# TAPE: Thermal-Aware Agent-Based Power Economy for Multi/Many-Core Architectures

Thomas Ebi, Mohammad Abdullah Al Faruque, and Jörg Henkel
University of Karlsruhe, Chair for Embedded Sytems
{ebi, alfaruque, henkel} @ informatik.uni-karlsruhe.de

## ABSTRACT

A growing challenge in embedded system design is coping with increasing power densities resulting from packing more and more transistors onto a small die area, which in turn transform into *thermal hotspots*. In the current late silicon era silicon structures have become more susceptible to transient faults and aging effects resulting from these thermal hotspots. In this paper we present an agent-based power distribution approach (TAPE) which aims to balance the power consumption of a multi/many-core architecture in a pro-active manner. By further taking the system's thermal state into consideration when distributing the power throughout the chip, TAPE is able to noticeably reduce the peak temperature. In our simulation we provide a fair comparison with the state-of-the-art approaches HRTM [19] and PDTM [9] using the MiBench benchmark suite [18]. When running multiple applications simultaneously on a multi/many-core architecture, we are able to achieve an 11.23% decrease in peak temperature compared to the approach that uses no thermal management [14]. At the same time we reduce the execution time (i.e. we increase the performance of the applications) by 44.2% and reduce the energy consumption by 44.4% compared to PDTM [9]. We also show that our approach exhibits higher scalability, requiring 11.9 times less communication overhead in an architecture with 96 cores compared to the state-of-the-art approaches.

**Categories and Subject Descriptors**:
C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

**General Terms**: Algorithms

**Keywords**: Dynamic Thermal Management, Agent-Based Systems, Multi-Core Architectures, Pro-Active Algorithm

## 1. INTRODUCTION AND RELATED WORK

Research has shown that if we want to keep on the track of the technology roadmap and meet the design challenges then system-level design using a multi/many-core paradigm is the most viable solution [1, 2]. The decrease in feature sizes from technology scaling is increasingly leaving these systems susceptible to several adverse effects such as the problem of *thermal hotspots* due to higher power densities in a particular area of a chip (e.g. on average from $2W/mm^2$ in $65nm$ technology to $7.2W/mm^2$ for $45nm$ [3]).

In recent studies [4, 5] *Dynamic Thermal Management* (DTM) has been discussed for multi/many-core architectures to tackle the problems of *thermal hotspots*. The goal of runtime dynamic thermal management is to keep the system below a safe operating temperature while maintaining real-time constraints for time/performance critical applications. The most common approaches for runtime *DTM* involve low-power techniques using *Dynamic Voltage and Frequency Scaling* (DVFS) which achieves a quadratic power reduction with an expense of linear performance degradation [6, 7]. Some pro-active approaches, i.e. approaches which act before actual *thermal hotspots* occur, have also been emerging for runtime *DTM* in multi/many-core architectures. In [8] the authors have presented an approach which uses an online learning strategy for thermal management. The authors of [9] introduce a predictive thermal model. However, these approaches lack scalability (discussed below) which is important when the number of cores increases.

Traditionally, low-power techniques have focused on minimizing energy or power. These approaches alone cannot achieve optimal *DTM* as they do not consider heat-specific factors such as the spatial distribution of power over the chip, the transfer of heat from one area to another, etc. Newer methods have taken these effects into consideration and achieved better results for runtime *DTM* by using a *Central Control Instance* (CI) to react to pending *thermal hotspots* by distributing power over the entire chip near-uniformly using runtime task migration [7, 8, 9, 10]. Task migration can be coupled with *DVFS* to maintain the heat generated by the power consumption of a *Processing Element* (PE) well below the temperature threshold determined at design time [9, 19]. However, as multi/many-core architectures are expected to soon incorporate hundreds or thousands of cores [12], runtime *DTM* will become increasingly difficult. Due to scalability problems, e.g. the increasing amount of data from thermal sensors to be communicated and the computational overhead for managing *DTM* in hundreds of cores, the *central control instances* used in [7, 8, 9, 10] will not be able to perform efficiently.

In order to deal with this growing complexity needed to manage multi/many-core architectures, new approaches have been proposed for system design. With the autonomic computing initiative introduced by IBM [13], there has been a consent that future systems will need to be *self-configuring* and *self-optimizing* that is system components must configure themselves and optimize their resource usage independently. Therefore, the *self-x* properties that are employed in the system may facilitate *DTM* in a pro-active manner. Agent-based systems are one of the most promising ways of realizing these properties.
In the scope of this paper we are the first to deploy an agent-based runtime *DTM* (details of agents and the mechanism are explained in Section 3). Our novel agent-based power distribution approach is inspired by the techniques deployed in *agent-based computational economics* [11].

### 1.1 Problem Formulation

Given a multi/many-core architecture $M$ made up of tiles $n \in M$, we define a region $R_j$ as the area covered by a rectangular connected subset of $M$ (i.e. groups of adjacent tiles forming rectangles), the temperature $T(t_1, r)$ for $r \in R_j$ at time $t_1$ can be expressed as:

$$T(t_1, r) = kh \int_0^{t_1} P(r,t)dt - \int_0^{t_1} \dot{Q}_{R_j}(t)dt + T(0,r) \quad (1)$$

*ICCAD'09,* November 2–5, 2009, San Jose, California, USA.
Copyright 2009 ACM 978-1-60558-800-1/09/11...$10.00.

where $P(r,t)$ is the power used at $r$ at time $t$, $\dot{Q}_{R_j}$ is the heat transfer rate to and from $R_j$, $h$ is the thickness of the silicon layer, $T(0,r)$ is the temperature at time 0, and $k$ is a unit constant. We aim to minimize the temperature of a region by avoiding power being concentrated in smaller regions,

$$\forall j,k\colon\ |R_j| < |R_k| \Rightarrow \int_{R_j} P(r,t)dr < \int_{R_k} P(r,t)dr \qquad (2)$$

and by spreading power equally over regions of the same area $A$, that is *minimizing* $\text{MAX}_{R_i} T(t,r)$.

$$\forall R_i\colon\ |R_i| = A\colon\ \text{MAX}_{R_i} T(t,r) = \text{MAX}_{R_i} \int_{R_i} P(r,t)dr \qquad (3)$$

Most *DTM* techniques, e.g. [9],[10], etc., do not consider managing power distribution until the temperature reaches a threshold. The approaches also rely on a *CI* for performing *DTM* which inversely affects the scalability of the next-generation thousand-core chip. For instance, the authors in [14] show that a distributed approach rather than using a *CI* requires 10.7 times lower traffic for collecting the state of the whole chip and 7.1 times lower computational effort for a $64 \times 64$ and $64 \times 32$ multi/many-core architecture respectively. The further consequences of a single *CI* are as follows:
(1) the *CI* needs to (re-)map many tasks at once due to lack of pro-active load balancing,
(2) the *CI* itself causing a *thermal hotspot* and becoming
(3) a central point of failure.
Considering the need of scalability and pro-active behavior of the *DTM* we are the first to propose an agent-based distributed thermal management technique which solves the above mentioned problems.
**Our novel contributions are as follows:**
(1) We present a highly scalable distributed agent-based power distribution approach (*TAPE*) which focuses on distributing power and the resulting heat evenly over a multi/many-core architecture. This results in minimizing Eq. 3.
(2) The approach incorporates thermal penalties in the agent negotiation to deal pro-actively with potentially developing *thermal hotspots* explicitly.
(3) We use an agent-based negotiation as a cost function for runtime application (re-)mapping which aids in avoiding the development of *thermal hotspots* in our proposed *DTM* approach.

