Prediction-Based Task Migration on S-NUCA Many-Cores

Martin Rapp*, Anuj Pathania†, Tulika Mitra†, Jörg Henkel*
*Karlsruhe Institute of Technology, Germany, †National University of Singapore, Singapore
martin.rapp@kit.edu, pathania@comp.nus.edu.sg, tulika@comp.nus.edu.sg, henkel@kit.edu

***Abstract*—Performance of a task running on a many-core with distributed shared Last-Level Cache (LLC) strongly depends on two factors: the power budget needed to guarantee thermally safe operation and the LLC latency. The task's thread-to-core mapping determines both the factors. Arrival and departure of tasks on a many-core deployed in an open system can change its state significantly in terms of available cores and power budget. Task migrations can thereupon be used as a tool to keep the many-core operating at the peak performance. Furthermore, the relative impacts of power budget and LLC latency on a task's performance can change with its different execution phases mandating its migration on-the-fly.**

We propose the first run-time algorithm *PCMig* that increases the performance of a many-core with distributed shared LLC by migrating tasks based on their phases and the many-core's state. *PCMig* is based on a performance-prediction model that predicts the performance impact of migrations. *PCMig* results in up to 16 % reduction in the average response time compared to the state-of-the-art.

***Index Terms*—Cache Memory, Processor Scheduling, Power Dissipation, Thermal Stability**

I. Introduction

Increasing power densities ended the race to higher and higher processor frequencies. Instead, the number of cores in processors kept on rising to satisfy the never-ending need for more computational power. Many-core processors with dozens to hundreds of cores housed on a single die emerged as promising parallel processing platforms [1]. With such large numbers of cores, the one-thread-per-core model is commonly used [2]. The task-to-core mappings are the key to derive maximum performance out of many-cores.

The task arrivals and departures, as well as the tasks themselves, are not known in advance in open systems [3]. Furthermore, the mappings need to be constantly readjusted at run-time by task migrations to cope with changing state of the many-core and task requirements. Mapping multiple tasks to multiple cores optimally is an NP-hard problem in the general case [4]. *Therefore, we are required to develop fast heuristics for run-time task migrations in many-cores.*

Power Budgets: Increasing power densities cause elevated on-chip temperatures, which above a critical temperature can cause permanent damage to the chip and thereby necessitate run-time thermal management. One way to ensure thermal safety on many-cores is through the use of power budgets that constrain the power consumption of cores. We use in this work a mapping-determined non-uniform power budget called Thermal Safe Power (TSP) [5], which maximizes the sum of all per-core power budgets under a thermal constraint.

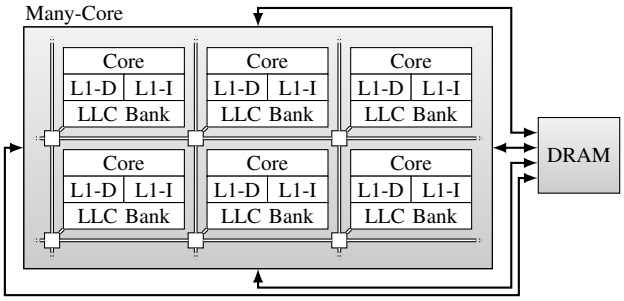


Fig. 1: An abstract block digram of a many-core architecture with physically-distributed, yet logically-shared LLC.

Many-Core Architecture with S-NUCA: Figure 1 presents an overview of a many-core architecture with a distributed shared LLC. The many-core consists of tiles, each containing a core with its associated caches. The L1 caches are private to the core on the tile. The shared L2 cache (LLC) is physically distributed among all tiles with each tile holding one LLC bank. All the tiles are connected by a Network-on-Chip (NoC) with a router in every tile. Memory controllers on the periphery provide DRAM access. Static Non-Uniform Cache Access (S-NUCA) is a scalable policy for managing physically distributed, yet logically shared LLCs. It determines a static mapping of memory address to LLC bank at design-time [6].

The latency of an LLC access under S-NUCA depends upon the hop count on the NoC between the tile where the thread is running and the tile where the LLC bank containing the memory address is located. The hop count is measured by the Manhattan Distance (MD). The average LLC latency experienced by a thread executing at a core depends upon its Average Manhattan Distance (AMD) to all tiles of a the many-core. The closer a core is to the center of the many-core, the lower is its AMD and thereby the lower is the average LLC latency for threads running on it. Therefore, the S-NUCA policy imposes an inherent heterogeneity on the otherwise perfectly homogeneous many-core. In order to minimize their average LLC latency, tasks should be mapped to cores as close as possible to the center of the many-core [7].

