Comparative Framework for the Analysis of Thermal and Resource Management Algorithms for Multi-Core Architectures

Moeez Akmal*[1], Muhammad Sarmad Saeed*[1], Muhammad Usama Sardar†[1], Hareem Shafi*, Osman Hasan*, Heba Khdr‡ and Jörg Henkel‡
*School of Electrical Engineering and Computer Science
National University of Sciences and Technology (NUST), Islamabad, Pakistan
Email: [13beemakmal, 13beemsaeed, 13beehshafi, osman.hasan]@seecs.nust.edu.pk
†Technische Universitat Dresden, Dresden, Germany
Email: muhammad_usama.sardar@msx.tu-dresden.de
‡Karlsruhe Institute of Technology, Karlsruhe, Germany
Email:[heba.khdr, henkel]@kit.edu

***Abstract*—On-chip multi-core architectures have emerged as a new paradigm for executing highly parallel and resource demanding applications. However, inefficient resource management and thermal issues in these multi-core architectures may lead to performance degradation. To cater for these issues, several resource and thermal management algorithms have been proposed. These algorithms are usually analyzed independently and under different environmental settings, which makes a fair comparison between them very difficult. In order to overcome this problem, this paper presents a generic framework for analyzing various resource and thermal management algorithms under similar settings. The proposed framework is developed using C++ and has been successfully used to compare mDTM and a variant of DsRem.**

I. Introduction

Dynamic resource management (DRM) [1] is used to allocate tasks to a core in a multi-core architecture to optimize the utilization of the available resources. Similarly, dynamic thermal management (DTM) [2] re-allocates tasks from heavily-loaded cores to the lightly-loaded ones to balance the thermal profile of a chip. Decreasing the feature size increases the number of transistors that fit in the same area, which directly increases the power dissipated [3], thus forming thermal hotspots around the cores that are being used. Since the temperature of a core affects both the speed and the reliability of the processor [4], DTM is concerned while ensuring that the cores do not exceed a certain critical temperature T_{crit}, and that the tasks are allocated such that the variation in the temperature of the cores across the chip is minimized.

A considerable amount of work has been done on developing DRM and DTM algorithms based on centralized [7], distributed [8] and hybrid [9] schemes. The schemes are usually evaluated using simulation environments, like Sniper [5] and Multi2Sim [6], which are used to model multi-core architectures using programming languages, like MATLAB and C++. Simulation is usually found to be better than emulation for analyzing DRM and DTM algorithms as it is cost efficient, it does not require any specialized equipment, and the analysis can be done in reasonable time as well by manipulating the simulation clocks.

For a fair comparison, all the variables in the simulation based models, corresponding to the environmental conditions and inputs, except the algorithm itself should be kept the same. However, to the best of our knowledge, this practice is not being followed and most of the DRM and DTM algorithm designers use their own thermal, application and system models to assess their algorithms. This kind of assessment cannot be trusted completely as an algorithm may have been tested in a scenario, which is more favourable for it. To overcome these limitations, we propose a framework that enables comparison of DRM and DTM algorithms under fair circumstances.

The proposed framework has been developed using C++. The main motivation behind choosing C++ compared to MATLAB for this purpose is the higher performance offered by the C++ based simulations. Aruoba et al. [10] observed that C++ provides almost ten times faster execution compared to MATLAB and in certain cases provides even more than 100 percent improvement in speed for the same executable code. We are making the code of the proposed framework available as open source for the benefit of the research community [15]. Our implemented system model is based on a scalable approach where any number of homogeneous cores can be modelled to test the given algorithm. Moreover, it provides generic thermal, application and system models. To illustrate the effectiveness of the proposed framework, we use it to analyze 2 algorithms, i.e. Multi-objective Dynamic Thermal Management (mDTM) [4] and a variant of Dark Silicon Resource Management (DsRem) [11]. The results were found to be consistent with the corresponding emulation based results.

[1]These authors contributed equally to the work and the names are presented in alphabetic order.

