Dynamic Scheduling on Heterogeneous Multicores

Ayobami Edun, Ruben Vazquez, Ann Gordon-Ross* and Greg Stitt*

Department of Electrical and Computer Engineering, University of Florida
Gainesville, Florida, USA
Email: {aedun, ruben.vazquez, anngordonross, gstitt}@ufl.edu
Also affiliated with the NSF Center for Space, High-Performance, and Resilient Computing (SHREC) at UF

***Abstract*—Heterogeneous multicore systems help meet design goals by using disparate hardware components that are suitable for different application requirements/design goals. The individual cores may also have different tunable hardware parameters for additional specialization. However, this complicates scheduling since to reap the benefits of specialization, applications should be scheduled to the core that offers the best configuration based on the application's requirements and design goals. This scheduling decision could be made by exploring the design space to evaluate different configurations to determine the best configuration, or by executing the application in a base configuration to gather execution statistics to predict the best configuration. However, given increasingly complex systems, these methods may be infeasible given extremely large design spaces or difficulty in choosing a representative base configuration. In this paper, we present a dynamic scheduling methodology that uses predictive methods to schedule applications to best configurations for reduced energy consumption for a system with configurable caches. We use an artificial neural network (ANN) to train our predictive model using hardware counters. The trained ANN can then be used to predict the best core and a tuning heuristic explores the design space to determine the best configuration on non-best cores. If the best core is busy, our scheduler considers alternative idle cores or the application is stalled depending on which decision is energy advantageous. Our experiments show that system energy can be reduced by 28% on average as compared to a fixed-core system where all cores offer the same configuration.**

***Index Terms* — Dynamic scheduling, embedded systems, Machine Learning.**

I. Introduction

Energy efficiency, high performance, quality of service, etc. are key design goals in the embedded computing domain. Application-specific hardware specialization can be leveraged to meet these goals. Many hardware components can be specialized, such as the cache subsystem, processor bit-width, instruction set architecture (ISA) type, instruction issue width, etc.

Heterogeneous multicore systems offer a powerful mechanism for specialization since each core can provide different configurable architectural components—different *configurations*. For example, a system with different core ISA types may include MIPS, ARM, PowerPC, etc., and even different versions of the ISAs can be offered, such as ARM big.LITTLE, Intel Quick IA, etc. This heterogeneity offers a coarse-grained specialization, but to further improve design goal adherence, each core can offer different configurable hardware parameters for more fine-grained specialization, such as voltage, frequency, cache size, etc.

All combinations of different core types and different per-core configurable parameter values constitutes the configuration design space, of which each configuration offers different design goal tradeoffs (e.g., energy versus performance). An application's *best* (optimal or near-optimal) configuration is the configuration that most closely adheres to design goals, and this configuration is offered by the application's *best* core.

In configurable heterogeneous multicore systems, scheduling can be viewed as two separate steps: scheduling the application to the core that offers the best configuration, then configuring that core to the best configuration. Both steps are considerably challenging since the scheduler must be aware of all of the cores' configurations and make scheduling decisions based on many factors such as core availability, application deadlines, design goals, etc. If the application's best core is idle, the application is scheduled to that core and if the best configuration is already known on that core, the core is immediately configured to the best configuration. However. if the best configuration is unknown, design space exploration introduces tuning overhead in the form of system degradation as the configuration(s) is/are physically executed to determine the best configuration. If the best core is busy, the scheduler must additionally consider the tradeoffs between stalling the application until the application's best core is idle or running the application on a non-best core, thus incurring additional overhead while stalling, and tuning, and/or executing in a non-best configuration. Therefore, ineffective scheduling significantly impacts system operation and quickly obviates any benefits from such a configurable system [13].

Many approaches have been proposed to determine an application's best configuration. Exhaustive and heuristic-based [9] approaches evaluate all or a subset of potential configurations, respectively. However, for large design spaces in complex systems, this physical exploration is infeasible. Even though the large design space helps to increase design goal adherence due to the presence of numerous configurations, searching too many configurations incurs too much overhead, reducing/eliminating the benefits that the system would gain from specialization, since executing non-best configurations can incur tremendous energy and performance overheads. This makes determining the best configuration while limiting tuning overhead an arduous task.

To reduce or eliminate tuning overhead, predictive methods have also been proposed, such as regression models [3][11][22], and recently, machine learning techniques [19][25], which have vast potential and are the focus of our proposed work. Based on the prediction power of many machine learning techniques, we base our design on an artificial neural network (ANN), which we train to predict the best core for an application using execution statistics (e.g., instructions per cycle (IPC), number of stall cycles, memory/computation intensity).

