Dynamic Runtime Optimizations for Systems of Heterogeneous Architectures

Geoffrey Phi C. Tran
Information Sciences Institute
Dept. of Electrical Engineering - Systems
Viterbi School of Engineering
University of Southern California
Los Angeles, CA 90007
Email: geoffret@usc.edu

Dong-In Kang
Information Sciences Institute
Viterbi School of Engineering
University of Southern California
3811 North Fairfax Drive, Suite 200
Arlington, VA 22203
Email: dkang@isi.edu

Stephen Crago
Information Sciences Institute
Dept. of Electrical Engineering - Systems
Viterbi School of Engineering
University of Southern California
3811 North Fairfax Drive, Suite 200
Arlington, VA 22203
Email: crago@isi.edu

***Abstract*— In today's embedded systems, engineers are trying to get as much performance out of designs while minimizing the energy consumed in order to maximize battery life. Furthermore, embedded systems and their computational sub-systems are becoming more heterogeneous, containing compute resources such as general-purpose processors, graphics processing units, and FPGAs. Because of this heterogeneity, there is a rich area for optimization, especially when considering the mapping of a dynamic, real-time application to these heterogeneous resources. One approach involves maximizing the performance of a task on a given architecture with a given energy constraint. However, this approach will not minimize power and energy consumption. Therefore, in this paper, we propose new dynamic runtime optimizations that can schedule dynamic tasks to a heterogeneous system while minimizing energy consumption and deadlines missed. Through experimentation, we found improvements in energy efficiency of up to 390x relative to a baseline greedy scheduler.**

I. Introduction

As technology finds its way into more areas now than ever before and embedded platforms become more sophisticated, users are now demanding more computational power for less energy consumed. Furthermore, systems of heterogeneous architectures are also becoming more prevalent in order to increase power efficiency. In order to meet requirements such as the aforementioned, a novel approach to take advantage of the strengths of different types of processing elements while balancing performance and power constraints becomes important. In this paper, we propose a number of dynamic runtime optimizations for the scheduling of incoming tasks to a heterogeneous computing platform.

The rest of this paper is organized as follows: Section II discusses previous work on optimizing energy consumption for processors. Section III presents the problem we have explored. Section IV presents the optimizations that we propose. Section V discusses our experimental procedure and results. Finally, Section VI discusses our conclusions and future work.

This work has been funded by DARPA cooperative agreement number HR0011-12-2-0023

II. Related Work

While there are many other works investigating runtime optimizations, we believe that that this is the first work that optimizes the assignment of dynamically occurring real-time tasks at run-time on a set of heterogeneous processing elements. In the static resource assignment problem, tasks are known prior to runtime. The resource assignment is performed on a fixed set of tasks and a resource assignment and schedule is generated for each task. In the dynamic resource assignment problem, tasks to be executed arrive during runtime. Therefore, assignments only take the current task and system state into account. There are also variations on the problem, such as whether task execution characteristics are stochastic or deterministic and whether tasks have any dependencies.

Rusu et al. look at the static scheduling of a set of tasks onto a single computational resource while maximizing the system value subject to energy and time delay [1]. Furthermore, the set of tasks to be executed is known at schedule time. Yang et al. proposed an approximation algorithm for energy-efficient real-time scheduling on a homogeneous chip multiprocessor [2]. They also proved that the energy minimization problem is NP-hard. Seo et al. proposed a dynamic repartitioning algorithm on multi-core multiprocessor system to optimize power consumption at run-time [3].

The problem of statically assigning tasks to a heterogeneous system has been explored in [4]. Here, it was proven that the *heterogeneous assignment problem* is NP-complete. An algorithm was proposed to minimize energy consumption while meeting a time constraint for the whole application. Another algorithm was then presented that minimized the hardware utilization. Energy optimization using Dynamic Voltage Frequency Scaling (DVFS) is not considered, though. In [5], the static assignment problem is explored. The work described in this paper utilizes similar models for architecture, communication, and tasks. We also pursue the goal of optimizing energy consumption while meeting a time constraint. However, our model does not assume static dependencies among the tasks and the knowledge of the tasks ahead. We use DVFS to reduce energy consumption further. Li and Wu explore partition-based energy-aware scheduling for independent tasks on heterogeneous platforms [6]. It is shown that workload-

978-1-4799-6233-4/14/$31.00

balanced partitioning methods are not optimal in terms of energy consumption. The algorithm proposed explicitly targets minimizing energy consumption using DVFS. Our work is also a partition-based scheduling using DVFS but considers dependent tasks. Again, tasks are known at scheduling time in [6], so this is another example of static scheduling. Young et al. investigates the problem of scheduling dynamically arriving independent tasks to a DVFS-enabled heterogeneous cluster computing environment [7]. In this work, several heuristics are presented that aim to minimize the deadlines missed while meeting specific energy and time of completion constraints. Unlike [7], our model considers a task model which may have dependencies between tasks, and our primary goal is to minimize energy consumption.

