ESTO: A Performance Estimation Approach for Efficient Design Space Exploration

Sparsh Mittal and Zhao Zhang
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa 50011, USA
Email: {sparsh,zzhang}@iastate.edu

***Abstract*—Recent advancements in CMOS fabrication and processor architectures have posed new challenges to the chip designers. The availability of a large number of design configurations, along with constraints such as energy consumption and real-time response demand exploring a large design-space for choosing the best possible configuration. However, the high cost of detailed simulators prohibit the designers from doing this. The existing techniques of performance estimation generally use intrusive methods or have high overhead or inaccuracy and hence, are not suitable for real-world processors.**

In this paper we present ESTO, a simulation-based approach for estimating application performance (execution time and energy) under multiple last level cache (LLC) configurations. ESTO uses a multi-level profiling cache which provides low-cost and non-intrusive dynamic profiling. A unique feature of ESTO is its ability to estimate performance of a cache of higher size than the baseline cache present. Experiments performed using a state-of-art simulator and benchmarks from SPEC2006 suite show the effectiveness of ESTO. For our workload, the average error in estimating execution time and memory subsystem energy are 3.7% and 3.3% respectively.

I. Introduction

Recent advancements in the field of processor architecture and chip design have opened new horizons for both architects and end-users. While these architectures promise high performance [1–4], they also pose significant challenges to the designers, due to the increasing number of design options (e.g. cache configurations) and design constraints (e.g. energy). Further, to bridge the widening gap between DRAM speed and processor speed, modern processors are using increasingly large LLCs and hence, LLCs have a significant influence on their performance. Over several years of CPU evolution, the size of L1 cache has stayed at 16KB or 32KB, while the size of the LLC has grown from nearly 256KB to 1, 2 or 4 MB in modern day processors, with future processors expected to have even larger LLC sizes. Hence, while translating a design from concept phase to a working chip, a designer must choose a suitable LLC size, based on the application requirements and also meet the constraints posed by chip power budget and real-life timing requirements. For this purpose, designers generally use detailed simulators for evaluating different design options, however, the high simulation time of these simulators makes it infeasible to use them for testing all possible configurations in the design space. This forces the designers to take decisions without considering all the design constraints or fully exploring the design space.

To address this challenge, several techniques have been proposed for performance estimation and fast design space exploration [5–7]. However, existing techniques of performance estimation have several drawbacks. Superscalar out-of-order processors use speculative execution and hence, the possible overlap between execution and different miss events such as cache misses and branch mispredictions etc. make it challenging to estimate performance under multiple design options. For this reason, several techniques use simplistic platforms or require offline profiling or multiple runs (e.g. [8, 9]) and hence these techniques are difficult to scale to real-world processors and applications, which execute trillions of instructions. Many performance estimation techniques use intrusive methods which have a large space/time overhead. A few other techniques have a large error of estimation and hence, the conclusions derived from them could be very misleading. Thus, an efficient and accurate performance estimation method is required for design space exploration and making crucial design decisions.

In this paper, we present ESTO, a dynamic profiling based technique for estimating the performance of an application program under a range of possible last level cache (LLC) sizes[1]. The key idea behind our approach is the use of a small *profiling cache*, to estimate the number of LLC misses under different cache configurations and to compute their effect on program performance. Profiling cache is a data-less cache which is based on the idea of set sampling [10] and has an energy overhead of less than 1% of that L2 cache. ESTO uses memory stall cycle model to take into account the possible overlap between different miss events and thus ESTO can be used in out-of-order processors with speculative execution support.

For a system with L2 cache size of X, we define any cache with size $\leq X$ as *sub*-sized cache and any cache of size $\geq X$ as *super*-sized cache. A unique feature of ESTO is its capability to estimate execution time and energy of both super-sized caches and sub-sized caches. Thus, for example, using a 4MB L2 cache, a designer can estimate the performance of 8MB L2, as well as 2MB, 1MB, 512KB and 256KB caches. This feature is extremely useful for making projections about a future configuration which may be presently unavailable.

[1]In this paper, we use the term performance to refer to execution time (ET) and energy consumption together.

Thus, ESTO helps a designer in choosing most suitable LLC configuration and fulfill the design constraints.

