Limiting Temperature Gradients on Many-Cores by Adaptive Reallocation of Real-Time Workloads

Matthias Becker*, Kristian Sandström†, Moris Behnam*, Thomas Nolte*†
*MRTC / Mälardalen University, Västerås, Sweden
{matthias.becker, moris.behnam, thomas.nolte}@mdh.se
†ABB Corporate Research, Västerås, Sweden
kristian.sandstrom@se.abb.com

***Abstract*—The advent of many-core processors came with the increase in computational power needed for future applications. However new challenges arrived at the same time, especially for the real-time community. Each core on such a processor is a heat source and uneven usage can lead to hot spots on the processor, affecting its lifetime and reliability. For real-time systems, it is therefore of paramount importance to keep the temperature differences between the individual cores below critical values, in order to prevent premature failure of the system. We argue that this problem can not be solved by traditional approaches, since the growing number of cores makes them intractable. We rather argue to split the problem in the spacial domain and control the temperature on core level. The cores control their temperature by rearranging the load in a predictable manner during runtime. To achieve this, a feedback controller is implemented on each core. We conclude our work with a simulation based evaluation of the proposed approach comparing its performance against a previously presented algorithm.**

I. Introduction

The transistor size, used to manufacture modern processors, is still shrinking like predicted by Moore's law. The growing complexity of the deployed applications lead chip designers to increase the number of computing elements on one die. This high number of computing elements does not allow a traditional shared interconnection medium, like a crossbar, due to increased contention delays. Instead the Network-on-Chip (NoC) is proposed as new interconnection medium [1]. The NoC consists of Intellectual Property (IP) cores which are located on so called tiles. Each tile is connected to the network via a dedicated router. A router has communication channels to the neighboring tiles and therefore communication among the tiles is possible. The individual IP cores can be computational cores, memory or special purpose cores. In this work we assume identical IP cores, consisting of one computational core together with local memory. Adaptevas Epiphany processor [2] is an example of such a processor.

With increasing popularity of many-core processors in embedded and real-time systems new challenges arise. The small manufacturing size of modern processor leads to a high energy density on the chip and hence to a high temperature. High temperature on the other hand affects the processor lifetime and reliability [3], [4]. An increased temperature of $10 - 15\,^{\circ}\mathrm{C}$ can already reduce the expected lifetime by 50% [5]. Since the different computing cores of many-core processors are arranged in a 2d-mesh, the different aging of the individual components has also to be taken into account. High temperature speeds the aging process of the cores. Therefore, an uneven heat distribution on the die can lead to different aging speeds of the cores.

Embedded systems are often constructed under high resource constraints. Heavy cooling equipment is not possible due to cost or space reasons. Most processors have the possibility to change the operating frequency and voltage of the core during runtime, called Dynamic Voltage and Frequency Scaling (DVFS). The consumed power of a CPU consists of a static and a dynamic part: $P = P_{static} + P_{dynamic}$. The static part can be approximated by $P_{static} = I_{leakage} \cdot V_{cc}$ where $I_{leakage}$ is the current caused by leakage in the transistors and V_{cc} is the operating voltage. Leakage current is tightly coupled with the temperature of the core, higher temperature leads to higher leakage. The dynamic power can be approximated by $P_{dynamic} = C \cdot f \cdot V_{cc}^2$, where C is the capacity switched at each cycle and f is the operating frequency. As can be seen, changing the frequency affects the power linearly and changing the operating voltage affects it quadratically [6]. Therefore DVFS is a popular and effective mechanism to regulate the consumed power, and therefore also the temperature, of the system. However, for real-time systems, changing the frequency during runtime is not without side effects. The execution time of the tasks changes with the frequency, introducing jitter and affecting the predictability of the system which in turn makes verification more difficult. Latency of the tasks is also affected by DVFS approaches. This can have a negative impact for systems used to control a process. Thus, in this paper we propose a new framework to dynamically reallocate real-time tasks in order to minimize heat gradients on the processor. The framework operates in a distributed manner with a feedback control loop on each core.

The remainder of this paper is structured as follows. Section II gives an overview of related work. Section III outlines the hardware and task model we assume for this work and Section IV discusses the relation between temperatures on the processor and the consumed power. Section V is the main part and describes the proposed framework which is then evaluated in Section VI. The paper concludes with Section VII.

II. Related work

Thermal aware scheduling is an important problem and thus much research has been conducted in that area. Most work applies DVFS, for both single and multicore systems.

The work presented in this paper is supported by the Swedish Knowledge Foundation via the research project PREMISE.

Recently Dasari et al. [7] examined commercially available Components-Off-The-Shelf (COTS) multicore systems with respect to their applicability for embedded and real-time systems. Their focus lies on different sources of unpredictability encountered in such processors. Among other points, thermal and power management is highlighted as such. They look at the different strategies applied by hardware vendors and by software, including a summary of existing work in both power and thermal management for multicore systems.