The rest of this paper is organized as follows: after presenting an overview of our system models in Section **2**, we introduce our agent-based approach of runtime *DTM* in Section **3**. We then present our implementation and results in Section **4** and Section **5** respectively before concluding in Section **6**.

## 2. SYSTEM MODELS

In this section we present the models of different parts for our complete multi/many-core architecture where we have used our proposed agent-based runtime *DTM* scheme. We assume our multi/many-core architecture is made up of tiles consisting mainly of a PE, a communication element, an agent for power negotiation, and a power management unit (see Fig. 1). A **task** is considered to be a computational entity running on the PE of a tile. There can be multiple tasks running simultaneously on each PE.

- Each task (task$_i$) has a gate parameter $\alpha_{z,i} = \text{AVG}(\frac{g_{z,i}}{c_{z,i} \cdot g_{z,tot}})$ for each PE type PE$_z$ (i.e. considering heterogeneous PEs), where $g_{z,i}$ are the number of gates switched when running task$_i$ on PE$_z$, $c_{z,i}$ are the cycles it takes to run task$_i$ on PE$_z$, and $g_{z,tot}$ is the total amount of gates in PE$_z$. It specifies the percentage of gate switches per cycle required on an average when running task$_i$ on a PE$_z$.
- Furthermore, it is assumed that each task has a deadline $D_{\text{task}_i}$ as well as a limit for the execution time measured in cycles: $0 < c_{z,i} \leq \text{MAX}(c_{z,i})$.

### 2.1 Determining Power Budgets

For each PE inside a tile, the given power budget limits the processing that can be accomplished. This power budget is measured in the number *power units* available in a tile which define the amount of power the tile is allowed to consume. Using the simplified power formula for CMOS gates: $P = k_1 \alpha_{z,i} C_{\text{PE}_z} V^2 f + k_2 V I_{\text{leak}}$, we may observe the influence of the power budget ($k_1$ and $k_2$ are constant). Since $I_{\text{leak}}$ and $C_{\text{PE}_z}$ are assumed to be constant for PE$_z$, the parameters that the agents can change are limited to $V$ and $f$. These two parameters are roughly proportional to each other and there is a maximum $f$ for a given $V$. Furthermore, in our particular model it is assumed that $V$ is limited to $n$ discrete values, while $f$ can have any value achievable by a *Digital Clock Manager* (DCM).

For a PE to be able to run a task, it must be running at a minimum frequency of $f_{\min}$ so that $D_{\text{task}_i} \geq f_{\min}^{-1} \cdot \text{MAX}(c_{z,i})$. Since a given frequency has a minimum required voltage (frequency and voltage being roughly linearly proportional) the voltage is set to the lowest voltage above the minimum available in the tile's power profile. Thus, **a power unit is defined by the frequency it is able to achieve on** PE$_z$. The characteristics of our power budgeting are explained below.

- The **power unit granularity** $gr_z$ is defined by the amount of power units needed to run at a given frequency $f_0$. If coarse-grained power units are used, i.e. when one power unit is able to achieve a higher frequency relative to $f_0$, less power units are required to run at $f_0$ (making trading, see below, less complex) but may allocate more power than is actually needed. On the other hand, using fine-grained power units, i.e. those only able to achieve low frequencies, allows more precise power allocation but results in more complexity by requiring more trading. A measure for the power unit granularity $gr_z$ for PE$_z$ can be given as:
$$gr_z = \frac{\sum_i \text{power units } \textit{per} \text{ task}_i}{\#\text{ of tasks}} \qquad (4)$$
We then use $gr_z$ to define the relative PE factor $\xi_z$ which describes the power unit utilization of $PE_z$ relative to other PEs (PE$_{z'}$, $z \neq z'$). This value is used later to calculate the *buy* and *sell* values (see Section 3.2)), and allows our approach to be used for heterogeneous multi/many-core architectures. Assuming there are $N$ total PEs, $\xi_z$ is defined as follows:
$$\xi_z = \frac{(N-1) \cdot gr_z}{\sum_{z'} gr_{z'}} \qquad (5)$$
- The **task parameter** $\alpha_{z,i}$ can reduce the number of power units required to achieve a given frequency $f_0$. To be able to determine the average $\alpha_{z,i}$, first the worst case number of power units are allocated ($\alpha_{z,i} = 1$). The power usage is then measured and averaged over time.
- The **longest execution time measured** $\text{MAX}(c_{z,i})$ along with its **deadline** $D_{\text{task}_i}$ determine the required frequency $f_0$. The longest execution time measured has no worst case scenario known *a priori*. For each missed task deadline, the exceeding time must be added to the current longest execution time measured. The frequency must then be changed accordingly.

## 3. AGENT-BASED POWER ECONOMY

Before incorporating the models presented in the previous section into our approach, we first introduce our novel agent-based power economy for managing a runtime *DTM*. The definition of an agent motivated by [11] is given below:

*Definition of an Agent:.*
An agent is a *situated* computational entity that acts on the behalf of others which is *autonomous* and *flexible*. For our approach it is necessary to situate our agents at the same level as at which power management is done (i.e. one agent per tile). Since we are doing power management and runtime

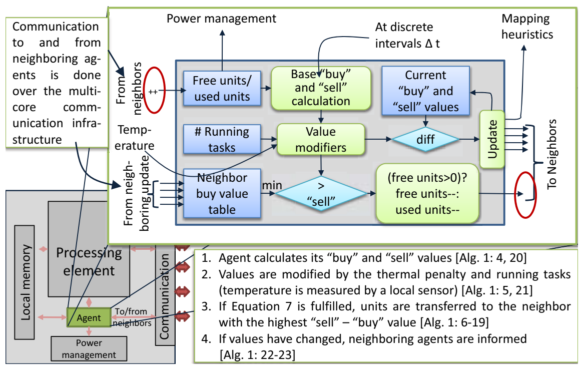

Communication to and from neighboring agents is done over the multi-core communication infrastructure
Power management
At discrete intervals Δ t
Mapping heuristics
From neighbors
++
Free units/ used units
Base "buy" and "sell" calculation
Current "buy" and "sell" values
Temp-erature
# Running tasks
Value modifiers
diff
Update
To Neighbors
From neighboring update
Neighbor buy value table
min
> "sell"
(free units>0)? free units--: used units--
Local memory
Processing element
Agent
To/from neighbors
Power management
Communication
1. Agent calculates its "buy" and "sell" values [Alg. 1: 4, 20]
2. Values are modified by the thermal penalty and running tasks (temperature is measured by a local sensor) [Alg. 1: 5, 21]
3. If Equation 7 is fulfilled, units are transferred to the neighbor with the highest "sell" – "buy" value [Alg. 1: 6-19]
4. If values have changed, neighboring agents are informed [Alg. 1: 22-23]

*Figure 1:* Functionality of an agent during negotiation

*DTM* at tile level,each tile also has a single agent in our architecture. Agents act autonomously in order to accomplish a common goal (in our case evenly distributing power) and are not dependent on other agents (although they react to the actions of each other). In particular, there is no higher instance controlling the agents. An agent is considered flexible when it is *responsive* (i.e. responds to input from other agents), *pro-active*, and *social* (i.e. communicates with other agents). In a nutshell our agents implement these properties where each of them acts on behalf of one tile. Plus, an agent trades power units with the agents of adjacent tiles using the agent negotiation presented below.

