to compensate the thermal unbalance of the MPSoC. Furthermore, if a very fine-grained timeout is requested to Rotation, it degenerates and shows a very significant decrease in QoS, namely, 26% of deadline misses on average. With respect to the TB policy, the experimental results show that it performs better than Rotation by having a lower standard deviation in critical thermal balancing constraints, but the values are only marginally better than `MiGra` (0.10 versus 0.17). Nonetheless, this values are achieved by TB at the cost of a large percentage of deadline misses (i.e., 7.62%) and QoS degradation, due to its large amount of required task migrations to balance the overall temperature, while `MiGra` does not generate any deadline miss. Finally, although TB-Th shows a lower number of deadline misses (1.62%) than TB or Rotation in the most fine-grained threshold temperature to detonate a task migration (316 $^\circ$ K), it still has deadline misses and experiences a larger standard deviation than `MiGra`.

TABLE V
EXPERIMENTAL RESULTS FOR SETUP II: TEMPERATURES UNBALANCED WITH REGULAR WORKLOAD CYCLES

	`MiGra`		Rotation		TB		TB-Th	
Timeout (ms)	10	10	10	20	10	20	10	10
Threshold ($^\circ$C)	2	1					316 $^\circ$K	318 $^\circ$K
Standard deviation	0.17	0.22	1.57	0.99	0.10	0.37	1.76	0.49
Deadline misses (%)	0	0	26.23	0	7.62	0	1.62	0.00
Migrations/ sec	5.89	8.07	30.25	14.98	19.94	9.98	12.02	8.51

E.3) Setup III: Highly-Variant Thermal Gradients at Irregular Intervals: In this last setup we have evaluated the ultimate reaction capabilities of `MiGra` to highly-dynamic workloads (i.e., variable frame rates in stream computing), which generate thermal gradients at very variable intervals. Furthermore, we have compared its behavior with respect to the best TB-Th configuration decided off-line as the best intermediate value for the SDR benchmark with different frame rates, after analyzing the thermal gradients derived from the execution of the application on the target 3-core MPSoC. As a result, we manually defined the minimum migration threshold value in TB-Th as 318-degree K, see Table V, and compared it with a fine-grained configuration threshold for `MiGra` (i.e., a threshold of 2 degrees around the average temperature). Then, we evaluated both policies using three frame rates: 30, 35 and 40 frames/sec.

Table VI summarizes the results. On one hand, this table shows that the numbers of migrations required by `MiGra` to guarantee the requested thermal balancing of less than 3 degrees at 30 frames/sec is very limited, although it is a valid frame rate for many stream computing applications. This limited number of migrations is due to the fact that at this frame rate, the workload of each task is below 50% for the 3-core platform under study. Thus, `MiGra` can effectively work and adapt the global thermal behavior of the system very fast by mapping two tasks in the same processing core at each moment in time, if this value can reduce the global energy of the system and balance the temperature, as indicated in the constraints of `MiGra` (cf. Section VI-A). Conversely, for 35 or 40 frames per second, the processors are always loaded more than 50%. Thus, several migrations are required to dynamically balance and swap one of the tasks between processors. Hence, `MiGra` performs about double number of migrations with input rates higher than 30 frames/sec, as it is shown in Table VI. Then, the differences in the number of migrations between 35 or 40 frames per second are not very significant for `MiGra`, no deadline misses exists, and the standard deviation can be well-adjusted to each case.

TABLE VI
EXPERIMENTAL RESULTS FOR SETUP III: `MiGra` VS. TB-TH IN A HIGHLY-VARIANT THERMAL GRADIENT CONTEXT

	`MiGra`			TB-Th		
Frame Rate (per sec)	30	35	40	30	35	40
Standard Deviation	0.27	0.12	0.04	0.15	0.49	0.10
Deadline Misses (%)	0	0	0	0	0	0
Migrations/ sec	2.49	4.62	4.42	3.17	8.51	2.74

