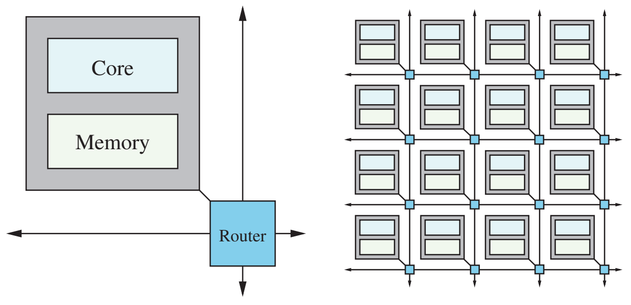
Dynamic Power Management for Thermal Control of Many-Core Real-Time Systems

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

***Abstract*—Many-core systems, processors incorporating numerous cores interconnected by a Network on Chip (NoC), provide the computing power needed by future applications. High power density caused by the steadily shrinking transistor size, which is still following Moore's law, leads to a number of problems such as overheating cores, affecting processor reliability and lifetime. Embedded real-time systems are exposed to a changing ambient temperature and thus need to adapt their configuration in order to keep the individual core temperature below critical values. In our approach a hysteresis controller is implemented on each core, triggering a redistribution of the cores' workload and the transition into an idle state allowing the core to cool down. We propose two approaches, one global and one local approach, to redistribute the tasks and relive overheating cores during runtime. We evaluate the two proposed approaches by comparing them against each other based on simulations.**



(a) One tile of a NoC (b) Tiled many-core architecture

Fig. 1. Architecture of a many-core processor using a 2D-Mesh NoC

I. Introduction

Thermal management becomes increasingly important for embedded real-time systems. The growing demand of computing power and the ongoing reduction of transistor size allows chip manufacturers to integrate multiple processor cores on one chip. The next generation of those parallel chips will accommodate numerous simple cores. This allows for more computing power but brings also new challenges due to the increasing power density on the die and the accompanying heat generation.

The shared bus as a traditional interconnection medium on the System-on-Chip (SoC) experiences scalability problems as the number of connected elements grows. Since the bus is a shared medium, contention delays start to dominate. The Network-on-Chip (NoC) was proposed to solve this problem and to allow for reliable interaction between the different elements on the SoC [1]. Each element is connected to a router, routers are connected to each other and thus a network is formed. The processor cores are located on so called tiles, together with a cache subsystem, Fig. 1(a). Most recent many-core processors [2]–[4] use a 2D-Mesh network to connect the different cores. Each router, besides the ones on the edges, has a connection in each cardinal direction plus a connection to the tile, Fig. 1(b). Since each router only needs to service those connections, it is independent of the number of tiles and thus large networks are possible.

Due to cost and space limitations most embedded systems need to operate without heavy cooling equipment. High temperatures on the die decreases the system reliability and lifetime [5], [6] and thus special attention should be given to thermal management in order to reduce cost and increase reliability. In this work we propose an approach to bound the maximum temperature of the individual processor cores by deploying a hysteresis controller on each core. If one core reaches a predefined threshold temperature, the proposed mechanism is activated and the core migrates all its tasks to other cores before it transits into an idle state in order to cool down.

II. Related Work

Much research has been conducted in the field of thermal aware single core scheduling. Most work applies Dynamic Voltage Scaling (DVS) on both single- and multicore processors [7]–[9]. Dasari et al. [10] look at the different sources of unpredictability of current off-the-shelf multicore systems. They first emphasize the importance of effective thermal control for safety critical systems and then look at the power saving strategies available on such hardware including a list of existing work in both power and thermal management for multicore processors. Fisher et al. [11] propose a thermal aware scheduling algorithm for sporadic real-time tasks targeting homogeneous multicore systems with the goal to reduce peak temperature. Fu et al. use thermal control to avoid overheating and thus frequency throttling of single core processors by applying feedback loops [9]. Both temperature and CPU utilization are controlled by their algorithm. They later present their novel Real-Time Multicore Control (RT-MTC) framework to handle the challenges introduced by multicore systems [8]. Yun et al. [12] predict the thermal dynamics of each core of a multicore processor using machine learning techniques. The thermal profile of each task is then used to predict if one core will overheat, before the next protocol invocation. If the core is expected to overheat, adequate steps like Dynamic Voltage and Frequency Scaling (DVFS) or clock gating are performed. However scaling the frequency during runtime is a crucial step for real time systems. A different frequency leads to variations in execution time, affecting latency and introducing jitter. Jeon et al. [19] propose a load unbalancing scheme to minimize power consumption of multicore processors. They adapt the number of active cores by concentrating the current load on as few cores as possible while disabling all additional cores. However they did not look at temperature effects.

