Reducing the Energy Cost of Computing through Efficient Co-Scheduling of Parallel Workloads

Can Hankendi　　Ayse K. Coskun
Electrical and Computer Engineering Department, Boston University, Boston, MA, 02215
{hankendi, acoskun}@bu.edu

***Abstract*—Future computing clusters will prevalently run parallel workloads to take advantage of the increasing number of cores on chips. In tandem, there is a growing need to reduce energy consumption of computing. One promising method for improving energy efficiency is co-scheduling applications on compute nodes. Efficient consolidation for parallel workloads is a challenging task as a number of factors, such as scalability, inter-thread communication patterns, or memory access frequency of the applications affect the energy/performance tradeoffs. This paper evaluates the impact of co-scheduling parallel workloads on the energy consumed per useful work done on real-life servers. Based on this analysis, we propose a novel multi-level technique that selects the best policy to co-schedule multiple workloads on a multi-core processor. Our measurements demonstrate that the proposed multi-level co-scheduling method improves the overall energy per work savings of the multi-core system up to 22% compared to state-of-the-art techniques.**

I. Introduction

Multi-core processors are broadly used in modern computing, from embedded to large-scale computing systems such as data centers and high performance computing (HPC) clusters. Multi-core processors offer performance improvements at lower power densities compared to complex single cores by utilizing simpler and more power-efficient cores on the same die. To take advantage of increasing number of cores on multi-core processors, parallel applications are expected to take over the application space in future computing clusters.

Although all parallel applications are designed to exploit various levels of parallelism on a multi-core system, performance scaling of some applications are limited by system resources (i.e., memory, bus, network, etc.), which lowers the system utilization. Lower utilization of multi-core systems increases the amount of idle power, which reduces the overall energy efficiency of the computing systems. As the energy consumption of the computing systems is among the major contributors to the total cost of ownership, reducing the energy cost is a significant goal for sustainable computing.

One promising method to reduce the energy cost is co-scheduling applications. By consolidating multiple workloads on the same physical nodes, energy spent per job can be reduced significantly. Nonetheless, energy savings due to co-scheduling varies depending on the characteristics of the applications that are being consolidated. In Figure 1, we show minimum and maximum energy consumed per useful work (*E/w*, energy per retired μOP) savings achieved by consolidation of PARSEC parallel benchmarks [1] compared to executing the same benchmarks on separate nodes. *E/w* metric considers both the energy consumption and the throughput of the system, thus it is a pertinent measure of energy efficiency. Figure 1 demonstrates a 60% range between maximum and minimum savings across benchmarks. While some of the benchmarks such as `bodytrack` or `dedup` consistently benefit from consolidation, some benefit minimally, such as `vips`. These results confirm that depending on the choice of consolidated applications, *E/w* savings vary dramatically. Thus, energy efficiency achieved through co-scheduling is a function of application characteristics; and it is important to consider such characteristics in designing co-scheduling policies.

In this paper, we first introduce a methodology to accurately evaluate the effect of co-scheduling on multi-core systems. We then analyze various co-scheduling policies (i.e., cache miss based, bus access based, instruction-per-cycle (IPC) based). We observe that success of the co-scheduling metric depends on the overall set of applications arriving at the cluster. In order to address this, we propose a novel multi-level technique to choose the best co-scheduling policy depending on the characteristics of the workload sets to improve the *E/w* savings. We evaluate 50 randomly created workload sets out of the PARSEC suite and show that our multi-level policy selection technique provides additional *E/w* savings up to 22% in comparison to state-of-the-art techniques.