On the other hand, TB-Th always swaps the tasks between the hottest and the coldest processors, without a complete knowledge of the influence of workload in the overall number of migrations, since it is not possible to define a minimum task migration threshold that works correctly for all possible variable working conditions. Therefore, this policy can create very anomalous conditions for some variable workloads, as it is the case of 35 frames/sec (see Table VI), where a large number of migrations are suddenly necessary to compensate for peaks of workloads accumulated in some processors. Indeed, in some cases, TB-Th reacts inappropriately to the gradient trends of parts of the MPSoC, as the minimum migration threshold defined in this policy cannot be dynamically changed. As a result, if a task migration timeout occurs for TB-Th before the last migration of a task from a hot core to a cold one has finished, as the system is beyond the minimum threshold to detonate new migrations, TB-Th can trigger a new migration phase that brings back more workload to the hot processing core, raising its temperature again. As a consequence, TB-Th performs an unnecessary number of migrations in certain situations with highly-dynamic workloads, and the perfect adjustment of its internal parameters is critical for a good behavior of this policy. Nonetheless, these highly-dynamic workloads are very difficult to predict at design time in order to suitably tune the thresholds and timeouts of the TB-Th algorithm for each target MPSoC.

Conversely, `MiGra` is only slightly affected by variable workloads, due to its fast run-time self-adaptation of the upper and lower thermal-based task migration thresholds. Thus, it can adapt to the thermal dynamics of each target MPSoC, and the standard deviation and number of deadline misses are largely insensitive to initial internal parameters tuning. Hence, it is easier to tune to any final MPSoC architecture.

VIII. Conclusions

As feature sizes decrease, power dissipation and heat generation density exponentially increase. Thus, temperature gradients in MPSoCs can seriously impact system performance and reliability. Thermal balancing policies based on task migration have been proposed to modulate power distribution between processors to achieve temperature flattening. However, in the context of MPSoC stream computing, the impact of migration on quality of service must be carefully studied. In this paper we have presented a new thermal balancing policy, i.e., `MiGra`, specifically designed to exploit dynamically workload information and run-time thermal behavior of stream computing architectures. `MiGra` keeps migration costs and deadline misses bounded to reduce on-chip temperature gradients via task migration, supporting further the application to local DVFS schemes on top of it. We have thoroughly evaluated the potential benefits of `MiGra` to balance the temperature in stream processing architectures with respect to state-of-the-art thermal management techniques using different versions of a software-defined radio multitask benchmark. We have run dynamic workloads of this benchmark on a complete cycle-accurate FPGA-based emulation infrastructure of a real-life 3-core stream platform, and the experimental results show that `MiGra` is able to reach a global thermal balance where the temperatures of the MPSoC components are within a range of 3 degrees around the average temperature. Furthermore, `MiGra` achieves this thermal balancing with a negligible performance overhead of less than 2% in MPSoC stream computing platforms, significantly less than state-of-the-art thermal management techniques.

