Thermal Balancing Policy for Multiprocessor Stream Computing Platforms

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

***Abstract*—Die-temperature control to avoid hotspots is increasingly critical in multiprocessor systems-on-chip (MPSoCs) for stream computing. In this context, thermal balancing policies based on task migration are a promising approach to redistribute 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-based emulation of an actual three-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*—Multiprocessor architectures, stream computing, systems-on-chip (MPSoCs), task migration, thermal balancing.**

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 an 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.

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 have gained attention in the context of chip multiprocessors [2], [4], [13]. 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 [5], [13]. In the case of embedded MPSoC architectures for stream computing (signal processing, multimedia, and 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 nonuniform noncoherent 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 the average system temperature conditions. This feature, rather than defining an absolute temperature limit as in hotspot-detection policies [2], [13], [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

Manuscript received January 20, 2009; revised June 4, 2009 and July 29, 2009. Current version published November 18, 2009. This work was supported in part by the Swiss Confederation through the Nano-Tera.ch NTF Project 123618—CMOSAIC—, the Spanish Government Research Grants TIN2005-5619 and TIN2008-00508, and the ARTIST-DESIGN Network-of-Excellence. This paper was recommended by Associate Editor N. Chang.

F. Mulas and S. Carta are with the Dipartimento di Matematica e Informatica, University of Cagliari, 09124 Cagliari, Italy (e-mail: fabrizio.mulas@sc.unica.it; salvatore@unica.it).

D. Atienza is with the Embedded Systems Laboratory (ESL), Institute of Electrical Engineering (IEL), School of Engineering (STI), Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland, and also with the Department of Computer Architecture and Automation (DACYA), Complutense University of Madrid (UCM), 28040 Madrid, Spain (e-mail: david.atienza@epfl.ch).

A. Acquaviva is with the Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy (e-mail: andrea.acquaviva@polito.it).

L. Benini is with the Dipartimento di Elettronica, Informatica e Sistemistica (DEIS), University of Bologna, 40136 Bologna, Italy, and also with Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland (e-mail: luca.benini@unibo.it).

G. De Micheli is with the Laboratoire des Systemes Integres, Institute of Computing and Multimedia Systems (ISIM), School of Computer and Communication Sciences (IC), Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland (e-mail: giovanni.demicheli@epfl.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2032372

necessary hardware and software extensions to allow designers to test different thermal-aware *multiprocessor operating systems (MPOSs)* implementation running onto emulated real-life multicore stream computing platforms. The framework has been developed on top of a cycle-accurate MPOS emulation framework for an MPSoC [14]. To the best of our knowledge, this is the first multiprocessor platform that supports operating system (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: 1) 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; 2) 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; and 3) `MiGra` is capable of very fast adaptation to changing workload conditions due to dynamic temperature-threshold adaptation.

The rest of this paper is organized as follows. In Section II, we overview a related work on thermal modeling and management techniques for MPSoC architectures. In Section III, we summarize the software/hardware characteristics of the 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 those of state-of-the-art thermal management strategies. Finally, in Section VIII, we summarize the main conclusions of this paper.

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. Reference [1] presents a thermal/power model for superscalar architectures. Furthermore, [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 a field-programmable-gate-array (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 [1], [10]. Furthermore, [9] combines load balancing with low-power scheduling at the compiler level to reduce the peak temperature in *very long instruction word* processors. In addition, [11] introduces the inclusion of temperature as a constraint in the cosynthesis and task allocation process for a platform-based system design. However, all these techniques are based on a 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 microarchitectural 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. Furthermore, [18] studies the thermal behavior of low-power MPSoCs and concludes that, for such low-power architectures, no thermal issues presently exist and that 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, which is 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. References [13] and [16] describe techniques for thread assignment and migration using performance counter-based information or compile-time precharacterization. Furthermore, 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 multithreaded architectures with a cache coherent memory hierarchy, which implies that the assumed performance cost of thread migration and misprediction effects is not adapted to MPSoC stream platforms. Conversely, in this work, we target specifically embedded stream platforms with a nonuniform 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 the temperature gradients because, in the presence of hotspots, there could be only one core that is very hot while others are cold. Moreover, Stop&go imposes large temporal gradients as the main countermeasure 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 bounding the 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 [19], [25], [27]. In typical stream computing platforms, the considered target MPSoC exploits the shared memory to implement communication between tasks. In homogeneous platforms, as the three-core streaming MPSoC that 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 OS, which is optimized for fast interprocessor communication. Then, to support MPOSs in stream computing, a dedicated hardware must be designed to support OS execution and communication among processes running on different processing cores. This includes the following: 1) an interprocessor interrupt controller; 2) a semaphore memory through hardware mutexes; 3) an address translator as the memories of each core have nonoverlapping address ranges; and 4) a 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.