The rest of the paper is organized as follows. Section

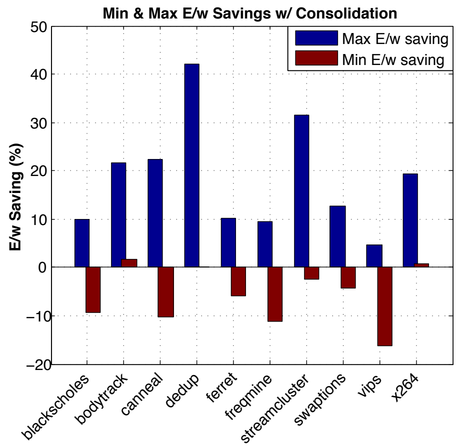

Fig. 1: Minimum and maximum change in *E/w* when a PARSEC application is consolidated with another PARSEC benchmark on an AMD Magny-cours server, each running with 6 threads, w.r.t. running the benchmarks separately with 12 threads.

II discusses the prior work. In Section III, we explain our methodology to evaluate the effect of co-scheduling on multi-core systems. Section IV discusses the factors affecting energy efficiency and provides the details of the proposed technique. In Section V, we evaluate our multi-level co-scheduling policy. Section VI concludes the paper.

II. Related Work

Dynamic power and energy management techniques such as controlling low-power/sleep modes and dynamic voltage-frequency scaling (DVFS) are well studied in the literature (e.g., [2],[3]). Both software and hardware-based power management strategies have been proposed in recent years. Kim *et al.* study on-chip regulators to achieve DVFS at a finer granularity reaching nanosecond range [4]. Shin *et al.* propose intra-task voltage scheduling through compiler-level static timing analysis [5].

Workload scheduling is another control knob that can achieve desirable performance and energy tradeoffs. Snavely *et al.* propose symbiotic job scheduling for single-threaded applications to improve system throughput [6]. Aydin *et al.* propose power-aware scheduling for real-time tasks [7].

Another group of energy management techniques focuses on improving the energy efficiency of large scale computing systems such as data centers and high performance computing (HPC) clusters (e.g., [8], [9]). Srikantaiah *et al.* study the correlation between energy consumption and resource utilization on the cloud by exploring performance and energy consumption of consolidated workloads through analyzing CPU and disk space usage [10]. They show that there is an optimal solution to the consolidation problem and provide a heuristic algorithm to find the consolidation scenarios providing minimal energy allocation. Romosan *et al.* propose algorithms for co-scheduling computation and data on clusters by load balancing frequently used files across multiple cluster nodes [11]. These analyses on large-scale multi-core systems provide important insights about the correlation between system utilization and energy efficiency; however, they do not consider application-level analysis during consolidation.

Frachtenberg *et al.* propose a co-scheduling technique through monitoring MPI (message passing interface) calls of parallel applications [12]. Their proposed co-scheduler identifies processes that communicate frequently through an MPI monitoring layer to make co-scheduling decisions. McGregor *et al.* present scheduling algorithms to improve performance by determining the best thread mixes [13]. They monitor workload behavior through performance counters and propose three scheduling policies which are based on bus transaction rate, stall cycle rate and last level cache miss rate. Raghavendra *et al.* propose a multi-level power management technique to coordinate different individual approaches such as virtual machine controller, enclosure manager etc. [14]. Bhadauria *et al.* propose resource-aware co-scheduling techniques for multi-core systems to improve energy-delay (ED) [15]. Their schedulers make decisions based on bus contention, last level cache miss rates and thread number to decide on time and space share of each workload. Dhiman *et al.* propose an energy efficient virtual machine scheduling technique based on CPU and memory usage of the workloads through performance counter monitoring [16]. Their technique estimates the CPU and memory usage information specific to virtual CPUs, and leverages the estimated data to make process migration decisions for better performance and power tradeoffs. Dey *et al.* propose a methodology to characterize the shared-resource contention of parallel applications [17]. They provide experimental analysis on inter- and intra-thread dependencies for PARSEC benchmarks.

