Dynamic Thread Mapping for Maximizing Performance in Power-Efficient Multi-core Systems

Veronia Iskandar*, Cherif Salama†,*, Mohamed Taher*
*Computer and Systems Engineering Department, Ain Shams University, Cairo, Egypt
†Computer Science and Engineering Department, The American University in Cairo, Egypt
Emails: veronia_bahaa@eng.asu.edu.eg, cherif.salama@aucegypt.edu,
mohamed.taher@eng.asu.edu.eg

***Abstract*—Efficiently mapping threads to system cores is critical for achieving high performance and power efficiency in multicore systems. Conventional scheduling techniques do not take system heterogeneity and varying DVFS states into account and therefore do not produce efficient mappings. Additionally, since thread behavior varies throughout its execution, a fixed thread-to-core mapping is suboptimal. Dynamically scheduling threads to appropriate core states can significantly improve system throughput and decrease energy consumption. Scheduling decisions are of utmost importance in systems that aim to maximize performance. The problem of thread assignment in order to meet performance or power constraints has been shown to be NP-complete and therefore exceedingly expensive to solve online. This paper proposes a heuristic for dynamically assigning threads to system cores such that performance is maximized while minimizing the energy consumed. The heuristic can be applied to heterogeneous systems and homogeneous systems with DVFS capabilities. Simulation results show that the heuristic can be used online in hundred-core systems, with runtime overhead of less than 1 millisecond in worst cases for 256-core systems.**

I. Introduction

The number of cores integrated on a single chip continues to grow with technology scaling. The cores integrated on a chip can either be homogeneous or heterogeneous. Systems with homogeneous cores usually include a number of identical complex cores, such as out-of-order and superscalar cores. These cores provide high serial performance for single-threaded applications. However, due to their sophisticated microarchitecture, they are highly power-inefficient. To decrease energy consumption, simple in-order cores can be used instead of the complex ones. Systems with simple homogeneous cores provide high parallel performance and power efficiency, but they are not useful except for applications with thread-level parallelism. Alternatively, a mix of simple and complex cores can be integrated on a single chip, thus producing a heterogeneous chip-multiprocessor. This heterogeneous system provides design flexibility and better power management, making use of the benefits of both simple and complex cores. Cores can differ in frequency, voltage, cache size, ROB size or other architectural features. Complex cores running at higher frequencies offer higher performance than simple cores but also consume more power. Existing commercial heterogeneous multicore systems include NVidia's Kal-El [1] and ARM's big.LITTLE [2] system. An important issue to note is that heterogeneous systems introduce a challenge to the conventional scheduling approaches used by most operating systems since conventional schedulers assume core homogeneity; thus not taking into account the difference in performance and power across different core types. Moreover, recent studies have shown that performance characteristics vary within a thread's execution [3], [4]. This suggests the need for a dynamic scheduler to dynamically migrate threads to suitable cores and thus exploit the advantages of the heterogeneous system.

This paper considers both heterogeneous chips and homogeneous chips with DVFS enabled that are used in systems that have constraints to achieve their maximum performance. In such systems, a performance requirement must be satisfied while keeping the system power-efficient. A case for this is embedded systems that are connected through sensors and actuators to an external environment. Interaction with this environment must happen at a certain pace and tasks must be executed within specified time limits. Examples are automatic control tasks, manufacturing systems, automotive/air/space applications, and signal processing tasks in general.

A traditional approach to address performance or power constraints is Dynamic Voltage and Frequency Scaling (DVFS) [5]. DVFS methods provide efficient power management since they allow controlling the amount of power provided to the core such that it is just enough for the running task to finish before its deadline. DVFS techniques are usually used for systems with homogeneous cores. Another approach that is suitable for heterogeneous systems is dynamic scheduling of threads. A thread scheduler generates a thread-to-core assignment that ideally maps every thread to the most suitable core to run on, considering the thread's performance characteristics. Such scheduling aims to maximize overall performance or satisfy a power budget.