In this work, we design a scheduler for a heterogeneous multicore system with disparate cores and disparate core configurations that optimizes energy consumption while considering performance. Without loss of generality, we consider a system with configurable caches, wherein each core offers a different total cache size, since cache size has the largest impact on energy and performance, and each core has configurable line sizes and associatvities. Our scheduler schedules applications to the application's best core based on the application's predicted best cache size using the trained ANN. If the

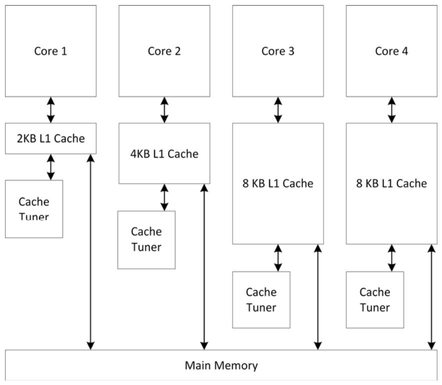

Figure 1: Core architecture.

best core is idle, the application is scheduled to that core, but if the best core is busy, an equation evaluates if it is energy advantageous to schedule the application to an idle non-best core or to stall the application until the best core is idle. After scheduling, if the best configuration on that core (best or non-best core) is known, the cache is directly configured to that configuration, and if the best configuration is not known, the application is profiled on the profiling core. Even though we focus on the cache system due to the cache's large contribution to system energy consumption [30], our work is generally applicable to any configurable system parameter. Our experiments show that total energy can be reduced by 28% on average as compared to a system that uses the same fixed cache configuration for all the cores.

II. Related Work

Multicore systems can execute applications efficiently since execution can be parallelized across the cores and the cores can be specialized to meet different application goals/requirements. Much research has focused on leveraging heterogeneous systems for energy efficient computing. Several algorithms scheduled applications to cores offline [10][23][24] while others do this scheduling at runtime [2][6][8]. Kumar et al. [9] used a heuristic to schedule applications to cores using a single ISA system. That method had a sampling phase where all possible scheduling of jobs to cores was examined and then the best scheduling was determined. However, the sampling approach is not realistic as the number of cores and applications increases since all possible combinations must be examined. Becchi et al. [2] also used a sampling technique where threads were executed periodically on all types of cores, however, that approach posed high tuning overhead.

Recent works have been able to build models that can predict what core configurations are suitable for an application [20][28]. Silva et al. [21] attempted to optimize both performance and energy using an offline data mining approach to determine the best core, then suggested the best scheduling of applications to cores, however, that work was limited due the offline scheduling that could not consider runtime variations. Sayadi et al. [20] used different machine learning algorithms for energy efficient prediction and scheduling. That work is most closely related to our proposed work, however, their work focused more on different core architecture compositions while we focus on configurable caches. The authors showed the impact of using several machine learning techniques, including neural networks, to obtain more accurate results. Van Craeynest et al. [29] used Performance Impact Estimation (PIE) to predict the scheduling of workloads to cores by extracting performance information, such as cycles per instruction (CPI), number of cache misses, instruction level parallelism (ILP), etc., to estimate the performance of a workload if the workload executed on a different core. This information was used to determine if the application should continue executing on the current core or if the application should be migrated to a different core based on the core's PIE prediction. Chen et al. [4] used Euclidean distance to schedule applications by mapping the core's configuration and the application's resource demands in to a multidimensional space. That work reduced energy by approximately 6.1%, which shows the limitation of heuristics since, as the core count increases, thus so will the design space, resulting in design space exploration becoming an increasingly arduous task.

Other works focused on optimizing different design goals using different scheduling techniques. Salamy et al. [18] used ILP to schedule applications on multicores under energy and power constraints. Munawar et al. [14] evaluated peak power management for real time tasks by introducing sleep cycles on each active core to ensure power consumption was within the thermal constraints. Other authors optimized for reliability [15] and temperature [27]. Liu et al. [12] used optimal cluster scheduling for real time task scheduling on heterogeneous systems to reduce energy, however, their system incurred a high tuning overhead due to task migrations.

