Power-aware Work Stealing in Homogeneous Multicore Systems

Shwetha Shankar
Intel Corporation
Austin, Texas, USA
shwetha.shankar@intel.com

Greg LaKomski, Claudia Alvarado, Richard Hay,
Christopher Hyatt, Dan Tamir, and Apan Qasem
Department of Computer Science,
Texas State University,
San Marcos, Texas, USA
{gl082,ca1015,rh1375,ch1662,dt19,apan}@txstate.edu

Abstract— **Excessive power consumption affects the reliability of cores, requires expensive cooling mechanisms, reduces battery lifetime, and causes extensive damage to the device. Hence, managing the power consumption and performance of cores is an important aspect of chip design. This research aims to achieve efficient multicore power monitoring and control via operating system based power-aware task scheduling. The main objectives of power-aware scheduling are: 1) lowering core's power consumption level, 2) maintaining the system within an allowable power envelope, and 3) balancing the power consumption across cores; without significant impact on time performance. In previous research we have explored power-aware task scheduling at the single core level referred to as intra-core scheduling. This paper reports on a research on a power-aware form of inter-core scheduling policy referred to as work stealing. Work stealing is a special case of task migration, where a "starving" core attempts to steal tasks from a "victim", i.e., a "loaded" core. We have performed experiments with ten variants of the work stealing that consider both the power and the performance attributes of the system in the process of selecting a victim core. The experiments conducted show that the power-aware inter-core stealing policies have high potential for power efficient task scheduling with tolerable effect on performance.**

Keywords-task scheduling; task migration; work stealing; power-aware task scheduling; energy delay product

I. INTRODUCTION

Power consumption is a dominant obstacle for performance improvements in the very large scale integration (VLSI) technology. Excessive power consumption affects the reliability of cores. High power dissipation results in high heat generation. This in turn, requires costly cooling mechanisms, affects battery lifetime, and causes damage to semi-conductor devices. Hence monitoring the power consumption is of high importance in the semiconductor industry [1]-[5]. This study aims to address power management issues by concentrating on scheduling techniques available at the Operating System (OS) level [6]-[10].

Task scheduling in a multicore system is composed of three components: task matching, intra-core task scheduling, and inter-core task migration. Task matching is the assignment of new tasks to cores. Intra-core task scheduling policies concentrate on selecting a ready task for execution at the single core level while inter-core task migration policies focus on moving ready tasks between cores. Work stealing, a specific type of task migration, is a multicore scheduling algorithm that can improve performance and achieve efficient dynamic load-balancing [3][5]. In the classical work-stealing environment, cores that are executing tasks are referred to as workers while idle cores are potential thieves (or stealers). Depending on the state (working or idle) cores make choices with regard to available tasks. Each worker must choose the next task to be executed. If an idle core becomes a thief, it must choose the victim core and the task to steal [3]-[5][12]-[19].

Maintaining a homogeneous multicore system within an allowable power envelope and/or balancing the power consumption across cores without drastically affecting performance are the main problems addressed in this paper. The main objective is to devise an efficient power-aware multicore OS task scheduler so that both execution and power consumption of the task are taken into consideration. In addition, this study aims to find mechanisms to lower a core's power consumption and support hot-spot elimination. These objectives are achieved by integrating power characteristics into inter-core work stealing policies.

There is significant amount of research on scheduling algorithms involving execution time as the optimization criteria, focusing on real-time applications, and interacting with hardware [1][2][9]-[11]. However, research on power-aware task scheduling strategies that focus on power consumption issues and integrate power and performance metrics in the scheduling optimization criteria has considerable opportunities for extension. This study incorporates both execution time and power considerations into the OS based task scheduling on homogeneous multicore systems.

The main contribution of this study is the introduction of power efficient inter-core work stealing policies that significantly reduce the energy consumption variance across cores and produce a noticeable improvement in the completion time for different workload scenarios.

This paper is organized in the following way. Section II provides a brief overview of relevant background information. Section III provides details concerning work stealing mechanisms and Section IV includes a review of literature related to research conducted. The literature survey shows that significant research is yet to be done and provokes studies seeking cost-effective power efficient OS task scheduling policies for single and multicore systems. Section V provides details on the experimental setup used to evaluate the devised power efficient policies. Section VI presents details of the set

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-339-1

of experiments conducted. Section VII includes evaluation of the experimental results. Finally, Section VIII provides conclusions and proposals for future research.

II. TASK SCHEDULING

In general, task scheduling in multicore systems is done by the OS. On the other hand power management, mainly through scaling of the frequency of cores via Dynamic Voltage and Frequency Scaling (DVFS), is done by firmware. Moreover, there is a significant amount of information concerning execution parameters and power performance that is readily available at the firmware-level, but is not readily available to the OS. Hence, there is semantic gap between the OS and the firmware. Figure 1 shows an ideal situation where the OS and the firmware have extensive hand-shaking utilized for power-aware scheduling and management by the OS. The hand shaking enables the OS to change the working frequency and states of cores. Furthermore, firmware-level execution information is used by the OS for improved scheduling decisions.

Many of the terms related to task matching and scheduling are overloaded. To remove ambiguity, we define the following terms [5]: Task Matching is the process of assigning incoming tasks to processing cores in order to optimize a given metric such as throughput (other terms for this operation are: task scheduling, task mapping, and task distribution). Intra-core task scheduling refers to the scheduling of tasks assigned to a core on that core. Task migration means moving tasks from the ready queue of one core to the ready queue of another core (e.g., work stealing). Often, task migration is referred to as task redistribution.