ESTO addresses several limitations of the existing approaches. Firstly, ESTO uses non-intrusive dynamic profiling and hence, does not require any changes to application source code or binaries. The profiling cache works in parallel with L2 and hence does not affect the access latency of the L2 cache. ESTO provides online estimates of performance and does not require offline profiling or any separate runs. To evaluate ESTO, simulations were performed using Sniper [11], simulator and benchmark programs from SPEC2006 suite. Across 80 combinations of benchmarks and configurations, the average error in execution time (ET) estimation is 3.7%. Further, the average error in memory subsystem energy (L2 cache+ main memory energy) is 3.3%. These results confirm the effectiveness of ESTO.

As computer systems are becoming increasingly power constrained, workload optimized system design is expected to become even more prominent, as seen through example of Intels Many Integrated Core (MIC) architecture and IBM's BlueGene processor. Hence, our approach is likely to become even more important in the design of future computer chips. Profiling cache can be easily used for saving cache energy, thus helping the designers in realizing the goals of sustainable and green IT.

II. Motivation and scope of the work

We present the motivation for using ESTO with a typical design scenario. Modern portable devices such as personal digital assistants, phones, laptops and iPODs etc are powered by the battery which supplies limited energy. Thus, the amount of battery dissipation which is induced by program execution becomes an important factor in assessing battery life and gives valuable information to take decision about recharging or replacement. This is especially important in situations such as traveling in flight etc. To address such needs, ESTO enables an architect to use a suitable cache size, taking into account the energy budget, usage scenario and quality of service (QoS) requirement. For example, if a certain delay in response is acceptable, the architect can use a smaller sized cache if that is more energy efficient. Similarly, within a same energy budget, an architect can use a larger sized cache if that is more performance efficient.

Our objective in this paper is to propose and experiment with the methods which enable exact program execution time estimation for a given input and hardware for different configurations of L2 cache sizes. The WCET analysis approach is different from our work. Worst-Case Execution Time (WCET) prediction approach seeks to estimate the upper bound of the program execution time under different program inputs or hardware platforms or system resources. Given the large number of possible inputs, only a range or bound is estimated for WCET. Moreover, such analysis has been done by assuming simplified/idealized platforms (e.g. perfect processor pipeline with no stalls [12] etc). In contrast, we estimate exact execution time, using a detailed out-of-order superscalar processor which presents challenges of its own.

III. Methodology

It is well-known that under different cache configurations, program applications show different number of cache misses, and hence different performance. Hence, to estimate the impact of multiple L2 cache configurations on performance, ESTO uses *profiling cache* to predict L2 misses under those configurations (Section III-A). Using these estimates, along with CPI stack model, ESTO estimates execution time of the application under those configurations (Section III-B). Finally, using these estimates, ESTO estimates both dynamic and leakage energy component of memory subsystem energy (Section III-C). Based on these estimates and the domain knowledge of design constraints, a designer can take suitable design decisions. Figure 1 shows the overall flow diagram of ESTO. In what follows, we explain each of these components in detail.

A. Profiling cache

Profiling cache is a small, dataless (tag only) cache, which is designed based on the well-known set sampling technique [13], which states that the miss rate characteristics of a set associative cache can be estimated by sampling only a few of its sets. The ratio of set count of L2 and that of a profiling cache is termed as sampling ratio (R_s). Profiling cache emulates L2 and thus, has same associativity, block size and replacement policy as L2. On an access to profiling cache, a hit or miss is decided and corresponding counters are updated. Note that, it does not store or communicate data and hence does not generate traffic. On a miss, the tag of missed address is copied and the victim is evicted. Thus, profiling cache is decoupled from L2 cache and as shown in Section IV, the size of this 'single level' profiling cache is only 0.10% of L2 cache size.

We use the above mentioned properties to extend profiling cache, such that it profiles multiple cache sizes in parallel; each size is referred to as a level. For our experiments, we choose six levels, each level profiling a cache of size 2X, 1X, X/2, X/4, X/8, X/16 respectively. These levels, also referred to as configurations, are, in general, shown as C and the baseline (1X) configuration is shown as $C^{\star}$. Also, note the unique capability of profiling cache: because of its decoupled operation with L2, it can also profile a cache of 2X size (double the baseline cache size) with reasonable accuracy, as we will see in the results section (Section VI). This feature is an important improvement over previous works based on profiling and it allows a designer to estimate program performance for a cache size which may be currently unavailable.