III. Assumptions and Setup

A. Hardware Assumptions

In this work we assume a 2D-Mesh network structure supporting n identical cores. Each tile contains one processor core and a local scratchpad memory, similar to Adaptevas Epiphany chip [4]. The NoC uses wormhole switching [13] as wormhole switching is currently used by most available hardware [2]–[4]. In wormhole switched networks the message is divided into small elements, so called flits. A header flit contains information about the destination of the message and is used to route the message through the network. It locks the channel during its traversal, so all other flits can follow in a pipelined fashion with a tail flit freeing the locked channels again. If the header flit encounters a channel already locked by a different message, it blocks till the channel gets free. The static and deterministic XY-routing is implemented to guarantee deadlock and livelock free operation. We further assume that the processor offers capabilities to adjust the clock speed at core level.

B. Task Model

We assume N independent periodic tasks to be scheduled on the many-core platform. Each task τ_i can be executed on any core. Migration to other cores is allowed after the execution of one job finished and before the beginning of a new period. Γ denotes the set of all N tasks. Each task can be described by the tuple $\tau = \{C_i, T_i\}$ where C_i is the worst-case execution time and T_i is the task period. The utilization for a task τ_i is calculated by $U_i = C_i/P_i$. We further assume deadlines equal to periods.

C. Mapping and Scheduling

We apply partitioned scheduling. Each core i schedules its own set of tasks $S_i \in \Gamma$. Without loss of generality we assign task priorities according to the rate monotonic priority assignment, and scheduling is done using fixed priority preemptive scheduling [14].

The initial task mapping is done according to the First Fit Decreasing Utilization (FFDU) algorithm. First the task set Γ is sorted by its utilization in decreasing order. Tasks are assigned one by one, starting with the first core as long as the task set on that core stays schedulable including the new task. If this is not the case, the next core is used to assign the task.

D. Thermal Model

We consider a many-core architecture as depicted in Fig. 1(b) with one heat sink, a cooling element, on each core. By modeling the cooling element out of multiple small connected elements, we get a more accurate temperature on the individual locations. The heating and cooling process is a complicated dynamic procedure since the neighboring cores affect each others temperature. A way to model this behavior is the usage of Fourier's Law, which is done in most related work [11], [15]. We use the model and notation defined in [11].

The thermal model consists of cores and heat sinks connected by thermal conductances. The temperature on the core j and heat sink h at time t is defined as $\Theta_j(t)$ and $\Theta_h(t)$. The core j consumes the power $\Psi_j(t)$ at time t and thus heats up. This heat energy is then emitted to the neighboring cores and to the connected heat sink. The heat sink itself is a passive heat sink emitting heat to the environment which we assume to be at a fixed temperature Θ_a.

Fisher et al. define the set of cores $\mathcal{M} = \{1, 2, 3, \ldots, n\}$. The thermal conductance between two cores $j \in \mathcal{M}$ and $l \in \mathcal{M}$ is defined as $G_{j,l}$ where $G_{j,l} = G_{l,j}$. The capacitance of core $j \in \mathcal{M}$ is C_j.

$\mathcal{H} = \{1, 2, 3, \ldots, n\}$ is defined as the set of heat sinks. We define the thermal conductance of each heat sink to the environment as $G^{\dagger}$. The set of heat sinks connected to core j is defined as $\mathcal{H}_j$. Similar to the cores, neighboring heat sinks affect each other. This is modeled by the thermal conductance $G_{h,g}$ between sinks h and g, where $G_{h,g} = G_{g,h}$.