Prior work on dynamic thread scheduling for heterogeneous systems focused on power-constrained environments where a power budget must be met [6], [7]. Scheduling decisions are taken such that the power consumed does not exceed the budget. Some studies have been conducted on scheduling threads to maximize the overall performance of the chip [8], [9], [10], [11]. Much work on dynamic scheduling relies on sampling CPI of workloads on every core type to determine the performance and power consumption and make scheduling decisions accordingly [12], [13] . While this approach works well with processors with small core count (4 – 8 cores), it does not scale well with increasing numbers of cores. With modern chips integrating hundreds of cores on them, this approach is highly inefficient. Other studies avoid the overhead of sampling by performing offline profiling of application threads to determine their characteristics beforehand [14], [15], [16].

This introduces an overhead for gathering offline performance and power measurements data. In this paper, we propose an approach to dynamic thread mapping that does not require sampling or offline profiling of the workloads to be run. Our approach is novel in that it considers satisfying maximum performance constraints with minimal energy consumption for multi-threaded and multi-program workloads on both heterogeneous and homogeneous platforms. We present an efficient scheduling algorithm that is executed online at specific time intervals to generate a mapping that maximizes performance while minimizing the energy consumed. This paper makes the following contributions:

1) We propose a thread scheduling framework for mapping threads to cores during runtime, such that performance is maximized while minimizing the power consumed.
2) We apply our heuristic to two different types of systems: heterogeneous systems that pack different core types on a single chip and homogeneous systems with identical cores and DVFS enabled. For homogeneous systems, we describe 2 DVFS schemes: using fixed voltage/frequency levels and using per-core DVFS.
3) We validate our findings using Sniper [17] architectural simulator. Simulation results show significant performance improvement and power saving as will be presented in details in section 6.

The rest of the paper is organized as follows: Section 2 summarizes related work; Section 3 describes the proposed thread mapping framework; Section 4 discusses the implementation details of our algorithm; Section 5 presents the experimental setup for our simulations. Section 6 shows results and an evaluation of the proposed method; Section 7 concludes this paper.

II. Related Work

Several studies have proposed static scheduling algorithms for heterogeneous systems to satisfy constraints or increase efficiency [14], [15], [16]. Most static schedulers rely on offline profiling of threads to decide the most suitable core for every thread. This introduces an overhead for the extra profiling run. While static approaches are mostly scalable and simple to implement, their limitation is that they pin threads to cores and therefore do not consider time-varying application behavior. Dynamic Scheduling addresses this limitation by taking scheduling decisions during runtime. Annamalai et al. propose a framework for determining thread-to-core assignment and operating frequencies of cores by predicting stable phases of applications using hardware performance counters [18]. A system with only two core types is used whereas our proposed system model is more general for any number of core types. Bias scheduling by Koufaty et al. aims at increasing performance by scheduling threads with low memory stalls to complex cores but does not consider power efficiency[8]. Van Craeynest et al. propose a model for performance estimation on a given core using CPI, ILP and MLP information [10]. It is worth noting that our dynamic thread assignment model can be built on top of any prediction model, such as the one proposed here. Birhanu et al. present a mechanism for modeling heterogeneity and scheduling threads to maximize performance [11]. Some studies are based on formulating the scheduling problem as an integer linear programming problem then solving it using fast heuristic approaches. Liu et al. propose MTS [6], a heuristic to maximize performance under a given power budget and further extend it by adding a DVFS framework to MTS scheduling [7].

Other approaches are based on DVFS for power management. Rangan et al. present a fine-grained power management method that moves threads across homogeneous cores with two VF-domains depending on application behavior [19]. Imamura et al. propose dynamic core and frequency scaling, an approach that runs the application with different numbers of threads and different core frequencies to determine the best configuration that meets power constraints [20]. Winter et al. propose DVFS and scheduling methods for dealing with process variations and systems that are randomly heterogeneous due to manufacturing flaws and gradual degradation of the chip components [21].