As shown in Section IV, even with this extension, the size of multilevel profiling cache is only 0.40% of L2 cache size. Thus, the multilevel profiling cache has a small size and access latency and since it does not lie on the critical access path, its latency is easily hidden. In what follows, we use the word

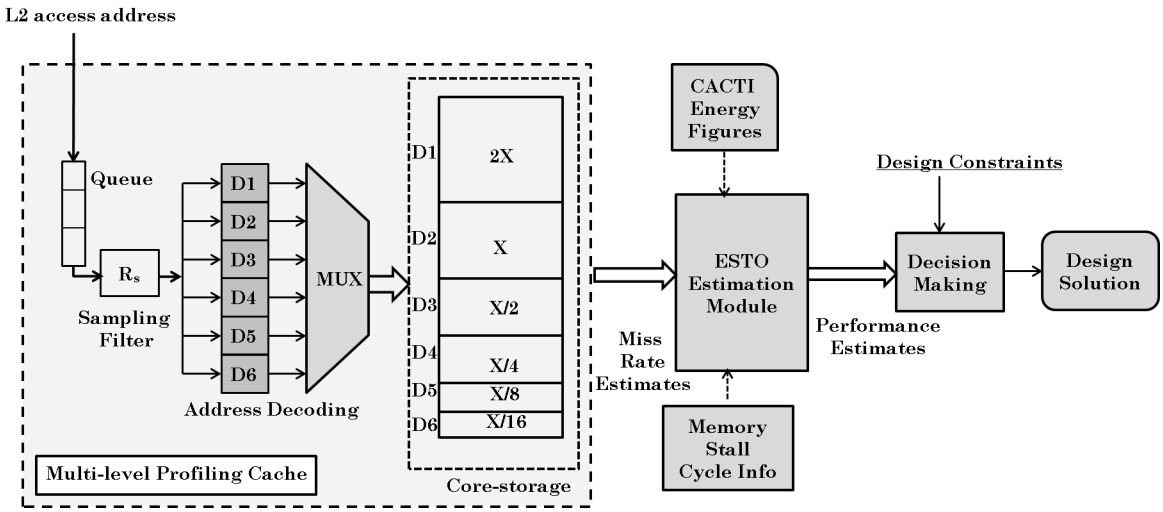

Fig. 1. ESTO flow diagram

profiling cache to refer to a multilevel profiling cache, unless otherwise mentioned.

The profiling cache works as follows (ref. Fig. 1). The L2 access addresses are passed through a small queue and then sampled using a sampling filter. Then these sampled addresses are passed through address decoding region for calculating the set (index) and tag values. Then these addresses are sent to the core storage component through a multiplexer (MUX). We mention that even though profiling cache is accessed multiple times for each sampled address, the presence of the queue and use of a large sampling ratio avoids the possibility of any congestion.

B. Execution time Estimation

For estimating both execution time and leakage energy under different cache configurations, we need to estimate memory stall cycles under those configurations as a function of L2 misses. However, modern out-of-order processors use several features for hiding latency (e.g. overlap between miss events such as branch misprediction and L2 miss), and hence the memory stall cycles cannot be computed as a linear function of the number of L2 misses.

To address this issue, ESTO uses a well known technique, called CPI stack model [11]. CPI stack shows the contribution of base execution along with different miss events, (such as branch mispredictions, cache misses) in the overall CPI of the program. For example, in any interval i, the memory stall cycle component of CPI stack (termed as $\text{StallCPI}_i(C^\star)$) shows the net contribution of memory stall cycles on overall cycles, *after* taking into account the overlap with other miss events. Let $\text{LoadMisses}_i(C^\star)$ show the number of load misses in interval i. Now, since memory stall cycles are primarily due to L2 load misses [10], we define $\text{K}_i(C^\star)$ as follows.

$$\text{K}_i(C^\star) = \frac{\text{StallCPI}_i(C^\star)}{\text{LoadMisses}_i(C^\star)} \tag{1}$$

Here $\text{K}_i(C^\star)$ shows memory stall CPI per load miss. We assume that $\text{K}_i(C^\star)$ value is independent of the number of load misses and hence remains same for different cache configurations, thus K_i=$\text{K}_i(C^\star)$, for all configurations. Further, we also use extra counters in profiling cache to record load misses, along with total misses, for different L2 configurations. Then, $\text{StallCPI}_i(C)$ for any configuration (C) can be computed, using

$$\text{StallCPI}_i(C) = \text{K}_i \times \text{LoadMisses}_i(C) \tag{2}$$

Then, using $\text{StallCPI}_i(C)$ and other components of CPI stack, total CPI value at any configuration C can be computed. Using total CPI, along with given frequency value and number of instructions, execution time under C can be easily estimated.