*Benefits of using Agents:*.
Agents exhibiting the above properties may be able to individually distribute power units between tiles of a multi/many-core architecture through local trade, i.e. only with their neighbors. They may exhibit an emergent behavior, i.e. many agents individually following simple rules are able to accomplish the power distribution when working together on a local basis only. As the hardware and software overhead of individual agents does not grow with system size (unlike the *CI* used in [7, 8, 9, 10]), agent-based approaches are highly scalable for next generation thousand-core chips [12]. The pro-active characteristics of agents allow our *DTM* to pro-actively minimize the possibility of *thermal hotspots* during execution time compared to [7, 8, 9, 10]. To the best of our knowledge, current usage of agents in multi/many-core architectures so far has been limited to task mapping [14].

## 3.1 Agent Negotiation

Our agents are able to negotiate with their immediate neighbors in all directions (e.g. 4 neighbors in the North, South, East, and West directions for a multi/many-core architecture having a regular 2D Mesh topology) to optimize power budgets.

Initially, a predefined number of power units are distributed evenly (depending on the PE factor $\xi_z$ in heterogeneous multi/many-core architectures) among all agents. Each of these power units belong to one of the following two categories: • *used* units are currently deployed by the PE to run tasks, i.e. *used* units are used to set the tile's frequency and voltage as specified above (Section 2.1), and • *free* units are power units available on a tile, but not currently needed by the tile to run its allocated tasks since the required frequency and voltage are already set by *used* units. These units are used by each agent to calculate *buy* and *sell* values for trading power units. These *buy* and *sell* values are not constant but vary depending on the *supply/demand* of power units calculated by each agent. The underlying main concept is that it becomes increasingly difficult to obtain power units if there are already many power units being used in a local area (high demand) even though there is no defined upper bound, as agents at the edge of a local area will trade with immediate neighboring agents outside this local area. This whole model may be compared to the classical ***supply/demand*** **model of computational economics**.

An overview of the agent negotiation is given in Fig. 1,2, and in Alg. 1. At each time interval (see Section 4 for details of the time interval) the agent first calculates its base *buy* and *sell* values. The calculation of these values at agent $n$ is given as follows (Alg. 1, L. 4-5):

$$\begin{aligned} \text{sell}: \quad & sell_n = w_{u,s} \cdot used_n + w_{f,s} \cdot free_n \\ \text{buy}: \quad & buy_n = w_{u,b} \cdot used_n - w_{f,b} \cdot free_n + \gamma \end{aligned} \tag{6}$$

where, $free_n$ and $used_n$ are the number of *free* and *used* power units respectively, $w_{i,j}$ are weight parameters, and $\gamma$ is a normalized offset parameter. $free_n$ may also be negative if new power units are required to execute a new task (see Section 3.3).

Once the base *buy* and *sell* values have been determined, these are then decreased based on the current temperature of the tile and increased based on the currently executing tasks. The final values are given as follows:

- *Buy* modifier:
  $buy_n = buy_n - a_b \cdot T_n$ (Alg. 1, L. 6)
  Here, $a_b$ is the normalizing factor showing the relationship between the temperature and the power units. The modifier of the *buy* value ensures that it becomes increasingly difficult for an agent whose tile has a high temperature to acquire additional power units. The modifier is only applied once the temperature reaches a given minimum temperature $T_0$ (a pro-active behavior) above which the modifier is to be applied as $T_n$ is measured in degrees above $T_0$ and is referred to as the *thermal penalty*.
- *Sell* modifier:
  $sell_n = sell_n + \sum_{\text{task}_i} p_i - a_s \cdot T_n$ (Alg. 1, L. 7)
  The *sell* modifier at the same time decreases the *sell* value when temperatures increases (*thermal penalty*) making it more likely that neighboring agents buy power units (again, the *thermal penalty* is only applied once the tile temperature surpasses the given minimum temperature $T_0$). At the same time it also considers the running tasks task$_i$ (with the weights $p_i$ dependent on the task execution time characteristics i.e. hard real-time constraints). This ensures that power units are first traded by tiles running less tasks or tasks with no hard real-time constraints.

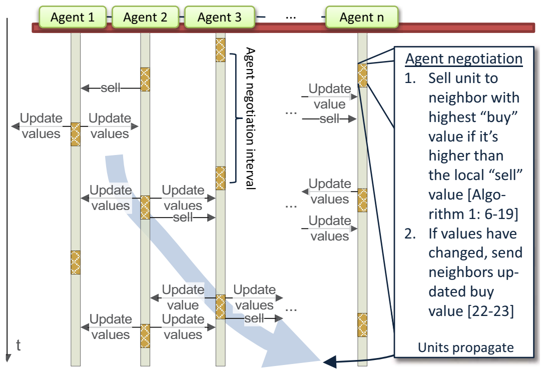

Agent 1
Agent 2
Agent 3
...
Agent n
Agent negotiation interval
sell
Update values
Update value
Units propagate
t
Agent negotiation
1. Sell unit to neighbor with highest "buy" value if it's higher than the local "sell" value [Algorithm 1: 6-19]
2. If values have changed, send neighbors updated buy value [22-23]

*Figure 2:* Sequence of the agent negotiation (for simplicity only 1 dimension is shown)

The neighbors of a specific tile $n$, i.e. all tiles adjacent to $n$, are informed when its modified *buy/sell* values have changed over time. The neighboring tiles then store the updated *buy/sell* values locally in order to minimize redundant communication. At the next trade time (i.e. after waiting the rest of their given interval time, see Section 4), the agents within the tiles compare their own resulting modified *buy/sell* values with the *buy/sell* values they have obtained from their neighbors ($i \in N$) to establish where more power units are needed and where there are "too many" power units being used, i.e. if the peak temperature has crossed a threshold or one PE is consuming a considerable amount of power while others are idle, using the following equation (Alg. 1, L. 8):

$$(sell_n - buy_n) - (sell_i - buy_i) > \tau_n \tag{7}$$

where $\tau_n$ is the sell threshold of tile $n$, with $\tau_n \approx w_{f,s}$. This threshold is to ensure that there is no redundant trading, i.e. two agents continuously trading one *free* unit back and forth (a ping-pong effect in trading).

If any of the neighbors $i \in N_n$ fulfill Eq. 7, agent $n$ relinquishes a power unit to the neighbor with the maximum $(sell_i - buy_i)$ value. If the local agents have *free* power units then this is simply a matter of decrementing its number of *free* power units. However, if there are no *free* power units then the local agent must give up one of its *used* power units. This means the agent must also reduce its power consumption through *DVFS* (Alg. 1, L. 12-13). If as a result of *DVFS* tasks will be unable to meet their deadlines then they must be (re-)mapped (Alg. 1, L. 15). This procedure is repeated until the system reaches a stable state, i.e. when the left side of Eq. 7 is less than $\tau_n$ in all tiles.

In this work we have assumed that the global amount of power units is a fixed number given at design time. Additional power scaling may be done by removing or adding power units to the multi/many-core architecture at runtime. However this is beyond the scope of this work.