Fisher et al. [8] propose a thermal aware global scheduling scheme for sporadic real-time tasks on a homogeneous multicore system. A first step derives the processor speed for each core, which is then set by DVFS techniques. The derivation of the processor speeds take the thermal effects from neighboring cores into account. This is formulated using Fourier's Law. Yu et al. [9] take the trade off between execution quality and temperature into account for their approach. They consider adaptive applications where the main goal lies on maximizing the executed cycles in the given deadline before reaching a threshold temperature where the core is turned off by hardware protection circuits. This is done by selecting the suitable core frequency considering the thermal behavior of the core itself and its neighboring cores. Wang et al. [10] also apply DVFS for their temperature constrained power control. They use a controller based on optimal control theory with the total power consumption of the processor as controlled variable and the DVFS levels on each core as manipulated variables. This way they consider both, power consumption and temperature in their control loop but the possible change in core frequency at each invocation makes it intractable for real-time workload where it has to be assured that the real-time constraints are met.

The algorithm proposed by Yun et al. [11] belongs to the class of algorithms which predicting future temperature values of cores and act based on that information. They apply machine learning algorithms to predict the thermal dynamics of each core of a multicore platform. Each application has a thermal profile which is then used to predict if a core will overheat before the next protocol invocation. If a possible overheating is detected adequate techniques like DVFS or clock gating are performed. An alternative technique for thermal management on processors with multiple cores is task migration. Temperature is controlled by keeping the workload of the system in a balanced state, allowing no core to overheat.

The DTM technique proposed by Liu et al. [12] is a hybrid between the algorithms applying DVFS techniques and the ones using task migration in order to cool the cores down. They predict the core temperature by using the local history and the effects of neighboring cores. In a emergency situation tasks are migrated to an other core. They take the temperature of the candidate cores and their changing rate into account but do not consider the time it takes to migrate the task. Additionally they use DVFS techniques if no appropriate core can be found. An other approach combining the DVFS and task migration techniques is presented in [13]. Subject of their work is to minimize the makespan of all tasks while meeting the thermal constraints of the processor. They show that the proposed approach is roughly linear with the number of computing elements on the die. Yeo et al. [14] use task migration to prevent overheating of cores. They use two thermal models, one for the application and one for the cores to predict their future temperature. The Linux standard scheduler is extended to incorporate this new information. If the predicted temperature for the next protocol invocation exceeds a migration threshold tasks are migrated to other cores. The core with the lowest future temperature is selected for task migration among all cores in the system. Their approach is similar to our previous work [15] where a hysteresis controller is used to initiate task migration. Additionally we turn overheating cores off in order to allow them to cool down until they reach a lower threshold where the hysteresis controller activates them again. In contrast to Yeo et al. we consider real-time characteristics of the scheduled tasks.

Most related work applies a central controller to prevent the system from overheating. Multiple Input Multiple Output (MIMO) controller are applied, with one input and output for each core. With the increasing number of computing elements on the many-core platform, this becomes intractable [16]. The core, hosting the controller, needs to collect temperature information of each core in periodic intervals, leading to a high network usage around that core and possibly to contention on the network. Therefore, it is proposed to split the problem in the spacial domain and apply a controller on each core [16], [15]. This distributed approach can scale with the number of cores since each core acts independently. The approach presented in [16] is similar to our approach. Tasks are migrated between direct neighbor cores in order to manage the temperature. Migration can be initiated trough different scenarios. Each core has a temperature sensor and also predicts future temperature. If one of those values exceeds a threshold value migration is initiated.

III. Assumptions for this work

In this section we outline the different assumptions we make for the hardware as well as for the software.

A. Hardware

As mentioned in the introduction, we assume a many-core processor comprised of m identical cores connected by a 2-dimensional mesh-bases NoC. Each core is located on a tile, together with private memory. During idle times, each core can transition into a low power state within a negligible time. And we further assume that each core is equipped with a temperature sensor. The NoC applies the wormhole switching technique [17]. Wormhole switching has several advantages compared to transaction based switching. A message is divided into equally sized parts, so called flits. A header and a tail flit are used to convey routing information. As the header flit travels through the network all other flits follow in a pipelined manner. Proceeding like this requires only enough buffer to accommodate one of those flits on each router at a time which is of advantage compared to transaction based techniques where the whole message is saved on each router. As the header flit travels through the network it locks each channel in order to allow the other flits to follow its route. The tail flit on the other hand frees the channel again.

Since routing of messages is an important step for efficient usage of the network, several routing algorithms are proposed [18]. Such algorithms can be deterministic or adaptive. For this work we assume XY routing, which can be applied by most current hardware [2], [19], [20]. In XY routing, a message first travels along the X axis until the destination X coordinate is reached. Then it turns and travels along the Y axis until the final destination is reached. The nature of this deterministic routing algorithm makes it deadlock and livelock free and therefore suitable for real-time systems.

B. Task model

In this work, we consider independent periodic real-time tasks. Each tasks τ can be described by the tuple $\{C_i, T_i, P_i\}$, where C_i is the tasks Worst Case Execution Time (WCET), T_i is the time between two consecutive instantiations of the task, called period, and P_i is the priority of the task. Deadlines are equal to the task period T_i. The instance of a task is called job and the arrival time is called α. Thus we can say that the h^{th} job of task τ_i has to be scheduled for C_i time units in the interval $[\alpha_{i,h}, \alpha_{i,h} + T_i)$.

Without loss of generality, priorities are assigned to each task based on the rate monotonic priority assignment [21]. Each task is able to execute on every core of the platform. The set of all tasks is described by $\Gamma = \{\tau_i | i \in n\}$, where n is the number of tasks in the system. The utilization U_i of a task τ_i is described by C_i/T_i.