The Energy Delay Product (EDP) is a performance measure that takes execution time and power into account. The EDP is defined to be $EDP = T \times E = T^2 \times P$, where T denotes the execution time of a task, E is the energy, and P is the average power consumed by the task throughout the execution [1]-[5].

The Highest Response Ratio Next (HRRN) is an intra-core scheduling measure that ranks tasks based on the equation $HRRN = \frac{w+s}{s}$. In this formula, s denotes the remaining task service time, and w is defined to be the amount of time the specific task has been waiting in any system queue.

The Highest Energy-delay-product based Cost Function (HECN) is a power-aware heuristic developed by our research team. It integrates the HRRN scheduling policy and EDP into the scheduling selection criteria. Several versions of the heuristics are used in our experiments. In this research we use the following version: $HECN = \frac{w+EDP}{EDP} = \frac{w+s^2P}{s^2P}$. Hence, the remaining EDP replaces the remaining execution time. Although, in our general matching, scheduling, and task migration framework, HECN is also used for intra-core task scheduling and task matching, this paper concentrates on task migration via work stealing. Task matching and task scheduling are not elaborated upon further. The reader is referred to [5] for further details on these topics. The HECN heuristics involves elements of different physical dimensions. Nevertheless, being a heuristic, this does not pose an issue.

Figure 2 shows the research and simulation framework through a snapshot of an exploratory simulator developed in this research. Figure 3 shows potential patterns of task generation (arrival). These topics are detailed in Section IV. The main components of the framework are described here.

A. Task Matching

Optimal task matching is an assignment of newly generated tasks to available cores that optimizes a cost function such as performance, power consumption, and thermal envelope [1][2][4]. Task matching is an NP complete problem [4]. Hence, numerous heuristics have been developed for finding a sub-optimal solution for the problem. Generally, these heuristics are referred to as matching polices. This topic is out of the scope of the current paper.

B. Intra-core Task Scheduling

Intra-core task scheduling is a scheduling component of single core and multicore systems. The process involves selecting the next task to be executed on a core from the current tasks allocate to that core. Numerous methods addressing different scenarios in preemptive and non preemtive operating systems, including round robin, first come first serve, and HRRN have been explored and implemented [1]-[3][5][9]-[13]. In previous research we have identified the HRRN and its power-aware heuristic variant HECN as the most promising approach for intra-core scheduling [3][5]. Again, this topic is out of the scope of the current paper which concentrates on task migration. Nevertheless, in our simulations, it is assumed that HECN is used for intra-core scheduling. Additionally, HRRN and HECN are used in work stealing decisions.

C. Inter-core Task Migration

Task migration occurs if the system is in extreme imbalance and certain cores experience an extremely high peak in a given parameter while other cores experience an extremely low peak in that parameter.

Under task migration policies, tasks are reallocated to cores. Classification of task migration policies includes 5 main parameters: 1) the trigger for reallocation, 2) the reallocation source core (or cores), 3) the destination core(s), 4) task selection criteria (which affects the set of tasks that are candidates for reallocation) and the set of tasks that are eventually migrated, and 5) the amount of available knowledge concerning the system's state. Work stealing, detailed in Section III, is a special form of task migration. In this case, the trigger for reallocation is the starvation of one or more cores, the source cores are cores that are considered to be loaded (see Section III) and the destination cores are the starving cores. Several core/task selection policies can be considered.

D. Core State Control

The firmware can control the state of a core and place it in several sleep-modes. Additionally, the firmware can control the frequency of cores. The current trend is to equip the OS with this type of control capabilities and this is reflected in our simulation framework depicted in Figure 2.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-339-1

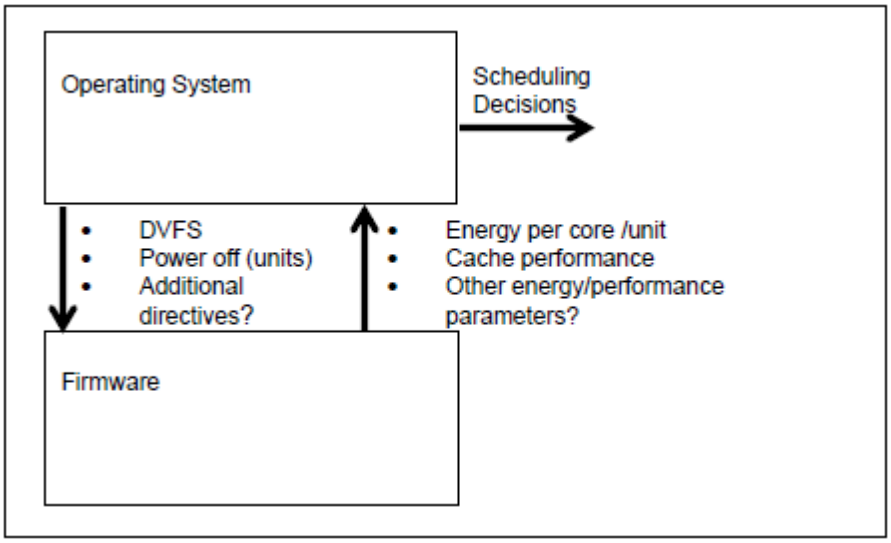

Figure 1. Desired OS and hardware interaction.