## 3.2 Weights and Buy/Sell Value Characteristics

An important factor in ensuring that the system reaches a stable state is the choice of weights used to determine the *buy* and *sell* values. These weights are constant over time. Different tiles have different weights depending on their PE type (a heterogeneous multi/many-core architecture is considered). The weights can be seen as the value of a power unit for each PE. When choosing weights, the following factors must be considered:

- The number of *used* units has a greater influence in the *buy/sell* value than the *free* units as these are the actual cause of supply/demand in a local area. In an abstract sense, the *used* units can be seen as generating "value" which the agents use to acquire *free* units. The weights for *used* units are proportional to the power unit granularity $gr_z$ and consequently also to $\xi_z$ to allow heterogeneous multi/many-core architectures (Eq. 4).
$$w_{u,s}, w_{u,b} \propto gr_z, \xi_z \tag{8}$$
- The number of *free* units however must also influence *buy/sell* values in order to prevent one agent from acquiring all *free* units from its neighbors. The weight parameters of *free* units are proportional to the total amount of units (i.e. the initial an maximum number of *free* units), and also to $gr_z$ (and $\xi_z$).
$$\begin{aligned} w_{f,s}, w_{f,b} &\propto \sum(used + free) \\ w_{f,s}, w_{f,b} &\propto gr_z, \xi_z \end{aligned} \tag{9}$$
$$\gamma = k \cdot \xi_z \cdot \frac{\sum(used + free)}{m} \tag{10}$$
where $m$ is the number of total tiles in the multi/many-core architecture $M$.

From these properties we get the following equations for the weights
$$w_{u,b} = a \cdot \gamma, \qquad w_{u,s} = b \cdot \gamma,$$
$$w_{f,b} = c \cdot \tfrac{\gamma}{k}, \qquad w_{f,s} = d \cdot \tfrac{\gamma}{k}$$
To normalize these weights we define $\gamma := 1$, which also gives us $k$, and choose $a \in [1, 2)$, $b, c, d \in (0, 1)$ with $a > b, c < d$.

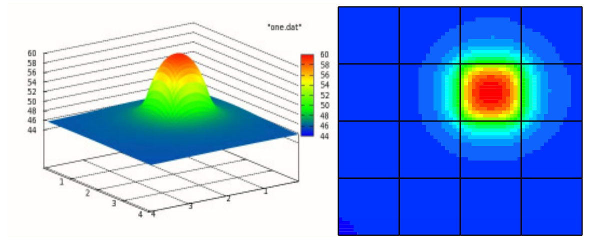
## 3.3 Runtime Application (Re-)Mapping

An agent-based runtime application (re-)mapping algorithm that is invoked due to power trading in the agents is integrated in our *TAPE* approach. Our (re-)mapping algorithm is based on the one presented in [14]. The goal of runtime application (re-)mapping is to determine which tile is best suited to migrate a given task to. The mapping agents are realized separately from the agents for power trading and are implemented in software running on one or more tiles (the implementation of the power trading agents is discussed in Section 4). The software entity that is responsible for our application (re-)mapping may be executed in any of the existing PEs inside the multi/many-core architecture. An overview of the interaction between the power trading agents and the mapping agents is presented in Figure 4. Depending on the situations during *DTM* the following scenarios may happen that require application (re-)mapping:

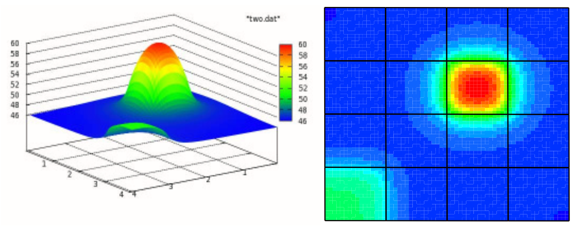
**Algorithm 1** Agent-based power economy for *DTM*

```
$sell_{base_n}$: Base sell value of a tile $n$ at time $t$
$buy_{base_n}$: Base buy value of a tile $n$ at time $t$
$T_n$: Temperature of a tile $n$ at time $t$
$sell_{T_n}$: Sell value of a tile $n$ at a temperature $T_n$
$buy_{T_n}$: Buy value of a tile $n$ at a temperature $T_n$
$N_n$: Set of all the neighboring tiles of tile $n$
$lastbuy$, $lastsell$: Last buy/sell values of a tile $n$ sent to all $i \in N$
$buy[N]$, $sell[N]$: List of buy/sell values of neighboring tiles stored in $n$
$free_n$: Free power units of tile $n$
$used_n$: Power units used for running tasks on tile $n$
$t_j$: Tasks running on tile $n$ at time $t$
$\tau_n$: sell threshold of tile $n$
 1: loop
 2:   for all tiles $n$ in parallel do
 3:     at every time interval $\Delta_n t$ do
          // Calculate base sell value
 4:       $sell_{base_n} \leftarrow (w_{u,s} \cdot used_n + w_{f,s} \cdot free_n)$
 5:       $buy_{base_n} \leftarrow (w_{u,b} \cdot used_n - w_{f,b} \cdot free_n + \gamma)$
          // The temperature increase may happen due to
          change in PE activity. Modify buy/sell value
 6:       $sell_{T_n} \leftarrow sell_{base_n}$ - $a_s \cdot T_n$ +
 7:       $buy_{T_n} \leftarrow buy_{base_n}$ - $a_b \cdot T_n \sum_{t_j} p_j$
 8:       if $\exists i \in N_n : ((sell_{T_n} - buy_{T_n}) - (sell[i] - buy[i]) >$
          $\tau_n$ then
 9:         if any free power units are left then
10:           decrement $free_n$
11:         else
12:           apply DVFS on $n$ to get more free power units
13:           decrement $used_n$
14:           if the task does not meet the given deadline as
              DVFS is used then
15:             (re-)mapping needs to be invoked
16:           else
17:             graceful performance degradation if allowed
18:           end if
19:         end if
20:         increment $free_i$
21:       end if
22:       if $buy_{T_n} \neq lastbuy$ or $sell_{T_n} \neq lastsell$ then
23:         send $buy_{T_n}$ to all $i \in N$
24:         send $sell_{T_n}$ to all $i \in N$
25:         $lastbuy \leftarrow buy_{T_n}$
26:         $lastsell \leftarrow sell_{T_n}$
27:       end if
          // This procedure will propagate until a stable state
          is reached.
28:     end at
29:     if received updated buy/sell values from any $l \in N_n$
        then
30:       update $buy[l]$, $sell[l]$
31:     end if
32:     if new task mapped to $n$ requiring $k$ power units then
33:       $free_n \leftarrow free_n - k$
34:       apply DVFS to PE on tile $n$
35:       $used_n \leftarrow used_n + k$
36:     end if
37:   end for
38: end loop
```

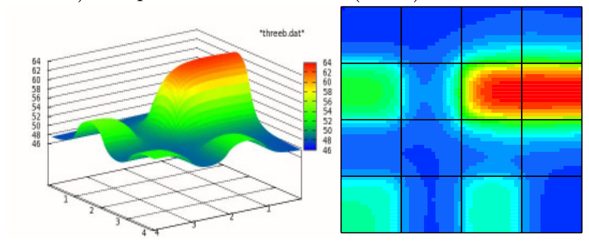
- If a new task needs to be mapped to a tile, it is mapped to one in which the most power units are available (either locally or through trade using agent-based negotiation). In our economic approach this is simply the tile where