In a simplified way we can say that the change in temperature is calculated by the energy put into the system minus the energy emitted to neighboring cores and to the connected heat sinks. The following equations describe the thermal process on the core using Fourier's Law 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

$$\begin{aligned} C_j \frac{\mathrm{d}\Theta_j(t)}{\mathrm{d}t} &= \Psi_j(t) - \sum_{h \in H} H_{j,h}(\Theta_j(t) - \Theta_h(t)) - \\ &\quad \sum_{l \in M} G_{j,l}(\Theta_j(t) - \Theta_l(t)) \qquad (1) \\ C_h \frac{\mathrm{d}\Theta_h(t)}{\mathrm{d}t} &= -G^{\dagger}(\Theta_h(t) - \Theta_a) \\ &\quad - \sum_{j \in \mathcal{M}} H_{j,h}(\Theta_h(t) - \Theta_j(t)) \\ &\quad - \sum_{g \in \mathcal{H}} G_{g,h}(\Theta_h(t) - \Theta_g(t)) \qquad (2) \end{aligned}$$

One of the parameters used by the thermal model is the power consumed by the respective core. The consumed power of one core can be divided into two parts, dynamic $\Psi_{dynamic}$ and static Ψ_{static} power, where the dynamic power depends on the frequency [16] and the static power depends on the leakage current. Since the static power grows with shrinking transistor size it can not be neglected [17]. We use the approximations $\Psi_{dynamic} \approx C \cdot f \cdot V_{dd}^2$ and $\Psi_{static} \approx I \cdot V_{dd}$ to describe the dynamic and static power consumption, where C is the switched capacitance at each clock cycle, the clock frequency is denoted by f, V_{dd} denotes the supply voltage and I the leakage current.

Since we consider only two processor states, *active* and *idle*, we can define two static power levels to be used (3). In order to account for the workload on the individual core and its impact on the power consumption we weight $\Psi_{dynamic}$ with the cores utilization. This simplification neglects the power characteristics of the individual applications and the relationship between temperature and consumed power; however it is sufficient for the purpose of this work.

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

IV. Thermal Aware Approach

The main goal of our approach is to keep the temperature of the individual cores below a critical value and thus improving system reliability and lifetime [5], [6] while meeting all deadlines. To achieve this, we apply feedback control on each core. As a first approach we use a hysteresis controller and limit the processor states to active and idle, achieved by clock gating. It is further assumed that each tile is equipped with a temperature sensor.

A. Hysteresis Controller

We implement an hysteresis controller on core level. Two thresholds are used, κ_{low} and κ_{high}, both are design parameters. If the temperature Θ_j of core j exceeds κ_{high}, the core initiates a migration of its load and transitions into an idle state in order to cool down. If Θ_j falls under κ_{low}, the core is activated again.

B. Migration Costs

Locality is a big factor for NoC based many-core processors. Thus we have to consider this for the migration, since migrating a task to a core far away might cause deadlines to be missed.

A task can only migrate between instances of job executions. Thus we do not need to take a context switch into account. The tasks' context is located in the local memory of the tile and thus needs to be transfered to the destination tile. We assume that the location of the code is in external memory and that it does not need to be migrated.

The lack of communication between the tasks allows us to assume a contention free network since task migration is only performed as part of the thermal management which should be enacted sparsely. Thus we can neglect contention delays and only consider the basic network latency as described by Shi and Burns [18]:

$$P_d = \left\lceil \frac{L_i}{f} \right\rceil \cdot \frac{f}{b} + H_{MD} \cdot S \quad (4)$$

where P_d is the time needed to migrate τ_i to core d, L_i is the size of the task context, f the flit size and b the bandwidth. H_{MD} denotes the hops, the manhattan distance, between the two cores and S the constant delay incurred on each router. τ_i can migrate to core d if inequation $t \mod T_i \geq P_d$ is true. Some tasks with high utilization might not be able to migrate to any core. In this case all other tasks are migrated and the core stays *active* handling only those tasks.

C. Deciding the Destination Cores

Before we can send a core i into an *idle* state we need to migrate all tasks $\in S_i$ to suitable cores so no deadlines are missed. As we assume rate monotonic priority assignment we can use a similar method as Jeon et al. [19] and exploit the utilization threshold $\sum_{\forall j \in S_i} U_j \leq n(\sqrt[n]{2} - 1)$ to check if one task can be scheduled on a core, where $n = |S_i|$.

1) Global Approach: Since global knowledge is difficult to maintain in such systems each core acts independently. If migration is initiated, the core sends a broadcast message to obtain the current temperature and utilization of all other active cores in the system. Proceeding like this gives the advantage of not needing to know what cores are active and thus if it is possible to respond at the moment.