Our work differentiates from the previous work as follows. First, we present an experimental approach to accurately evaluate the energy-performance tradeoffs for co-scheduling parallel workloads. We then show that best-performing co-scheduling policy varies depending on the overall characteristics of the workload sets running on a cluster. We design a multi-level selection mechanism that improves the energy efficiency by choosing the best co-scheduling policy, and demonstrate the benefits of our approach on a real-life multi-core system.

III. Methodology

This section presents the details of our experimental methodology. We run all experiments on an AMD 12-core Magny-cours (6172) server, running 2.6.29 Linux kernel OS. Magny-cours is a single-package CPU comprising two 6-core dies (each similar to AMD Istanbul architecture) attached side by side. There is a 1MB private L2-cache for each core and a 6MB shared L3-cache for each 6-core die.

We collect data from the performance counters to identify possible performance bottlenecks for parallel applications such as cache misses, stalls, memory accesses, etc. We use *pfmon 3.9* utility tool to collect the following core-level performance counters at a 100ms sampling interval: μ-OPs retired, unhalted CPU cycles, data cache misses, L2-cache misses, executed lock operations, mispredicted branch instructions, dispatch stalls, dispatched FP operations. In addition to per-core measurements, we collect system-level L3-cache misses. We also measure system power, CPU and memory utilization data at a 1s sampling interval. We measure the system power by using a *Wattsup PRO* power meter. As system power includes CPU, motherboard, and cooling power, it determines the total energy cost of computing. CPU and memory utilization percentages are collected using *mpstat* and *top*.

Parallel applications consist of serial I/O stages and the parallel phase, i.e., region-of-interest (ROI). In order to accurately measure the power and performance effects of consolidating parallel workloads on a single machine, it is important to consider only the ROI of the workloads. Parallel phases already dominate the execution time in HPC clusters and are expected to occupy a large portion of the cycles in future systems. Most of the prior work on consolidation, however, considers the full execution of the PARSEC applications including the serial phases (e.g., [15] [16]), which may cause inaccurate evaluation of co-scheduling tradeoffs.

In order to manage the start/finish times of application ROI phases and the data collection, we implement a consolidation management interface, `consolmgmt`, on top of the default PARSEC benchmark management interface, `parsecmgmt`. `consolmgmt` interface manages thread affinity settings to assign each thread to one core and the ROI-Synchronization routine (ROI-Synch) to synchronize the ROI of multiple workloads. We implement the ROI-Synch inside the `HOOKS` library routines of the default PARSEC package. In Figure 2, we demonstrate the synchronization flow for a hypothetical two benchmark consolidation case. ROI-Synch ensures that benchmarks A and B wait at the ROI checkpoints (ROI_A, ROI_B). After all benchmarks reach the ROI checkpoint, ROI-Synch calls *roi-Trigger()* function to synchronously trigger the benchmarks to resume their execution. Data loggers are triggered by the *start-Logging()* function to collect the ROI performance and power characteristics. We collect all experimental data for the first 20s of the parallel phase of the PARSEC 2.1 parallel workloads [1].

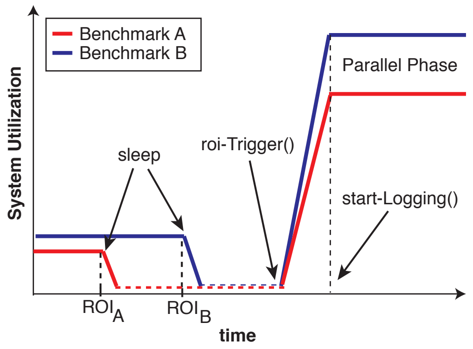

Fig. 2: ROI-Synchronization flow for the case of co-scheduling two benchmarks A and B.