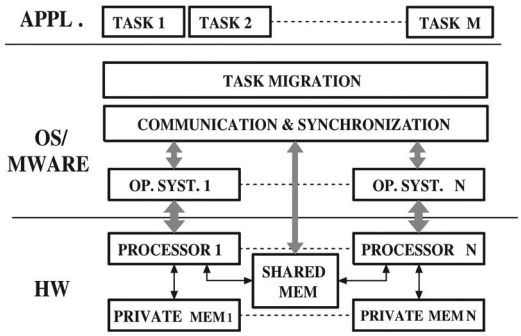


Fig. 1. Scheme of the software abstraction layer.

From the software viewpoint, streaming applications are composed of multiple tasks communicating data and synchronizing with each other using a message-passing paradigm. Then, tasks are spread on the various cores, depending on resource availability, to exploit the architecture parallelism. As such, communication takes place using the interprocessor buffers located in private memories or in a 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 shown in Fig. 1. It is based on three main components: 1) a stand-alone OS for each processor running in a private memory; 2) a lightweight middleware layer providing data sharing/synchronization and communication services; and 3) a 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 that we have adopted in our baseline MPSoC stream computing architecture. We developed a lightweight message-passing scheme that is 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 OS that we have extended for very fast interprocessor 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 additional support for communication, synchronization, and task migration using a 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 the 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 OSs 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 the 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 Fig. 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 reload the program code from the file system; thus, the offset 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

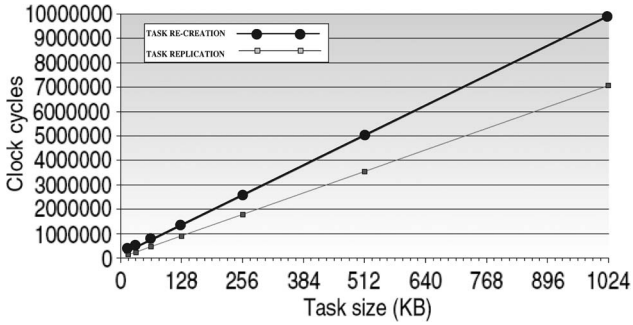


Fig. 2. Migration cost as a function of task size for task replication and task recreation.

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.

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 the detailed statistics of hardware components, OS, 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 in the following. An overview of the extended framework is shown in Fig. 3.

FPGA emulation is exploited to model the hardware components of the MPSoC platform at multimegahertz speeds. The hardware architecture consists of a variable number of soft cores (currently three cores, as required by the modeled MPSoC, shown in Fig. 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 the 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 dynamic voltage and frequency scaling (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 nonintrusive subsystem, which implements

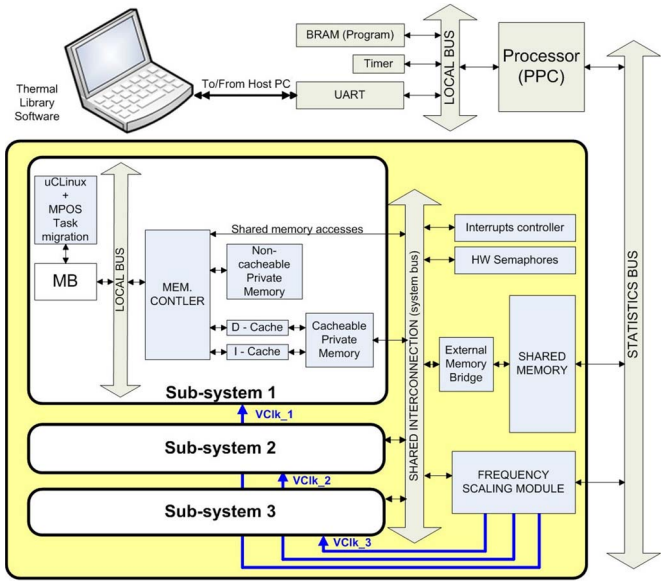


Fig. 3. Overview of the MPSoC thermal emulation framework for stream computing platforms.