A response message r_i of core i can be described as a tuple $r_i = \{\Theta_i, U_i\}$ where Θ_i is the temperature of the core and U_i is its current utilization. $\mathcal{R}$ denotes the set of all responses.

In order to select suitable destination cores for all tasks $\tau_k \in S_i$, the overheated core i sorts its task set in decreasing order by utilization (Algorithm 1). Similar to FFDU we try to allocate the largest task first. In order to find a suitable destination core the set of all cores which can provide enough resources for τ_k are selected according to the utilization threshold. We further select the subset containing all cores that are close enough to guarantee task migration before the next instance of τ_k starts. The core $d \in \mathcal{D}$ with least Θ is selected as destination core.

Algorithm 1 Global Task Distribution Approach

```
 1: $\mathcal{R} = request\ states$
 2: $sort_{dec}(S_i, U)$
 3: for $\forall \tau_k \in S_i$ do
 4:     $\mathcal{D}_{sched} = \{r_i | \sum_{\forall j \in S_i} U_j + U_k \leq (n+1)(\sqrt[n+1]{2} - 1)\}$
 5:     $\mathcal{D} = \{d | d \in \mathcal{D}_{sched} \wedge t \mod T_d \geq P_k\}$
 6:     if $\mathcal{D} \neq \emptyset$ then
 7:         $d = min(\mathcal{D}, \Theta)$
 8:         $migrate(\tau_k, d)$
 9:     end if
10: end for
11: if $S_i = \emptyset$ then
12:     $transition(idle)$
13: end if
```

2) Local Approach: To minimize the communication overhead we propose a second approach as described in Algorithm 2. A request message is sent to only one core, which is selected based on its distance to the overheated core i. As in the global approach this core replies the tuple $r_j = \{\Theta_j, U_j\}$ which is used by core i to evaluate if tasks can be migrated or not. For this decision S_i is sorted by utilization in decreasing order and tasks are assigned to core j as long as the core has efficient resources to schedule them. If the core can not support more tasks but S_i is not empty, the next core is selected.

V. Evaluation

To evaluate the two approaches, we compare the resulting average peak temperature at different average utilization levels against measurements without implemented heat controller. Additionally the message overhead of the global and local

Algorithm 2 Local Task Distribution Approach

```
 1: $\mathcal{C} = sort_{distance}(cores, H_{MD})$
 2: while $S_i \neq \emptyset \vee \mathcal{C} \neq \emptyset$ do
 3:     $c = min(\mathcal{C}, H_{MD})$
 4:     $r = requestState(c)$
 5:     while $checkSchedulability(c, min(S_i))$ do
 6:         $\tau_k = min(S_i)$
 7:         $S_i = S_i \setminus \tau_k$
 8:         $migrate(\tau_k, d)$
 9:     end while
10: end while
11: if $S_i = \emptyset$ then
12:     $transition(idle)$
13: end if
```

approach is compared. All measurements are based on simulations. The task sets were generated randomly with task utilization uniformly distributed by $(0, 0.7]$ and the tasks period uniformly distributed by $[20, 100]\,\mathrm{ms}$, with 1000 task sets for each data point. Like in [20] we pick $85\,^{\circ}\mathrm{C}$ as the maximum silicon temperature. Due to space limitations only a platform with 8×8 layout, $\kappa_{low} = 50\,^{\circ}\mathrm{C}$ and $\kappa_{high} = 70\,^{\circ}\mathrm{C}$ was evaluated. The thermal parameters were chosen based on similar hardware.