C. Scheduling

Task scheduling on many-core processors can be done in different ways. The partitioned scheduling algorithms assign tasks to cores. Those tasks then stay there during execution and well known single core scheduling algorithms can be used on core level. Global scheduling approaches manage all threads in a global run queue and jobs are assigned to processors during runtime. This can lead to the execution of consecutive jobs of the same tasks on different cores. A combination of both approaches assigns tasks to cores, but migration of tasks is allowed in a predefined manner during runtime. A recent survey by Davis and Burns [22] gives an overview of the current state of the art.

Our approach can be counted as combination of partitioned and global scheduling. Tasks are scheduled on each core by a fixed priority scheduling algorithm [21]. Task migration is allowed, but only as part of our approach to bound the peak temperature on the die.

In this work we denote the set of tasks assigned to a core j by $\mathcal{S}_j$, where $\Gamma = \bigcup \mathcal{S}_i$ and $i \in n$. The $\mathcal{NP}-hard$ problem of mapping tasks to cores leads to the exploration of heuristics and meta-heuristics [23]. In this work we apply the First Fit Decreasing Utilization (FFDU) heuristic to produce the initial mapping. FFDU starts by sorting all tasks $\tau_i \in \Gamma$ according to their utilization U_i in decreasing order. This way, big tasks are allocated on an empty system. Allocation is done by assigning tasks one by one to a core, until this core is not schedulable any more. If schedulability is not given, the algorithm assigns the following tasks to the next core.

IV. Temperature and power

This section describes the important aspects and relations of the temperature on the processor. Since temperature is tightly coupled with energy consumption and thus with the consumed power of the cores we also discuss the power model. We use both for the simulation based evaluation of this work in Section VI.

A. Temperature

As already mentioned, we assume a many-core processor with tiles arranged on a two dimensional grid. Since we target embedded-systems, we assume a passive heat sink covering all cores. A heat sink is an element used to remove thermal energy from the active elements and thus cool them down. To get more accurate simulation results, the heat sink is conceptually split into equally sized elements which cover the respective tiles.

There are several ways to model the relations between the different elements on the processor. It is possible to build the equivalent thermal network of the hardware [24]. This method describes the system by thermal resistance and capacitance analogue to electrical circuits. An other way of describing the thermal behavior is to apply Fourier's Law. The basic one dimensional form is given by the equation: $\vec{q} = -k\nabla T$ and states, that the change in heat transfer $\vec{q}$ is proportional to the thermal conductivity of the material k and the negative temperature gradient ∇T [25].

Like in [8] and [26], we use Fourier's Law to describe the thermal dependencies in the system. In the remainder the model and notation is described as in Fischer et al. [8].

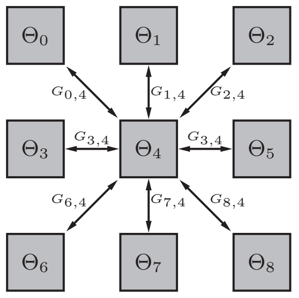
The temperature on the core j at time t is represented by $\Theta_j(t)$ and the temperature on the heat sink h connected to that core is represented by $\Theta_h(t)$. A constant ambient temperature of Θ_a is assumed throughout the lifetime of the system. The individual cores are consuming power during their execution which in turn leads to the change in temperature we want to describe. The power consumed by core j at time t is represented by $\Psi_j(t)$.

We group the elements into the set of all cores $\mathcal{M} = \{1, 2, \ldots, m\}$ and the set of all heat sinks $\mathcal{H} = \{1, 2, \ldots, m\}$. To describe the effect that the elements have on each other we define the respective thermal conductivity. The thermal conductivity between cores is defined as $G_{i,j}$ where $i \in \mathcal{M}$ and $j \in \mathcal{M}$. We further assume that $G_{i,j} = G_{j,i}$. The thermal conductivity between a core i and a connected heat sink l is described as $H_{i,l}$ and the thermal conductivity between heat sink h and g is described with $R_{h,g}$. Analogue to the conductance between cores we have $H_{i,l} = H_{l,i}$ and $R_{h,g} = R_{g,h}$. The thermal conductance of the heat sink to the environment is given by $G^{\dagger}$.

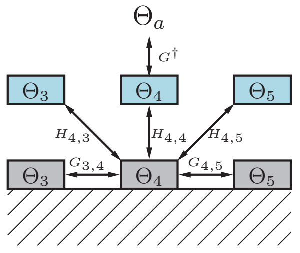
We visualize these relations in Fig. 1, where we have the top view on the silicon layer (Fig. 1(a)) and the side view, cut through cores 3,4 and 5 (Fig. 1(b)). For better understanding we omitted the relations between most other elements, only the thermal conductivity $\Psi^{\dagger}$ is shown on one heat sink to illustrate the connection to the ambient air.

Now we can formulate equations to describe the thermal process on the core (1) and on the heat sink (2).

$$C_j \frac{\mathrm{d}\Theta_j(t)}{\mathrm{d}t} = \Psi_j(t) - \sum_{h \in H} H_{j,h}(\Theta_j(t) - \Theta_h(t)) - \sum_{l \in M} G_{j,l}(\Theta_j(t) - \Theta_l(t)) \quad (1)$$



(a) Top view



(b) Side view

Fig. 1. Temperature relations on the many-core at the example of core 4

$$C_h \frac{\mathrm{d}\Theta_h(t)}{\mathrm{d}t} = -G^\dagger(\Theta_h(t) - \Theta_a) - \sum_{j \in \mathcal{M}} H_{j,h}(\Theta_h(t) - \Theta_j(t)) - \sum_{g \in \mathcal{H}} R_{g,h}(\Theta_h(t) - \Theta_g(t)) \tag{2}$$

where $\frac{\mathrm{d}\Theta_j(t)}{\mathrm{d}t}$ is the derivative of the temperature on core j and $\frac{\mathrm{d}\Theta_h(t)}{\mathrm{d}t}$ is the derivative of the temperature of the heat sink h. C_j and C_h are the thermal capacitance of core j and heat sink h respectively.