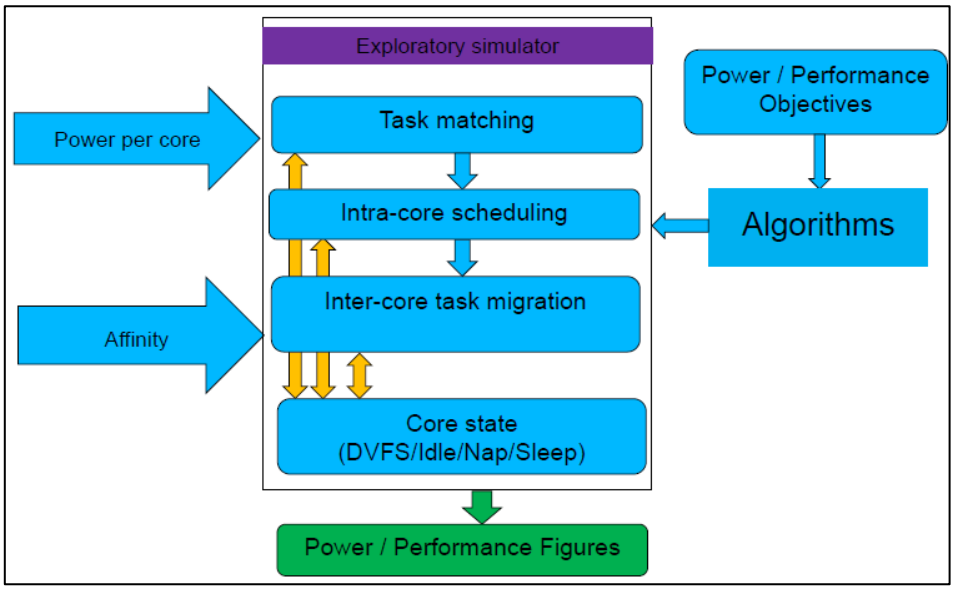

Figure 2. A snapshot of the research and simulation framework developed in this project.

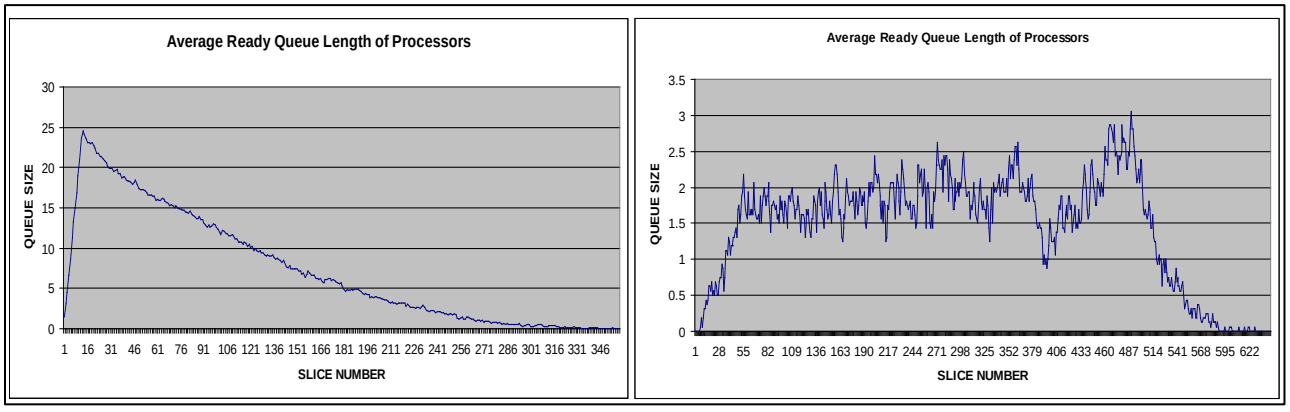

Figure 3. Task arrival modes (a) early saturation mode (b) Steady state task arrival modes

III. POWER-AWARE WORK STEALING

In a multicore system, task migration refers to moving tasks from the ready queue of one core to the ready queue of another core in order to improve performance metrics. This paper concentrates on one type of migration referred to as work stealing [3].

Cores that experience extreme (high or low) imbalance values of a given parameter might initiate a task migration transaction. In this study, the ready queue size is considered as the parameter that indicates imbalance. A core is considered as starved if the number of tasks in its ready queue falls below a threshold T_s. On the other hand, a core is considered as loaded if the number of tasks in the ready queue is above a threshold T_l. A core is considered as normal if it is neither starving nor loaded. This type of core does not participate in work stealing.

A starving core is a potential stealer and a loaded core is a potential victim of stealing. A stealer initiates the stealing process by seeking a victim. The stealer identifies a victim. The victim volunteers a task to be stolen. The stealer steals this task by migrating it to its own ready queue. This process is referred to as work stealing. In a homogeneous multicore system, there is no architecture difference between cores. Hence, all the stealers and all the potential victims can only be distinguished based on execution parameters and not on architecture parameters. This research report concentrates on homogeneous multicore systems.

The process of work stealing involves three steps. The first step is identifying starving and loaded cores. Next, a specific victim has to be selected from the loaded cores. Finally, a specific task to be migrated from the victim to the stealer has to be identified. There are numerous variants and options related to each of these steps. One consideration is the amount of knowledge available to cores. Under the local knowledge model, each core is only aware of its own current status [3][5]. In the global knowledge model, there is an entity (e.g., the OS) that has and utilizes knowledge about the status of each core [15]-[20]. The selection of the victim core and the migrated task can be done in a way that optimizes performance objectives. For example, if there is more than one potential victim, the OS might choose the most loaded core as the victim and the task with the highest wait time in the ready queue of that core as the task for migration.