Our work focuses on optimizing energy consumption. We build on prior work [1] where the authors proposed a scheduling and tuning (SaT) algorithm that used data mining to subset core configurations and scheduled applications to both best and non-best cores. However, that work executed applications with each configuration on each core, while our work uses machine learning to reduce tuning overhead. Our scheduler uses a single application execution to predict the best core and a tuning heuristic to determine best configuration on non-best cores. To the best of our knowledge, this is the first dynamic scheduling solution that combines machine learning techniques and a heuristic for energy optimization.

III. System Architecture and Core Configurations

Figure 1 depicts our sample quad-core heterogeneous system architecture, which is suitable for our experimental benchmarks (Section VI). This general structure could be scaled up or down for different system requirements. Each core has a dedicated private level one (L1) cache, a cache tuner that configures the L1 cache parameters, and a private non-configurable level two (L2) cache.

Table 1:
Cache configuration design space

2KB_1W_16B	2KB_1W_32B	2KB_1W_64B
4KB_1W_16B	4KB_1W_32B	4KB_1W_64B
4KB_2W_16B	4KB_2W_32B	4KB_2W_64B
8KB_1W_16B	8KB_1W_32B	8KB_1W_64B
8KB_2W_16B	8KB_2W_32B	8KB_2W_64B
8KB_4W_16B	8KB_4W_32B	8KB_4W_64B

Table 1 shows the complete cache configuration design space, which lists the cache size in Kbytes (KB), the associativity in number of ways (W), and the line size in Bytes (B). Since the cache size has the largest impact on energy and performance, tuning overhead is reduced by subsetting the design space across all cores, wherein each core has a fixed cache size. Core 1, Core 2, Core 3, and Core 4 use 2KB, 4KB, 8KB, and 8KB cache sizes, respectively. On each core, the line size can be configured to 16B, 32B, or 64B, and the associativity can be configured to 1-, 2-, or 4-way. These values were chosen based on prior work on cache configuration [1] and the requirements of our experimental benchmarks, however, we note that these values can be altered for different application requirements.

To gather execution statistics for the ANN predictor, we profile the application and predict the best configuration - cache size, line size and associativity using Core 4 (a core with the largest cache size) while executing in a base cache configuration of 8KB_4W_64B. This configuration provides a pessimistic view with respect to energy consumption, has the lowest number of cache misses, and has been shown in prior work to be an ideal general configuration [1]. Even

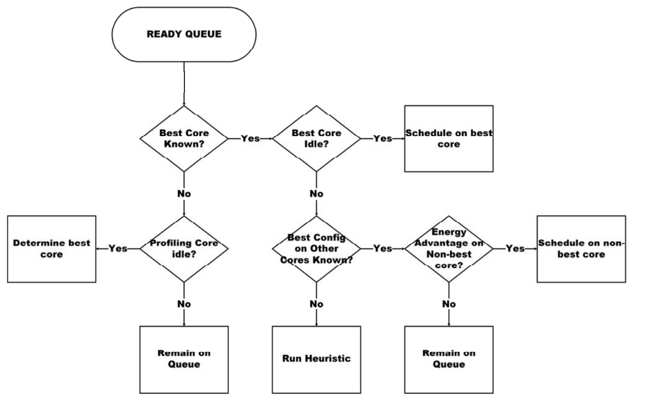

Figure 2: Our proposed scheduler

though for simplicity Core 4 is the primary profiling and prediction core, Core 3 can also be used as a secondary profiling core if Core 4 is busy. Likewise, when Core 4 is idle (not profiling), Core 4 can execute applications.

IV. Scheduler

A. Overview

Cores 3 and 4 are profiling cores and both cores can run our scheduler, the ANN predictor, and orchestrate tuning, which is part of the operating system kernel. Since we designate Core 4 as the primary profiling core, Core 4 also contains a profiling table that stores profiling information for all applications, including the execution statistics for the base configuration, and the performance and energy consumption of any core configurations that have been explored during design space exploration (Section IV.E). This storage eliminates future profiling executions and enables the tuning heuristic to operate across multiple application executions. We assume that when Core 3 is used as the secondary profiling core, Core 3 can access Core 4's profiling information using the existing communication network.

Figure 2 shows our *proposed system's* operational flow. Incoming applications are stored in a ready queue and are processed in first-come first-served (FIFO) order. When an application is to be executed, the application is dequeued from the ready queue. If the application has no profiling information, the application is profiled on Core 4 (Section IV.B) and the ANN (Section IV.C) predicts the best core (Section IV.D). If there is profiling information, a scheduling decision is made. If the best core is idle, the application is scheduled to that core. If the best core is busy, and the best configurations are not known for the idle core(s), the scheduler is unable to determine if it is energy advantageous to schedule to a non-best core (Section IV.E) because there is not enough information about the design space.