In a simplified way, we can say that the change in temperature is affected by the energy put into the system, minus the energy emitted to the neighboring cores and to the connected heat sink. A similar description holds for the heat sink, where the temperature is affected by the heat transfer between the connected cores, heat sinks and additionally by the ambient air.

B. Power

As mentioned before, we consider two power states of the processor, *active* and *idle*. Therefore we can define the two power levels. With shrinking transistor size the leakage current grows and we can not neglect it anymore [27], thus we formulate the two levels based on static and dynamic power:

$$\Psi_i = \begin{cases} \Psi_{active} \cdot \sum_{\forall j \in S_i} U_j + \Psi_{static} & \text{if } active \\ \Psi_{static} & \text{if } idle \end{cases} \tag{3}$$

We account for the different utilization on the cores by multiplying Ψ_{active} with the utilization of the cores. This simplification neglects the different power characteristics of the applications but it is sufficient for this work.

V. Methodology

In this section we propose a framework to bound the peak temperature of individual cores on a mesh based many-core processor, which in turn leads to reduced temperature gradients on the die. The substantially higher number of cores accommodated on a many-core processor make the mostly global control approaches used for multi-core processors intractable, the periodic collection of sensor values at a central node in the system leads to a communication hot spot and possibly to contention in the network. For large many-core platforms, the framework would also need to take the communication delays into account in order to control the system like expected. The popular Multi Input Multi Output (MIMO) controller rise in complexity with an increasing number of input output pairs. Because of the mentioned challenges we argue to tackle the problem by splitting it in the spacial domain. Therefore we propose a distributed approach. Each tile is controlled independently, limiting the complexity of the controller on each core.

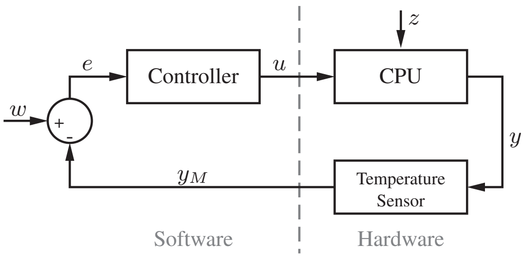


Fig. 2. Control loop for the thermal management

Fig. 2 illustrates the control loop on one core. The division into hardware components and software components is shown. The manipulated control input u is used to set the processor utilization needed to reach the desired set point w. This is done by measuring the control output y and converting it into temperature y_M. The error value e is then computed by subtracting y_M from w and used as input variable for the controller. Since we have to consider the temperature effect of surrounding cores we model this effect as disturbance z.

We control the utilization of the core in order to regulate the temperature. Thus we have several steps to perform:

1) Compute the needed change in utilization u.
2) Select a subset $\mathcal{U}_j \in \mathcal{S}_j$ with a total utilization $\geq u$.
3) Find a suitable destination core for each $\tau_i \in \mathcal{U}_j$.
4) Migrate each $\tau_i \in \mathcal{U}_j$ without missing the deadline.

These control actions are invoked periodically. The invocation interval is based on the desired trade off between accuracy and overhead. We further discuss each point in more detail.

A. Feedback control

PID controller are used in over 95% of all process control loops today, with most of them being PI controllers [28]. A PID controller is built of three different parts, proportional, integral and derivative. In this work we only use the proportional part. The missing differential part makes the system act slower. A slower system might be undesirable for most scenarios however, in our case, where a change in the control variable leads to task migration, a slower controller might be of advantage, leading to less migrations. We chose to spare the integral part as well, since it is used to remove the constant control error of the P-controller which in our case is neglectable. The parameter K_p is used to adjust the behavior of the controller. The following equation describes the controller in the time domain:

$$u(t) = K_p \cdot e(t) \tag{4}$$

The goal of most control loops is to keep a certain reference value by manipulating the system input. In our case those variables are temperature and utilization. However our objective is to bound the maximum core temperature, thus we do not need to reach this value and all values below are acceptable as well.

If we look at the control variable u (see Fig. 3), we can split the possible values in two regions. A negative region, where the controller needs to decrease the load and thus needs to migrate tasks, and a positive region where we can receive additional load without overheating. We call those regions overload and budget respectively.

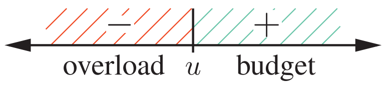


Fig. 3. Regions of the control variable u

B. *Selecting tasks for migration*

If the control variable u is computed and the value is located in the overload region, we have to reduce load on the core. This is done by migration. In order to migrate tasks we first need to decide which tasks we are going to migrate, which is described in this section. The problem at hand is equivalent to the knapsack problem. We have to find a subset of tasks $\mathcal{U}_k \in \mathcal{S}_k$ with total utilization close to u at a preferably low number of tasks in $\mathcal{U}_k$. Since the knapsack problem is known to be $\mathcal{NP}$ − hard, heuristics are applied.