We compute energy consumed per work done (*E/w*) to evaluate the impact of consolidation on energy efficiency. *E/w* measures the required energy for the useful work, which is basically the efficiency of the system. Our comparison is based on two scenarios on the 12-core server. In the consolidation scenario, a benchmark is co-scheduled with another benchmark, each running with 6 threads on a distinct 6-core die of the Magny-cours system. In the second scenario, benchmarks are executed alone with 12 threads on 12 cores. While similar methodology can be applied to other consolidation scenarios, in this work we target co-scheduling two applications as *E/w* savings decrease for more aggressive consolidation cases. *E/w* values for consolidated and 12-thread cases are derived from Equations 1 and 2, respectively. $Thrput_{6T}$ and $Thrput_{12T}$ denote the throughput of the application at 6 and 12 threads, which is equal to total number of retired μOPs per second. We then compute the percentage of *E/w* decrease with respect to the execution with 12 threads.

$$E/w_{con} = \frac{Energy_{A,B(con)}}{Thrput_{A-6T} + Thrput_{B-6T}} \tag{1}$$

$$E/w_{12T} = \frac{Energy_{A-12T} + Energy_{B-12T}}{Thrput_{A-12T} + Thrput_{B-12T}} \tag{2}$$

IV. Energy-Efficient Co-Scheduling of Parallel Workloads

In this section, we first discuss the factors that impact energy efficiency of multi-core systems. We then present our multi-level co-scheduling policy to improve the energy efficiency during consolidation of parallel workloads.

A. Factors Affecting Energy Efficiency

Energy consumption of a multi-core system varies as a function of the characteristics of the parallel workloads running on the system. As a result, the range of potential *E/w* savings show significant variations across workloads. To understand these variations, next we analyze the energy tradeoffs for a set of performance events for the PARSEC benchmarks.

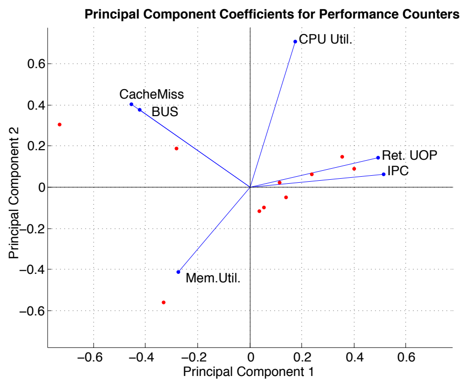

Fig. 3: First two principal component coefficients for various performance metrics.

We use Principal Component Analysis (PCA) to determine which performance events vary considerably across the benchmark suite. Figure 3 shows the coefficients for various performance events for their first two principal components (PCs). First two PCs together explain more than 85% of the overall variations. Figure 3 demonstrates that performance events cover the PC space in four distinct directions. Cache misses and bus accesses almost have the same coefficients. Similarly, retired μOPs and IPC are closely related. These two groups of events cover distinctive features on the x axis of the PC space. On the other hand, CPU and memory utilization cover other distinct features of the applications on the y-axis of the PC space. Note that memory utilization is located at a different quadrant than the cache misses, motivating investigating the impact of both cache misses and memory utilization metrics separately. Following the results of the PCA, we focus on the performance events discussed below. We focus on the average values of the performance counters, since in this paper we target co-scheduling the applications at the job allocation stage, without considering run-time workload migration scenarios.

CPU utilization: CPU utilization measures percentage of active (non-idle) time of the CPU. Figure 4 shows average CPU utilization percentages of the PARSEC benchmarks while running 12 threads. Applications that heavily utilize the CPU resources are CPU-bounded as their performance improvement

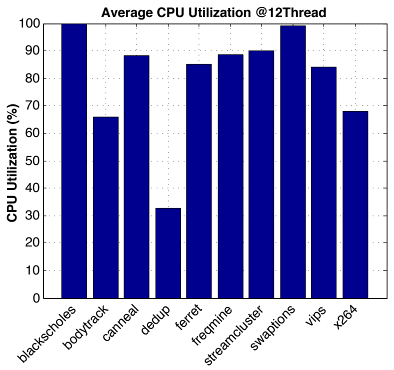

Fig. 4: CPU utilization of the benchmarks running 12 threads.

is only limited by the CPU resources. CPU-bounded applications are expected to benefit marginally from co-scheduling in terms of energy efficiency because running CPU-bounded benchmarks with a higher number of threads improve performance significantly. Therefore, the energy increase for running a larger number of threads is compensated with an increase in throughput, without the need of consolidation. On the contrary, applications that are not CPU-bounded benefit from co-scheduling. Applications such as `dedup`, `bodytrack` and `x264` poorly utilize the CPU resources and they are the only PARSEC benchmarks that consistently benefit from consolidation as Figure 1 implies.