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 soft-core processor. Then, the temperatures coming out from the simulator provide a real-time temperature information that is visible by the running uClinux in each processor through the 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, due 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 multimegahertz speed and the observation of the real thermal transients of MPSoC stream platforms.

VI. Thermal Balancing for Stream Computing

In general, thermal balancing does not come as a side effect of energy balancing. In Fig. 4(a), a typical situation where a two-core system running three tasks (A, B, and 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 an FSE of 50% and 40%, respectively; core 2 runs task C that has an FSE of 40%. In this case, core 1 can ideally scale its frequency to 90% of its maximum value, whereas core 2 can scale it to 40%. No better task mapping exists that further

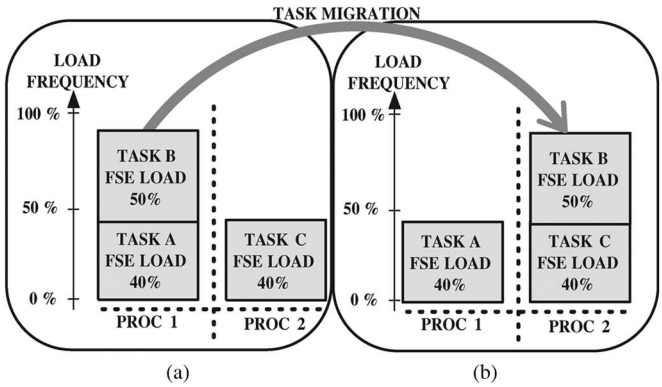


Fig. 4. Simple thermal balancing example between two cores.

reduces the 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 Fig. 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 that is 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 the migration costs bounded.

A. `MiGra`: *Thermal Balancing Algorithm*

The thermal balancing strategy that we propose in this paper, `MiGra`, is inspired by [14]. To prevent impact on QoS caused by migration, `MiGra` is based on the 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 *DVFS* policy [5]. Thus, the power consumption of a task can roughly be estimated at run time by assuming that it is proportional to its load (cf., Fig. 2).

`MiGra` implements a strategy that tries to bound the temperature of each processor around the current average temperature, as well as to minimize the overhead in terms of a 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 controls thermal runaway by stopping the core that reaches a temperature that is 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 the 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 that is 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 is 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 the thermal gradients without increasing the overall energy dissipation when tasks are migrated, as well as minimize the 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, whereas, 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.

1) If the temperature of the source core is beyond the average temperature (t_{mean}), the destination core has to be below: $(t_{src} - t_{\text{mean}}) * (t_{dst} - t_{\text{mean}}) < 0$.
2) The frequency of the source core must be higher than the average if the one of the destination core is below: $(f_{src} - f_{\text{mean}}) * (f_{dst} - f_{\text{mean}}) < 0$.
3) 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: $(f_{src} * {v^2}_{src} + f_{dst} * {v^2}_{dst})_{\text{before\_migr}} \geq (f_{src}{v^2}_{src} + f_{dst} * {v^2}_{dst})_{\text{after\_migr}}$.

The first condition is about temperature and assures that the migration achieves reduction in the 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 an 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 the temperatures are stabilized (constraint 1) and that no oscillations are caused by the workload reallocations (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. Furthermore, no pairs of candidates may exist, which occurs in the 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 are 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:

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

where ${C^{src}}_i$ is the amount of data to be moved for the ith of I tasks running on the source processor and ${C^{tgt}}_j$ is the amount of data to be moved for the jth 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 five–ten 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 precharacterization 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.

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 three-core MPSoC running a multitask streaming application. Therefore, in the next sections, we first describe the concrete instance of a used MPSoC architecture, as well as the power figures and the 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 three-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– 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 shown in Fig. 5. In particular, we consider a system based on three 32-b reduced instruction set computer processors without MMU support to access cacheable private memories and a single non-cacheable shared memory. It follows the structure envisioned for noncache-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 90-nm 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).