C. Energy Estimation

We now discuss the energy model used in ESTO and also show the procedure for estimating program energy value under any configuration using the estimates of miss rates and execution time. Since other components of processor are minimally affected by change in L2 cache size, we only consider memory subsystem energy, which is given as the sum of L2 and memory energy.

$$Energy = E_{L2} + E_{mem} \tag{3}$$

We use the symbols E_{L2}^{dyn} and P_{L2}^{leak} to show the dynamic energy *per access* and leakage energy *per second*, respectively, consumed in L2 cache. For memory, these parameters are shown by E_{mem}^{dyn} and P_{mem}^{leak} respectively.

To calculate L2 energy, we assume that an L2 miss consumes twice the energy as that of an L2 hit [13]. Thus,

$$E_{L2} = E_{L2}^{dyn} \times (2M_{L2} + H_{L2}) + P_{L2}^{leak} \times Time \tag{4}$$

Here, for any configuration, we have corresponding M_{L2}=L2 misses, H_{L2}=L2 hits, $Time$=execution time. The L2 energy values are obtained using CACTI 5.3

(http://quid.hpl.hp.com:9081/cacti/) for 4 bank, 8-way caches with 64 byte block size at 45nm. These values are shown in Table I.

TABLE I
L2 CACHE ENERGY VALUES

	$E_{L2}^{dyn}(nJ/access)$	$P_{L2}^{leak}(Watt)$
8MB	1.525	5.588
4MB	1.148	2.848
2MB	0.985	1.568
1MB	0.912	0.966
512KB	0.872	0.664
256KB	0.848	0.500

To calculate memory energy, we note that P_{mem}^{leak}=0.18 Watts and E_{mem}^{dyn}=70 nJ [10]. Thus, we get,

$$E_{mem} = E_{mem}^{dyn} \times A_{mem} + P_{mem}^{leak} \times Time \quad (5)$$

where A_{mem} denotes the number of memory accesses. From Eq. 4 and 5, it is clear that using miss rate and execution time estimates, program energy under any configuration can be estimated.

IV. OVERHEAD OF ESTO

ESTO uses profiling cache and computations for performance estimation, and hence the overhead of ESTO comes from these two components. ESTO does computations for ET and energy only at the end of a large interval length (e.g. 5M instructions). Thus, the cost of these calculations is amortized over interval length. In remainder of this section, we first compute the size of single level and multilevel profiling cache and then compute the energy consumption of multilevel profiling cache, to show that the overhead of ESTO is extremely small. We use the subscripts $Single$ and $Multi$ to represent any quantity (e.g. size) for single level and multilevel profiling cache respectively. For a W way L2 cache having Q sets, B byte cache block and G bit tag, the total cache size in bits is

$$\text{Size}_{L2} = Q \times W \times (B \times 8 + G) \quad (6)$$

Since profiling cache is a dataless cache, its size is

$$\text{Size}_{Single} = \frac{Q}{R_s} \times W \times G \quad (7)$$

If Θ_{Single} shows the size of single level profiling cache as a percentage of L2 size, we get

$$\Theta_{Single} = \frac{G}{R_s(G + B \times 8)} \times 100 \quad (8)$$

For R_s=64, B=64 and G=36 we get Θ_{Single}=0.10%.

For computing size of multilevel profiling cache, we first compute the number of sets (Sets_{Multi}) in it, as follows.

$$\text{Sets}_{Multi} = \frac{2Q}{R_s} + \frac{Q}{R_s} + \frac{Q}{2R_s} + \frac{Q}{4R_s} + \frac{Q}{8R_s} + \frac{Q}{16R_s} \quad (9)$$

$$\text{Sets}_{Multi} = \frac{63Q}{16R_s} < \frac{4Q}{R_s} \quad (10)$$

Using above equations, we compute the size of multilevel profiling cache as a percentage of L2 size (Θ_{Multi}) as follows.

$$\Theta_{Multi} = \frac{4G}{R_s(G + B \times 8)} \times 100 \quad (11)$$

Thus, for R_s=64, B=64 and G=36 we get Θ_{Multi}=0.40%. To cross-check, we have computed the area of L2 and multilevel profiling cache using CACTI, for the cache sizes used in our experiments (Section V). Since multilevel profiling cache is a tag only structure, we take 8B block size, which is smallest allowed block size in CACTI and only take the area values for tag arrays. From these values, we compute Θ_{Multi} and find that Θ_{Multi}=0.29%, which is in the same range as that obtained above.