III. The Thread Mapping Framework

A. Problem Definition and System Model

The goal of this work is to find a mapping for n threads running on a many-core system with heterogeneous cores or homogeneous cores with DVFS enabled. The mapping aims to satisfy a given performance requirement T, which is the maximum achievable performance, while minimizing the consumed power. T is specified as a throughput value measured in ips, instructions per second. The mapping has to computed periodically at runtime to be aware of thread phase changes and variations. As such, we have a scheduling problem that is NP-complete. An ILP solver could be used to generate an optimal solution; but as the number of cores and number of threads increase, the time required by the ILP solver will also increase and it will not be feasible to use online. We propose a heuristic method that iteratively searches for a mapping that maximizes performance.

We define a core state as a core type with given microarchitectural characteristics for heterogeneous systems. For a homogeneous system, a core state is a specific voltage/frequency setting. Hence, we have n threads to be mapped to m core states. We also define the assignment matrix A as an $n * m$ matrix where $A[i][j]$ = 1 if thread i is assigned to core state j. Else, $A[i][j]$ = 0. The objective of our algorithm is to generate a mapping A where each active thread t is assigned to one core state j such that T is satisfied.

B. Proposed Solution

Our proposed method is an iterative algorithm that consists of three phases: Initialization, Move Generation and Exchange. Figure 1 shows a pseudo-code for the algorithm. In the Initialization phase, an initial assignment for the threads is chosen such that the consumed power is minimized. This is done by assigning the threads from the smallest core states to the bigger ones in order of increasing power consumption (i.e., threads with the highest power consumptions are assigned to smallest core states). This initial mapping is considered the current solution. In the Move Generation phase, we consider every core state at a time starting with the smallest. We determine all possible thread moves for such a core state. A possible move is defined as an exchange of a thread $t1$ running on a given core state with a thread $t2$ running on the next

Input:
core_states : set of available core states
threads : set of running threads
Output:
A : $n*m$ assignment matrix.
Initialization Phase
sort core_states in ascending order
for each core_state $c \epsilon$ core_states **do**
  $Power_c \leftarrow \phi$
  for each thread $t \epsilon$ threads **do**
    if t is not assigned to a core **then**
      calculate $P_{t,c}$
      $Power_c \leftarrow Power_c \cup P_{t,c}$
    end if
  end for
  construct max-heap from $Power_c$
  assign top n/m heap elements to c
end for
Move Generation Phase
for each core_state $c \epsilon$ core_states **do**
  $exchanges_c \leftarrow \phi$
  for each thread $t1$ running on c **do**
    for each thread $t2$ running on c+1 **do**
      if Δthroughput > 0 **then**
        priority($t1$,$t2$) $\leftarrow$ Δthroughput
        $exchanges_c \leftarrow exchanges_c \cup exchange(t1, t2)$
      end if
    end for
  end for
  construct max-heap from $exchanges_c$
Exchange Phase
  while $exchanges_c$ is not empty **do**
    $bestSwap \leftarrow$ popheap($exchanges_c$)
    perform $bestSwap$ in A
  end while
end for
return A

Fig. 1. Algorithm for the proposed heuristic solution

bigger core state such that this exchange results in an increase in the overall throughput. For every possible move, a priority value is calculated as the throughput gain achieved by such an exchange. Then we start the Exchange phase. The highest-priority move between this core state and the next is performed. Then we perform the next moves in order of priority until all moves for this core state are performed. Afterward, we perform Move Generation and Exchange phases for the next bigger core states in order, in a similar manner. The iterations continue until there are no more possible moves. All moves during the algorithm are not actual thread migrations or voltage/frequency switchings. Thread migration and VF switching are performed only after reaching the final solution. After every move is performed, all sets of possible moves are updated to reflect this change. For example, after exchanging thread $t1$ from core state 1 with thread $t2$ from core state 2, the sets of possible moves need to be updated since all exchanges between $t1$ and any thread running on core state 2 would not be possible now.

C. Heterogeneous Cores

We consider a system with m core types that are different in architectural features. The scheduling relies on thread motion across cores. The heuristic calculates the best thread-to-core mapping then threads are migrated to their appropriate cores.