However, mapping all the tasks to the center of the many-core creates a thermal hotspot, which reduces their power budgets and therefore degrades their performance. The tasks need to be mapped as far away as possible from each other to maximize their power budgets. However, this causes cores near the corners of many-core to be used excessively, which are

Due to space constraints, not all of the originally submitted content is included in this IP paper.

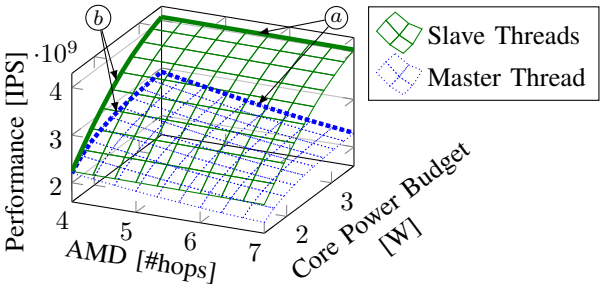


Fig. 2: Performance of both *blackscholes* master and slave threads strongly depends on the power budget and AMD. However, the master thread is more sensitive to the AMD, while the slave threads are more sensitive to the power budget.

also the cores with a high AMD and therefore a high average LLC latency. *Therefore, a trade-off needs to be made between minimizing the LLC latency and maximizing the power budget for a task in order to maximize its performance [8].* In our previous work [8], we proposed a task mapping algorithm *PCMap* which exploits this trade-off. However, this algorithm does not employ task migration and hence is not able to react to changing workload and thread characteristics.

The relative impacts of power budget and LLC latency in this trade-off depend upon the thread characteristics. Figure 2 shows the performance (measured in Instructions per Second (IPS)) for *PARSEC* [9] *blackscholes* master and slave threads depending on the power budget and the AMD of the core on which they are executing. The master threads prepares data, which is processed by the slave threads. Due to more frequent LLC accesses, the master thread is more sensitive to the AMD. With a high power budget, the performance of the master thread drops significantly with increasing AMD whereas slave threads' performance remains nearly constant for all AMD values (Lines a in Figure 2). Contrarily, the slave threads, which mainly perform computations, are more sensitive to the power budget. For a low AMD, the performance of the master thread saturates early with increasing power budget whereas the performance of the slave threads saturates at a higher power budget (Lines b in Figure 2). *Thread characteristics must be considered to determine the performance-maximizing trade-off between power budget and LLC latency.*

Arriving or departing tasks in an open system can cause sudden changes in the workload that require adjustments to the current mappings in order to maximize the many-core's performance. Furthermore, the characteristics of a certain task may change over time as the task proceeds through various phases in its execution. This may change the trade-off between power budget and LLC latency at run-time for the task and can only be satisfactorily addressed by changing its mapping. *Therefore, performance of a many-core in an open system can only be maximized by dynamically adapting the mappings of its executing tasks using task migrations.*

Authors of [10] propose to perform *defragmentation* of the many-core. Thereby, already running tasks are migrated in order to let the idle cores form a contiguous shape. This allows new arriving tasks to be mapped in a contiguous shape. *Defrag*

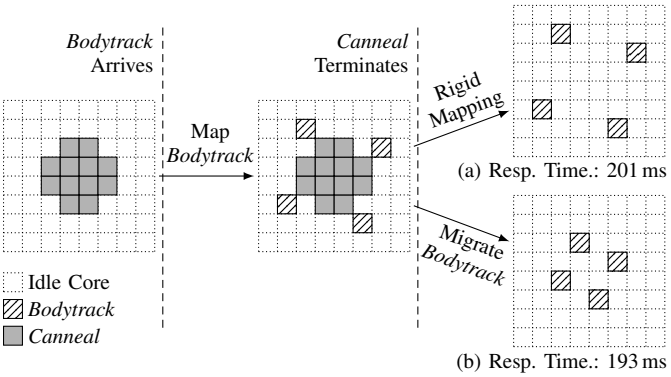


Fig. 3: *Canneal* finishes during the execution of *bodytrack*. Migrating *bodytrack* to better suited cores (b) decreases its response time by 3.8 % compared to the rigid mapping (a).

assumes message passing between threads of a task. On such a system, a contiguous mapping reduces the communication latency and thus increases the performance. However, this is not the case in S-NUCA many-cores. Furthermore, they do not consider thermal effects.