II. Methodology

The proposed framework utilizes a scalable multi-core grid, composed of N homogeneous cores, for conducting the analysis. We have used generic application and thermal models from [11] in the proposed analysis framework, which can be used to analyze most of the existing DRM and DTM algorithms. The applications are mapped to the cores and their resources, i.e., the number of threads, voltage/frequency (v/f) levels are decided by the decision maker. When the application resources are decided, we find the corresponding steady-state temperatures for the time when that application is mapped to a core. Once the application has been mapped, the decision maker checks for any thermal violations. In case a thermal violation is found, it stops the running application and sends it to the waiting queue. These queued applications are then remapped via the decision maker. The iterator decreases the application execution time in every iteration and the temperature of the cores is updated. When a new application arrives, we again go to the decision maker and the process is repeated. The sub-modules of the proposed analysis framework are described in detail now:

A. Application Model

As mentioned earlier, we have used a multi-threaded application model [11], where TH_i represents the number of threads of an application *i*. For simplicity, we assume that only one thread runs on a single core and all the cores of one application consume the same power. This can, however, be extended to multiple threads' handling on a single core and/or use of heterogeneous cores. Each application i is characterized by its number of threads, voltage frequency levels vf_i, power consumption P_{app} and time t_i required for execution, as shown in Fig. 1. The execution time corresponds to the selected v/f levels, and thus we develop a table containing these attributes. The v/f levels for a core are selected depending on the constraints set by an algorithm, such as TDP, from the table of attributes. Under the given conditions, we maximize the throughput W_i of an application to get the best performance and execution times, where: $W_i = 1/t_i$.

B. Thermal Model

Our thermal model [11] employs the well-known RC thermal network, defined in [12]. This thermal model calculates the steady-state temperatures of the cores. When an application is running on a core, we consider the contributions of the ambient temperature T_{amb} and the power consumption of the other cores to calculate the steady-state temperatures using $T^{core} = B.P^{core} + C$, where T^{core} is a column vector representing the steady state temperature of the cores. The amount of heat generated by the cores is represented by the matrix B, and P^{core} is the power consumption of the core while running an application that is given by $P^{core} = P_{app}/TH_i$. Similarly, $C = T_{amb}.B_2.G$, where B_2 contains the amount of heat generated by the rest of the thermal nodes and G contains the heat contribution of T_{amb} to the thermal nodes. The core temperatures determine if an application can be mapped or needs to be stopped.

C. Algorithms

The proposed model of our system uses an iterative approach where, in every iteration, the remaining execution time for a running application decreases. The temperature of a core is also updated in every iteration using the slope formula [13] $\frac{dT}{dt} = b \times (T_{ss} - T)$, where T_{ss} is the steady-state temperature of the core, T is the current temperature of the core and b is an empirically determined constant. This way, the temperature is incremented or decremented in every iteration, if required. The T_{ss} term in this equation is equal to T_{core}, which is updated every time an application completes its execution. For the analysis purpose, a set of applications are pre-loaded which are to be executed. Once an application completes its execution, a new application awaits to be mapped on the core.

Algorithm 1 Working in Iterator

Input: Application model, temperature model, algorithm to be tested

Output: Algorithm feasibility analysis

1: **Initialize the system:** Cores, Temp. Model, App Model
2: Load app **threads** on the cores
3: Populate the **application queue**
4: Calculate **Total TDP** (Algorithm dependent)
5: Determine steady-state temp. & **slopes** with respect to $T_{present}$
6: **while** loop **do**
7: **if** App time > 0 **then**
8: App time decrements by a **constant time-stamp**
9: **end if**
10: **if** App thread running **then**
11: Core Temperature increment based on calculated slope
12: **end if**
13: **if** no app running on core **then**
14: Core Temperature decrement based on calculated slope
15: **end if**
16: **end while**
17: Decide on the new app to be loaded on cores based on available ***resources**

*empty cores, temperature constraint, power quota, v/f levels

The proposed framework allows to model a wide range of attributes of cores and applications, which can be changed in the core class as required. Once the attributes are finalized, they may or may not need to be varied in every iteration. For this purpose, those parameters which need to be varied with every iteration are placed in the main iterator, presented in Algorithm 1. The parameters which need to be varied at the complete execution of an application may be placed in the pre-existing functions or new functions may be created as required. Thermal management techniques for an algorithm may be applied in the T_{crit} avoidance function or by creating a