We apply a FFDU inspired heuristic (see Algorithm 1) to find $\mathcal{U}_k$. First, all tasks are sorted by their utilization. Then we add the largest tasks to $\mathcal{U}_k$, one at a time, as long as the utilization of $\mathcal{U}_k$ does not exceed u. We continue trying to add smaller task to $\mathcal{U}k$ even after we encountered a task that could not be added. This is done because we want to reach u as close as possible.

Algorithm 1 Selecting $\mathcal{U}_k$

```
1: $\mathcal{U}_k = \emptyset$
2: $\mathcal{T} = \text{sort}_{dec}(\mathcal{S}_k, U)$
3: while $\text{size}(\mathcal{U}_k) \leq u \wedge \mathcal{T} \neq \emptyset$ do
4:     $\tau_t = \mathcal{T} \setminus \max\{U_p | \forall \tau_p \in \mathcal{T}\}$
5:     if $u - \text{size}(\mathcal{U}_k) \geq \text{size}(\tau_t))$ then
6:         $\mathcal{U}_k = \mathcal{U}_k \bigcup \tau_t$
7:     end if
8: end while
```

C. *Task migration*

Deciding if a real-time task τ_i can migrate from a core k to a core j at the current time t without missing its deadline is an important criteria. We apply the basic network latency for 2d-mesh based wormhole networks as described by Shi and Burns [29]. We do not consider lower and higher priority interference, since we need to apply the computation during runtime, we do not consider communication between tasks and since we can expect infrequent migrations as result of our approach.

The transmission latency $P_{k,j,i}$ is the time it takes to migrate a task τ_i of size L_i from core k to core j. This is described in Eq. (5). The first part of the equation computes the time it takes for the first flit to reach core j, which is dependent on the manhattan distance $H_{MD}(k, j)$ between the two cores and the constant switching delay encountered on each router S together with the constant delay for each hop based on flit size f and network bandwidth b. This value is then augmented by the time it takes for all consecutive flits to follow.

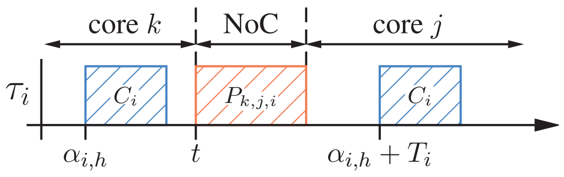


Fig. 4. Execution and migration times of a task τ_i

$$P_{k,j,i} = H_{MD}(k, j) \cdot (S + \frac{f}{b}) + \left\lceil \frac{L_i}{f} \right\rceil \cdot \frac{f}{b} \qquad (5)$$

We assume that the core flushes all task related data at the beginning of the migration process from local into off-chip memory. Since off-chip memory transactions are handled by a different NoC [2] we do not need to consider this for the calculation of $P_{k,j,i}$. Therefore, the message size L_i is reduced to the reference of the location of τ_i in the off-chip memory.

Fig. 4 depicts two consecutive jobs of a task τ_i at times $\alpha_{i,h}$ and $\alpha_{i,h} + T_i$. At time t the task is migrated between core k and core j, which takes $P_{k,j,i}$ time units as described above. We allow task migration only between two instances of the task. Therefore the task execution has to be finished and task migration has to be completed before the next instance of τ_i at time α_{i+1}.

Based on that, we can now define an inequality (6) which needs to be fulfilled in order to not affect the execution of the next instance of τ_i.

$$t \leq \alpha_{i,h} + T_i - P_{k,j,i} \qquad (6)$$

D. *Selecting destination cores*

We need to take different characteristics of the system into account when deciding for the destination core of a task τ_k. Since we can not neglect locality, if working with many-core processors, we need to consider the migration time. In prior work [15] we showed evidence that a neighborhood probing approach can perform as good as a global broadcast approach in order to find suitable cores for task migration in a framework to control peak temperature of cores, while drastically reducing the number of sent messages. Thus we pursue the neighborhood probing approach in this work.

Analogue to the control approach we handle the selection of destination cores in a distributed manner. We first describe the communication between two cores and then the algorithm to select those cores.

1) Communication between two cores: As depicted in Fig. 5, the overheated core k sends a request to a core j.

If core j accepts τ_i, it reserves the required amount of utilization for the task. This is needed in case another core sends a request before the actual migration takes place. Finally, if the response was positive, core k migrates the task as described before.

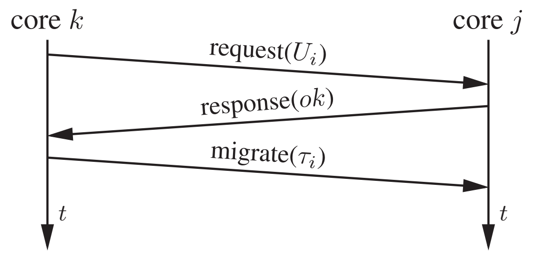


Fig. 5. Protocol to select destination core

2) Neighborhood probing algorithm: Here we describe the algorithm we use to select the possible destination cores for a task τ_i.

Several things need to be considered. The distance between the two cores affects the migration time which in turn affects the successfulness of a migration. A decision should also take the schedulability of the new task set on core k into account as well as the affects on the cores temperature, avoiding imminent overheating.