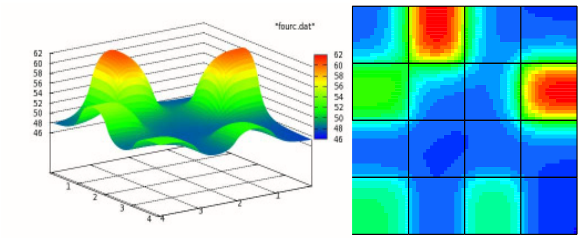
$a)$ Temperature for one task (task$_1$) at time $t_1$



$b)$ Temperature for two tasks at time $t_2$



$c)$ Five tasks at time $t_i$



$d)$ Task$_1$ (re-)mapped at time $t_{i+1}$

$$\underbrace{\begin{array}{cc}\underset{used}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}} & \underset{buy}{\begin{pmatrix}0.5&\cdots&0.5\\\vdots&\ddots&\vdots\\0.5&\cdots&0.5\end{pmatrix}}\\ \underset{free}{\begin{pmatrix}3&3&3&3\\3&3&3&3\\3&3&3&3\\3&3&3&3\end{pmatrix}} & \underset{sell}{\begin{pmatrix}0.6&\cdots&0.6\\\vdots&\ddots&\vdots\\0.6&\cdots&0.6\end{pmatrix}}\end{array}}_{t_0}
\underbrace{\begin{array}{cc}\underset{used}{\begin{pmatrix}0&0&0&0\\0&0&2&0\\0&0&0&0\\0&0&0&0\end{pmatrix}} & \underset{buy}{\begin{pmatrix}0.5&0.5&0.6&0.5\\0.5&0.6&2.4&0.5\\0.5&0.5&0.6&0.5\\0.5&0.5&0.5&0.5\end{pmatrix}}\\ \underset{free}{\begin{pmatrix}3&3&2&3\\3&2&5&3\\3&3&2&3\\3&3&3&3\end{pmatrix}} & \underset{sell}{\begin{pmatrix}0.6&0.6&0.4&0.6\\0.6&0.4&2.2&0.6\\0.6&0.6&0.4&0.6\\0.6&0.6&0.6&0.6\end{pmatrix}}\end{array}}_{t_1}
\underbrace{\begin{array}{cc}\underset{used}{\begin{pmatrix}0&0&0&0\\0&0&2&0\\0&0&0&0\\1&0&0&0\end{pmatrix}} & \underset{buy}{\begin{pmatrix}0.5&0.5&0.6&0.5\\0.5&0.6&0.3&0.5\\0.6&0.5&0.6&0.5\\1.2&0.6&0.5&0.5\end{pmatrix}}\\ \underset{free}{\begin{pmatrix}3&3&2&3\\3&2&5&3\\2&3&2&3\\4&2&3&3\end{pmatrix}} & \underset{sell}{\begin{pmatrix}0.6&0.6&0.4&0.6\\0.6&0.4&2.2&0.6\\0.4&0.6&0.4&0.6\\1.4&0.4&0.6&0.6\end{pmatrix}}\end{array}}_{t_2} \cdots$$

$$\underbrace{\begin{array}{cc}\underset{used}{\begin{pmatrix}0&0&0&0\\1&0&2&2\\0&0&0&0\\1&0&1&0\end{pmatrix}} & \underset{buy}{\begin{pmatrix}0.6&0.5&0.6&0.6\\1.2&0.7&-1.5&2.5\\0.6&0.5&0.6&0.7\\1.2&0.6&1.3&0.6\end{pmatrix}}\\ \underset{free}{\begin{pmatrix}2&3&2&2\\4&1&5&4\\2&3&2&1\\4&2&3&2\end{pmatrix}} & \underset{sell}{\begin{pmatrix}0.4&0.6&0.4&0.4\\1.4&0.3&-1.1&1.9\\0.4&0.6&0.4&0.3\\1.4&0.4&1.3&0.4\end{pmatrix}}\end{array}}_{t_i}
\underbrace{\begin{array}{cc}\underset{used}{\begin{pmatrix}0&2&0&0\\1&0&0&2\\0&0&0&0\\1&0&1&0\end{pmatrix}} & \underset{buy}{\begin{pmatrix}0.7&2.5&0.6&0.6\\1.2&0.7&0.7&2.4\\0.6&0.5&0.6&0.6\\1.2&0.6&1.2&0.6\end{pmatrix}}\\ \underset{free}{\begin{pmatrix}1&4&2&2\\4&2&2&5\\2&3&2&2\\4&2&4&2\end{pmatrix}} & \underset{sell}{\begin{pmatrix}0.3&1.9&0.4&0.4\\1.2&0.4&0.4&2.0\\0.4&0.6&0.4&0.4\\1.2&0.4&1.2&0.4\end{pmatrix}}\end{array}}_{t_{i+1}} \xrightarrow{}$$

*time*

*Figure 3:* An Exemplary Runtime Scenario explaining TAPE's Functionality (Section 3.4)

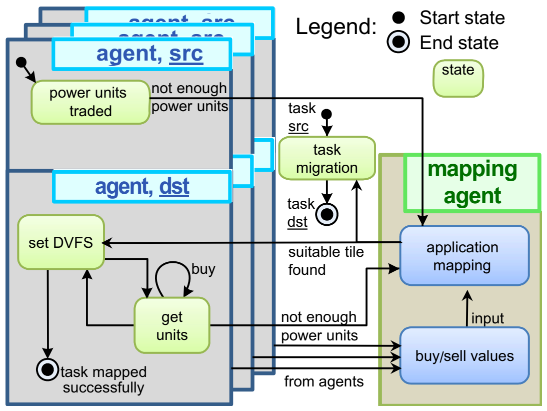


*Figure 4:* Runtime application (re-)mapping overview

the difference between its *sell* value and its *buy* value is maximal: MAX$_n(sell_n - buy_n)$ (if more than one tile fulfills this criterion, one is chosen randomly among the ones with the lowest absolute *buy* value). Once a tile is chosen, that particular tile decrements its *free* power units and increments its *used* power units. If not enough *free* power units are available, the '*free*' count becomes negative and the agent must acquire new power units from its neighbors.

- If during runtime a tile does not have enough power units for a currently allocated task to meet its deadline (i.e. the number of *used* power units is less than the required number of power units – meaning that the PE is thereby not operating at a high enough frequency to complete the task by its deadline – and there are no *free* power units) then the task needs to be (re-)mapped. A new tile that maximizes the difference between *sell* and *buy* values is chosen to migrate the task to.
- In case of failure to (re-)map a task the system rejects the new task and in case of a currently executing task it uses the concept of graceful performance degradation similar to [14].

An important consideration that comes with task (re-)mapping is how the task migration will be done. In *TAPE* we have adopted the approach presented in [14] during *DTM*. It realizes *task-relocation* by shutting down a task running on tile $n$ and transferring its context to the destination tile. Here, the context is then loaded and the task is resumed. In [15] it is shown that the task migration overhead similar to our approach ranges from 10 000 to 10 000 000 cycles depending on the task size. Details of task migration are not in the scope of this paper.