To compute the energy values for (multilevel) profiling cache, we take R_s=64 and use CACTI 5.3. As explained above, we only take the energy figures for tag arrays. For a profiling cache corresponding to a baseline L2 of 2MB, we get the energy values as E_{Multi}^{dyn}= 0.004 nJ/access and P_{Multi}^{leak}=0.007 Watt. Noting that, profiling cache is accessed only 6 times for every 64 L2 accesses, we find that profiling cache energy consumption is a very small fraction of L2 cache energy consumption. Thus the overhead of ESTO is indeed very small. Moreover, by taking large value of sampling ratio (e.g. R_s=128), the overhead of ESTO can be even further reduced.

V. EXPERIMENTAL PLATFORM

For evaluating ESTO, we have used Sniper [11], which has been validated against the real hardware. We model 4-way processor with 1GHz frequency. L1I and L1D are 32KB, 4-way caches with 4 cycle latency. L2 is 4MB, 8-way cache with 12 cycle latency. All caches use LRU and 64B block size. Memory has 90 cycle latency, 6GB/s peak bandwidth and memory request queue is also modeled. The performance estimates are collected after every 5M instructions.

Our workload consists of 16 benchmark programs from SPEC2006 (astar, bwaves, cactusADM, gamess, gemsFDTD, gobmk, h264ref, hmmer, lbm, leslie, libquantum, mcf, perlbench, sjeng, sphinx and tonto), which represent a wide range of cache usage characteristics. Each benchmark program was fast forwarded for 10B instructions and then simulated for 100M instructions.

VI. RESULTS

In this section, we present the results on accuracy of estimation of program execution time and memory subsystem energy. Further, to be strict in evaluation, we compare execution time and energy values only for cache sizes other than 1X, since, for 1X size (i.e. baseline), these values are easily predicted with high accuracy. ESTO provides performance and energy estimates for five cache sizes (other than baseline) and with 4 MB cache as baseline, these caches have the size of 8MB, 2MB, 1MB, 512KB and 256KB (4MB itself is baseline and is skipped). Hence, using 4MB cache, a single run was performed for each benchmark, and performance estimates