Traditionally, work stealing has been applied under performance optimization criterion. For example, the stealing decisions (choosing the victim core and the task to be migrated) might attempt to optimize wall to wall time of an entire workload. In this research, the stealing decisions incorporate power and performance objectives. Three main objectives were considered 1) lowering a core's power consumption level, 2) maintaining the system within an allowable power envelope, and 3) balancing the power consumption across cores; without significant impact on time performance. In each set of experiments, one or more of these goals was used in the process of victim and migrated task selection. For example, in order to achieve balance in power consumption, the loaded core that has consumed the most amount of energy in the last K time slices is most likely to be selected as one of the victims. Additional considerations include the amount of global knowledge assumed, affinity between tasks and cores (e.g., due to recent use of the core cache), and the power consumption characteristics of the tasks in the ready queue of potential victims.

IV. LITERATURE REVIEW

This section discusses the relevant research available on single and multicore task scheduling policies that consider the energy consumption of cores.

Kashif et al. and Kim et al. propose a Priority-based Multi-level Feedback Queue Scheduler (PMLFQS) for mobile devices [11][12]. Their papers, however, focus on the firmware role rather than the OS role in power management.

Wu et al. propose Low Thermal Early Deadline First (LTEDF), a temperature-aware task scheduling algorithm for real-time multicore systems [13]. If cores are thermally saturated, task migration is performed to alleviate the saturation. The paper is focused on real-time systems and on lowering the peak power and temperature consumptions. Our study, however, concentrates on general applications. Moreover, rather than limiting the consideration to peak power, this research considers balancing the power consumption across cores in the system.

Zhou et al. propose an algorithm referred to as THRESHHOT [14]. At each step, THRESHHOT selects the hottest task that does not exceed the thermal threshold using an online temperature estimator, leveraging the performance counter-based power estimation. The paper, however, focuses on batch processes on a single core and is intended to lower final core temperature. Our study aims to consider varying type of processes (beyond batch processes) on a multicore system.

Quintin et al. detail the Classic Work Stealing Algorithm. [15]. In addition, they propose the idea of grouping cores as *Leaders* or *Slaves* and restricting the stealing according to the grouping. In our research, stealing policies are devised for a homogeneous system such that all cores (that have load imbalance) can participate in stealing with the help of one efficient central unit.

Guo et al. propose two policies that fit high granular parallel processing environment [16]. Our work aims at developing power-aware policies for all types of workload including high and low granularity parallelism workloads.

Agarwal et al. propose a Central Task Scheduler that can maintain information of all the cores in the system [17]. Sudarshan et al. discuss a similar policy that mainly consists of a dispatcher and nodes [18]. Our research considers

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-339-1

several levels of information sharing from local to global knowledge sharing.

Robison et al. propose considering task to core affinity as a part of the task matching. They use a "Mailbox" and FIFO mechanism to handle the affinity [19]. Our research involves affinity at all levels of scheduling, not only at the matching stage.

Faxén et al. suggest Sampling Victim selection where a thief samples several potential victims and selects the one with the task that is closest to the root of the computation [20]. In addition, they propose a Set Based Victim selection where each thief only attempts to steal from a subset of the other workers. We have implemented their methods in addition to other policies reported below.

V. EXPERIMENTAL SETUP

An exploratory software simulator (see Figure 2) is developed to rapidly assess the utility of different matching and scheduling procedures. The simulator is a time based simulator which uses two important "atomic" time units. The operating system atomic unit is referred to as a slice. A typical slice time is 1 – 30 milliseconds. Scheduling decisions are performed on a slice boundary. In addition, the simulator employs an atomic time unit referred to as a tick. System updates occur on a tick boundary. To mimic a realistic scenario we assume that a slice time is 20 milliseconds; we further assume that a tick represents 100 microseconds hence there are 200 ticks per slice. Other configurations have been used as well.

The simulator can be easily altered to evaluate a number of different configurations and task parameters. The overall system environment is equally flexible. Variables like the number of cores, power consumption per core, core frequencies, idle power consumption, slice time (in ticks), intra-core scheduling algorithms, stealing policies, and termination conditions can all be changed for individual experimental runs. Additional parameters include: 1) thresholds for the starvation/loaded status, 2) workload size, 3) task arrival rate, which in general follows a Poisson distribution, 4) task serving time, which in general follows an exponential distribution, and 5) task power consumption per tick, which is assumed to have a uniform distribution. These parameters have been selected based on discussion with experts from leading chip design companies.

The simulations are performed for all the formulated stealing policies. Each simulation is repeated several times with different random number generation seeds. Every simulation provides performance figures on a slice time basis for all the cores. Data is gathered on slice boundaries for each simulation of each stealing policy.

We report on two sets of experiments: Experiment 1: task scheduling with high initial rate of task generation, and Experiment 2: task scheduling with a steady arrival rate. The first scenario is typical of highly parallel loads, where in the first steps of computation many tasks are being generated. Initially, the system is saturated with new tasks, but after a while the system completes the processing of all the tasks in the current load. We refer to this scenario as the parallel workload scenario. The second mode is typical to communication and networking scenarios where tasks are generated at a more or less fixed rate and the system is usually in a steady state where the rate of processing tasks is about the same as the rate of task generation. This is referred to as the steady state scenario. Figure 3 illustrates the two scenarios via the size of the ready queue per slice time.