Our work looks at the problem of dynamically assigning tasks to a heterogeneous processing system. Arriving tasks may have dependencies that must be enforced. Each task also has an explicit deadline. The goal is to minimize energy consumed while meeting the deadlines for each task. Factors such as inter-task communication are also considered.

III. Problem Description

Many of the optimizations today focus on improving the performance or energy efficiency of a single task on a given architecture. However, many systems today are composed of a system of heterogeneous architectures. Examples of the processing elements in these heterogeneous architectures include general-purpose processors, graphical processing units, and FPGAs. Being able to take advantage of the strengths of each resource could result in large energy savings. We propose a dynamic runtime scheduler that can process tasks as they arrive to the system and schedules each task to a given computational resource based on the current usage using heuristics. The runtime system cannot look into the future to determine the optimal schedule, and thus must consider only what is available at the current time.

Using a runtime scheduler allows the handling of dynamic task arrivals. In embedded applications, tasks often have dynamic behavior. Dynamic behavior may have multiple sources. First, embedded systems that are processing sensor data can have dynamic behavior driven by a dynamic external environment. For example, a system that tracks targets that are contained in a video feed will have behavior and performance requirements based on the number of targets that are present in the sensor field within a given time window. Image processing techniques may depend on lighting conditions or the frequency content of the input images. Second, user input or requirements can change behavior dynamically. For example, a user may choose between modes of an application by sending a command to a system. Modes might include a standby mode and an active processing mode, or different modes could determine types of targets being tracked, or different types of signal or image processing being performed. Another example is that a user might turn communication channels on and off in a smart radio/phone type of application. Third, dynamic behavior might arise simply because an application or system behavior is too complex to analyze and optimize at static design or compile time. For example, memory hierarchy (e.g. cache) behavior may be too complex to determine from program

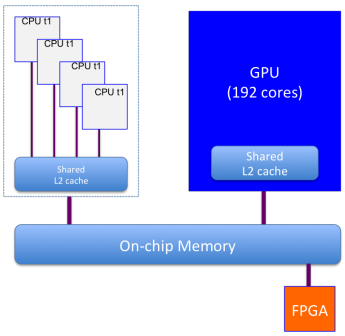

Fig. 1: Target Architecture (28nm)

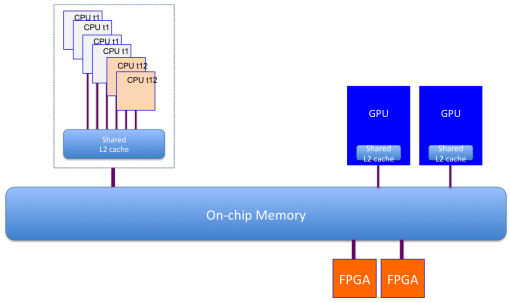

Fig. 2: Target Architecture (14nm)

input. The experiments in this paper consider the first two sources: a dynamic environment and user input.

A. Architecture Description

In order to evaluate the different optimizations proposed, we defined a heterogeneous target architecture that represents several generations of embedded heterogeneous processors. Heterogeneity is becoming predominant for embedded processing architecture, since heterogeneous processing elements can be adapted toward specific processing types to gain efficiency and processing elements that are not used for a specific application or phase can be turned off[8]. In order to estimate reasonable numbers of processing elements as well as their performance characteristics and power efficiency, we started with a modern mobile architecture and projected expected performance and efficiency improvements from 28nm to 7nm nodes.

We use the NVIDIA Tegra K1 mobile processor as the reference design for our target architectures. The NVIDIA Tegra K1 mobile processor has a Kepler™ GPU with 192 NVIDIA CUDA® cores and the NVIDIA 4-Plus-1™ quad-core ARM Cortex™-A15 CPU[9]. Figures 1, 2, and 3 illustrate the architectures at 28nm, 14nm, and 7nm respectively. The differences between each technology node include not just the numbers of each type of element, but characteristics such as V_{dd} and frequency.