Cache misses: Cache miss rates are measures of the memory stall cycles. Figure 5 shows the weighted cache miss per μOP. Since different levels of cache differ in access times, we evaluate the weighted cache misses by using the specific miss penalty cycles for L1 and L2 caches. High cache miss rates cause large number of stall cycles, resulting in sub-linear performance scaling of the applications. Thus, applications with high cache miss rates are expected to benefit more substantially from co-scheduling compared to running with a higher number of threads. Figure 5 shows that `canneal` and `streamcluster` have significantly higher cache miss rates than other benchmarks, as `canneal` operates on significantly large datasets and `streamcluster` solves a clustering problem on continuously streaming data points as its input. These two benchmarks have relatively higher *E/w* savings when they are co-scheduled with other benchmarks.

Memory utilization: Memory utilization percentage shows the utilization of the DRAM modules. Figure 6 shows the memory utilization percentages of PARSEC benchmarks.

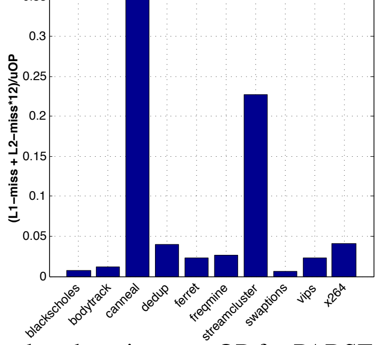

Fig. 5: Weighted cache miss per μOP for PARSEC benchmarks running 12 threads.

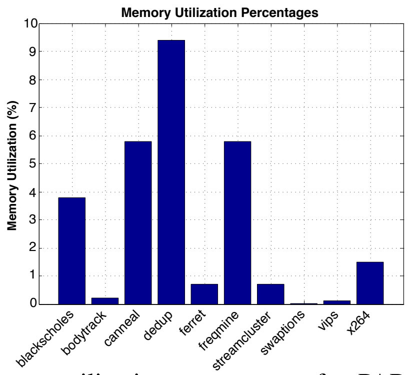

Fig. 6: Memory utilization percentages for PARSEC benchmarks running 12 threads.

When consolidating multiple benchmarks on a multi-core system, it is important to consider the degree of memory utilization, as memory is a shared resource across applications. `canneal`, `dedup` and `freqmine` utilize the memory significantly higher compared to the rest of the benchmarks. Note that despite having low cache miss rates, `freqmine` and `dedup` utilizes the memory heavily. On the other hand, `streamcluster` does not utilize the memory heavily, although it generates high cache misses. These observations confirm memory utilization and cache misses both needs to be considered while making consolidation decisions.

Our analysis on performance events show that considering a single performance event would not be sufficient to make optimum consolidation decisions. In addition, PARSEC benchmarks differ significantly in their performance bottlenecks. Therefore, as the overall set of applications (i.e., workload set) running on a cluster changes, we would potentially select a different metric for grouping the benchmarks during consolidation. This motivates having a meta-policy to select which metric to use for consolidation. Next, we explain our multi-level consolidation technique.