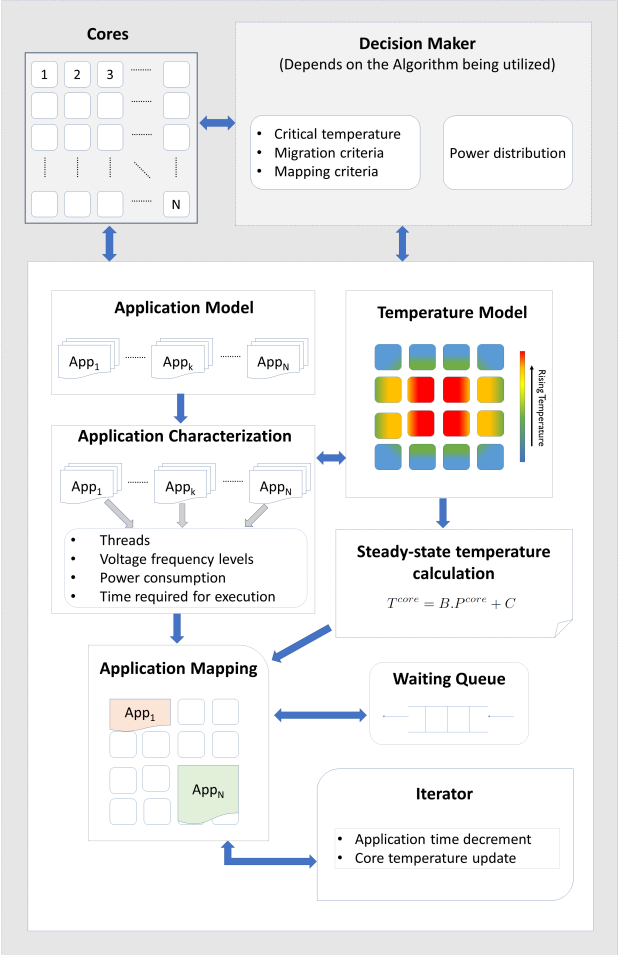


Fig. 1: Overview of our System

new function, like the Central Management Unit for Achieving Thermal Balance (CU-AB) [4].

Application related changes, like frequency of application cores, power consumption, number of threads and other similar parameters can be made in the application functions. We get the steady-state temperatures from the get temperature function whereas the decisions regarding incoming applications are made in the incoming application function.

III. Case Studies

A. mDTM

mDTM focuses on solving the thermal problem by using a multi-objective approach, i.e., resolving the thermal issues by first rectifying the thermal violations and then balancing the temperature throughout the chip, by migrating applications from the hottest to the coldest cores. mDTM avoids the thermal violations by predicting when the temperature of a core exceeds the critical temperature, i.e., T_{crit}. At a given time, this algorithm predicts core temperature for the next instant, and we use this temperature to avoid thermal violations.

We used the proposed analysis framework to model mDTM [4], by defining new functions to accommodate its multi-objective nature besides the regular built-in functions. The Central Management Unit for Avoiding Threshold (CU-AT) is the main function responsible for checking all the violations and the availability criteria for an application to be mapped on to the core. We place the CU-AT function in the main iterator function, defined in Section II. The main iterator function reduces the time T_i, by the calculated fixed amount, of every running application. The temperature of every core is also updated in each iteration. The temperature increases if an application is running on a core and decreases if no application is mapped to the core.

CU-AT uses the core temperature at every instant to see if a thermal violation will occur. If a thermal violation occurs then the application is stopped and moved to the waiting queue. Once a core becomes available and fulfills the availability criteria for running an application, then the applications from waiting queue are mapped to it.

When the waiting queue is empty, we get a new application from the application function. All the attributes for the cores are updated in the application function according to the new application mapped. This function chooses the maximum values for the v/f levels to run an application on the core in the case of mDTM.

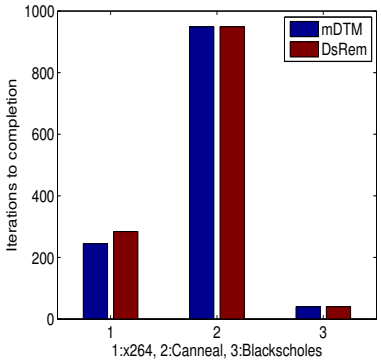
Once all the thermal violations are avoided and the waiting queue for the applications is empty, the function CU-AB is called to balance the temperatures throughout the chip via task migration. Temperatures are balanced throughout the chip until there is no running application.

B. Variant of DsRem

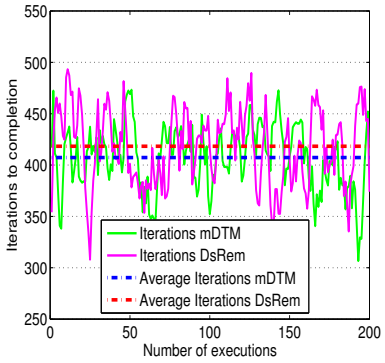
As the number of cores on the chip increases, the number of dark cores also increases to mitigate the on chip thermal emergencies. DsRem [11] focuses on solving the thermal issues while considering this dark silicon problem as well.