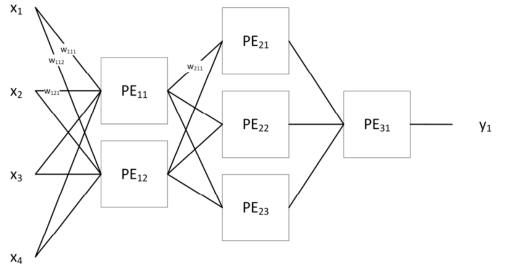

Figure 3: 3-layer ANN to predict the application's best core

In this situation, the application is scheduled to an arbitrary idle core. Regardless of the scheduled idle core, if the best configuration on that core is known, the core is directly configured to that configuration, otherwise a tuning heuristic (Section IV.F) iteratively explores the design space on that core to determine the best configuration on that core. If the application's best configuration is known for the idle non-best core(s), the scheduler can evaluate whether it is energy advantageous to stall the application until the best core is idle or to run the application on a non-best core. If the application is stalled, the application is enqueued back into the ready queue.

B. Profiling

During profiling, the application's execution statistics while executing in the base configuration are recorded using built in hardware counters, such as memory access counts, cache misses, etc. This profiling could be eliminated if the applications were known a priori with profiling-based statistics recorded at design time and this profiling information can be pre-loaded. However, since this does not provide much flexibility, we assume that the profiling is done dynamically during runtime. After profiling the application, the execution characteristics are stored in the profiling table. This information is used to predict the best core using the ANN predictor.

C. ANN predictor

Figure 3 shows the 3-layer ANN that our scheduler uses to predict the best core (i.e., best cache size). The ANN is made up of the inputs (**x**), the weights (**w**), the processing elements (**PEs**) that compute an output from the given inputs and weights, and the final output (**y**). The inputs to the ANN are the cache-relevant execution statistics for each of the training applications. The vertically aligned PEs constitute a layer and the number of layers defines the ANN's size and is notated as $\{n_1, n_2, \ldots, n_m\}$, where n_m is the number of PEs in *m*-th layer. Empirical analysis determined that the best ANN size for the best cache size prediction was {10, 18, 5, 1}.

D. Best core prediction

To make accurate best core predictions, the ANN must be trained using applications from a similar application domain using features that reflect execution statistics of the configured hardware components. Prior work showed that applications from similar application domains have similar execution statistics [16]. Since we target embedded systems, we trained the machine learning algorithm offline using the EEMBC benchmark suite as training data and an energy model (Section V).

The training data consisted of 270 total inputs—18 different cache-relevant execution statistics for each of the 15 benchmarks. We note that for diverse systems executing different application domains, the scheduler could have multiple ANNs each of which would be specialized for a different domain. After feature selection, the execution statistics most relevant for cache size prediction where the total number of instructions, the number of cycles for one complete benchmark execution, the number of load and store instructions, the number of branches, and the number of integer and floating-point instructions.

We split our training data into three sets: the ANN was trained with 70% of the input data, 15% was used as the validation set, and 15% was used as the test set. We used bagging to improve the ANN's accuracy and generalization, which trains several different ANNs using a subset of the input data and averages the ANNs' outputs to determine the final prediction. We trained 30 ANNs and initialized the model weights randomly, resulting in ANNs that may result in more accurate predictions for certain inputs. Results obtained using methods described in Section V showed that the ANNs predicted best cache sizes (indicating the best core) only degraded the average

```
Algorithm 1: Tuning Heuristic
  Data: Core to test, application
  Result: Configuration with lowest energy
  initialization: Cache size is fixed, associativity and line size set to min;
  run application with min values;
  update config, energy;
  calculate energy;
  while energy is less than previous do
      increase associativity;
      if associativity is max then
          increase line size, make associativity min;
      else
          continue;
      end
      run application on core;
      calculate energy;
      update best config;
  end
```

Figure 5: Cache tuning heuristic

energy consumption by less than 2% over all the benchmarks as compared to the optimal cache size.

E. Scheduling on a non-best core

If the application's best core is busy, and there are other cores that are idle, the application can be scheduled to run on a non-best core. Since these cores are expending idle energy anyway, it may be energy advantageous to utilize this idle energy to execute the application on a non-best core. However, since the application will require more dynamic energy executing on a non-best core, this extra energy must be considered when making scheduling decisions.