We considered two different packaging solutions. The first package shows temperature variations of around 10° in a few seconds [27], whereas 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

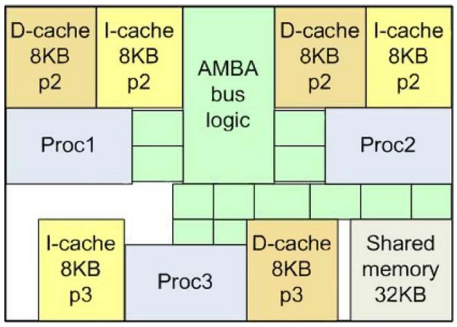


Fig. 5. Emulated three-core MPSoC streaming architecture.

TABLE I
Power of the Components in a 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

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	$1.945e-12 J/um^3 K$
silicon thickness	$300um$
copper thermal conductivity	$400 W/mK$
copper specific heat	$3.55e-12 J/um^3 K$
copper thickness	$1000um$
package-to-air conduct. (low-cost)	$12 K/W$
package-to-air conduct. (high-cost)	$1 K/W$

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 a printed circuit board), we have tuned these figures according to the experimental figures measured in our industrial three-core case study [27], according to the final MPSoC working conditions indicated by our industrial partners.

B. Benchmark Application Description

We ported to our emulation framework different multitask variations of the *software FM defined radio (SDR)* benchmark (Table III), 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 first-in–first-out queues, as shown in Fig. 6, where the tasks are graphically represented as blocks. As this figure shows, the output data of the tasks of the SDR application are 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.

We performed two sets of experiments. In the first set, we used a very dynamic workload that is made of multiple instances of the SDR application, using versions divided in three or six tasks (as in Fig. 6). The input data to the SDR

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

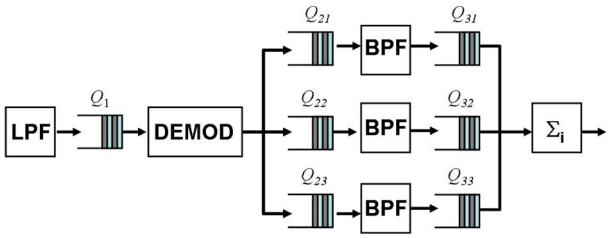


Fig. 6. SDR case study (six-task version).

application represent the samples of the digitalized pulse-code-modulation radio signal to be processed in order to produce an equalized baseband audio signal. In the first step, the radio signal passes through a *low-pass filter* 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 *bandpass filters* (BPFs) 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 $core_{(j+1) \bmod N}$.

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 $core_{N-j-1}$.

TB 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 offline 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 MPOSs, using a low- and a high-cost thermal package. DVFS was also always active in the MPOS to adjust the power dissipated by each core to the required workload.

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 s), the temperatures of the three cores stabilize. However, it is not balanced, and an approximately 10-°C 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 cores 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 Fig. 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 3° 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 s 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. Fig. 7 shows the temperature standard deviation for the three policies as a function of the threshold values. The X-axis indicates the distance of the 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

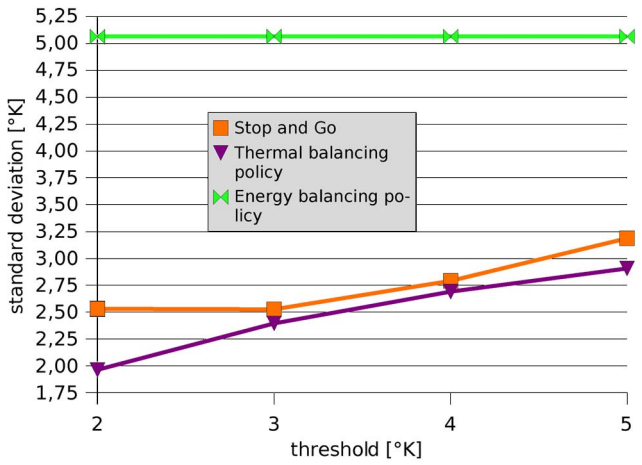


Fig. 7. Temperature standard deviation in low-cost embedded SoCs from the mean on-chip temperature (337 K).