## 3.4 An Exemplary Runtime Scenario explaining TAPE's Functionality

In order to further illustrate the agent-based power economy, we present an exemplary scenario detailed in Fig. 3. The $4 \times 4$ 2D-Mesh architecturecan be seen as a part of a larger multi/many-core architecture, however to keep the example clear and be able to present all of the *buy/sell*, *free/used* values at each time $t$, only the $4\times4$ tiles are shown. First, we consider a multi/many-core architecture where all tiles are idle. All agents have only *free* units and all tiles have the same *buy/sell* values (a homogeneous multi/many-core architecture is considered here). At time $\boldsymbol{t_0}$ with no tasks running on the system, the left side of Eq. 7 is zero (i.e. less than $\tau_n$) for all agent pairs, resulting in a stable state. Now we assume at time $\boldsymbol{t_1}$ task$_1$ needs to be mapped onto the system. At this point, from a thermal and power perspective, all tiles are equally suited to run task$_1$. The (re-)mapping algorithm randomly maps the task to tile $n$ (Fig. 3a).[1] Tile $n$ then converts the number of power units

[1]These example figures include power traces from two applications (gcc and FFT, using the *wattch* simulator), the thermal values are calculated using the *Hotspot thermal simulator* [16], and the temperature $T_0$ after which the thermal penalty is applied is assumed to be 62 °C.

needed to run task$_1$ from *free* to *used*. This results in both an increase in *buy* and *sell* values and a decrease in the difference ($sell_n - buy_n$). Assuming that the difference of $n$'s *buy/sell* values ($sell_n - buy_n$) is now more than $\tau_n$ less than the difference in *buy/sell* values of any of its neighbors $N_n$, $n$ buys *free* power units from $l \in N_n$ in the next trade intervals of the neighbors until Eq. 7 again holds. At the same time, any $l$ that has given up power units will have increased *buy* (and *sell*) values. These $l$ then in turn trade with their other neighbors. The trading thus propagates throughout the chip until once again a stable state is reached. Now, assuming a second task, task$_2$ needs to be mapped on the chip, it will be mapped to a tile away from $n$ at time $\boldsymbol{t_2}$ (Fig. 3b).

Next, we will assume that at a future time $\boldsymbol{t_i}$ several tasks are running on the system and that, unfortunately, there is a potential *thermal hotspot* forming at tile $n$ (seen in Fig. 3c, the tile temperature exceeds 62 °C). The rise in temperature will cause $buy_n$ as well as $sell_n$ to decrease (shown in red in the timeline). Once the temperature has become high enough to decrease $sell_n$ to the point where the left side of Eq. 7 is larger than $\tau_n$ for one of the $l$, $n$ will be forced to give up a power unit. Assuming $n$ has no *free* power units, $n$ must give up a *used* unit. We assume that $n$ is now not able to accommodate task$_1$. Thus the mapping agent is informed and task$_1$ is (re-)mapped (Fig. 3d) at time $\boldsymbol{t_{i+1}}$. The remaining *used* units are converted into *free* ones.

## 4. IMPLEMENTATION DETAILS AND HARDWARE PROTOTYPE

The agents store information (e.g. *buy/sell* values of neighboring agents, the current *free* and *used* units, and tasks currently allocated to the tile) in local memory. They receive thermal information from a thermal sensor also located in the tile. Each tile has one sensor as one thermal value is needed for the *thermal penalty* in the agent negotiation and the area requirement is marginal (see hardware prototype below). Neighboring agents communicate (i.e. trade negotiation) with each other through the multi/many-core architecture's communication infrastructure. Agents can be implemented either in hard- or software. Both have their advantages and disadvantages. However, software agents may not be available in all tile types, e.g. for a dedicated DSP or ASIC.

- **Hardware Agents** When not using a general purpose PE, it is necessary to implement agents in hardware. Hardware agents require additional on-chip area. Our hardware agent prototype takes up 143 slices of a Xilinx FPGA (i.e. less than 1% of the slices of a Virtex-4). Hardware agents do not interfere with tasks running on the tile's PE.
- **Software Agents** are realized as non-transferable tasks running on the PE of a tile. Therefore, they take power and processing time away from the power units allocated to the tile to run its other tasks. Software agents can also serve as a fall-back for hardware agents, that is if a hardware agent were to fail, its functionality may be replaced through a software agent. If this is not possible (i.e. software agents are not available) the power manager may simply choose to use a predetermined voltage/frequency setting, thus the agents do not present a central point-of-failure in a tile.

The frequency with which agent negotiation occurs, i.e. due to the time interval between agent activity is based on the maximum rate of temperature increase: it takes at least 100K cycles to raise the temperature by 0.1 °C [7]. We choose the negotiation interval to have a maximum increase of 0.1 °C − 1.0 °C, which is enough to detect significant increases in temperature. Even when choosing a small interval (e.g. every 100$\mu$s) the power consumed by the agents is still negligible compared to the rest of the architecture. It should also be noted that *buy/sell* values in general do not change at each interval and thus agents do not have to communicate at each interval, thus further lowering the overhead.

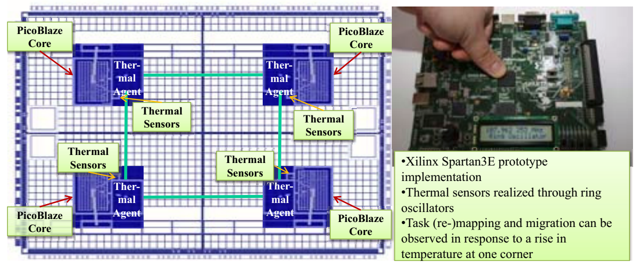

PicoBlaze Core
PicoBlaze Core
Ther-mal Agent
Ther-mal Agent
Thermal Sensors
Thermal Sensors
Thermal Sensors
Thermal Sensors
Ther-mal Agent
Ther-mal Agent
PicoBlaze Core
PicoBlaze Core
•Xilinx Spartan3E prototype implementation
•Thermal sensors realized through ring oscillators
•Task (re-)mapping and migration can be observed in response to a rise in temperature at one corner

*Figure 5:* Hardware implementation (PlanAhead)

To test the effectiveness of our agent-based power economy in avoiding *thermal hotspots* we have also implemented a small-scale version of our *TAPE* approach on a Xilinx Spartan3E FPGA. For convenience, the agents of all tiles in our example implementation are realized in hardware. Each tile also includes its own thermal sensor implemented as a ring oscillator which occupies one CLB (*Configurable Logic Block*) on the FPGA. The layout of the hardware implementation is shown in Fig. 5. In this case there are 4 PicoBlaze processors. As these processors do not produce a lot of heat, we have decreased the minimum temperature $T_0$ (and thereby increase the influence of the temperature on the *buy/sell* values) in order to achieve an effect. We have observed tasks migrating away from a corner when placing a thumb on that corner of the chip to increase the temperature showing that the agent-based power negotiation is successful.

## 5. EXPERIMENTAL SETUP AND RESULTS

For the thermal model we have used the following values: Each tile occupies a $1mm \times 1mm$ area on the chip. The silicon and copper (heat sink) layers are $0.6mm$ and $1mm$ thick respectively. The specific heat capacity of the two materials are $0.7kJkg^{-1}K^{-1}$ and $0.385kJkg^{-1}K^{-1}$ (with a material density of $2330kgm^{-2}$ for silicon and $8920kgm^{-2}$ for copper) and the thermal conductivity $148Wm^{-1}K^{-1}$ and $401Wm^{-1}K^{-1}$ respectively. The ambient temperature is 45 °C. We use the weights $w_{b,u} = 0.8, w_{s,u} = 1$ and $w_{b,f} = 0.1, w_{s,f} = \tau_n = 0.2$ which were determined at design time based on the equations in Section 3.2. The *thermal penalty* values are $a_b = 0.1$ and $a_s = 0.2$ and the temperature used ($T_n$) is the difference between the measured temperature $T_m$ and the ambient temperature $T_0$ ($T_n = T_m - T_0$). All agents are simulated in software and thus their own power usage is explicitly considered.