References

[1] K. Skadron, et al., "Temperature-aware microarchitecture: Modeling and implementation." *ACM TACO*, vol. 1, no. 1, pp. 94–125, 2004.
[2] T. Sato, et al.,"On-chip thermal gradient analysis and temperature flattening for SoC design," Proc. *ASP-DAC*, pp. 1074–1077, 2005.
[3] C. Isci, et al.,"Live, run-time phase monitoring and prediction on real systems with application to dynamic power management," Proc. *MICRO*, pp. 359–370, 2006.
[4] I. Yeo, et al., "Predictive dynamic thermal management for multicore systems," Proc. *DAC*, pp. 734–739, 2008.
[5] J. Donald, et al., "Techniques for multicore thermal management: Classification and new exploration," Proc. *ISCA*, pp. 78–88, 2006.
[6] J. Hu, et al., "Energy-aware communication and scheduling for NoC SoCs under real-time constraints," Proc. *DATE*, pp. 10234, 2004.
[7] P. Rong, et al., "Power-aware scheduling and DVS for tasks running on a hard real-time system," Proc. *ASPDAC*, pp. 1–6, 2006.
[8] StreamIt-MIT Research, "StreamIt Benchmarks," 2009, http://www.cag.lcs.mit.edu/streamit/shtml/benchmarks.shtml
[9] M. Mutyam, et al., "Compiler-directed thermal management for VLIW functional units," Proc. *LCTES*, pp. 163– 172, 2006.
[10] A. K. Coskun, et al., "Temperature aware task scheduling in MPSoCs," Proc. *DATE*, pp. 1659–1664, 2007.
[11] W-L. Hung, et al., "Thermal-aware task allocation and scheduling for embedded systems," Proc. *DATE*, pp. 898–899, 2005.
[12] A. Kumar, et al., "HybDTM: a coordinated hw-sw approach for dynamic thermal management," Proc. *DAC*, pp. 548–553, 2006.
[13] P. Chaparro, et al., "Understanding the thermal implications of multi-core architectures," *IEEE TPDS*, vol. 18, no. 8, pp. 1055–1065, 2007.
[14] S. Carta, et al., "Multi-processor OS emulation framework with thermal feedback for SoCs," Proc. *GLSVLSI*, pp. 311–316, 2007.
[15] R. Mukherjee, et al., "Physical aware frequency selection for dynamic thermal management in multi-core systems," Proc. *ICCAD*, pp. 547–552, 2006.
[16] J. Donald, et al. "Power efficiency for variation-tolerant multicore processors," Proc. *ISLPED*, pp. 304–309, 2006.
[17] F. Bellosa, et al. "Event-driven energy accounting for dynamic thermal management," Proc. *COLP*, pp. 41–50, 2003.
[18] G. Paci, et al., "Exploring temperature-aware design in low-power MPSoC," Proc. *DATE*, pp. 838–843, 2006.
[19] Y. Xie, et al., "Temperature-aware task allocation and scheduling for embedded MPSoC design," *J. VLSI-SPS*, pp. 177–189, 2006.
[20] H. Su, et al., "Full chip leakage estimation considering power supply and temperature variations," Proc. *ISLPED*, pp. 78–83, 2003.
[21] J. Li, et al., "Power-performance implications of thread-level parallelism in chip multiprocessors," Proc. *ISPASS*, pp. 124–134, 2005.
[22] "UClinux: Embedded linux/microcontroller project," 2006, http://www.uclinux.org/.
[23] D. Atienza, et al., "Hw-sw emulation framework for temperature-aware design in MPSoCs," *ACM TODAES*, vol. 12, no. 3, pp. 1–26, 2007.
[24] "XUP Virtex-II Pro development system," Xilinx, 2006, http://www.xilinx.com/univ/xupv2p.html.
[25] H.-J. Stolberg, et al., "Hibrid-soc: A multi-core SoC architecture for multimedia signal processing," *J. VLSI-SPS*, vol. 41, no. 1, pp. 9–20, 2005.
[26] P. Van der Wolf, et al., "Design and programming of embedded multiprocessors: An interface-centric approach," Proc. *CODES+ISSS*, pp. 206–217, 2004.
[27] Freescale,"IMX31 - Multimedia Applications Processors," 2006, www.freescale.com/imx31.

Fabrizio Mulas is currently Ph.D. student in Computer Science at Universita degli Studi di Cagliari, Italy. He received the Electronic Engineer degree from Universita degli Studi di Cagliari, Cagliari, Italy. In 2007/2008 he spent six months as guest researcher at EPFL (Ecole Polytechnique Fédérale de Lausanne) at Integrated Systems Laboratory about conception and development of software algorithms and policies for dynamic resource management in Multiprocessor Systems. His scientific activities mostly concern soft real-time scheduling, power management in wireless sensor networks, Linux kernel activities monitoring, management techniques for addressing variability/reliability and aging problems in next generation hardware components.

David Atienza received his M.Sc. and Ph.D. degrees in Computer Science and Engineering from Complutense University of Madrid (UCM), Spain, and Inter-University Microelectronics Center (IMEC), Belgium, in 2001 and 2005. Currently he is Professor and Director of the Embedded Systems Laboratory (ESL) at Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland, and adjunct professor at the Computer Architecture Department of UCM. He is also scientific counselor of long-time research of IMEC Nederland (IMEC-NL). His research interests focus on design methodologies for high-performance Multi-Processor Systems-on-Chip (MPSoCs) and embedded systems, including new 2D/3D thermal-aware design, wireless sensor networks, dynamic memory optimizations and Network-on-Chip (NoC) design. In these fields, he is co-author of more than 100 publications in prestigious journals and conferences. Dr. Atienza is also Associate Editor of IEEE Transactions on Computer-Aided Design of Circuits and Systems, IEEE Embedded Systems Letters, and Elsevier Integration. He is an elected member of the Executive Committee of the IEEE Council of Electronic Design Automation (CEDA) since 2008.