No approach has been proposed that employs task migration to increase the performance on S-NUCA many-cores.

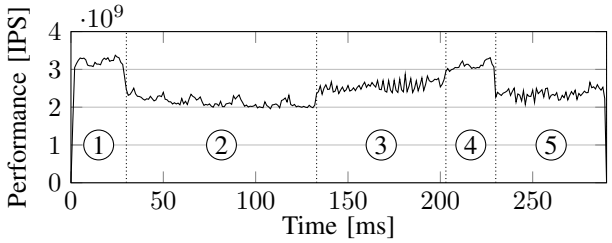
Contributions: We are first to explore the performance potential of thread migrations on a many-core with distributed shared LLC and within this context we propose a lightweight run-time task migration algorithm that uses a performance-prediction model to improve the many-core's performance.

We highlight the importance of our contributions using two examples in the following section.

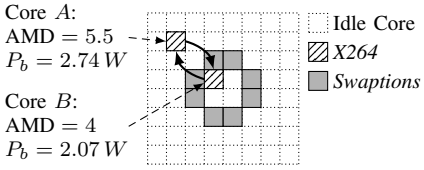
II. Motivational Examples

Figure 3 shows an example where task migration allows a many-core to react to workload changes. *PARSEC canneal*, which is very sensitive to the LLC latency, is mapped to the cores close to the center. During its execution, an instance of *PARSEC bodytrack* arrives and needs to be mapped. *Canneal* then finishes execution while *bodytrack* is still continuing execution, which results in two changes on the many-core. First, the cores in the center are now idle, which changes the power budgets. Second, these cores are now free to be used by another task. We can now improve the performance of *bodytrack* by migrating its threads closer to center as shown in Figure 3b. This migration results in 3.8 % improvement in performance compared to rigid mapping shown in Figure 3a.

Figure 4 shows an example of exploiting changing thread characteristics at run-time using task migration. Single-threaded *PARSEC x264* shows five distinct execution phases, which are distinguishable in the IPS trace shown in Figure 4a. Phases 1 and 4 have higher IPS due to relatively fewer LLC accesses. Contrarily, during Phases 2, 3 and 5, higher LLC accesses cause the core to wait and therefore reduces the IPS. Figure 4b shows the many-core state during the execution of *x264*. An 8-threaded instance of *PARSEC swaptions* is running in parallel to *x264* and is mapped to cores near the center of the many-core. Two candidate cores were examined for the map-



(a) Execution phases of *x264* (if running on Core A)



(b) Mappings of the tasks

Phase	1	2	3	4	5
Core A: IPS [$\cdot 10^9$]	**3.22** (+9.6 %)	2.13	2.54	**3.08** (+6.1 %)	2.36
Core B: IPS [$\cdot 10^9$]	2.94	**2.43** (+13.7 %)	**2.64** (+4.1 %)	2.90	**2.53** (+7.0 %)

(c) IPS of the phases under different mappings

Fig. 4: *X264* shows different phases during its execution, which results in changing impacts of power budget and LLC latency on its performance. Migrating *x264* in each phase to the core best suited for that phase increases the per-phase performance by up to 14 %.

ping of *x264*. Core A is farther away from other active cores and therefore has a higher power budget. Core B has a lower AMD and therefore a lower average LLC latency. Figure 4c shows the performance of the different phases when mapped to Core A or B. During Phases 1 and 4, *x264* performs better on Core A, where it benefits from the higher power budget. Contrarily, a low AMD is more beneficial during phases 2, 3 and 5, which results in higher performance on Core B. Most importantly, the optimal mapping for *x264* changes during its execution. A rigid task mapping algorithm cannot address this exhibited phase behavior, but task migrations are needed to maximize the performance.

III. IPS Prediction Model

A key necessity of our task migration approach is a model that predicts the IPS of a thread if migrated to another core. This model needs features for both the target core and the thread characteristics to distinguish different thread behaviors. Cores of a many-core with S-NUCA differ in their AMDs, as well as in their mapping-dependent power power budgets P_B. Hence, we use these two values to describe the target core.