As noted, the work stealing procedure requires identifying a potential victim and selecting a task to be migrated. For the victim selection we have taken into account the amount of global knowledge, the set of potential victims, and the specific power performance goal. In terms of selecting the migrated task, it makes sense to assume that a victim core would like to volunteer its "worst" task as the task to be migrated. In this sense, we have identified HRRN as the most promising criteria for power agnostic work stealing. The victim is volunteering the task with the minimal HRRN for stealing. The HECN which takes into account EDP rather than expected execution time has been used as the basis for selecting the task to be migrated for the power-aware policies. In this case, the victim is volunteering the task with the minimum HECN as the task to be stolen.

A. *Stealling Policies, Legend and Avbbreviations.*

The following legend is used in the text and figures for the Power-aware (PAW) variants of the work stealing policies where HECN is used to determine the task to be volunteered by the victim core.

Random_MinHECN_Task; the stealer chooses a random core as a potential victim without knowledge of the core's load. If that randomly chosen core is not loaded, then no stealing occurs. Otherwise, this victim core volunteers a task with the lowest HECN.

MaxLoaded_MinHECN_Task; the stealer identifies a loaded core with the largest ready queue as a victim. This victim core volunteers a task with the lowest HECN.

MaxMin_ HECN_Task; each loaded core (a potential victim) volunteers a task with lowest HECN. The stealer considers the tasks volunteered by all potential victims and finds a task with the highest HECN among all volunteered tasks. Hence the name MaxMin, implies that the MaxHECN task is selected from the available MinHECN tasks.

MaxRemainingService_MinHECN_Task; the service time of tasks remaining in the ready queue can be used to estimate the remaining core execution time and the energy that might be consumed. Therefore, the stealer picks the core with a ready queue that has the maximum remaining task service or execution time. The victim core volunteers a task with the lowest HECN.

MaxRemainingEnergy; the energy of tasks remaining in the ready queue indicates the energy that the core might

consume. Hence, the stealer selects the core with a ready queue that has the maximum remaining task energy. In this case the victim has two options for volunteering tasks: 1) **MinHECN_Task;** the victim core volunteers a task with the lowest HECN. 2) **MaxEnergyTask;** the victim core volunteers a task with maximum energy.

MaxEnergyInLastKSlices; the stealer chooses a core that has consumed the maximum amount of energy in the last k slices of the simulation. Again, can choose between the: **MinHECN_Task** or the **MaxEnergyTask.**

MaxEnergyConsumed; the stealer opts for a core that has consumed the maximum energy so far in the simulation, and the victim has the same two options as in the previous policies: **MinHECN_Task or MaxEnergyTask**.

The PAG version of the above inter-core work stealing policies uses the HRRN ratio in place of the HECN to determine the task to volunteer.

VI. EXPERIMENTS AND RESULTS

This section reports the two types of experiments with work stealing conducted as part of this study and provides the results of these experiments.

A. *Experiment 1 - Multicore Task Scheduling for a Parallel Workload Scenario.*

In this set of experiments, a fixed workload simulation is performed in a system having a fast task arrival rate (parallel workload). These experiments are intended to study the behavior of the formulated policies and identify the policy that performs the best under this specific scenario. The four main performance figures provided from this experiment are the energy consumption variance, the average turnaround time, the peak ready-queue length, and the completion time of all the policies. The parallel workload scenario is depicted in Figure 3(a). According to the figure, the ready queue length is rapidly increasing in the first few time slices of the simulation and then gradually decreasing as the simulation progresses. Figure 4 shows the cores' energy consumption variance. This is used as an indicator of load balancing. It can be observed that, work stealing provides a reduction of about 18% in variance compared to PAG_NoSteal policy. The PAW_MaxMin_HECN_Task is the best stealing policy. The power-aware policies provide a marginally better power performance than the power agnostic method.

Figure 5 displays the turnaround time. In this case, the PAW_NoSteal policy has a lower turnaround time than PAG_NoSteal policy. This implies that power-aware intra-core task scheduling, without any stealing, lowers turnaround time by about 4%. By including stealing, the PAW_MaxMin_ HECN_Task is the best stealing policy and it improves (reduces) turnaround time further by approximately 31% compared to PAG_NoSteal policy. This shows that in the process of trying to gain power efficiency, the time factor is improved as well. This can be due to the fact that the EDP metric used in the selection criteria considers time along with power attributes. Again, power-aware is slightly better than power agnostic.

An experiment that measured the maximal size of the ready queues in each simulation of each stealing procedure has shown that the ready queues had reasonable sizes (up to 40 tasks per queue). Another experiment performed measured the task completion time. In this case, PAW_NoSteal policy increases the total completion by about 3.5%. This can be attributed to the fact that power-aware scheduling may increase task wait time and there is no stealing to help reduce wait time. On the other hand, stealing significantly reduces the completion time with PAW_MaxMin_ HECN_Task policy being the best stealer as it reduces the completion time by about 17%. Further experimental results are reported in [5].

From all of the results of this experiment, it can be seen that the PAW_MaxMin_ HECN_Task is the best stealing policy for a fast task arrival rate scenario. It significantly improves three important metrics, namely, energy consumption variance, turnaround time, and completion time.