1) DVFS Model: The models used for Dynamic Voltage and Frequency Scaling are based on the alpha-power law[10][11]. Equation 1 is derived using these models. Equation 1 is then used to show the relationship of execution times and energy consumed between V_{dd} and V_i in equations 4 and 5.

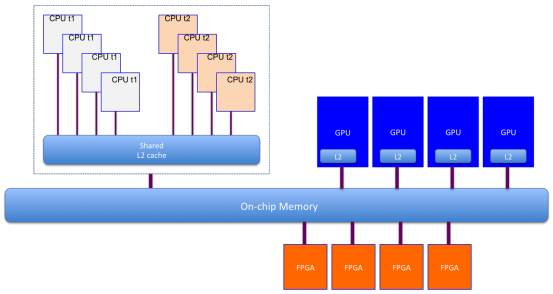

Fig. 3: Target Architecture (7nm)

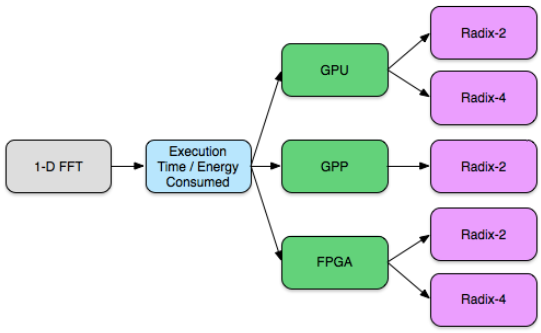

Fig. 4: Hierarchical View of Task Model: Energy Consumed and Execution Time

Our architecture model includes values such as V_{dd} and the other possible voltage levels as a factor of V_{dd}, such as $0.9V_{dd}$ and $0.8V_{dd}$.

$$\frac{V_i^2}{V_{dd}^2} = \frac{f_i}{f_{dd}} \tag{1}$$

$$P_{dyn}(V_{dd}) = C_{eff} V dd^2 f_{dd} \tag{2}$$

$$P_{dyn}(V_i) = P_{dd}(\frac{V_i^2}{V_{dd}^2})(\frac{f_i}{f_{dd}}) \tag{3}$$

$$T_i = T_{dd}(\frac{f_i}{f_{dd}}) \tag{4}$$

$$E_i = E_{dd}(\frac{V_i^2}{V_{dd}^2}) \tag{5}$$

2) Communication Model: For the current experiments, all communication is done via memory in the target architecture model. Sending and receiving data is done via writing and reading from memory. We assume that data at the sender side is already in cache and that the major cost is writing data from the receiver to the sender memory space. The time and energy cost of memory access to transfer data from one address space to another address space is modeled using the characteristics of the LPDDR2-800 DRAM. Power consumption of DRAM changes as its channel utilization changes [12]. We approximated the average speed of memory access as 3.5GB/s and energy consumption as 40pJ/bit for the data in this paper.

B. Task Models

One of the fundamental assumptions of our work is that tasks will arrive in a dynamic manner, and will be scheduled to execute on a certain processing element in a heterogeneous system. Therefore, we have constructed models that describe various characteristics of the tasks. Section III-B1 discusses the models of various runtime parameters while section III-B2 describes the inter-task dependencies that may appear in a dynamic scenario.

1) Task Description: The task description model for each possible task requires at a minimum the energy consumed per resource, the execution time of the task, and the size of the input and output data. Furthermore, a deadline that the task needs to complete execution by may be defined. Figure 4 shows the hierarchical view of a portion of the task model. In this example, a 1D FFT has multiple implementations that could execute on either a GPU, GPP, or FPGA. For each of those implementations, the energy consumed and execution time are defined. In this case there are five different designs. A runtime scheduler that receives the incoming tasks will use this data to make a determination on where the task should be scheduled for execution.

2) Task Dependencies: This section will detail the dependencies between tasks.

Each task may be dependent on the output of other tasks. In order to allow exploration on the effects dependencies may play in affecting the performance of our optimizations, a dependency model was defined. Figure 5 illustrates three types of dependencies we model: one-to-one, one-to-many, and many-to-one. The number of dependent tasks may be either deterministic or stochastic. Tasks may also be independent.