Cycle stacks [11] divide the cycles needed to execute a set of instructions into the base cycle count and a set of components that represent cycles losses, such as cache misses. The thread characteristics are represented by the thread's current IPS on its current core, as well as the LLC-related component $\tilde{c}_{LLC}$ of the normalized cycle stack, which reflects the relative performance loss due to LLC accesses. The IPS and $\tilde{c}_{LLC}$ depend on the core on which the thread is currently running. Hence, we add the AMD and power budget P_B of this core to the set of our features. Details on the internals of this model have been omitted due to space constraints.

IV. Online Task Migration Algorithm

Our run-time task migration algorithm *PCMig* is based on the IPS prediction model. Task migration periodically traverses the following steps:

First, migration candidates are created. There are $O(n^{|T|})$ possible migrations in a many-core with n cores and set of threads T. The number of migrations is too large to be explored in its entirety. We therefore limit our algorithm to only migrate one thread at a time or swapping two threads. This reduces the number of possible migrations to $O(n{\cdot}|T|)$.

All these migration candidates need to be rated in order to select the most promising one. Let M_0 be the mapping before migration. A migration candidate m is defined by the changed mapping M_m. The IPS prediction model is used to rate each migration candidate. The AMDs and power budgets before and after migration can be determined from the mappings M_0 and M_m. The IPS and $\tilde{c}_{LLC}$ before migration are obtained from performance counters. Migration of a thread does not only affect this thread, but also all other threads because their power budgets may change as well. Therefore, the model is used to predict the IPS of all threads before and after migration. In order to be able to select one migration, we calculate the average relative IPS improvement $\Delta_{IPS}(m)$ among all threads for all migration candidates m.

$$\Delta_{IPS}(m) = \sum_{t \in T} \left(\frac{\mathrm{IPS}(t, M_m)}{\mathrm{IPS}(t, M_0)} - 1 \right)$$

Note that no actual migration is performed to calculate this metric, but the IPS prediction model $\mathrm{IPS}(t, M)$ *described in Section III is used instead.* Finally, the migration with the highest IPS improvement $\Delta_{IPS}(m)$ is selected. However, this migration is executed only if $\Delta_{IPS}(m) > \delta$. The threshold δ prevents migrations with only very little expected speedup, where the performance penalty of the migration itself is likely to outweigh the expected benefit. Also, it prevents oscillations due to small prediction inaccuracies. We set δ to 3 %.

System Integration: Thread characteristics are not available during the initial mapping of a task. Therefore, our IPS model cannot be used. Instead, we use *PCMap* [8], which is a task-agnostic mapping algorithm to decide the initial mapping.

Migrating a thread causes a performance penalty due to cold caches. It takes some time until the caches are warm and the thread reaches its peak performance, which strongly depends on the size of the L1 caches and the LLC latency. On our infrastructure, it takes less than 0.2 ms for the IPS to saturate after a migration. The migration interval is set to 10 ms, which is small enough to react fast to changes, but large enough to maintain a reasonable overhead.

V. Evaluation

Experimental Setup: We perform the evaluation using *HotSniper* [12], which offers multi-threaded multi-program simulation of many-cores with full modeling of shared resource contentions. *HotSniper* combines the *Sniper* many-core

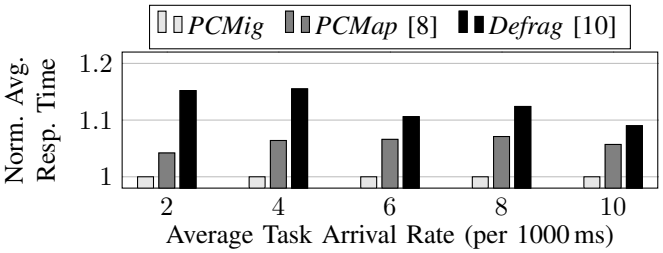


Fig. 5: Performance with different management algorithms.

simulator [13] with McPAT [14] for power estimation and HotSpot [15] for thermal simulation. We simulate a 8×8-core many-core. The L1-I and L1-D caches have a capacity of 16 KiB, each. The LLC has a total capacity of 8 MiB and consists of 64 LLC-banks with 128 KiB each. The NoC latency is 1.5 ns per hop (6 CPU cycles @ 4.0 GHz). The NoC link width is 256 Bit. The many-core is modeled to be fabricated in the 14 nm technology node. The area of each core is 0.81 mm^2. Thermal Design Power (TDP) is set at 100 W. Idle cores consume 0.3 W due to the active LLC cache bank. Ambient and maximum temperature are 45°C and 70°C, respectively. Dynamic Voltage and Frequency Scaling (DVFS) sets core frequencies from 1.0 GHz to 4.0 GHz in steps of 100 MHz.