B. *Experiment 2 - Multicore Task Scheduling for a Steady State Workload Scenario.*

For this test, a fixed workload simulation is performed in a system having a moderate task arrival rate. This emulates a steady state workload scenario as illustrated in Figure 3(b). In the first few time slices of the simulation, the ready queue length gradually increases. Then as the simulation progresses, the queue length remains steady for several slices thereby simulating a steady state workload scenario.

Figure 6 shows the cores' energy consumption variance. It is noticed that PAW_NoSteal policy performs slightly better than PAG_NoSteal policy by lowering the energy consumption variance by about 2%. By including stealing, the PAW_MaxEnergyInKSlices_MaxEnergyTask is seen as the best power-aware stealing policy. This policy further reduces the variance by 5% compared to PAG_NoSteal policy.

The PAG_MaxEnergyConsumed_MinHRRN work stealing policy provides a marginally better power performance than the PAW_MaxEnergyInKSlices_MaxEnergyTask method but it is not considered significant since it does not perform as well for the turnaround time metric discussed next.

Figure 7 displays the turnaround time. Again, PAW_NoSteal policy is better than PAG_NoSteal policy by almost 13%. The power-aware intra-core task scheduling coupled with inter-core work stealing further improves the turnaround time. The policy PAW_MaxEnergyInKSlices_MaxEnergyTask is again the best stealing policy with approximately 17% lower turnaround time compared to PAG_NoSteal policy. The power-aware policies are noticeably better than the power agnostic policies.

As in the case of parallel load, an experiment that measured the maximal size of the ready queues in each

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-339-1

simulation of each stealing procedure has been performed. The result shows that the ready queues had reasonable sizes (up to 20 tasks per queue). Another experiment performed measured the task completion time. In this case, an important difference was noted compared to the previous experiment. The PAW_NoSteal policy is better than PAG_NoSteal policy with a 3% lower completion Furthermore, the best power-aware stealer of this experiment is again the PAW_MaxEnergyInKSlices_MaxEnergyTask policy with about 8% reduction in completion time compared to PAG_NoSteal policy.