Figure 6 shows the task graph for the application scenario used to evaluate the optimizations in this work. Here, histogram equalization forms the root task from which all others are generated. When a histogram equalization task is completed, either a 2D FFT or Discrete Wavelet Transform (DWT) is executed next with probabilities of 0.4 and 0.6 respectively. When either the 2D FFT or DWT completes execution, a number of 2D convolution tasks are generated using a probability distribution function designated by n_1. The parameter n_1 for this scenario is a uniform distribution between one and ten tasks (mean = 5.5 tasks). The successor task after the 2D convolution is a 1D FFT, using a one-to-one relationship (all of the 2D convolution tasks lead to one 1D FFT task). The next step is to merge the data from each of the $n1$ 1D FFTs. This gather step is done by a task known as merge.

The next dependency is a one-to-many relation. After the merge step has gathered all the required data, it then generates n_2 2D FFT tasks. n_2 is a value that is dependent on the mode the system is currently in. There are three modes: HI/MD/LO. The maximum duration of each mode is five periods, ten periods, and 20 periods, respectively. The actual duration for a given instance of a mode is uniformly distributed between zero and the maximum duration. Each mode has a maximum level of parallelism, n_{max}. It is assumed that n_2 will vary between 0 and nmax. n_2 at a current time can increase, not change, or decrease based on the probabilities for each mode shown in the table I. These modes allow a relation to user inputs on the system.

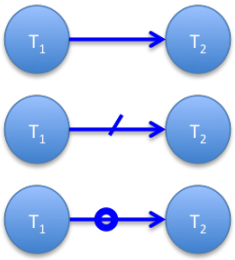

Fig. 5: Graph Notation for Task Dependencies

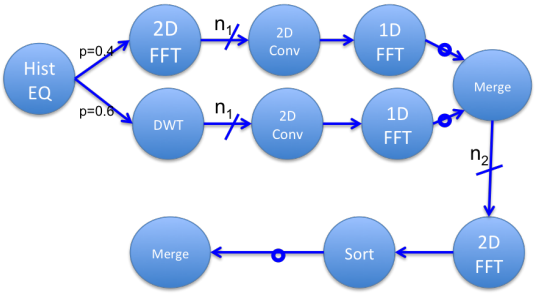

Fig. 6: Scenario Dependency Graph

TABLE I: n_2 Probability Distribution

Mode	P^+	P^0	P^-	n_{max}
HI	0.5	0.3	0.2	20
MD	0.3	0.5	0.2	8
LO	0.3	0.5	0.2	2

Each 2D FFT will then generate a sorting task after completion. Finally, the n_2 sorting task results will be gathered by another merge task.

In the tested scenario, the histogram equalization task has a period of 33ms, to correspond with the requirement of realtime systems to maintain a 30 frame per second throughput. Thus the deadline is 33ms after the task arrives. The deadline for each dependent task is $D_{parent} + 33ms$.

C. Performance Metrics

The metrics used to evaluate the performance of the optimizations are the energy consumed by the system executing the tasks and the number of missed deadlines. The goal is to minimize energy consumed and the number of missed deadlines. In our results, the primary metric reported is the energy efficiency ratio, defined as $\frac{EnergyConsumed_i}{EnergyConsumed_{greedy}}$. The relative standard deviation over the greedy is also reported, to verify that enough simulations have been run to account for the stochastic behavior of task dependencies.

IV. Optimizations

The following subsections describe the different algorithms that the runtime system uses to schedule tasks for execution. Once again, it is important to note that in a runtime system, knowledge of future tasks is unavailable. Therefore, all algorithms discussed here consider tasks that have arrived at the system.

A. Greedy

The greedy scheduling algorithm is the baseline against which the other scheduling algorithms are compared. This scheduler takes the earliest available task at the given system time and looks for the earliest available computational resource. If that resource is available, then the task is scheduled. If not, then the scheduler will wait until it becomes available before scheduling. When more than one resource is available at the next earliest time, the resource that consumes the least energy is selected. The greedy scheduler does not take any deadlines into consideration when scheduling tasks.