In DsRem [11], the Thermal Design Power (TDP) is considered as an input, and thus the applications can be mapped to the cores such that the total power of the system does not exceed TDP. For this purpose, we modify the incoming application function, described in Section II, to cater for the TDP constraint. DsRem optimally distributes TDP to all applications, so that the maximum performance is obtained. We do not remove the TDP constraint and keep it to see its impact on the results. So when mapping an application to a core, we choose such a v/f level that allows the power consumption of the application within the TDP constraint. DsRem jointly determines the number of the cores (threads) and the v/f levels of these cores for each application, using a dynamic programming algorithm. Once the applications have been mapped, we remove the TDP constraint and exploit the thermal headroom such that no thermal violation occurs.

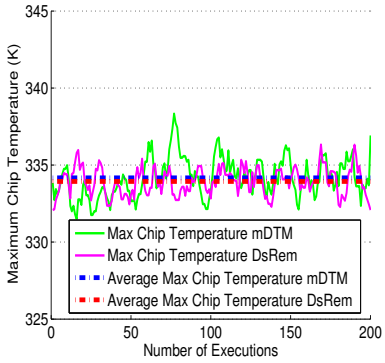
The incoming application function, described in Section II, calls the get temperature function, described in Section II, to get the steady state temperatures. In turn, the get temperature function calls the T_{crit} avoiding function to ensure that no



(a) Result for each Application

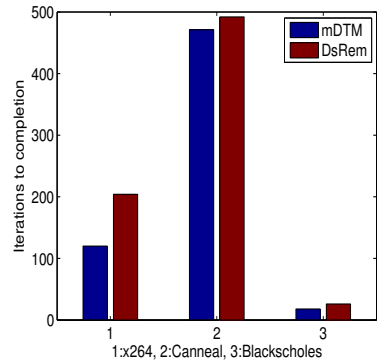


(b) Iterations to Completion

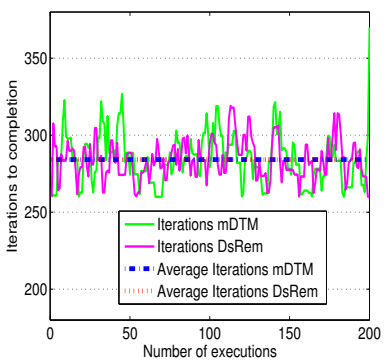


(c) Maximum Chip Temperature

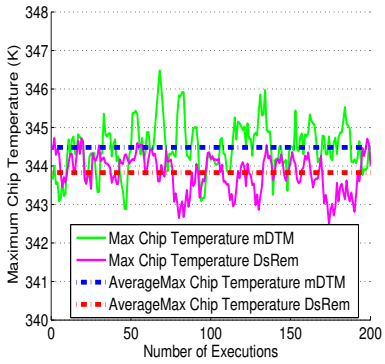
Fig. 2: Execution results for 4 x 4 grid



(a) Result for each Application



(b) Iterations to Completion



(c) Maximum Chip Temperature

Fig. 3: Execution results for 8 x 8 grid

thermal violation occurs once the application is mapped, unlike CU-AT in mDTM [4].

C. Experimental Results

To demonstrate the usefulness of the proposed analysis framework, we analyzed mDTM [4] and DsRem [11] under the same conditions for a fair comparison. The algorithms are implemented on a homogeneous 4 x 4 and 8 x 8 grid. All the cores are initially considered to be at room temperature. As the execution of applications begins, the temperature of the cores starts to change. A core with no applications running tends to reach T_{amb} eventually.

We considered T_{amb} to be 318.15 Kelvin. Similarly, for both the algorithm and grid size, we chose T_{crit} to be 343.15 Kelvin for the 4x4 grid and 347.15 Kelvin for the 8x8 grid. For DsRem, we chose the TDP to cover a wide range, and thus selected 70 W and 280 W, respectively, for the 4x4 and 8x8 grids.. DsRem considers dark cores as resources and allocates them spatially among the applications to reduce the temperature of the chip. However, in our analysis we have not utilized dark cores because of relatively small grid sizes. Nevertheless, our system is capable of accommodating the dark cores as the grid size increases. The initial mapping configuration of applications is the same for both algorithms. We took a total of 13 randomly selected applications, in each case, from a pool of 3 unique applications, i.e., x264, canneal and blackscholes, from the parsec benchmark suite [14] are used in the testing phase of these algorithms. For each application, we have a set of four threads: 1, 2, 4 and 8. The number of threads for an application are chosen depending upon the number of available cores.

Each application has 5 different v/f levels for each set of threads, each corresponding to a different power consumption. The v/f levels in case of mDTM [11] are chosen according to the available TDP while in DsRem [4], since we do not have a TDP constraint, we run the applications at the maximum possible v/f levels. In mDTM [4], we notice that it takes more time to map a new application to a core, because the core needs to fulfill the availability criteria for mapping a new application, as compared to DsRem [11], which is almost instantly ready.