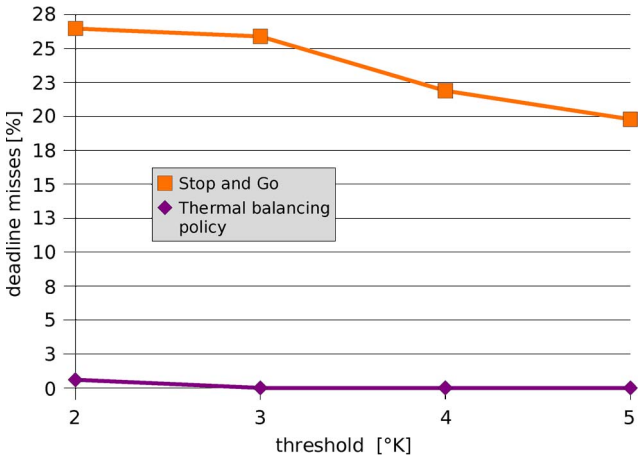


Fig. 8. Deadline misses for the embedded mobile system.

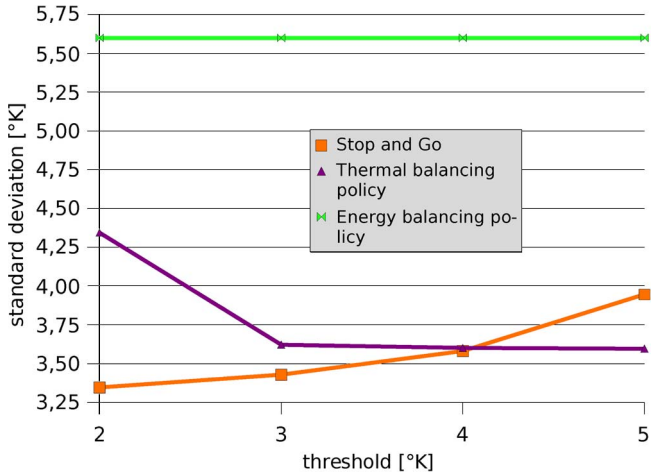


Fig. 9. Standard deviation in high-performance SoCs from the mean on-chip temperature (314 K).

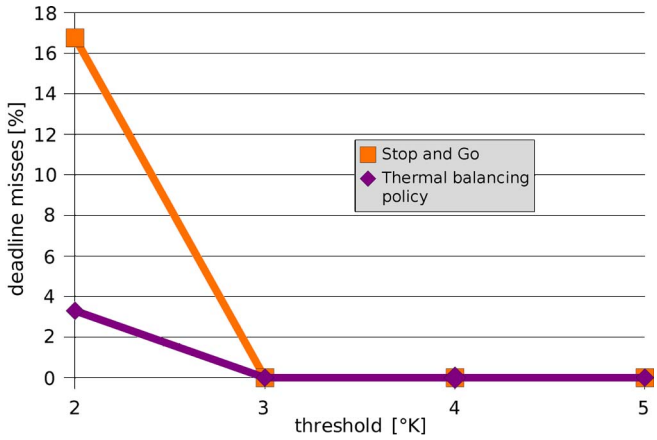


Fig. 10. Deadline misses for high-performance systems.

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 [5], [13], considering the highest supported temperature for the low-cost package as the panic threshold, higher temperature swings are observed, which leads to a worst standard deviation value (3.70 K more) with respect to those shown in Fig. 7.

Then, Fig. 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, interprocessor queues are depleted during migration, and, if the queue of the last stage gets empty, a deadline miss occurs. However, as Fig. 8 shows, 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.

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 six times faster than the previous model. Hence, the three-core case study experiences gradients of more than 10°, i.e., the coolest core typically operates at 56 °C and the hottest one can reach 67 °C.

Fig. 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 (Fig. 10). Moreover, using the original version of Stop&Go [5] with the highest supported temperature of the high-performance package as the panic threshold, a worst standard deviation value of 4.48 K more is observed with respect to Fig. 9.