B. Time-Window(TW)

The time window scheduling algorithm manages a single global job queue and schedules the ready tasks. However, the scheduler may wait for a window of time for the most energy-efficient resource to become available instead of scheduling the task immediately on a resource that is available right away. The length of the time window is user configurable before simulation. If the most energy-efficient resource for the task becomes available within the time window, it will be given to the task. If not, the time-window scheduler defers scheduling of the task by as much as the time window length. If any event happens before the time-window expires, the scheduler tries to schedule the task again. If the task is not scheduled until the end of the time window, the scheduler schedules the task on the best available resource at that time. The time-window scheduler will try to enforce deadlines by considering the resource where the task can finish before deadline. When the deadline cannot be met, a resource where the task can finish earliest will be chosen.

C. Time-Window with DVFS

The time-window with DVFS scheduler will assign a task to a computational resource with a selected implementation in the same way as the time-window scheduler. The difference is that at schedule time, the scheduler will attempt to use the slowest frequency that will still meet the deadline of the task.

D. Time-Window with Local Queues

The TWS manages a single global job queue, and there are no local job queues for the resources. In the TWS scheduler, a task cannot be assigned to a resource unless it is currently available. The time-window scheduler with local queues has a local job queue for each computing element. The scheduling decision is made at the time of the arrival of a task. When a task is ready to be executed and has been scheduled, it will be assigned to a resource. If the resource is currently available, it will begin execution. If the resource is busy, then the task will be put into the local job queue of the processing element. Those tasks that have been assigned to a resource do not have to be further processed by the scheduler. In the original time-window scheduler, the scheduler is assumed to have no knowledge of the future available time of a computing element. Here we assume that the scheduler does know when a resource will become available based on the expected execution times of any currently executing tasks and those tasks currently in the resources local queue.

E. Time-Window with Local Queues and DVFS

The time-window scheduler with both queues and DVFS attempts to utilize optimizations that are possible when local queues and DVFS are enabled. The scheduler will determine the task and frequency-voltage parameters based on information about the availability of the resource and information in the local queue. It chooses the lowest energy-consuming resource and the corresponding frequency-voltage pair. Contrary to TW+DVFS, this scheduler will still assign the task even if the resource is busy. When the resource receives the task assignment, it can start executing, if possible, or put the task into a local queue for execution at a later time.

F. Runtime DVFS Adjustment (RA)

In the aforementioned algorithms where a local processing queue exists for each resource, tasks do not change the speed that is selected at schedule time. When the task arrives in a local processing queue, it may miss the deadline because of the tasks that are already in the job queue. Allowing runtime DVFS adjustment allows the scheduler to modify the frequency and voltage of previously scheduled tasks, if needed, to ensure that new tasks meet their deadlines. This technique can be enabled only when local processing queues exist.

V. Experiments and Results

A. Runtime Scheduler Simulator

We implemented a simulator to measure the performance of each scheduling algorithm. The simulator was constructed to be modular to allow future expansion. Figure 7 shows the input and output of the simulator. The task description contains information discussed in section III-B1, task dependencies such as discussed in section III-B2, and architecture description describes the system as in section III-A. The scenario description holds the occurrences of root nodes and independent tasks.

Figure 8 illustrates the modularity and function of the simulator. The scheduler object encompasses the algorithm and can be swapped out to allow for different optimizations. The system object contains one or more resources. Each resource contains information on the currently executing task and runtime parameters such as DVFS levels. Local queues for each resource may or may not be present depending on the configuration. Finally, a global event queue keeps track of when tasks arrive, are scheduled, or start and end execution.

B. Results

For each of the six algorithms, 10 runs were executed for each generation (28nm, 14nm, and 7nm). Below, we present the energy efficiency ratio and the standard deviations over the 10 runs. Experiments with 10000 periods of the histogram equalization kernels were evaluated and the results are shown here. The time window for TW scheduler is 0.015 seconds.

It can be seen that in each scenario, the TW with LQ and DVFS, and the TW with LQ, DVFS, and RA showed the best performance.