D. Homogeneous Cores

Homogeneous systems pack together a number of similar cores. A traditional approach to DVFS is to combine the needs of all cores into a single voltage/frequency (VF) setting. The limitation of this method is that different applications have different performance and power characteristics that require different VF settings. As an alternative, we use our heuristic with two schemes for DVFS: fixed VF levels and per-core DVFS. Section 6 shows a comparison between both DVFS methods.

1) Fixed VF levels: As traditional DVFS methods fail to obtain an optimal VF setting for every core while per-core DVFS may be expensive [22], we utilize fixed VF as a compromise between these two methods. Some recent studies have suggested that this method performs nearly as well as per-core DVFS [19], [23]. In this method, we assign fixed VF levels for every group of cores. As such, we have a system of homogeneous cores but with heterogeneous VF settings. Threads move across cores based on the mapping produced from our heuristic such that every thread runs on the core with the most suitable VF setting for it. Providing more VF domains requires more added hardware and therefore incurs more cost. Hence, the aim is to satisfy maximize performance with as few VF domains as possible. As detailed in Section 6, we test our method with a 2-VF and a 4-VF configuration.

2) Per-core DVFS: With the emergence of many-core chips, per-core DVFS depending on per-core voltage regulators has been suggested to overcome the limitations of traditional DVFS. Each core has its independent VF setting according to the needs of the thread running on it. No thread migration is needed in this method. Our mapping heuristic produces the required VF setting for each core and voltage is switched accordingly. However, this method introduces more costs of the added hardware. In addition to the hardware cost, this method assumes 1 thread per core. Having more than one thread with different VF requirements running on the same core would result in inaccurate VF mappings.

IV. Implementation Details

A. Prediction of Throughput and Power

Prediction of power and throughput values is done based on the nearest neighbour classification method. This needs an initial offline profiling step where we run several PARSEC [24] benchmarks on each core state to get throughput values measured by the number of instructions executed per second (IPS) and power values. We choose a diverse set of benchmarks with throughput and power characteristics ranging from very low to very high values. For every run, we use 4 threads per benchmark and 4 cores of the same state. We record $power$ and ips during the parallel region of the benchmark then we compute the average values per run. These values are used to predict the throughput and power of any thread t on all core

TABLE I. CORE STATES FOR HETEROGENEOUS SYSTEM

Core Type	Big	Mid2	Mid1	Small
Voltage	1V	0.86V	0.71V	0.56V
Frequency	3.5GHz	3GHz	2.5GHz	2GHz
ROB window size	128	64	32	16
Issue width	8	4	2	1
L2 cache	512KB	256KB	256KB	128KB
LI cache: 32KB , L1 cache: 32KB				

TABLE II. CORE STATES FOR HOMOGENEOUS SYSTEM

Core State	High	Mid2	Mid1	Low
Voltage	1V	0.86V	0.71V	0.56V
Frequency	3.5GHz	3GHz	2.5GHz	2GHz
LI cache: 32KB, L1 cache: 32KB, L2 cache: 256 KB ROB window size: 32, Issue width: 2				

states. The throughput of t is measured on the core state c that it is running on. Then the benchmark N whose throughput on core state c is nearest to that of t is determined. The throughput of t on the rest of the core states is predicted to be equal to that of N on them. Power values are computed in a similar way. Since prediction modeling is not the focus of this work, we have chosen to use this simple binning method. More accurate methods could be integrated as well with the scheduling algorithm of our heuristic solution.

B. Control Intervals

Our proposed algorithm aims to determine the best thread-to-core mapping that maximizes performance while keeping power consumption as low as possible. This mapping is not static; it changes as threads behavior changes over time. Thus, the heuristic is called periodically every time interval, called $control\ interval$, and threads are dynamically migrated, or VF values are set accordingly. We choose $1\ ms$ as our control interval. The granularity of the interval can be adapted per run. However, it is reasonable to assume a $1\ ms$ interval as it is fast enough to keep track of changes such as phase variations that could affect the mapping decisions. At the same time, it allows some time for threads to run on the cores states they are mapped to before the next mapping decision is made.