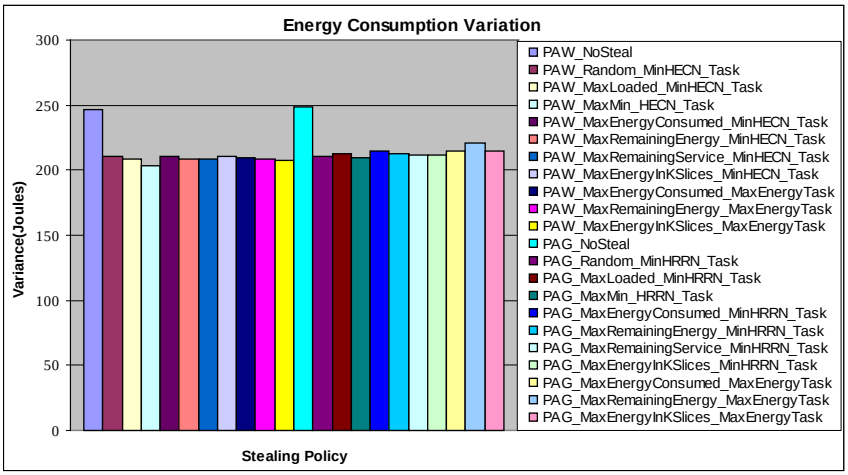

Figure 4. Energy Consumption Variance of the parallel load experiment

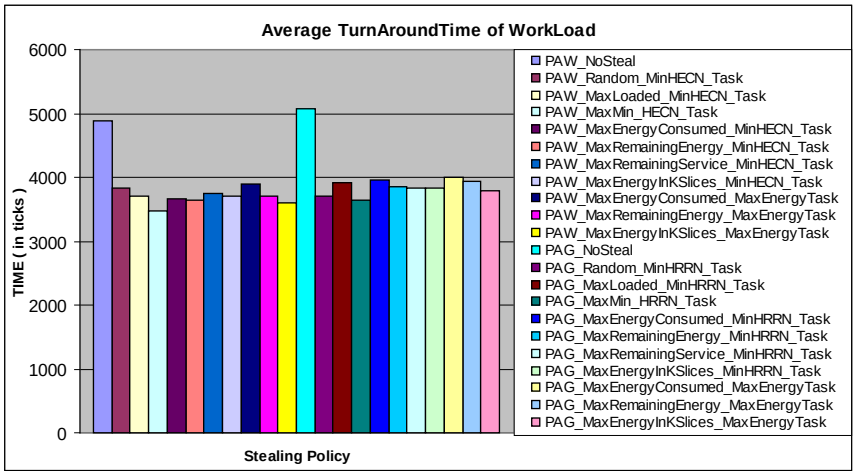

Figure 5. Average Turnaround time of the parallel load experiment

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-339-1

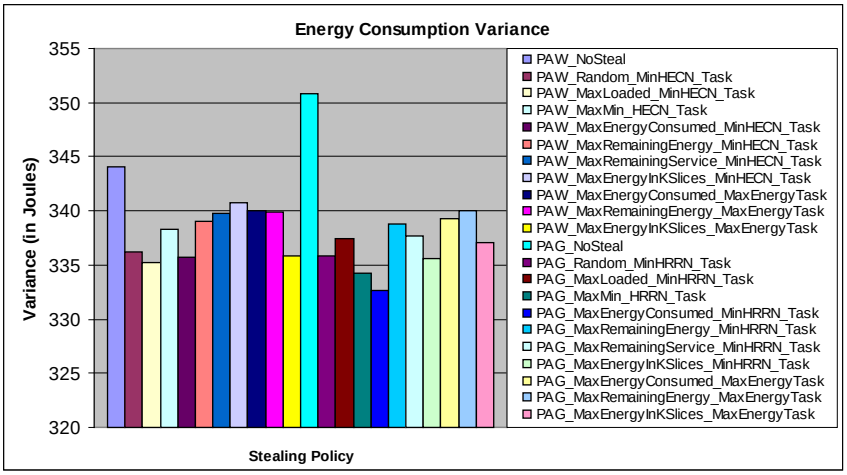

Figure 6. Energy Consumption Variance of the steady state load experiment

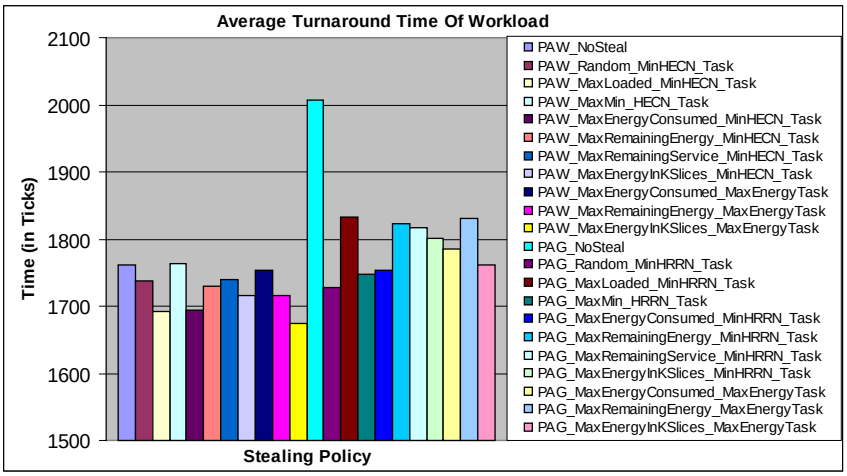

Figure 7. Average Turnaround time of the steady state load experiment

The PAG_MaxMin_HRRN_Task policy shows slightly better completion time compared to the PAW_MaxEnergyInKSlices_MaxEnergyTask policy, but it has a disadvantage since it fails to be the best in terms of efficiency in the energy consumption variance and turnaround time metrics.

VII. RESULT EVALUATION

According to the data gathered, in every experiment, a power-aware policy emerges as the policy that successfully reduces energy consumption variance, turnaround time and completion time. In addition to accomplishing energy efficiency, the performance time has been improved as well. The PAW_MaxMin_ HECN_Task policy shows the highest potential with 18% reduction in energy consumption variance, 31% improvement in turnaround time, and 17% more efficiency in completion time compared to the PAG_NoSteal policy.

Furthermore, the key points noted from the combined results of all the experiments are as follows.

1. The PAW_MaxMin_ HECN_Task policy emerges as the best policy in Experiment 1. The reason for this might be because the MaxMin policy is the only policy that directly selects a task to steal by choosing the least power consuming task among the high power consuming tasks of all potential victim cores. All the other stealing policies, first select a potential victim core and then select a task from that chosen core. Therefore, a stealing policy that considers all the tasks in the system such as the MaxMin policy outperforms other policies.
2. PAW_MaxEnergyConsumedInKSlices_MaxEnergy_ Task policy is the most efficient policy in Experiment 2. This can be best explained by the following analysis. Excluding the MaxMin policy, all the stealing policies

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-339-1

first consider the power properties related to a core to determine a victim. Most properties are related to the number of tasks or type of tasks in the ready queue but only two of the policies consider the history of the core, namely, the MaxEnergyConsumedInKSlices policy which uses recent past data and the MaxEnergyConsumed policy which uses all the past data. Hence, a policy that considers properties related to the recent history of potential victim cores might have advantage over policies that ignore this information.

3. Based on the experiment results, the PAW_MaxMin_HECN_Task procedure is the policy with the overall most potential for power efficiency even if the task arrival rate is unknown. It is observed that this policy performs the best for cases with fast task arrival rate and also performs reasonably well in situations with steady task arrival rate.

4. In all the experiments, there is no significant difference in performance amongst many of the stealing policies. This could be attributed to the fact that the variations in work stealing are very minute and have subtle differences.

5. The turnaround time is improved much more than the power efficiency level in all the experiments. This implies that the EDP metric integrated into the HECN policy might be giving more consideration to the task time rather than to the task power.

VIII. CONCLUSIONS AND PROPOSALS FOR FURTHER RESEARCH

The primary goal of this research work is to develop efficient power-aware work stealing policies. In an attempt to achieve the desired goal, the following steps have been implemented. First, based on previous research [3][5], we have selected the HRRN as the benchmark for intra-core task scheduling and the derived HECN cost function that extends the HRRN policy to include power characteristics of tasks in the system.

Next, building on the new intra-core HECN policy, various inter-core work stealing policies have been explored. Several different power-aware variations of work stealing that consider power features of the cores and its tasks before identifying the task to steal have been formulated. Finally, an in-house exploratory simulator has been developed solely to evaluate the potential of the policies devised.