We tested the same set of applications on both the algorithms and grids. We noted that for the case of x264, mDTM performed better than DsRem by almost 24% when we ran the algorithms on a 4x4 grid and a further performance improvement of 41% was recorded for the 8x8 grid. This happens in x264 because the TDP constraint in our variant of DsRem does not allow to select the maximum v/f levels

in all the cases. This eventually leads to this difference in the number of iterations. Canneal and blackscholes showed almost the same performance, as depicted in Figs. 2a and 3a. The execution times for canneal are similar because the power consumption for this application is less, so the considered DsRem variant tends to run this application at the highest possible v/f level. In the case of blackscholes, with an increase in the v/f levels and the number of threads, we do not notice a very significant decrease in the number of iterations as is the case with the other two applications.

In the second case, we vary the order and frequency of an application randomly for each run, while the number of applications remain the same. We run these applications and note the number of iterations for the considered algorithm to execute all those applications. We also note the maximum chip temperature in every iteration.

We executed both, the mDTM and the DsRem, programs 200 times each and recorded the iterations, as shown in Figs. 2b and 3b. We found that both the algorithms exhibit a very similar performance. Similarly, in every execution, we noted the maximum chip temperature, as shown in Figs. 2c and 3c. We notice that our DsRem variant tends to keep the temperatures lower than mDTM.

IV. Conclusion

The thermal and resource management in multi-core architecture becomes quite challenging as the size of the grid increases. Various algorithms have been proposed to cater for these challenges, but, to the best of our knowledge, there is no common comparative platform where a fair comparison between their performance can be made. In order to overcome this limitation, we have developed a comparative framework, using C++, which provides the flexibility to implement and compare different DRM and DTM algorithms. We also demonstrated the efficiency of our model by implementing and analyzing two algorithms, i.e., mDTM [4] and a variant of DsRem [11], and observing their results. In the future, we plan to analyze further DRM and DTM algorithms using the proposed framework.

References

[1] M. Fattah, M. Daneshtalab, P. Liljeberg, J.Plosila, "Smart Hill Climbing for Agile Dynamic Mapping in Many-Core Systems", *DAC*, 39, 2013.

[2] Y. G. Kim, M. Kim, J. M. Kim, S. W. Chung, "M-DTM: migration-based dynamic thermal management for heterogeneous mobile multi-core processors", pp. 1533-1538 *DATE*, 2015.

[3] R. W. Keyes, "Physical limits of silicon transistors and circuits", vol. 68, no. 12, *Progress in Physics*, 2005.

[4] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, J. Henkel, "mDTM: Multi-Objective Dynamic Thermal Management for On-Chip Systems", pp.1-6, *DATE*, 2014.

[5] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multicore simulations", *High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1-12, 2011.

[6] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, "Multi2Sim: A simulation framework to evaluate multicore-multithreaded processors", *Computer Architecture and High Performance Computing*, pp. 62–68, 2007.

[7] V. Hanumaiah, S. Vrudhula, "Temperature-Aware DVFS for Hard Real-Time Applications on Multicore Processors", vol. 61, issue: 10, *IEEE Transactions on Computers*, 2012

[8] M. Mandelli, G. Castilhos, G. Sassatelli, L. Ost, F. G. Moraes, "A Distributed Energy-aware Task Mapping to Achieve Thermal Balancing and Improve Reliability of Many-core Systems", *Integrated Circuits and Systems Design*, pp. 1-7, 2015

[9] C. A. Floudas, P. M. Pardalos, "Encyclopedia of Optimization", pp. 1711-1715, *Springer*, 2009

[10] S. B. Aruoba, J. Fernández-Villaverde, "A Comparison of Programming Languages in Economics", paper no. 20263, *NBER*, 2014

[11] H. Khdr, S. Pagani, M. Shafique, J. Henkel, "Thermal Constrained Resource Management for Mixed ILP-TLP Workloads in Dark Silicon Chips", . 179 *DAC*, 2015.

[12] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, M. Stan, "HotSpot: a compact thermal modeling methodology for early-stage VLSI design", vol. 14, issue: 5, pp.501–513, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2006.

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

[14] C. Bienia, S. Kumar, J. P. Singh, K. Li, "The PARSEC benchmark suite: Characterization and architectural implications", pp. 72–81, *Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[15] M. S. Saeed, M. Akmal, H. Shafi, http://save.seecs.nust.edu.pk/projects/csfatrma/crftrma.html, 2020.