B. *Proposed Multi-level Co-scheduling Policy*

The goal of our consolidation technique is to improve energy efficiency, or in other words, to decrease *E/w*. Our multi-level co-scheduling technique utilizes principles from two previously proposed co-scheduling policies: *cache miss* based [15] and *IPC*CPU-Utilization* based [16] consolidation. We follow two main steps:

1) Selecting the co-scheduling policy: The goal is to choose the best-fitting policy for co-scheduling depending on the overall workload set running at the cluster. We first distinguish between computation and communication-intensive benchmarks by computing the sum of IPC, CPU-Utilization and bus access of each workload and derive the computation-to-communication ratio for the given workload sets, $IPC * CPU_{Util}/BUS_{acc}$. Workload sets that have lower computation-to-communication ratio mostly suffer from high bus accesses causing lower IPC. Therefore, balancing the useful work done lowers *E/w* by increasing the *w*. We experimentally derive a threshold (Th_1) of 5000 and classify the workload sets according to their cumulative $IPC * CPU_{Util}/BUS_{acc}$ values. If the computation-

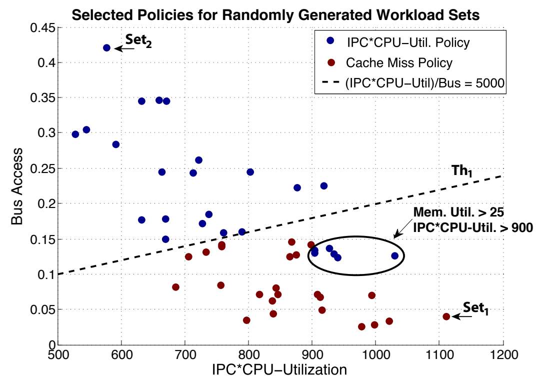

Fig. 7: Performance characteristics and the selected co-scheduling policies for 50 randomly generated workload sets.

to-communication ratio of the workload set is below Th_1, we choose the $IPC * CPU_{Util}$ policy to balance the useful work done. Workload sets that have high computation-to-communication ratio ($> Th_1$) are co-scheduled using either $IPC * CPU_{Util}$ or cache miss based policy. Before choosing the policy, we first evaluate the $IPC * CPU_{Util}$ and memory utilization (MEM_{Util}) separately. In this step, we identify whether there is a stronger bottleneck compared to the cache misses for computationally intensive workloads. Computationally intensive workload sets ($> Th_2 = 900$) that have high memory utilization ($> Th_3 = 25$) are scheduled using $IPC * CPU_{Util}$ policy, as higher memory utilization lowers the useful work done. Rest of the workload sets would have only cache misses as their major bottleneck. Thus, we co-schedule them using the cache miss based policy. Algorithm 1 demonstrates the pseudo-code of the policy selection algorithm.

2) Co-scheduling benchmarks in a given workload set: After choosing the appropriate policy, we rank benchmarks in each workload set according to the selected metric. If the selected policy is $IPC * CPU_{Util}$, we rank the benchmarks from high to low according to their $IPC * CPU_{Util}$ values. We then co-schedule the benchmarks starting by pairing up the two benchmarks with the lowest and the highest ranking and progressing through the list this way. We use the same sorting and balancing principle for the cache miss based policy.

Algorithm 1 Policy Selection

```
if $(IPC * CPU_{Util}/BUS_{acc}) > Th_1$ then
    if $(IPC * CPU_{Util} > Th_2)$ & $(MEM_{Util} > Th_3)$ then
        Balance: $IPC * CPU_{Util}$
    else
        Balance: $CacheMiss$
    end if
else
    Balance: $IPC * CPU_{Util}$
end if
```

V. Experimental Results

This section presents the experimental evaluation. To compare our multi-level co-scheduling policy against previously

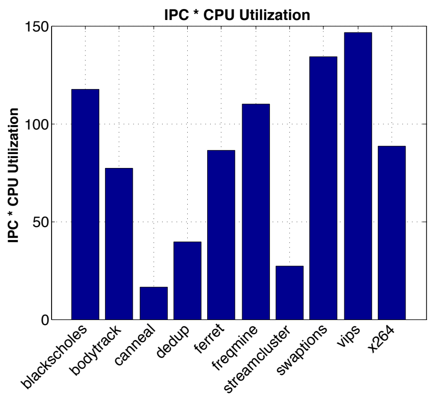

Fig. 8: IPC * CPU Utilization for PARSEC benchmarks running 12 threads.