V. EXPERIMENTAL SETUP

We show simulation results for 64-core systems. For the heterogeneous systems we test, we have four core states available: Big, $Mid2$, $Mid1$ and $Small$. Core configurations are shown in table I. All cores have private L1 and L2 caches and no L3 cache. For the homogeneous system, the configuration of cores and available VF settings are shown in table II.

We use Sniper simulator to model our heterogeneous and homogeneous systems. We generate random mixes of PARSEC and SPLASH-2 multi-threaded benchmarks and aim to achieve the maximum possible throughput. A representative sample of the workloads used for testing is shown in table III. We present results of using our heuristic on both heterogeneous systems and homogeneous systems with DVFS. We compare between two variations of heterogeneous systems: a system with 4 core states, 16 cores per state; and a system with just two core states ($Mid1$ and Big), 32 cores per state. For homogeneous systems, we implement the 2 DVFS schemes detailed in Section 3: Fixed VF levels and per-core DVFS. For the first method, we show results for using 4 VF domains and 2 VF domains ($High$ and $Mid1$).

TABLE III. WORKLOADS

1	Fluidanimate, lu.cont, lu.ncont, water.nsq
2	ocean.ncont, bodytrack, ocean.cont, ocean.ncont
3	Barnes, ocean.cont, radix, ocean.ncont
4	ocean.cont, barnes, blackscholes, ocean.ncont
5	lu.ncont, fluidanimate, radix, ocean.ncont
6	canneal, streamcluster, swaptions, streamcluster

VI. RESULTS AND EVALUATION

A. Comparison between Approaches

We examined several systems to use with our proposed heuristic: heterogeneous systems with 4 core types, heterogeneous systems with 2 core types, homogeneous systems with fixed VF levels per core and homogeneous systems with per-core DVFS. Given the tradeoff between maximizing performance and minimizing power, we use EDP (energy-delay product) as an appropriate metric for showcasing the benefits of our heuristic. Table IV presents power, performance and EDP comparison results for all experiments we conducted, normalized to a baseline that uses static scheduling and no DVFS. Figure 2 shows EDP results for the 6 representative workloads we selected. Our baseline system is a heterogeneous system packing the four core states shown in table I, 16 cores per state.

The average EDP for heterogeneous systems is around 24% better than the baseline. Among the configurations we tested, heterogeneous cores offered the best performance improvement (2X improvement on average). However, since Big cores consume high amounts of power, the performance boost came at the cost of increasing energy. For this reason, EDP, in this case, is not as high as expected. Varying the number of core states available has some effect on EDP depending on the workloads. For example, workloads 2 and 6 perform better with 4 core states while workloads 1,4 and 5 perform better with 2 core states. Workload 3 has worse EDP than the baseline because the thread scheduling led to small improvement in execution time with a significant increase in consumed energy, hence leading to degradation of EDP. In general, the average EDP calculated for all workloads for 4 core states was slightly higher than that of 2 states. We note here that DVFS methods can also be integrated with heterogeneous systems to gain better results concerning power consumption and thus improve the overall EDP. Using our heuristic with heterogeneous systems requires negligible hardware cost.

Homogeneous systems offered much more significant EDP improvement (an average of 60%). Our experiments show that for all 3 configurations of homogeneous systems we implemented, using our heuristic leads to decreasing both execution time and energy consumption. Using fixed VF levels per core achieved very similar results to using per-core DVFS. Furthermore, using 2 VF domains instead of 4 also achieved comparable results to per-core DVFS. Less hardware cost and design complexity are required for fixed levels VF due to