We can divide the necessary actions into parts executed on the sender and parts executed on the receiver side (see Algorithm 2 and 3 respectively.)

The overheated core starts to send requests to cores based on their distance $H_{MD}(k, j)$, starting with the closest cores. This neighborhood based search approach is done to minimize the communication overhead on the network.

Algorithm 2 Neighborhood probing, sender side

```
1: $\mathcal{C}$ = sort$_{acc}(cores, H_{MD})$
2: $c$ = min($\mathcal{C}, H_{MH}$)
3: $\tau_r$ = min($\mathcal{U}_k$)
4: while $\mathcal{U}_k \neq \emptyset \land \mathcal{C} \neq \emptyset$ do
5:     response = request($c, \tau_k$)
6:     if response = $true$ then
7:         $\mathcal{U}_k = \mathcal{U}k \setminus \tau_r$
8:         migrate($\tau_k, c$)
9:         $\tau_r$ = min($\mathcal{U}_k$)
10:    else
11:        $\mathcal{C} = \mathcal{C} \setminus c$
12:        $c$ = min($\mathcal{C}, H_{MH}$)
13:    end if
14: end while
```

After reception of the request, core j has to decide if $\mathcal{S}_k$ stays schedulable after τ_i is added. Since this check is done at runtime, we exploit the utilization threshold for rate monotonic priority assignment, similar as it is done by Jeon et al. in [30]. To increase readability of Eq. (7), we write $n = |\mathcal{S}_j|$.

$$U_i + \sum_{\forall \tau_z \in \mathcal{S}_j} U_z \leq (n+1)(\sqrt[n+1]{2} - 1) \tag{7}$$

Additionally, to guarantee the schedulability we want to prevent the core from overheating as effect of the load rearrangement. Thus we take the load budget, provided by the controller, into account. A positive value of the control variable u states that the core can increase its load in order to reach the temperature set by the reference variable. We use this value as budget, which can be received without imminent overheating.

Algorithm 3 Neighborhood probing, receiver side

```
1: $\tau_k$ = receiveRequest()
2: if $u > U_k \land$ checkSchedulability($\mathcal{S}_j, \tau_k$) then
3:     response(true)
4:     reserve($U_k$)
5: else
6:     response(false)
7: end if
```

VI. Evaluation

This section evaluates the proposed approach. We compare our result with two static mapping solutions and our previous work [15], where we used a simple hysteresis controller to bound the peak temperature. Instead of decreasing the load we moved all load and turned the respective cores off for cooling purposes. New destination cores for the tasks are found based on two proposed communication protocols. One global broadcast based protocol and one protocol which searches for new destination cores in a serial way based on the core distance, leading to less communication overhead.

A. Simulation setup

To evaluate our approach we use simulation. We generated random task sets, with each task having a uniformly distributed utilization between $(0, 0.7]$ and a uniformly distributed period between $[20, 100]$ ms. As described before, the initial task set is mapped to the cores by FFDU. This is done to activate a small number of cores at startup and then rearrange the load during runtime based on the temperature development on the processor.

We then changed the global utilization in steps of $5\,\%$ throughout the measurements, with 100 schedulable task sets for each data point. In order to stay below the schedulability threshold for rate monotonic we cover the range from 5% to 60% utilization in our experiments. We set the maximal temperature for a core to $72\,^{\circ}\mathrm{C}$. If this temperature is reached then the core is throttled by hardware circuits in order to protect it.

We simulate the temperature and energy consumption based on the theory explained in Section IV. The desired temperature value w is set to $60\,^{\circ}\mathrm{C}$ and the ambient temperature Θ_a is assumed to be constant at $25\,^{\circ}\mathrm{C}$. Our approach is then evaluated against our previous approach [15] which is based on a hysteresis controller. The parameter of the hysteresis controller where chosen as follows: $\kappa_{low} = 55\,^{\circ}\mathrm{C}$ and $\kappa_{high} = 65\,^{\circ}\mathrm{C}$. All control algorithms where periodically invoked with a period of 20 ms. We did no comparison against other work because of different assumptions either in the workload model or in the objective of the approach.

B. Controller parametrization

Since we have one parameter, K_p, that we need to set for the P-controller we need to define it. There are several possible ways to determine this value [28]. For our experiments, we choose to record the temperature step response of one core at a load step of $100\,\%$.

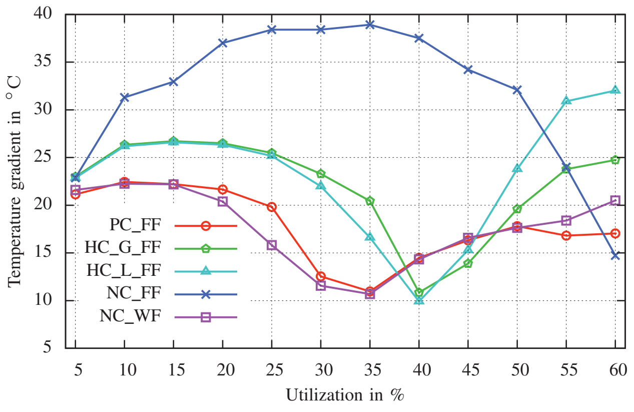


Fig. 6. Maximal temperature gradient on the die, for the P-controller approach (PC_FF) presented in this paper, hysteresis controller with local (HC_L_FF) and global (HC_G_FF) migration approach, and two data sets without controller. One with First Fit (NC_FF) and one with Worst Fit (NC_WF) mapping of the task set.