To evaluate *TAPE*, we run our agent-based thermal management approach with diverse applications of the MiBench benchmark suite [18] (e.g. calculating SHA hashes or encoding mp3s with LAME). These results are then compared to the same applications running without thermal management, using the *Heat and Run Thermal Management* (HRTM) technique described in [19], and using the state-of-the-art *Predictive Dynamic Thermal Management* (PDTM) from [9] which, is reported to achieve the highest reduction in peak temperature by using a predictive thermal model to migrate tasks before a thermal threshold is reached to tiles where the predicted temperature is the lowest. The HRTM technique is chosen to serve as a basis for comparison with other state-of-the-art techniques as it is a simple responsive DTM approach – migrating tasks when a threshold temperature is reached. The power traces used as input are generated by *SimpleScalar/Panalyzer 2.0* [17] and the temperature is calculated by the *Hotspot* [16] simulator. For the simulation we have used a $3 \times 3$ multi/many-core architecture and used agents implemented in software. Fig. 6 shows that *TAPE* is successfully able to keep the maximum temperature below a dynamic threshold of 47 °C and thus

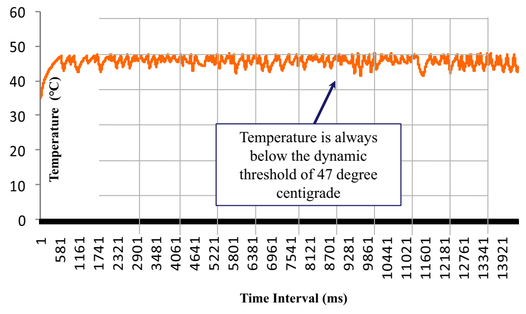


*Figure 6:* Maximum core temperature in simulation

avoid *thermal hotspots*. It should be noted however that this dynamic threshold is not given but dependent on the agent economy (or effectively on the power distribution on the chip).[2] When running the same scenario without our thermal-aware power economy, we see that the on-chip temperature peaks at 51.47 °C. Our results are similar to other *DTM* methods [9] achieving a 7.7% drop in peak temperature compared to having no *DTM* [14] (e.g. here there is a gain of 1.9% compared to [9]). Therefore, besides being highly scalable without added complexity, the pro-active behavior of our *DTM* approach keeps the maximum temperature well below the design-time-determined threshold at runtime compared to other approaches. The peak temperatures for further applications are shown in Fig. 7 where our approach achieves a similar decrease in peak temperature as the HRTM and PDTM approaches (e.g. 54.32 °C for *TAPE* versus 54.12 °C and 53.89 °C for HRTM and PDTM respectively for the LAME benchmark).

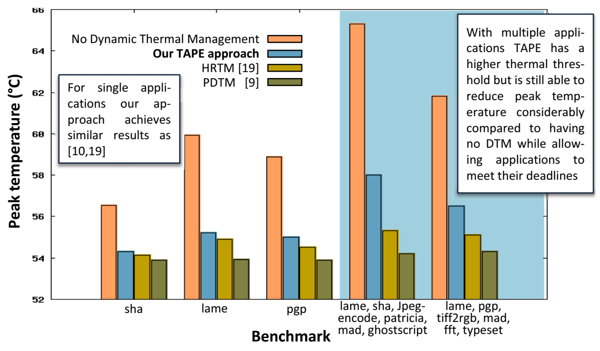


*Figure 7:* Peak temperature

When multiple applications are running on the multi/many-core architecture our approach exhibits a higher dynamic thermal threshold as there are more *used* units which raises the average *sell* (and *buy*) values and thereby decreases the impact of the *thermal penalty* (see Fig. 7). Here, our approach does not achieve the same reduction of peak temperature as the HRTM/PDTM approaches (TAPE: 58 °C, HRTM: 55.3 °C, PDTM: 54.2 °C) with PDTM achieving 4.65% less peak temperature, but it still reduces the peak temperature by 11.23% compared to having no *DTM* (Fig. 7).

The 15.9% increase in execution time compared to having no *DTM* seen in Fig. 8 is based on two reasons. First, the application is interrupted in discrete intervals (in our simulation every 4ms) to run the agent negotiation. This interval is based on the rate at which thermal measurements are received; in our simulation every 10ms. The second factor is the task migration which takes considerably more time than

[2]This is by design, a hard threshold can be implemented by not allowing tasks to be (re-)mapped to tiles whose agents have negative *buy/sell* values.

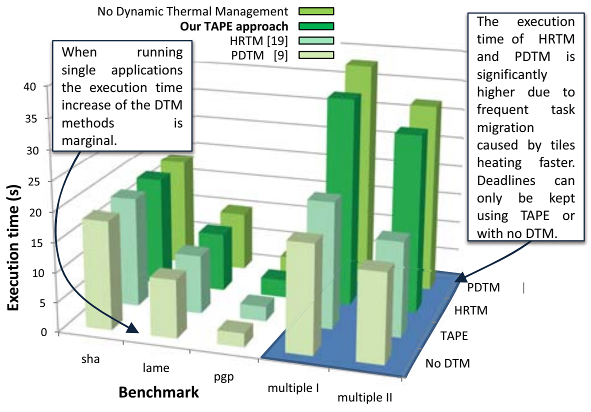


*Figure 8:* Total execution time

the agent negotiation (in our simulation, 100 000 cycles are assumed). This factor is responsible for the HRTM and PDTM approaches having a longer execution time/energy usage than our approach with HRTM and PDTM taking 39.4% and 44.2% longer respectively, while using 31.59% and 44.4% more energy.

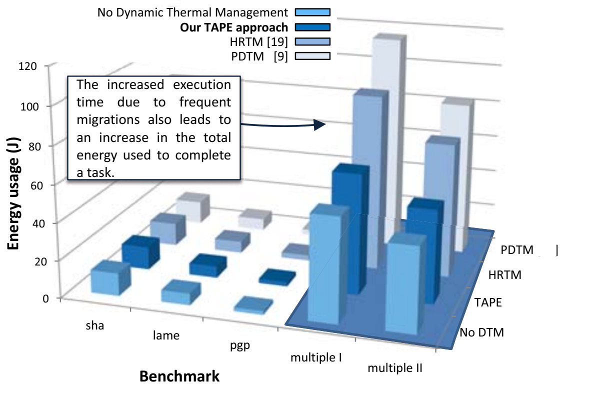


*Figure 9:* Total energy usage

The reason behind this is that our approach requires a lesser number of task migrations than the HRTM and PDTM approaches because our thermal threshold value is not fixed and has a certain degree of tolerance. Even though there are idle cores, tasks need to be migrated more frequently in HRTM and PDTM approaches to ensure that the temperature does not reach a strict threshold. This is due to the heat sink. As it is already spreading heat from other tiles, it is unable to conduct heat efficiently away from a new task on a new tile making the tile heating up significantly faster. For instance, when the thermal sensor reaches a threshold, it will immediately migrate the task to another tile when using HRTM. In our approach however, the agent will begin selling some *free* units when it starts approaching the threshold. The agent must first sell any *free* units it may still have before being forced to migrate a task. The energy used for each application (Fig. 9) directly correlates to the execution time of the application.