To make this decision, the scheduler must know the energy consumption of the application executing with the best configuration offered on the non-best core. If this configuration is known, the configuration and the configuration's energy and performance are stored in the profiling table. The scheduler evaluates whether or not it is energy advantageous: stalling the application until the best core is idle or scheduling the application to a non-best core.

This decision can be explained using a simple example. Assuming a system with two applications *A* and *B* where *A* and *B* have the same best core *C1,* the energy advantageous decision (E_{adv}, a Boolean value where 1 indicates that stalling is energy advantageous) is evaluated using the following equation:

$$E_{adv} = Energy_{C_1}^{A} + \ Energy\ _{C_1}^{B} + Idle\ Energy_{C_2} > \ \ Energy\ _{C_2}^{B} + Energy\ _{C_1}^{A}$$

where $Energy_{C_1}^{A}$ and $Energy\ _{C_1}^{B}$ are the energy consumptions of *A* and *B* executing on these applications' best core C_1, $Energy\ _{C_2}^{B}$ is the energy of *B* executing on a non-best core C_2, and $Idle\ Energy_{C_2}$ is the idle energy of core C_2.. If *A* is executing on the best core C_1 when *B* arrives and C_2 is idle, the scheduler compares the energy expended by executing *B* on C_2 ($Energy\ _{C_2}^{B}$) to the energy expended by stalling *B* until *A* completes. The stall energy is calculated as the energy that will be consumed while executing the remaining portion of *A*, plus the idle energy expended by C_2 for that same period, plus the energy to execute *B* on C_1 after *A* completes execution. The remaining execution time in cycles for *A* is equal to the total number of cycles to execute *A* for one complete execution minus the number of cycles that *A* has already been executing for. The remaining energy consumption can be estimated by multiplying this remaining number of cycles by the average energy consumption per cycle. This stall energy is evaluated for all idle cores where the best configuration on that core is known. If this stall energy is greater than the energy expended by running *B* on C_2 ($Energy\ _{C_2}^{B}$), and *A* on *C1, B* will be scheduled to the non-best core C_2 if and only if the best configuration is known for all cores, else *B* will be enqueued back into the ready queue and will stall for *B*'s best core.

Even though this evaluation may indicate that *B* should be scheduled to a non-best core, the scheduler must gather information about all system cores to make more accurate future scheduling

$$E(total) = \ \ E(sta) + E(dynamic)$$
$$E(dynamic) = cache_{hits} * E(hit) + cache_{misses} * E(miss)$$
$$E(miss)$$
$$= E(off\ chip\ access) + miss_{cycles} * E(CPU_{stall}) + E\left(cache_{fill}\right)$$

$$Miss\ Cycles$$
$$= cache_{misses} * miss_{latency} + \left(cache_{misses} * \left(\frac{linesize}{16}\right)\right)$$
$$* \ memory\ bandwidth$$
$$E(sta) = total\ cycles * E(static\ per\ cycle)$$
$$E(static\ per\ cycle) = E(per\ Kbyte) * cache_{size}(in\ KBytes)$$
$$E(per\ Kbyte) = \frac{E(dyn\ of\ base\ cache) * 10\%}{base\ cache\ size\ in\ Kbytes}$$

Figure 4: Energy model

decisions. If there any core's best configuration is unknown, the scheduler will schedule *B* to one of these cores arbitrarily.

F. Cache tuning heuristic

When an application is scheduled to a non-best core, if the best configuration is unknown, the cache tuning heuristic depicted in Figure 5 determines the best configuration. This tuning process takes several application iterations since the application is executed in a single configuration each time the application is scheduled that same core, and that configuration's energy consumption and performance are stored in the profiling table. Each time the application executes on a core, the heuristic can continue where the exploration left off using the configuration information stored in the profiling table.

The tuning heuristic explores the associativity followed by the line size, since the associativity has the second largest impact on energy after the size. Each parameter is explored from the smallest to the largest value to minimizing cache flushing. Exploration begins with the smallest value for both parameters, and the energy consumption of that configuration is the currently known lowest energy configuration. The associativity is iteratively increased while there is a reduction in energy, which updates the currently known lowest energy configuration to that configuration, or the maximum associativity is reached. After determining the lowest energy associativity, the associativity is fixed to that associativity and the line size is similarly iteratively increased and the currently known lowest energy configuration is updated. At the end of this tuning process, the best configuration on that core is known.