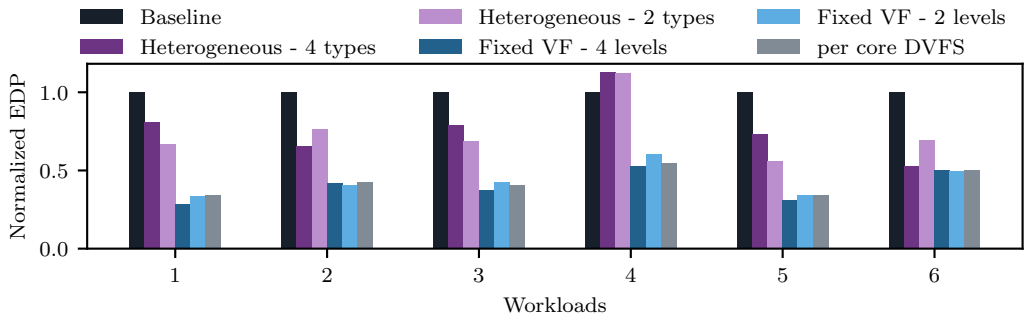

Fig. 2. EDP results comparison of the 5 methods implemented for 6 representative workloads

the need for fewer voltage regulators. Also, the overhead of switching VF values every control epoch is not present.

B. Heuristic Optimality

The time complexity of the algorithm is $O(n^2/m)$ where m is the number of core states in the system and n is the number of active threads. The complexity of the algorithm is determined by the second phase, generating all possible moves. For every core state, the maximum number of assigned tasks is n/m. The number of moves evaluated per level is $(n/m)^2$. Thus, the complexity for all core states is $O((n/m)^2 * m) = O(n^2/m)$. Compared to the recently proposed Extended MTS heuristic which has a time complexity of $O(mn^2)$ [7], our heuristic is more efficient.

Additionally, we compare our heuristic to Gurobi [25], a commercial ILP solver in terms of solution optimality and runtime. On average, the runtime of our heuristic was 3 orders of magnitude faster than the time consumed by Gurobi. Also, the solution provided by our heuristic was on average 0.6% away from the optimal one produced by Gurobi.

C. Scalability

Our heuristic can be used for any number of threads. If the number of active threads is greater than the number of cores, more than 1 thread is assigned per core, and they time-share the core. Since it is designed to be used online, the heuristic should be fast enough to fit into several milliseconds-long intervals. We ran simulations of our heuristic for systems with up to 1024 cores and recorded the heuristic runtime. Figure 3 shows that our solution scales well with increasing numbers of cores. It can be easily implemented for systems with hundreds of cores.

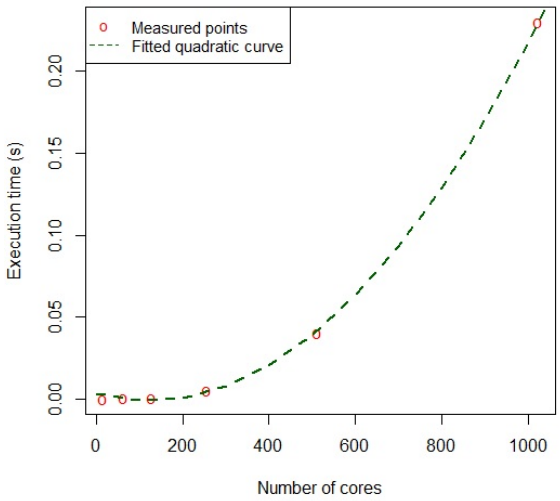

Fig. 3. Heuristic runtime with increasing number of cores

VII. Conclusion

We present a dynamic thread mapping algorithm for maximizing performance that can be used for heterogeneous and DVFS-enabled homogeneous multi-core systems. Our heuristic runs periodically online and needs no additional profiling runs. We show that our algorithm results in significant performance improvement and power saving. Simulation results show that we can achieve an average of 60% EDP improvement on homogeneous systems and 24% improvement on the heterogeneous system compared to benchmarks runs with fixed thread-to-core mapping. The proposed scheduler is scalable for systems with hundreds of cores and can accommodate any number of active threads.

Future work includes integrating DVFS methods into heterogeneous cores to improve power efficiency. Additionally, another enhancement would be to add fair scheduling methods to our scheduler to avoid thread starvation.