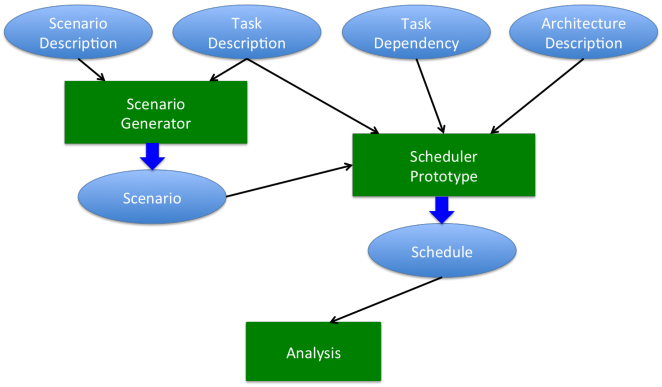

Fig. 7: Simulator Usage Diagram

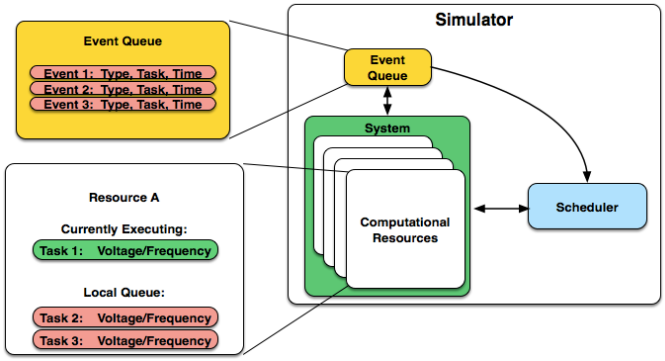

Fig. 8: Simulator Block Diagram

VI. Conclusion and Future Work

In this paper, a number of dynamic runtime optimizations that map tasks to heterogeneous architectures are proposed and evaluated. We used simulation and three different heterogeneous architecture models in conjunction with a dynamic task application scenario to evaluate the effectiveness of our dynamic optimizations. The simulations modeled task execution time, power consumption, and communication overhead. The simulation results show that dynamic optimization techniques

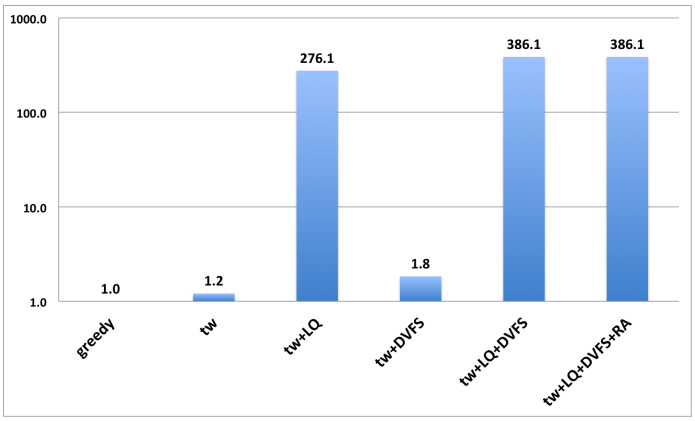

Fig. 9: Energy Efficiency Ratio (28nm, 10000 Periods)

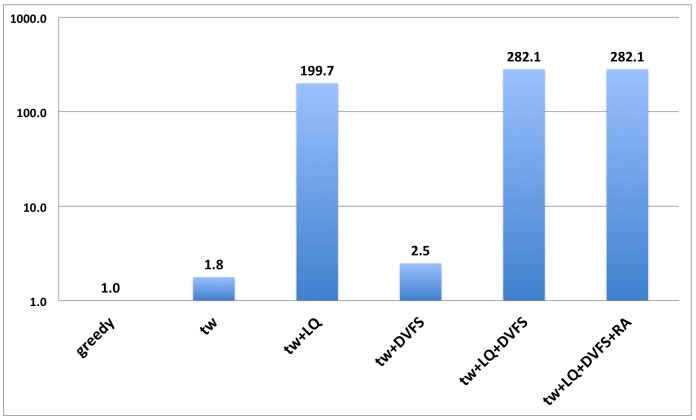

Fig. 10: Energy Efficiency Ratio (14nm, 10000 Periods)