V. Experimental Setup

We evaluated our proposed system by simulating different systems using MATLAB, the architecture in Figure 1, and the complete EEMBC suite [5]. Each benchmark was assigned an identification number, which indexed into the profiling table. We created 5000 uniform distribution arrival times of these benchmarks to ensure that the system executed long enough to depict stable results. On arrival, benchmarks were enqueued and processed on a FIFO basis. The scheduler was invoked to make scheduling decisions each time a benchmark arrived or when a core became idle.

Figure 4 shows our energy model [1]. We used SimpleScalar [26] to record the benchmarks' cache accesses and miss rates for every cache configuration. We used CACTI [17] and assumed a 0.18um technology to obtain dynamic energy values and estimated off-chip memory access energy using a standard low-power Samsung memory. We assumed a fetch from main memory took forty times longer than an L1 cache fetch and the memory bandwidth was 50% of the miss penalty [1], which are reasonable assumptions for our application domain.

To measure the impact of our proposed system on energy and performance, including the ANN predictor, the tuning heuristic, and the energy advantageous scheduling decisions, we created three different systems assuming no form of preemption or priority, which is the focus of future work. The *base system*'s cores all used the base configuration of 8KB_4W_64B, thus there was no profiling, and the ANN and tuning heuristic were not used. The *optimal system* used the core configuration subsets in Figure 1, profiled the benchmark on the profiling core, but did not use the ANN predictor to determine the best core. This system executes each benchmark using all possible configurations to determine what the best configuration is and only schedules to the best core when that core is idle. Even though we denote this as the *optimal system,* we are referring to the fact that the benchmark only executed in the optimal configuration on the core that the benchmark was scheduled to, and not a system where all possible scheduling decisions (e.g., stall rather than run on a non-best core) were made to determine the lowest energy. This system enabled evaluation of the effect of eliminating excess stall energy incurred while waiting for best cores and comparison with ANN enabled system. The e*nergy-centric system* only scheduled benchmarks to the benchmark's best core even if idle cores were available. Even though it may have been energy advantageous to schedule the benchmark to a non-best core, this stall decision left non-best cores free to execute future benchmarks that otherwise might stall due to a benchmark executing on a non-best core. This system profiled the benchmark on the profiling core, used the ANN predictor to determine best configuration for each benchmark. This system enabled evaluation of our system's energy advantageous scheduling decision.

Finally, we note that direct comparison could not be made with past work that used machine learning techniques since these prior works evaluated tuning the core type and frequency and not the cache configuration.

VI. Results

In this section, we present our experimental results comparing our proposed system to the systems defined in Section V in terms of idle energy, dynamic energy, total energy, and performance in number of cycles. Figure 6 shows the idle, dynamic, and total energy consumed for each system normalized to the base system. The optimal system reduced idle, dynamic, and total energy by approximately 3%, 35%, and 6%, respectively. We note that even though this system is denoted as the optimal system, as discussed in Section VI, this system may not have the lowest energy. Since the energy-centric system used the ANN predictor, the energy-centric system reduced dynamic energy by 58%, with a slight increase in idle and total energy of 6% and 2%, respectively. Comparing the optimal and energy-centric systems showed that the optimal system had lower idle energy and higher dynamic energy compared to the energy-centric system, the dynamic energy revealed that, scheduling benchmarks using a system with the ANN predictor (energy-centric system) outperforms a system with an exhaustive search (optimal system). This advantage is even more pronounced when the design space is extended beyond cache configuration and a predictor is used. Our system reduced idle, dynamic, and total energy by 27%, 55%, and 29%, respectively. These results reveal the importance of the energy advantageous scheduling decision, revealed in the lower total energy even though there was a slight increase in dynamic energy.

Figure 7 depicts the total number of cycles, idle energy, dynamic energy, and total energy of the energy-centric system and our system normalized to the optimal system. The energy-centric system and our system reduced the total number of cycles by 17% and 25%, respectively. Even though the energy-centric system stalled benchmarks until the best core was idle, the energy-centric system was able to improve performance, showing the advantage of predictive schemes in a system. Our system revealed an additional 10% improvement even though performance was only partially considered in the energy advantageous decision.

Comparing to the optimal system showed a more detailed analysis of the energy effects shown in Figure 6. The energy-centric system decreased the dynamic energy by 35%, but increased the idle and total energy by 10% and 9%, respectively, resulting in an overall increase in total energy. As compared to the optimal system, our system reduced idle, dynamic, and total energy by 26%, 31%, and 24%, respectively. As compared to the energy centric system, our system reduced the idle and total energy by 32% and 31%, respectively, but increased the dynamic energy slightly by 7%.