Extensive multicore experiments with work stealing policies have been performed. The outcome suggests that several power-aware policies have promising results where power efficiency is being attained along with a minimal effect on performance.

We plan to extend the reported research in several ways: first, we plan to examine the utility of additional heuristic evaluation functions. Next, we plan to consider affinity between tasks and cores (e.g., due to recent use of cache) in the stealing decisions. Additionally, we plan to connect our simulator to a vendor multicore board and use parameters of power and performance available at the hardware/firmware in the process of making scheduling decisions (as per the model depicted in Figure 1). This will enable fast and realistic exploratory simulations. Finally, we plan to incorporate DVFS policies and changing core states (e.g., shutting down cores) as a part of the scheduling decisions.

ACKNOWLEDGMENT

This research was partially supported by a grant from the Semiconductor Research Consortium. Additionally, this work was partially supported by the NSF grants: NSF CNS-1253292 and NSF CNS-1305302.

REFERENCES

[1] A. Merkel and F. Bellosa, “Balancing Power Consumption In Multiprocessor Systems,” in Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, 2006, pp. 403-414.

[2] A. K. Coskun, R. Strong, D. M. Tullsen, and T.S. Rosing, “Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors,” in Proceedings Of The Eleventh International Joint Conference on Measurement and Modeling of Computer Systems, 2009, pp. 169-180.

[3] S. Shankar, D. E. Tamir, and A. Qasem, “Towards an OS-centric Framework for Energy-Efficient Scheduling of Parallel Workloads,” PPTDA-2013, the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, July, 2013, pp. 21-28.

[4] C. Hyatt, G. LaKomski, C. Alvarado, R. Hay, A. Qasem, and D. E. Tamir, “Power-aware Task Matching and Migration in Heterogeneous Processing Environments,” the 2014 International Conference on Computational Science and Computational Intelligence, Las Vegas, US, 2014, pp. 3-8.

[5] S. Shankar, Power-aware Task Scheduling on Multicore Systems – Thesis Report, Texas State University, Computer Science, December, 2012.

[6] A. Silberschatz, P. B. Galvin, and G. Gagne, “CPU Scheduling,” in Operating System Concepts,8th ed.,John Wiley and Sons, 2008, pp.183-223.

[7] D. Tam, R. Azimi, and M. Stumm, “Thread Clustering: Sharing-Aware Scheduling On SMP-CMP-SMT Multiprocessors,” in Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, 2007, pp. 47-58.

[8] S. Boyd-Wickizer, M. F. Kaashoek, and R. Morris, “Reinventing Scheduling For Multicore Systems,” in Proceedings of the 12th Conference on Hot topics in Operating Systems, 2009, pp. 21-21.

[9] M. Rajagopalan, B. T. Lewis, and T. A. Anderson, “Thread Scheduling For Multicore Platforms,” in Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems, 2007, pp. 35-44.

[10] J. Donald and M. Martonosi, “Techniques for Multicore Thermal Management: Classification and New Exploration,” in Proceedings of the 33rd International Symposium on Computer Architecture, 2006, pp.78-88.

[11] M. Kashif, T. Helmy, and E. El-Sebakhy, “A Priority-Based MLFQ Scheduler for CPU Power Saving,” in

Proceedings of the IEEE International Conference on Computer Systems and Applications, 2006, pp.130-134.

[12] K. H. Kim, R. Buyya, and J. Kim, "Power-aware Scheduling Of Bag-Oftasks Applications With Deadline Constraints On DVS-Enabled Clusters," in Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, ser. CCGRID '07, 2007, pp. 541-548.

[13] G. Wu, Z. Xu, Q. Xia, J. Ren, and F. Xia, "Task Allocation and Migration Algorithm for Temperature-constrained Real-time Multicore Systems," in Proceedings of the IEEE International Conference on Cyber,Physical and Social computing, 2010, pp.189-196.

[14] X. Zhou, J. Yang, M. Chrobak, and Y. Zhang, "Performance-Aware Thermal Management via Task Scheduling," The Journal of ACM Transactions on Architecture and Code Optimization, vol. 7 issue 1, April. 2010, pp. 5:1-5:31.

[15] J. Quintin and F. Wagner, "Hierarchical Work-Stealing," in EuroPar'10 Proceedings of the 16th International Euro-Par Conference on Parallel Processing, 2010, pp. 217-229.

[16] Y. Guo, J. Zhao, V. Cave, and V. Sarkar, "SLAW: a Scalable Locality-aware Adaptive Work-stealing Scheduler," in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 2010, pp.1-12.

[17] S. Agarwal, G.K. Mehta, and Y. Li, "Performance- based Scheduling with Work Stealing." Internet: http://www.cs.ucsb.edu/~gaurav_mehta/reports/cs290b.pdf, 2009 [retrieved: March, 2014].

[18] D. Sudarshan and D. Pooja, "LIBRA:Client Initiated Algorithm for Load Balancing Using Work Stealing Mechanism," in Proceedings of 2nd International Conference on Emerging Trends in Engineering and Technology, 2009, pp. 636-638.

[19] A. Robison, M, Voss, and A. Kukanov, "Optimization via Reflection on Work Stealing in TBB," in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 2008, pp.1-8.

[20] K. Faxén and J. Ardelius, "Manycore Work Stealing,"in Proceedings of the 8th ACM International Conference on Computing Frontiers ACM, 2011, pp. 10:1-10:3.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-339-1