proposed approaches, we generate 50 workload sets. Each workload set contains 10 randomly selected PARSEC applications. For example, Set_1 in Figure 7 consists of 4 instances of blackscholes, 2 instances of vips and one instance of bodytrack, freqmine, streamcluster and swaptions applications. On the other hand Set_2 consists of 3 instances of canneal and ferret, 2 instances of bodytrack and one instance of dedup and vips applications. As Figure 7 shows, randomly generated workload sets cover a wide range of bus accesses and CPU usage values. In Figure 7, we also show the selected policies and the computation-to-communication ratio threshold (Th_1). For a set of 50 randomly generated workload sets, proposed multi-level technique achieves 82% accuracy for selecting the best co-scheduling policy within a 2% error margin of the best-case consolidation in terms of *E/w* savings.

We implement three co-scheduling algorithms based on previous approaches [16] [15]. These three approaches allocate applications on given machines by balancing the cache misses (L1 and L2), bus accesses (L3 cache misses) or computational intensity ($IPC * CPU_{Util}$) across co-scheduled application pairs (e.g., co-scheduling a benchmark with a high cache miss rate together with a benchmark with low cache miss rate on the same machine). Figure 8 shows $IPC*CPU_{Util}$ values and Figure 9 shows L3-cache misses of 10 PARSEC benchmarks.

In Figure 10, we show average *E/w* savings for 50 workload sets. We report savings due to consolidating applications with

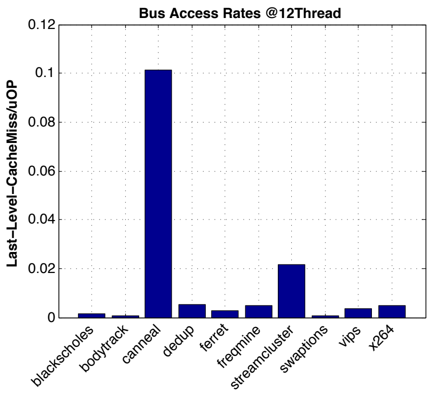

Fig. 9: Last level cache (L3) misses per μOP for PARSEC benchmarks running 12 threads.

various co-scheduling policies with respect to each application running with 12 threads separately on a single node. Bus access based policy performs worst among all policies, since bus accesses per μOP does not necessarily capture the frequency of the bus accesses. Our multi-level co-scheduling policy provides 31.3% *E/w* savings on average, which is 8.6% higher than the best performing previous policy.

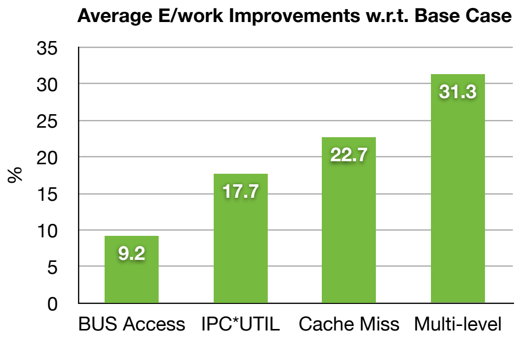

Fig. 10: Average *E/w* saving improvements for 4 policies w.r.t 12 thread execution on a single node.

In Figure 11, we show maximum *E/w* saving improvements achieved by our multi-level policy across the 50 workload sets. We also report energy savings and throughput difference for the maximum savings case. Our proposed approach improves the *E/w* savings by 14.1% with respect to best performing cache miss based policy with a 0.1% throughput (retired μop per second) decrease. *E/w* savings reach up to 45.2% with respect to the bus access based policy along with a throughput loss of 16.9%.