On the contrary, although our algorithm makes the 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

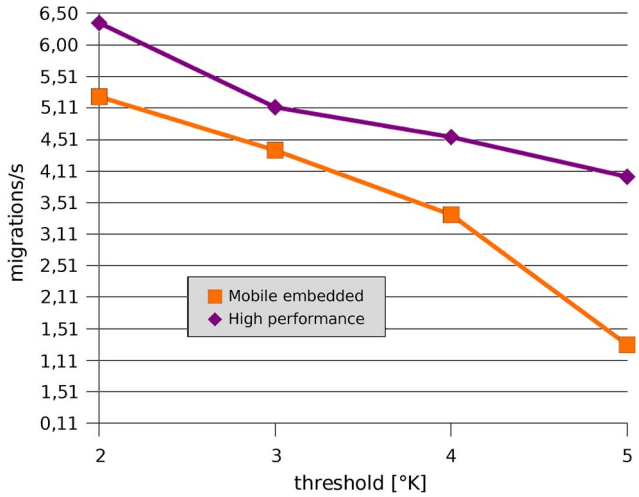


Fig. 11. Migrations per second of MiGra for both types of packages.

threshold increases, as less migrations are triggered. Furthermore, 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 codesign approach is needed.

Finally, Fig. 11 shows 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 kB of data (the minimum memory space allocated by the OS), the required three migrations per second are equivalent to $64 * 3 = 192$ kB/s, which means that our task migration policy implies only a negligible overhead in system performance (1% overall).

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 partition 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/s 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/s, respectively.

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 (°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

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 the task migration timeout values, and, for TB-Th, we varied its minimum temperature unbalance threshold to force the migrations.

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 three-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.

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 three-core MPSoC platform under study experiences thermal gradients,

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 (°C)	2	1					316 °K	318 °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

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 the average temperature between the cores from 4° and 1° 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 three-core MPSoC (i.e., less than 1° 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 those of 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 the critical thermal balancing constraints, but the values are only marginally better than those of MiGra (0.10 versus 0.17). Nonetheless, these 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 K), it still has deadline misses and experiences a larger standard deviation than MiGra.

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 offline 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 three-core MPSoC. As a result, we manually defined the minimum migration threshold value in TB-Th as 318 K (see Table V) and compared it with a fine-grained configuration threshold for MiGra (i.e., a threshold of 2° around the average temperature). Then, we evaluated both policies using three frame rates: 30, 35, and 40 frames/s.

TABLE VI
EXPERIMENTAL RESULTS FOR SETUP III: MiGra VERSUS 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

Table VI summarizes the results. On the one hand, this table shows that the number of migrations required by MiGra to guarantee the requested thermal balancing of less than 3° at 30 frames/s 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 three-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/s, 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/s, as it is shown in Table VI. Then, the differences in the number of migrations between 35 and 40 frames/s are not very significant for MiGra, no deadline misses exist, and the standard deviation can be well-adjusted to each case.

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/s (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 the parts of the MPSoC, as the minimum migration threshold defined in this policy cannot dynamically be 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 parameter tuning. Hence, it is easier to tune to any final MPSoC architecture.

VIII. Conclusion

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 QoS 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 three-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° 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 that of state-of-the-art thermal management techniques.

References

[1] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM TACO*, vol. 1, no. 1, pp. 94–125, Mar. 2004.

[2] T. Sato, J. Ichimiya, N. Ono, K. Hachiya, and M. Hashimato, "On-chip thermal gradient analysis and temperature flattening for SoC design," in *Proc. ASP-DAC*, 2005, pp. 1074–1077.

[3] C. Isci, G. Contreras, and M. Martonosi, "Live, run-time phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. MICRO*, 2006, pp. 359–370.

[4] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Proc. DAC*, 2008, pp. 734–739.

[5] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proc. ISCA*, 2006, pp. 78–88.

[6] J. Hu and R. Marculescu, "Energy-aware communication and scheduling for NoC SoCs under real-time constraints," in *Proc. DATE*, 2004, p. 10 234.

[7] P. Rong and M. Pedram, "Power-aware scheduling and DVS for tasks running on a hard real-time system," in *Proc. ASPDAC*, 2006, pp. 473–478.

[8] StreamIt-MIT Research, *StreamIt Benchmarks*, 2009. [Online]. Available: http://www.cag.lcs.mit.edu/streamit/shtml/benchmarks.shtml

[9] M. Mutyam, F. Li, N. Vijaykrishnan, M. T. Kandemir, and M. J. Irwin, "Compiler-directed thermal management for VLIW functional units," in *Proc. LCTES*, 2006, pp. 163–172.