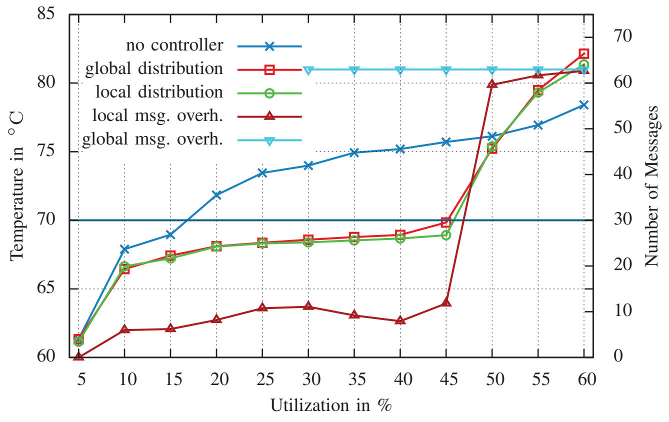


Fig. 2. Simulation results of different core utilizations

Fig. 2 shows the respective peak temperature curves for the three measured scenarios. The peak temperature of the two proposed approaches behave very similar, which is to expect since they only differ in the distribution strategy. Both outperform the system without thermal control up until the point when no suitable cores for load migration can be found. Looking at the average number of messages that the two approaches send at average we see that the local approach clearly outperforms the global approach.

VI. Conclusion and Future Work

The high energy density on many-core chips lead to a number of proposed approaches to manage the emerging heat. We first give suitable models to represent the involved components and then propose a method to increase the reliability and lifetime of many-core embedded real-time systems based on thermal management of the individual cores. Each core is equipped with a hysteresis controller used to switch between the processors *idle* and *active* state, allowing the core to cool down in the *idle* periods. Two strategies to safely migrate the real-time tasks before transitioning into *idle* state are proposed.

Future research will investigate if earlier migration of a subset of tasks can prevent the core of reaching the critical temperatures and thus allowing more reliable execution. We further want to extend the presented approach to support a more realistic task model and we want to consider dependencies between tasks. This is important especially for many-core processors where contention delays and longer distances between the two communicating cores can cause messages to miss their deadlines. This has to be taken into account in the decision of selecting the destination core before migration.

Acknowledgment

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

References

[1] L. Benini and G. De Micheli, “Networks on chips: a new soc paradigm,” *IEEE Comp. J.*, vol. 35, no. 1, 2002.

[2] Intel. Single chip cloud computer. http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html, Retrieved February 5, 2014.

[3] Tilera. Tile64 processor. http://www.tilera.com/products/processors, Retrieved February 5, 2014.

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

[5] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, “The case for lifetime reliability-aware microprocessors,” in *31st ISCA*, 2004.

[6] F. J. Mesa-Martinez, E. K. Ardestani, and J. Renau, “Characterizing processor thermal behavior,” *SIGPLAN Not.*, vol. 45, no. 3, 2010.

[7] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, “Power-aware scheduling for periodic real-time tasks,” *IEEE Transactions on Computers*, vol. 53, no. 5, 2004.

[8] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos, “Feedback thermal control of real-time systems on multicore processors,” in *10th EMSOFT*, 2012.

[9] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. Koutsoukos, and H. Wang, “Feedback thermal control for real-time systems,” in *16th RTAS*, 2010.

[10] D. Dasari, B. Akesson, V. Nelis, M. Awan, and S. Petters, “Identifying the sources of unpredictability in cots-based multicore systems,” in *8th SIES*, 2013.

[11] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, “Thermal-aware global real-time scheduling on multicore systems,” in *15th RTAS*, 2009.

[12] B. Yun, K. Shin, and S. Wang, “Predicting thermal behavior for temperature management in time-critical multicore systems,” in *19th RTAS*, 2013.

[13] L. Ni and P. McKinley, “A survey of wormhole routing techniques in direct networks,” *IEEE Comp. J.*, vol. 26, no. 2, 1993.

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

[15] T. Chantem, R. Dick, and X. Hu, “Temperature-aware scheduling and assignment for hard real-time applications on mpsocs,” in *DATE*, 2008.

[16] E. Grochowski, R. Ronen, J. Shen, and P. Wang, “Best of both latency and throughput,” in *22nd ICCD*, 2004.

[17] 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 Comp. J.*, vol. 36, no. 12, 2003.

[18] Z. Shi and A. Burns, “Real-time communication analysis for on-chip networks with wormhole switching,” in *2nd NOCS*, 2008.

[19] H. Jeon, W. H. Lee, and S. W. Chung, “Load unbalancing strategy for multicore embedded processors,” *IEEE Transactions on Computers*, vol. 59, no. 10, 2010.

[20] W. Huang, K. Skadron, S. Gurumurthi, R. Ribando, and M. Stan, “Exploring the thermal impact on manycore processor performance,” in *26th SEMI-THERM*, 2010.