References

[1] NVIDIA, "Variable smp a multi-core cpu architecture for low power and high performance." NVIDIA, 2011. [Online]. Available: http://www.nvidia.com

TABLE IV. COMPARISON BETWEEN APPROACHES

Method	Heterogeneous - 4 states	Heterogeneous - 2 states	Fixed VF - 4 levels	Fixed VF - 2 levels	Per-core DVFS
Average Speedup	2.12	1.80	1.85	1.64	1.72
Average Power increase	1.63	1.31	0.73	0.70	0.71
Normalized EDP	0.77	0.75	0.39	0.43	0.41

[2] P. Greenhalgh, "Big.little processing with arm cortex-a15 & cortex-a7: Improving energy efficiency in high-performance mobile platforms," September 2011. [Online]. Available: http://www.cl.cam.ac.uk/ rdm34/big.LITTLE.pdf

[3] T. Sherwood, Calder, and B. Calder, "Time varying behavior of programs," Tech. Rep., 1999.

[4] B. Xu and D. H. Albonesi, "Runtime reconfiguration techniques for efficient general-purpose computation," *IEEE Des. Test*, vol. 17, no. 1, pp. 42–52, Jan. 2000. [Online]. Available: http://dx.doi.org/10.1109/54.825676

[5] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*, Feb 2002, pp. 29–40.

[6] G. Liu, J. Park, and D. Marculescu, "Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 54–61.

[7] ——, "Procrustes1: Power constrained performance improvement using extended maximize-then-swap algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1664–1676, Oct 2015.

[8] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 125–138. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755928

[9] Q. Chen, Y. Chen, Z. Huang, and M. Guo, "Wats: Workload-aware task scheduling in asymmetric multi-core architectures," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 249–260.

[10] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 213–224.

[11] T. M. Birhanu, Z. Li, H. Sekiya, N. Komuro, and Y.-J. Choi, "Efficient thread mapping for heterogeneous multicore iot systems," *Mobile Information Systems*, vol. 1565, p. 8, 2017.

[12] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd Conference on Computing Frontiers*, ser. CF '06. New York, NY, USA: ACM, 2006, pp. 29–40. [Online]. Available: http://doi.acm.org/10.1145/1128022.1128029

[13] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ser. ISCA '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 64–. [Online]. Available: http://dl.acm.org/citation.cfm?id=998680.1006707

[14] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Proceedings of the 46th Annual Design Automation Conference*, ser. DAC '09. New York, NY, USA: ACM, 2009, pp. 927–930. [Online]. Available: http://doi.acm.org/10.1145/1629911.1630149

[15] M. A. Awan, D. Masson, and E. Tovar, "Energy efficient mapping of mixed criticality applications on unrelated heterogeneous multicore platforms," in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016, pp. 1–10.

[16] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "Hass: A scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1531793.1531804

[17] T. E. Carlson, W. Heirmant, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2011, pp. 1–12.

[18] A. Annamalai, R. Rodrigues, I. Koren, and S. Kundu, "An opportunistic prediction-based thread scheduling to maximize throughput/watt in amps," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, Sept 2013, pp. 63–72.

[19] K. K. Rangan, G.-Y. Wei, and D. Brooks, "Thread motion: Fine-grained power management for multi-core systems," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 302–313. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555793

[20] S. Imamura, H. Sasaki, N. Fukumoto, K. Inoue, and K. Murakami, "Optimizing power-performance trade-off for parallel applications through dynamic core and frequency scaling," in *Proceedings of the RE-SoLVE'12*, 2012.

[21] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept 2010, pp. 29–39.

[22] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, Feb 2008, pp. 123–134.

[23] Q. Cai, J. Gonzlez, G. Magklis, P. Chaparro, and A. Gonzlez, "Thread shuffling: Combining dvfs and thread migration to reduce energy consumptions for multi-core systems," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, Aug 2011, pp. 379–384.

[24] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 72–81. [Online]. Available: http://doi.acm.org/10.1145/1454115.1454128

[25] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com