[10] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. DATE*, 2007, pp. 1659–1664.

[11] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Thermal-aware task allocation and scheduling for embedded systems," in *Proc. DATE*, 2005, pp. 898–899.

[12] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha, "HybDTM: A coordinated HW–SW approach for dynamic thermal management," in *Proc. DAC*, 2006, pp. 548–553.

[13] P. Chaparro, J. González, G. Magklis, Q. Cai, and A. González, "Understanding the thermal implications of multi-core architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 8, pp. 1055–1065, Aug. 2007.

[14] S. Carta, A. Acquaviva, P. G. Del Valle, D. Atienza, G. De Micheli, F. Rincón, L. Benini, and J. M. Mendias, "Multi-processor OS emulation framework with thermal feedback for SoCs," in *Proc. GLSVLSI*, 2007, pp. 311–316.

[15] R. Mukherjee and S. O. Memik, "Physical aware frequency selection for dynamic thermal management in multi-core systems," in *Proc. ICCAD*, 2006, pp. 547–552.

[16] J. Donald and M. Martonosi, "Power efficiency for variation-tolerant multicore processors," in *Proc. ISLPED*, 2006, pp. 304–309.

[17] F. Bellosa, S. Kellner, M. Waitz, and A. Weissel, "Event-driven energy accounting for dynamic thermal management," in *Proc. COLP*, 2003, pp. 41–50.

[18] G. Paci, P. Marchal, F. Poletti, and L. Benini, "Exploring temperature-aware design in low-power MPSoC," in *Proc. DATE*, 2006, pp. 838–843.

[19] Y. Xie and W.-L. Hung, "Temperature-aware task allocation and scheduling for embedded MPSoC design," *J. VLSI-SPS*, vol. 45, no. 3, pp. 177–189, Dec. 2006.

[20] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full chip leakage estimation considering power supply and temperature variations," in *Proc. ISLPED*, 2003, pp. 78–83.

[21] J. Li and J. F. Martínez, "Power-performance implications of thread-level parallelism in chip multiprocessors," in *Proc. ISPASS*, 2005, pp. 124–134.

[22] *uClinux: Embedded Linux/Microcontroller Project*, 2006. [Online]. Available: http://www.uclinux.org/

[23] D. Atienza, P. G. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, J. M. Mendias, and R. Hermida, "HW–SW emulation framework for temperature-aware design in MPSoCs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 1–26, Aug. 2007.

[24] *XUP Virtex-II Pro Development System*, Xilinx, San Jose, CA, 2006. [Online]. Available: http://www.xilinx.com/univ/xupv2p.html

[25] H.-J. Stolberg, M. Berekoviæ, S. Moch, L. Friebe, M. B. Kulaczewski, S. Flügel, H. Klußmann, A. Dehnhardt, and P. Pirsch, "HiBRID-SoC: A multi-core SoC architecture for multimedia signal processing," *J. VLSI-SPS*, vol. 41, no. 1, pp. 9–20, Aug. 2005.

[26] P. van der Wolf, E. de Kock, T. Henriksson, W. Kruijtzer, and G. Essink, "Design and programming of embedded multiprocessors: An interface-centric approach," in *Proc. CODES +ISSS*, 2004, pp. 206–217.

[27] Freescale, *IMX31—Multimedia Applications Processors*, 2006. [Online]. Available: www.freescale.com/imx31

Fabrizio Mulas received the B.S. degree in electronic engineering from the University of Cagliari, Cagliari, Italy, where he is currently working toward the Ph.D. degree in computer science.

In 2007–2008, he spent six months at the Integrated Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, as a Guest Researcher 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 activity monitoring, management techniques for addressing variability/reliability, and aging problems in next-generation hardware components.

David Atienza (M'05) received the M.Sc. degree in computer science and engineering from the Complutense University of Madrid (UCM), Madrid, Spain, in 2001 and the Ph.D. degree in computer science and engineering from the Inter-University Microelectronics Center (IMEC), Leuven, Belgium, in 2005.