We require 4 bytes to encode the *buy/sell* values in our approach. The communication overhead required for power trading in Fig. 10 is shown for both the fully distributed *TAPE* approach as well as for an approach using a *CI* to implement the same functionality, comparable to [7, 8, 9, 10]. The exponential increase in overall communication of the *CI* clearly shows their lack of scalability for larger multi/many-core architectures. Our *TAPE* approach already requires

11.9 times less communication with as few as 96 tiles compared to an approach using a *CI*. The communication overhead in collecting the state information for (re-)mapping is caused by: 1) each tile reporting its state (*buy/sell* values) to a mapping agent and 2) synchronization between mapping agents, which only requires the transfer of $\text{MAX}_n(sell_n - buy_n)$. Thus the scalability of the runtime application (re-)mapping communication depends largely on the amount of mapping agents deployed. Fig. 11 shows the communication overhead for gathering the system state of the multi/many-core architecture dependent on the number of tiles and the number of mapping agents. The overall computational overhead for (re-)mapping and power trading remains the same, i.e. both power trading and (re-)mapping are done in $O(N)$ using a *CI*, whereas *TAPE* requires $O(1)$ for power trading in each of the $N$ tiles (and similarly mapping is done in $O(\log N)$), but is spatially distributed and parallelized when using agents thereby eliminating a central point of failure.

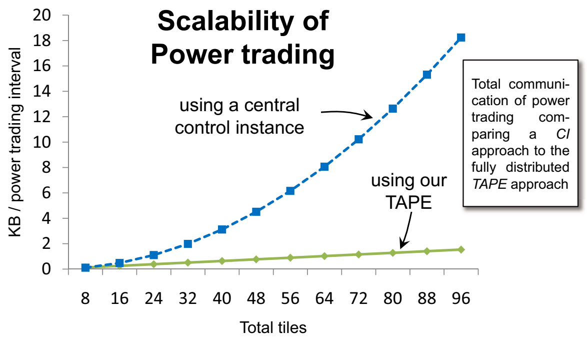


*Figure 10:* Scalability of power trading communication

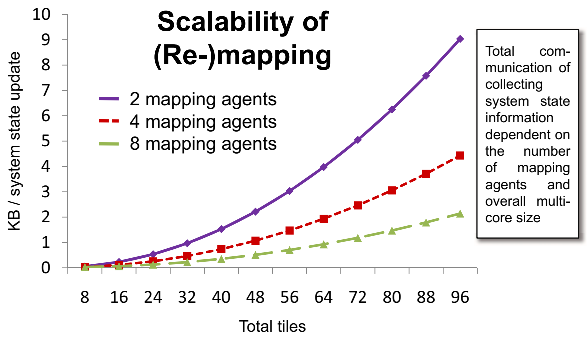


*Figure 11:* Scalability of (re-)mapping communication

In summary our approach is able to pro-actively distribute power over a multi/many-core architecture and achieve on average a 7.7% decrease in peak temperature compared to having no thermal management; when running multiple applications this is increased to 11.23%. While this is 4.65% more than when using PDTM [9], our approach requires 44.2% less execution time and uses 44.2% less energy compared to PDTM. These savings, together with the scalability of *TAPE* result in an ideal solution for future multi/many-core architectures.

## 6. CONCLUSION

In this paper we have presented the novel *TAPE* approach for minimizing the possibility of *thermal hotspots* through power distribution in a multi/many-core architecture pro-actively. We have utilized a classical *supply/demand* model from an *agent-based economic* approach [11] where neighboring agents trade power units locally. By limiting power trading to immediate neighbors, we are able to reduce the communicational overhead by 11.9 times compared to a central approach considering a 96-core architecture. These agents can be implemented in hardware as well as in software with hardware agents requiring chip area (143 slices) and software agents requiring execution time on a PE. We have simulated our approach using power traces obtained from the MiBench benchmark suite. We found that when running multiple applications simultaneously our approach uses a dynamic thermal threshold in order to keep deadlines and while PRTM [9] achieves 4.65% less peak temperature compared to our approach, we are still able to achieve an 11.23% reduction compared to having no *DTM* and are able to reduce the execution time by 44.2% and energy consumption by 44.4% compared to PRTM [9]. Therefore, the pro-active behavior stemming from using an agent-based approach as well as the higher scalability due to a distributed management scheme for our *TAPE* approach out-performs the current state-of-the-art approaches presented in [9, 19].

## 7. REFERENCES

[1] J. M. Rabaey and S. Malik, *Challenges and Solutions for Late- and Post-Silicon Design*, IEEE Design and Test, Vol. 25, No. 4, pp. 296–302, 2008.

[2] J. M. Rabaey, *Design at the End of the Silicon Roadmap*, ASP-DAC, pp. 1–2, 2005.

[3] G. M. Link and N. Vijaykrishnan, *Thermal Trends in Emerging Technologies*, ISQED, pp. 625–632, 2006.

[4] D. Brooks and M. Martonosi, *Dynamic Thermal Management for High-Performance Microprocessors*, HPCA, pp. 171–182, 2001.

[5] P. Chaparro, J. González, G. Magklis, Q. Cai, and A. González, *Understanding the Thermal Implications of Multi-Core Architectures*, IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 8, pp. 1055–1065, 2007.

[6] S. Herbert and D. Marculescu, *Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors*, ISLPED pp. 38–43, 2007.

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

[8] A.K. Coskun, T.S. Rosing, K.C. Gross, *Temperature Management in Multiprocessor SoCs using Online Learning*, DAC, pp. 890–893, 2008.

[9] I. Yeo, C. C. Liu, E. J. Kim, *Predictive Dynamic Thermal Management for Multicore Systems*, DAC, pp. 734–739, 2008.

[10] S. Heo, K. Barr, and K. Asanović, *Reducing Power Density through Activity Migration*, ISLPED, pp. 217–222, 2003.

[11] L. Tesfatsion, *Agent-based Computational Economics: Growing Economies From the Bottom Up*, Artificial Life, pp. 55–82, 2002.

[12] S. Borkar, *Thousand core chips: a technology perspective*, DAC, pp. 746–749, 2007.

[13] P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IBM Corporation, 2001.

[14] M. A. Al Faruque, R. Krist, and J. Henkel, *ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication*, DAC, pp. 760–765, 2008.

[15] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, and D. Atienza *Thermal Balancing Policy for Streaming Computing on Multiprocessor Architectures*, DATE, pp. 734–739, 2008.

[16] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusam, *Compact Thermal Modeling for Temperature-Aware Design*, DAC, pp. 878–883, 2004.

[17] T. Austin, T. Mudge, and D. Grunwald. Sim-panalyzer. http://www.eecs.umich.edu/~panalyzer/.

[18] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, *MiBench: A free, commercially representative embedded benchmark suite*, IEEE International Workshop on Workload Characterization, pp. 3–14, 2001.

[19] M. D. Powell, M. Gomaa, and T. N. Vijaykumar, *Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the System*, ASPLOS, pp. 260–270, 2004.