VI. Conclusions

Improving energy-efficiency of multi-core systems is one of the major challenges of future computing clusters. Co-scheduling multiple applications on the same node provides opportunities to improve the energy-efficiency. In this paper, we propose an experimental methodology to accurately evaluate the effects of co-scheduling on multi-core systems. Using this methodology, we show that none of the existing policies consistently outperforms the others as the success of consolidation is dependent on the overall workload set running on the cluster. Also, we demonstrate that the benefits of consolidation vary dramatically depending on which applications are co-scheduled on the same resources. In order to address these challenges, we propose a novel multi-level technique that first selects a co-scheduling policy by evaluating various cumulative characteristics of the workload sets and then balances the workload on the machines in terms of a performance metric determined by the selected policy. Our technique provides additional 22% *E/w* savings on average and improves the *E/w* savings up to 14% in comparison to the best performing state-of-the-art co-scheduling policy.

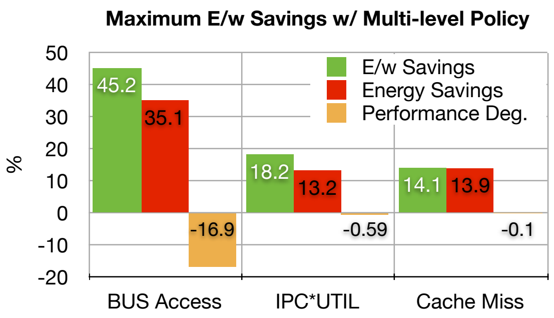

Fig. 11: Maximum *E/w* savings, energy savings, and performance loss of our multi-level policy w.r.t. previous methods.

Acknowledgment

This research has in part been funded by Oracle, Inc., VMware, Inc. and Dean's Catalyst Award at College of Engineering, Boston University.

References

[1] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[2] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS '09, 2009.

[3] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 299 –316, june 2000.

[4] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *International Symposium on High-Performance Computer Architecture*, 2008.

[5] D. Shin, J. Kim, and S. Lee, "Low-energy intra-task voltage scheduling using static timing analysis," in *Proceedings of the 38th Conference on Design Automation*, 2001.

[6] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreaded processor," *SIGARCH Comput. Archit. News*, vol. 28, pp. 234–244, November 2000.

[7] H. Aydin, R. Melhem, D. Moss, and P. Meja-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of Real-Time Systems Symposium*, 2001, pp. 95–105.

[8] X. Fan, W. dietrich Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *In Proceedings of ISCA*, 2007.

[9] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP '01, 2001, pp. 103–116.

[10] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 conference on Power aware computing and systems*, ser. HotPower'08, 2008.

[11] R. Romosan, D. Rotem, A. Shoshani, and D. Wright, "Co-scheduling of computation and data on computer clusters," in *In Proceedings of the 17th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2005.

[12] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fern, "Adaptive parallel job scheduling with flexible coscheduling," *IEEE Trans. Parallel and Distributed Syst*, pp. 1066–1077, 2005.

[13] R. L. Mcgregor and C. D. Antonopoulos, "Scheduling algorithms for effective thread pairing on hybrid multiprocessors," in *In Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, 2005.

[14] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: coordinated multi-level power management for the data center," in *ASPLOS XIII*, 2008, pp. 48–59.

[15] M. Bhadauria and S. A. McKee, "An approach to resource-aware co-scheduling for cmps," in *Proceedings of the 24th ACM International Conference on Supercomputing*, ser. ICS '10, 2010, pp. 189–199.

[16] G. Dhiman, G. Marchetti, and T. Rosing, "vgreen: A system for energy efficient computing in virtualized environments," in *ISLPED*, 2009, pp. 243–248.

[17] T. Dey, W. Wang, J. W. Davidson, and M. L. Soffa, "Characterizing multi-threaded applications based on shared-resource contention," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2011, pp. 76–86.