He is currently a Professor and the Director of the Embedded Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, and an Adjunct Professor with the Computer Architecture Department, UCM. He is also a Scientific Counselor of long-time research with IMEC Netherlands. His research interests focus on design methodologies for high-performance multiprocessor systems-on-chip and embedded systems, including new 2-D/3-D thermal-aware design, wireless sensor networks, dynamic memory optimizations, and network-on-chip design. In these fields, he is the coauthor of more than 100 publications in prestigious journals and conferences.

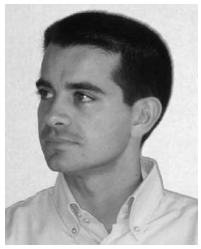
Dr. Atienza is also an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE LETTERS ON EMBEDDED SYSTEMS, and *Elsevier Integration*. He has been an elected member of the Executive Committee of the IEEE Council of Electronic Design Automation since 2008.

Andrea Acquaviva received the B.S. degree (*summa cum laude*) in electrical engineering from the University of Ferrara, Ferrara, Italy, in 1999 and the Ph.D. degree in electrical engineering from the University of Bologna, Bologna, Italy, in 2003.

He was an Assistant Professor in computer science with the University of Urbino, Urbino, Italy, and with the Department of Computer Science, University of Verona, Verona, Italy. In 2001 and 2002, he was a Research Intern with Hewlett Packard Laboratories, Palo Alto, CA. He is currently an Assistant Professor with the Computer Science and Automation Department, Politecnico di Torino, Torino, Italy. He is also a Visiting Professor with the Laboratoire des Systemes Integres, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. Since 2004, he has been collaborating with Freescale Semiconductor, Inc., Glasgow, U.K. His 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 human–computer 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.

Since 2005, he has been an Assistant Professor in computer science with the Dipartimento di Matematica e Informatica, University of Cagliari. His research interests focus mainly on architectures, software and tools for embedded and portable computing, with particular emphasis on operating systems, middleware, and software for multiprocessor systems on chips; networks on chip; and reconfigurable computing. He is the author of more than 20 papers in these fields in the last three years. In the last two years, he has also been working in the fields of recommendation systems and social networks. He is the author of several papers in these fields.



Luca Benini (S'94–M'97–SM'04–F'06) 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 Polytechnique Fédérale de Lausanne, 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 silico* biology. 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 the Program Chair and the Vice Chair of the Design Automation and Test in Europe Conference. He was a member of the 2003 MEDEA and EDA Roadmap Committee. He is a member of the IST Embedded System Technology Platform Initiative (ARTEMIS), a working 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 a member of the Technical Program Committee and Organizing Committee of several technical conferences, including the Design Automation Conference, the 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 INTEGRATED CIRCUITS AND SYSTEMS and of the *ACM Journal on Emerging Technologies in Computing Systems*.

Giovanni De Micheli (S'79–M'83–SM'89–F'94) received the Nuclear Engineer degree from the Politecnico di Milano, Milan, Italy, in 1979 and the M.S. degree and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983, respectively.

He was a Professor of electrical engineering with Stanford University, Stanford, CA. He is currently with Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, where he is the Director of the Institute of Electrical Engineering and the Integrated Systems Centre and a Professor with the Laboratoire des Systemes Integres, Institute of Computing and Multimedia Systems (ISIM), School of Computer and Communication Sciences (IC). His research interests include design technologies for integrated circuits and systems, such as synthesis, HW/SW codesign, and low-power design, as well as systems on heterogeneous platforms.

Prof. De Micheli is a Fellow of the Association for Computing Machinery. He was the recipient of the 2003 IEEE Emanuel Piore Award. He was also the recipient of the Golden Jubilee Medal for outstanding contributions to the IEEE Circuits and Systems (CAS) Society in 2000 and the 1987 D. Pederson Award for the best paper on the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS (TCAD/ICAS). He was the Division-1 Director (2008–2009), the Cofounder, and the President Elect of the IEEE Council on Electronic Design Automation (2005–2007), the President of the IEEE CAS Society (2003), and the Editor-in-Chief of the IEEE TCAD/ICAS (1987–2001). He also chairs the Scientific Committee of the Centre Suisse d'Electronique et de Microtechnique, Neuchatel, Switzerland.