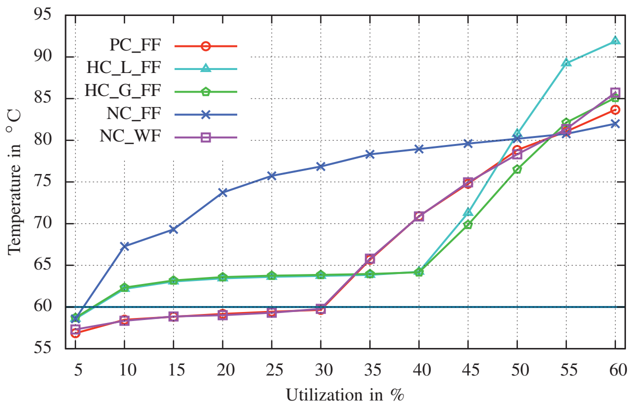


Fig. 7. Peak temperature on the die, for the P-controller approach (PC_FF) presented in this paper, hysteresis controller with local (HC_L_FF) and global (HC_G_FF) migration approach and measurements without controller and First Fit mapping of the tasks set (NC_FF). Additionally the control variable w is shown at 60 °C

The plant was identified as a 1st-order element [28], therefore we can obtain the parameters to describe the plant from the graph, which lead to the values $K = 22.75\,^\circ\mathrm{C}$ and $T = 337\,\mathrm{ms}$. The controller parameter K_p was set to 22.

C. Performance analysis

In this section we evaluate the main functionality, namely reducing the maximum temperature gradient on the die and limiting peak temperature.

Fig. 6 shows the maximal temperature gradient on the die for task sets with different utilization. This was obtained by subtracting the minimal temperature from the maximal core temperature. We compare the new approach with the previous hysteresis controller approach. Additionally, we compare the data against systems without any additional controller, once with FFDU mapping and once with worst fit (WF) task mapping, which is known to perform best for an even temperature distribution [32]. FFDU locates the tasks on as few cores as possible and thus adds to possible large temperature gradients. WF on the other hand spreads the tasks onto as many cores as possible leading to an even distribution and thus to low temperature gradients. The figure shows that the

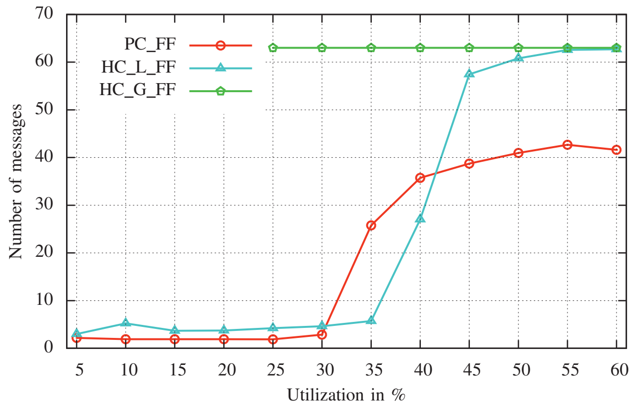


Fig. 8. Message overhead introduced by the proposed approach (PC_FF) and the previously proposed hysteresis controller based approaches with local (HC_L_FF) and global (HC_G_FF) task migration.

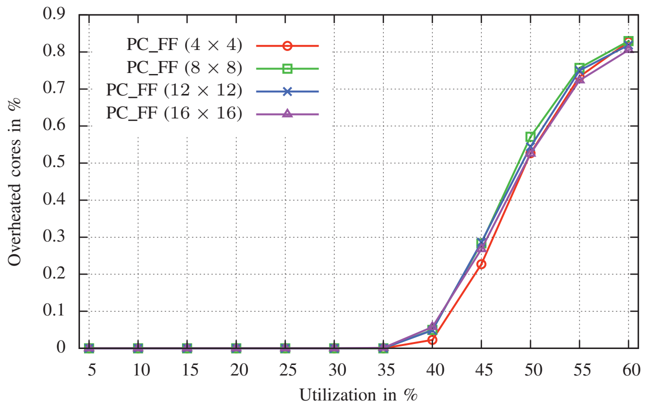


Fig. 9. Comparing the percentage of overheated cores for the proposed approach with platforms of different size, 4×4, 8×8, 12×12 and 16×16.

proposed approach performs close to the WF performance with lower temperature gradients than the hysteresis controller based approaches. All controller based approaches perform significantly better than the FFDU static mapping, even though this was chosen as the initial mapping for those experiments.

Fig. 7 shows the maximal core temperature during the experiments. Additionally to the measured curves, we show the desired maximal temperature w at 60 °C. The proposed approach performs as good as the WF mapping despite the initial FFDU task placement. We can see that at our approach performs slightly better than WF at a utilization level of 60%.

D. Message overhead

Communication bandwidth on the NoC is a valuable resource of the many-core and therefore we need to look at the overheads introduced by our approach. Since the worst case message overhead is by design equal to an global broadcasting approach we use the average message overhead and not the worst case. The new approach uses in average less messages compared to the hysteresis controller based approaches, see Fig. 8. The new approach even uses less messages, before the saturation point at 30 %.