Andrea Acquaviva graduated (summa cum laude) in Electrical Engineering at the University of Ferrara in 1999. He received a Ph.D. degree in electrical engineering from Bologna University in 2003. Andrea Acquaviva has been an Assistant Professor in Computer Science at the University of Urbino (Italy) and in the Computer Science Department at University of Verona (Italy). He is currently an Assistant Professor in the Computer Science and Automation Department at Politecnico di Torino, Italy. He is also Visiting Professor at Laboratoire de Systemes Integres of the Ecole Politechnique Fédérale de Lausanne (EPFL), Switzerland. He has been research intern at Hewlett Packard Laboratories (HPLabs), Palo Alto, CA (USA) in 2001 and 2002. Since 2004 he collaborates with Freescale Semiconductor (UK). Andrea Acquavivas research interests mainly concern software for multiprocessor and distributed systems, with particular emphasis on operating systems and middleware for Multiprocessor Systems on Chip and wireless body sensor networks for humancomputer interfaces, with particular emphasis on energy conservation aspects.

Salvatore Carta received the B.S. degree (summa cum laude) in electronic engineering and the Ph.D. degree in electronics and computer science from the University of Cagliari, Cagliari, Italy, in 1997 and 2003, respectively. In 2005, he became an Assistant Professor with the Computer Science Department, University of Cagliari. His research interests mainly include architectures, software and tools for embedded and portable computing, with particular emphasis on operating systems, middleware and applications modeling for multiprocessor-systems-on-chips, networks-on-chip, and reconfigurable computing. He is the author of several papers in these fields.

Luca Benini received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1997. He is currently a Professor with the University of Bologna, Bologna, Italy. He also holds a Visiting Faculty Position with the Ecole Polytecnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. His research interests include the design of systems for ambient intelligence, from multiprocessor systems-on-chip/networks-on-chip to energy-efficient smart sensors and sensor networks. From there, his research interests have spread into the field of biochips for the recognition of biological molecules, into bioinformatics for the elaboration of the resulting information, and further into more advanced algorithms for in silicobiology. He has published more than 300 papers in peer-reviewed international journals and conferences, three books, several book chapters, and two U.S. patents. Dr. Benini has been Program Chair and Vice-Chair of Design Automation and Test in Europe Conference. He has been a member of the 2003 MEDEA and EDA Roadmap Committee. He is a member of the IST Embedded System Technology Platform Initiative (ARTEMIS), aworking group on Design Methodologies, a member of the Strategic Management Board of the ARTIST2 Network of Excellence on Embedded System, and a member of the Advisory Group on Computing Systems of the IST Embedded Systems Unit. He has been member of the Technical Program Committee and organizing committee of several technical conferences, including the Design Automation Conference, International Symposium on Low Power Design, and the Symposium on Hardware-Software Codesign. He is an Associate Editor of the IEEE Transactions on Computer-Aided Design of Circuits and Systems, and of the ACM Journal on Emerging Technologies in Computing Systems.

Giovanni De Micheli is Professor and Director of the Institute of Electrical Engineering and of the Integrated Systems Centre at EPFL, Switzerland. He also chairs the Scientific Committee of CSEM, Neuchatel, Switzerland. His research interests include design technologies for integrated circuits and systems, such as synthesis, HW/SW co-design, low-power design, as well as systems on heterogeneous platforms. Prof. De Micheli is the recipient of the 2003 IEEE Emanuel Piore Award. He is a Fellow of ACM and IEEE. He received the Golden Jubilee Medal for outstanding contributions to the IEEE CAS Society in 2000 and the 1987 D. Pederson Award for the best paper on the IEEE TCAD/ICAS. He was Division 1 Director (2008-9), co-founder and President Elect of the IEEE Council on EDA (2005-7), President of the IEEE CAS Society (2003), Editor in Chief of the IEEE TCAD/ICAS (1987-2001).