These results revealed an interesting observation about our system and the energy-centric systems. When designing these systems, we hypothesized that the fixed scheduling decisions (never stall vs. always stall, respectively) would result in the intended solution. Another words, the hypothesis that stalling benchmarks until the benchmarks' best core was idle, thereby leaving non-best cores free to execute future benchmarks did not result in the best total energy savings, showing that this decision can not be made naively, and requires an energy advantageous evaluation—our proposed system.

Finally, we evaluated the profiling overhead and tuning heuristic's efficiency. Profiling only introduced less than .5% overhead in total energy consumption. Even though our heuristic may explore a minimum of three configurations and a maximum of nine configurations, out of 18, no benchmark explored more than six configurations, thus our tuning heuristic explored significantly fewer configurations than the optimal system

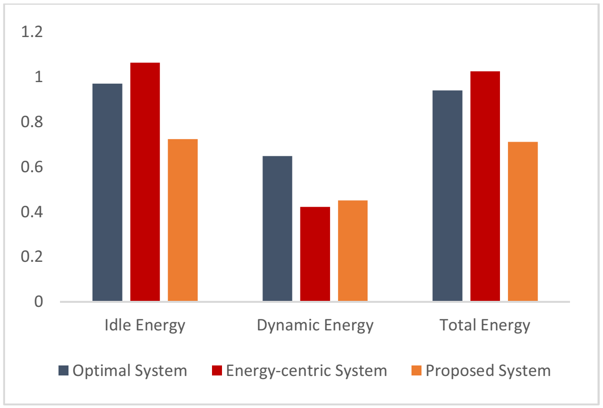

Figure 6: Energy consumption normalized to base system

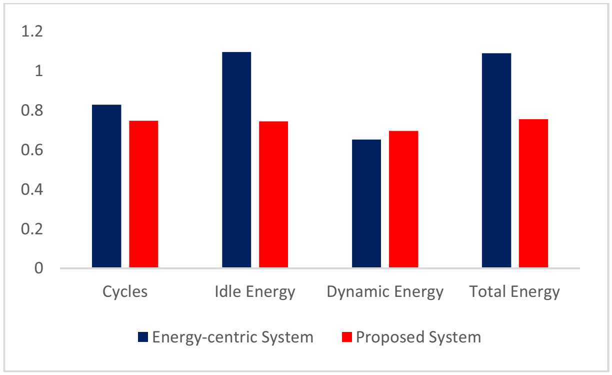

Figure 7: Performance in number of cycles and energy normalized to the optimal system

VII.

VIII. Conclusion and Future Work

Scheduling on heterogeneous cores is a very challenging problem given the wide variety of configurable hardware parameters, thus leaving large design spaces that are difficult to explore. Configuring these parameters to specific application requirements and scheduling applications to cores that offer the best configuration can reveal large optimization potential. In this paper, we used an ANN predictor to predict the best core for an application given profiling statistics for heterogeneous cores that offered different cache sizes, and a tuning heuristic to determine the best configuration on non-best cores. We also presented a scheduling method that scheduled applications to the applications best core if the core was idle. If the core was not idle, the scheduler evaluated whether or not it was energy advantageous to schedule applications to an idle non-best core, or stall the application until the best core was idle. Results showed a reduction in total energy of 28% and a 17% increase in performance.

Future work includes evaluating different machine learning techniques and considering systems with preemption, priority, and deadlines, and additional levels of private and shared caches. We will also evaluate overheads introduced by the machine learning algorithm.

IX. Acknowledgements

This work was supported by the National Science Foundation (CNS-0953447 and CNS-1718033). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

X. References

[1] M. H. Alsafrjalani; A. Gordon-Ross; "Dynamic Scheduling for Reduced Energy in Configuration-Subsetted Heterogeneous Multicore Systems," Int. Conf. on Embedded and Ubiquitous Computing. 2014

[2] M. Becchi and P. Crowly, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers*. New York, NY, USA: ACM, 2006, pp. 29–40.

[3] J. L. Berral, R. Gavalda, and J. Torres, "Adaptive scheduling on power aware managed data-centers using machine learning," in Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing, ser. GRID '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 66–73.

[4] J. Chen and L. K. John. Efficient program scheduling for heterogeneous multi-core processors. In Proceedings of the 46th Design Automation Conference (DAC), pages 927–930, July 2009.