E. Scalability

The number of cores on one die increases exponentially. Scalability of the proposed framework is therefore of special importance. For these experiments, we compare the performance of the approach presented in this paper on different grid sizes. Namely on a 4×4, 8×8, 12×12 and 16×16 grid, with 16, 64, 144 and 256 cores respectively. Our experiments show the ratio of overheated cores, since this is a good indicator of the system state. It is shown that the ratio of overheated cores is unrelated to the grid size and therefore to the number of cores (see Fig. 9).

VII. Conclusion

In this work we proposed a framework to increase the reliability of many-core processors with hard real-time workload by controlling the core temperature. This is done at core level, in order to reduce the overhead introduced by global approaches to keep the system state. Temperature is controlled by changing core utilization, which requires run time task migration in order to meet all deadlines. To do this, we proposed a localized migration scheme, reducing the number of average messages required compared to global approaches.

Our simulation based evaluation showed a lower message overhead compared to the hysteresis controller based approach. The maximal temperature gradient is close to the measurements obtained by an even load distribution through WF mapping and significantly lower than the gradients caused by the hysteresis controller based approaches. The static mapping obtained by WF performs good for constant workload. However, for most systems, workload is not constant. The experiments showed that our approach obtains close results to the WF mapping even though the initial mapping was FFDU based which is worst if the goal is to obtain an even temperature distribution.

In our future work we want to investigate techniques to predict the core temperature for processors without temperature sensors on each core. This is done by looking into ways of estimating the current per core temperature based on given sensor values as most COTS many-core processors do not provide reliable temperature sensors on each core. We also want to investigate the performance of our approach with different realistic workload types under consideration of dependencies between tasks.

References

[1] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[2] *Epiphany Architecture Reference*, Adapteva Inc., Adapteva Inc. 1666 Massachusetts Ave, Suite 14 Lexington, MA 02420 USA, 2012.

[3] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *31st ISCA*, pp. 276–288, 2004.

[4] F. J. Mesa-Martinez, E. K. Ardestani, and J. Renau, "Characterizing processor thermal behavior," *SIGPLAN Not.*, vol. 45, no. 3, pp. 193–204, Mar. 2010.

[5] R. Viswanath, V. Wakharkar, A. Watwe, V. Lebonheur, M. Group, and Intel Corporation, "Thermal performance challenges from silicon to systems," 2000.

[6] E. Grochowski, R. Ronen, J. Shen, and P. Wang, "Best of both latency and throughput," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, pp. 236–243, 2004.

[7] D. Dasari, B. Akesson, V. Nelis, M. Awan, and S. Petters, "Identifying the sources of unpredictability in cots-based multicore systems," in *8th SIES*, pp. 39–48, 2013.

[8] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *15th RTAS*, pp. 131–140, 2009.

[9] H. Yu, R. Syed, and Y. Ha, "Thermal-aware frequency scaling for adaptive workloads on heterogeneous mpsocs," in *DATE '14*, 2014.

[10] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *36th ISCA*, pp. 314–324, 2009.

[11] B. Yun, K. Shin, and S. Wang, "Predicting thermal behavior for temperature management in time-critical multicore systems," in *19th RTAS*, pp. 185–194, 2013.

[12] G. Liu, M. Fan, and G. Quan, "Neighbor-aware dynamic thermal management for multi-core platform," in *DATE '12*, pp. 187–192, 2012.

[13] V. Hanumaiah, A. Vrudhula, and K. Chatha, "Performance Optimal Online DVFS and Task Migration Techniques for Thermally Constrained Multi-Core Processors" in *IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, pp. 1677–1690, 2011.

[14] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *45 DAC*, pp. 734–739, 2008.

[15] M. Becker, K. Sandström, M. Behnam, and T. Nolte, "Dynamic power management for thermal control of many-core real-time systems," in *6th APRES*, 2014.

[16] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *47th DAC*, pp. 579–584, 2010.

[17] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, no. 2, pp. 62–76, 1993.

[18] Y. Xu, J. Zhou, and S. Liu, "Research and analysis of routing algorithms for noc," in *3rd ICCRD*, vol. 2, March 2011, pp. 98–102, 2011.

[19] Intel. Single chip cloud computer. http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html, Retrieved April 15, 2014.

[20] Tilera. Tile64 processor. http://www.tilera.com/products/processors, Retrieved April 15, 2014.

[21] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, 1973.

[22] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011.

[23] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *50th DAC*, pp. 1:1–1:10, 2013.

[24] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *14th VLSI*, vol. 14, no. 5, pp. 501–513, 2006.

[25] F. P. Incropera, A. S. Lavine, and D. P. DeWitt, *Fundamentals of heat and mass transfer*. John Wiley & Sons, 2011.

[26] T. Chantem, R. Dick, and X. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," in *DATE '08*, pp. 288–293, 2008.

[27] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *IEEE Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[28] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.

[29] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *2nd NOCS*, pp. 161–170, 2008.

[30] H. Jeon, W. H. Lee, and S. W. Chung, "Load unbalancing strategy for multicore embedded processors," *IEEE Transactions on Computers*, vol. 59, no. 10, pp. 1434–1440, 2010.

[31] W. Huang, K. Skadron, S. Gurumurthi, R. Ribando, and M. Stan, "Exploring the thermal impact on manycore processor performance," in *26th SEMI-THERM*, pp. 191–197, 2010.

[32] H. Aydin and Qi Yang, "Energy-Aware Partitioning for Multiprocessor Real-Time Systems," in *IPDPS '03*, 2003.