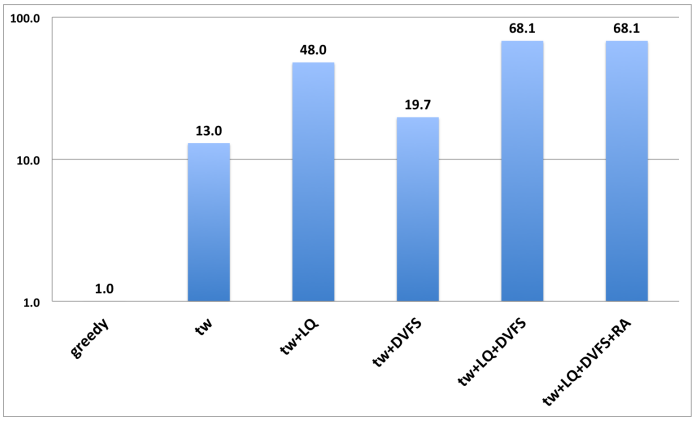

Fig. 11: Energy Efficiency Ratio (7nm, 10000 Periods)

have the potential to greatly improve energy efficiency, showing an improvement of 390x over a baseline greedy algorithm in the best case, while still satisfying task deadlines.

As seen in the results, the time-window with local queues and DVFS adjustment scheduling heuristic shows the greatest improvement over a baseline greedy scheduler. It can be noted that for this class of scenarios, the additional runtime adjustment of DVFS parameters by each computational node does not add value to the performance. Therefore, we can conclude that the complexity of implementing runtime adjustment by each node should not be added for this class. However, that may not be the case with other classes of scenarios, and will be explored in future work. Future work also includes experimenting with other time window values relative to the period of input tasks. The work may also be expanded to include running experiments on real hardware heterogeneous systems versus simulation. We also plan to explore other heuristics while also expanding upon the task and architecture models. An example is exploring a more complicated communication model, where direct communication between computational nodes is possible. Additionally, we will take into account distance and placement of nodes in the communication model. Lastly, we would like to explore other metrics that have been used by other work, such as response time for tasks or robustness in meeting the constraints.

TABLE II: Relative Standard Deviations for 1000 Periods

	28nm	14nm	7nm
Greedy	0.00%	0.00%	0.00%
TW	0.99%	1.62%	0.90%
TW+LQ	1.52%	2.69%	3.69%
TW+DVFS	0.89%	1.34%	0.90%
TW+LQ+DVFS	1.50%	2.68%	3.66%
TW+LQ+DVFS+RA	1.50%	2.68%	3.66%

TABLE III: Relative Standard Deviations for 10000 Periods

	28nm	14nm	7nm
Greedy	0.00%	0.00%	0.00%
TW	0.21%	0.48%	0.32%
TW+LQ	0.40%	0.71%	1.01%
TW+DVFS	0.20%	0.45%	0.30%
TW+LQ+DVFS	0.40%	0.71%	1.00%
TW+LQ+DVFS+RA	0.40%	0.71%	1.00%

REFERENCES

[1] C. Rusu, R. Melhem, and D. Mosse, "Maximizing the system value while satisfying time and energy constraints," in *23rd IEEE Real-Time Systems Symposium*, 2002.

[2] C. Yang, J. Chen, and T. Luo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *the Design, Automation and Test in Europe Conference and Exhibition (DATE05)*, 2005, pp. 468–473.

[3] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy-efficient scheduling of real-time tasks on multicore processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1540–1552, 2008.

[4] Z. Shao, Q. Zhuge, C. Xue, and E. H.-M. Sha, "Efficient assignment and scheduling for heterogeneous dsp systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 516–525, June 2005.

[5] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints," *IEEE Transactions on Emerging Topics in Computing*, 2014.

[6] D. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, 2014.

[7] B. Young, J. Apodaca, L. Briceno, J. Smith, S. Pasricha, A. Maciejewski, H. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou, "Energy-constrained dynamic resource allocation in a heterogeneous computing environment," in *40th International Conference on Parallel Processing Workshops*, 2011.

[8] H. Esmaelizadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011.

[9] NVIDIA, "Tegra k1 whitepaper." [Online]. Available: http://www.nvidia.com/content/PDF/tegra_white_papers/Tegra_K1_whitepaper_v1.0.pdf

[10] T. Sakurai and A. Newton, "Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, Apr 1990.

[11] S. Park, J. Park, ..., M. Pedram, and N. Change, "Accurate modeling of the delay and energy overheard of dynamic voltage and frequency scaling in modern microprocessors," *IEEE Tran. On Computer-Aided Design*, vol. 32, no. 5, May 2013.

[12] K. Malladi, F. Notahft, K. Periyathambi, B. Lee, C. Kyzyrakis, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile dram," *ACM SIGARCH Computer Architecture News*, vol. 40.3, pp. 37–48, 2012.