[5] EEMBC. The Embedded Microprocessor Benchmark Consortium. https://www.eembc.org/ benchmark/automotive_sl.php, Sept. 2013.

[6] C. Goh, E. Teoh, and K. Tan, "A hybrid evolutionary approach for heterogeneous multiprocessor scheduling," *Soft Computing*, vol. 13, no. 8-9, pp. 833–846, March 2009.

[7] V. J. Jimenez, L. Vilanova, I. Gelado, M. Gil, G. Fursin,and N. Navarro, "Predictive runtime code scheduling for heterogeneous architectures," in *Lecture notes in Computer Science*, vol. 5409, 2009, pp. 19–33.

[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*, April 2010, pp. 125–138.

[9] 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*, June 2004, pp. 64–75.

[10] N. Lakshminarayana, J. Lee, and H. Kim, "Age based scheduling for asymmetric multiprocessors," in *Super Computing*, November 2009.

[11] M. Y. Lim, A. Porterfield, and R. Fowler, "Softpower: fine-grain power estimations using performance counters," in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 308– 311.

[12] D. Liu, J. Spasic, G. Chen, and T. Stefanov, "Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs," in Proceedings of ESTIMedia, Oct 2015, pp. 1–10.

[13] A. Lukehahr et al., "Composite cores: Pushing heterogeneity into a core", In *MICRO-45,* pp. 317-328, 2012.

[14] W. Munawar et al., "Peak power Management for scheduling real-time tasks on heterogeneous many-core systems," in proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS) pp. 200-209, Dec. 2014.

[15] A. Naithani, S. Eyerman, and L. Eeckhout, "Reliability-aware scheduling on heterogeneous multicore processors," in Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA), Feb. 2017, pp. 397–408

[16] M. Rawlins and A. Gordon-Ross. An Application Classification Guided Cache Tuning Heuristic for Multi-core Architecture (ASP-DAC), Jan 2012

[17] G. Reinman, and N.P. Jouppi, COMPAQ Western Research Lab: CACTI2.0: An Integrated Cache Timing and Power Model, 1999.

[18] H. Salamy, S. Aslan, D. Methukumalli, "Task scheduling on multicores under energy and power constraints," in proceedings of the 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). May 2013.

[19] R. Sarikaya, C. Isci, and A. Buyuktosunoglu, "Program behavior prediction using a statistical metric model," in Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems, ser. SIGMETRICS '10. New York, NY, USA: ACM, 2010, pp. 371–372.

[20] H. Sayadi, N. Patel, A. Sasan, H. Homayoun, "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures," ICCD-17, Nov. 2017.

[21] B. Silva, L. Cuminato, A. Delbem, P. Diniz, and V. Bonato. Application oriented cache memory configuration for energy efficiency in multi-cores. IET Computers & Digital Techniques. 1 (9). January 2015. pp. 73–81.

[22] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," SIGARCH Comput. Archit. News, vol. 37, no. 2, pp. 46–55, Jul. 2009.

[23] D. Shelepov, J. Carlos, S. Alcaide, S. Jeffery, A. Fedorova,N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "HASS: a scheduler for heterogeneous multicore systems," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 2, pp. 66–75, April 2009.

[24] D. Shelepov and A. Fedorova, "Scheduling on heterogeneous multicore processors using architectural signatures," in *Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture*, June 2008.

[25] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," in Proceedings of the 30th annual international symposium on Computer architecture, ser. ISCA '03. New York, NY, USA: ACM, 2003, pp. 336–349.

[26] SimpleScalar: http://www.simplescalar.com, July 2018.

[27] Q. Tang, S. K. Gupta, and G. Varsamopoulos. Energy-efficient thermal aware task scheduling for homogeneous high-performance computing data centers: a cyber-physical approach. IEEE Transactions on Parallel and Distributed Systems, 19:1458–1472, 2008.

[28] M. K. Tavana et al., "ElasticCore: enabling dynamic heterogeneity with joint core and voltage/frequency scaling," DAC-15, pp. 1-6, June 2015.

[29] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in ACM SIGARCH Computer Architecture News, vol. 40, no. 3. IEEE Computer Society, 2012, pp. 213–224.

[30] C. Zhang; F. Vahid; W. Najjar, A highly configurable cache architecture for embedded systems. In Proceedings of the 30th Annual International Symposium on Computer Architecture, San Diego, CA, USA, 9–11 June 2003; pp. 136–146.