State-of-the-Art Comparison: We compare our proposed algorithm *PCMig* to the state-of-the-art algorithms *Defrag* [10] and *PCMap* [8]. *Defrag* is the closest approach that uses task migration, while *PCMap* is the closest approach on S-NUCA many-cores. We adapted them to work on our infrastructure.

We use the *PARSEC* tasks *blackscholes*, *bodytrack*, *canneal*, *dedup*, *fluidanimate*, *streamcluster*, *swaptions*, and *x264* with *simsmall* inputs for the evaluation. The tasks *facesim* and *raytrace* were skipped since they do not offer a *simsmall* input and hence take unreasonably long to execute. The tasks *ferret*, *freqmine*, and *vips* were skipped due to unresolvable instrumentation errors. Two workloads consisting of 20 randomly selected tasks with randomly selected numbers of threads are executed with varying arrival rates. Thereby, the average (peak) utilization is varied from 10 % (58 %) to 38 % (100 %).

Figure 5 presents the average response time with the three management algorithms. For all task arrival rates, the proposed *PCMig* results in the lowest average response time with up to 7 % and 16 % improvement over *PCMap* and *Defrag*, respectively. *PCMap* uses task-agnostic rigid mappings and therefore does not consider task characteristics and cannot react to changing workloads or thread characteristics. *Defrag* assumes message passing between threads and does not consider temperatures. In order to obtain a contiguous shape of the idle cores, active tasks are migrated to cores near the border of the system, which have a igh average LLC latency with S-NUCA. Furthermore, contiguous mappings cause thermal hotspots and hence reduces the power budget. *PCMig* avoids both these drawbacks and thus achieves a higher performance.

VI. Conclusion

To maximize the performance on a many-core deployed in an open system, task migration is required to react to changing workloads and thread characteristics. Performance of a thread strongly depends on the power budget and the AMDs of its core. However, both factors affect the performance of different threads differently. We proposed a run-time task migration algorithm *PCMig* that uses a performance-prediction model to rate potential migration candidates prior to actual migration. Our algorithm results in up to 16 % improvement in the response time compared to the state-of-the-art.

Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89), by the National Research Foundation, Prime Ministers Office, Singapore under its Industry-IHL Partnership Grant and by Huawei International Pte. Ltd. NRF2015-IIP003.

References

[1] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari *et al.*, "Invasive Manycore Architectures," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012.

[2] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, M. F. Kaashoek, R. Morris *et al.*, "Corey: An Operating System for Many Cores," in *Symp. Operating System Design and Implementation (OSDI)*, 2008.

[3] D. G. Feitelson and L. Rudolph, "Metrics and Benchmarking for Parallel Job Scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1998.

[4] M. R. Garey and D. S. Johnson, "Complexity Results for Multiprocessor Scheduling under Resource Constraints," *SIAM Journal on Computing*, 1975.

[5] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal Safe Power (TSP): Efficient Power Budgeting for Heterogeneous Manycore Systems in Dark Silicon," *IEEE Transactions on Computers (TC)*, 2017.

[6] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[7] A. Pathania and J. Henkel, "Task Scheduling for Many-Cores with S-NUCA Caches," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[8] M. Rapp, A. Pathania, and J. Henkel, "Pareto-Optimal Power- and Cache-Aware Task Mapping for Many-Cores with Distributed Shared Last-Level Cache," in *Int. Symp on Low Power Electronics and Design (ISLPED)*, 2018.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[10] J. Ng, X. Wang, A. K. Singh, and T. Mak, "Defragmentation for Efficient Runtime Resource Management in NoC-Based Many-Core Systems," *Transactions on Very Large Scale Integration (VLSI) Systems*, 2016.

[11] W. Heirman, T. E. Carlson, S. Che, K. Skadron, and L. Eeckhout, "Using Cycle Stacks to Understand Scaling Bottlenecks in Multi-Threaded Workloads," in *Int. Symp. on Workload Characterization (IISWC)*, 2011.

[12] A. Pathania and J. Henkel, "HotSniper: Sniper-Based Toolchain for Many-Core Thermal Simulations in Open Systems," *IEEE Embedded Systems Letters (ESL)*, 2018.

[13] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[14] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing," *Transactions on Architecture and Code Optimization (TACO)*, 2013.

[15] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *Transactions on Very Large Scale Integration (VLSI) Systems*, 2006.