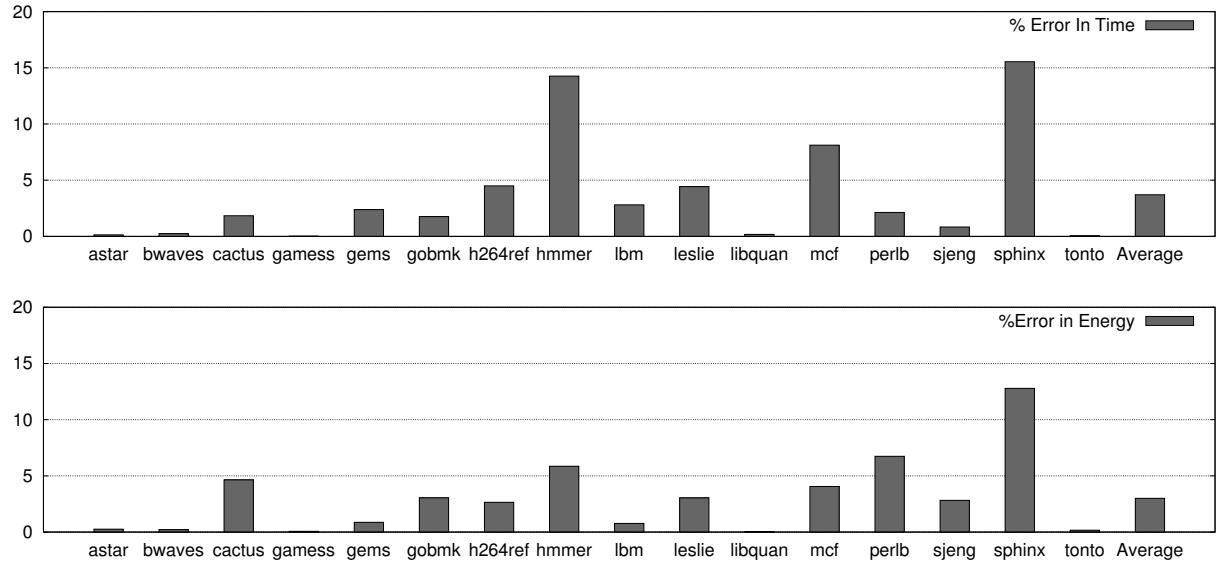

Fig. 2. %Error in Exeuction Time and Energy Estimation

were obtained using ESTO. These estimates were compared with the corresponding actual values obtained using 8MB, 2MB, 1MB, 512KB and 256KB caches and percentage errors were computed with respect to baseline values.

Figure 2 shows the average error for each benchmark, across all cache sizes. Across all benchmark/configuration combinations, the average errors in execution time estimates and energy estimates are 3.7% and 3.3% respectively.

Figure 3 presents the same result; this time for each cache size, across all benchmarks. Clearly, for 2X (8MB) and X/2 (2MB), the accuracy is the highest, which decreases gradually as we move to cache sizes farther from 1X.

We have also tested ESTO for sampling ratio value of 128 and observed that ESTO still provides high estimation accuracy. Further, for approximate LRU schemes, such as round-robin replacement policy also, ESTO provides high accuracy, which implies that ESTO does not require implementation of true-LRU policy.

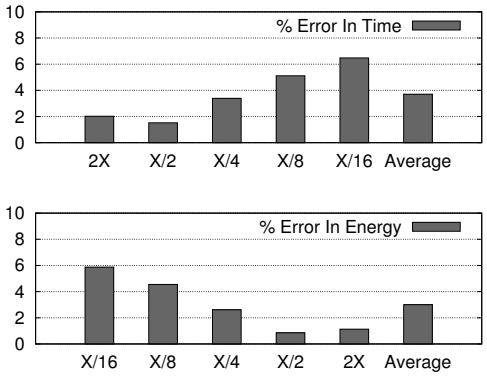

Fig. 3. %Error in Execution Time and Energy Estimation

We have shown the effectiveness of ESTO in execution time and energy estimation. ESTO can be easily extended to estimate total system energy by simply including the energy model of processor core in total energy equations. Also, since ESTO predicts both energy and execution time; using these estimates the energy delay product (EDP) of the program can also be estimated, although with higher error.

VII. Related Work

Recently, several methods have been proposed for estimating cache miss rate, execution time and energy of a program. In the following, we review them briefly.

Miss Rate Estimation: Tam et al. [14] present a software based L2 miss rate prediction approach. This technique works by recording data addresses of memory accesses to a data address register and later feeding the log of addresses to an LRU stack simulator to generate the miss rate curve (MRC) using the Mattson stack algorithm. This technique only takes into account L1 data cache misses and does not take into account L1 instruction cache misses and L1 data write-backs. This, however, leads to loss of accuracy and hence the miss rate curve generated using this approach need to be vertically shifted to better match real MRC. Moreover, this approach only works for fully-associative caches, while the modern processors use set-associative caches with finite (e.g. 8 or 16 way) associativity.

Qureshi et al. [15] propose Utility Monitors (UMONs) for tracking miss rate of L2 caches for different ways of an LRU cache, using Mattson stack algorithm. However, due to the high cost of implementation of true-LRU technique, most real-world processors use an approximation of LRU (e.g. pseudo-LRU). Hence, true-LRU based miss rate prediction approaches are not suitable for real-world processors. In contrast, ESTO

uses set-based profiling, and hence, it can easily work with different "approximate-LRU" replacement policies.

Execution Time Estimation: Techniques for estimation of execution time is especially important for high-performance computing applications [16, 17]. Yamamoto et al. [18] propose an execution time prediction method which combines measurement-based execution time analysis and simulation-based memory access analysis. As for memory access analysis, the memory access latency value is estimated in terms of the memory access pattern of a function level and the properties of the target processor cache architecture. However, the authors observe an error up to 64% in ET estimation on Pentium-M processor.

Most methods of computation of L2 cache latency require running the program twice (e.g [8, 9]). Once the program is run, with the assumption of infinite cache and then with finite (real) cache. This method, however, introduces large overhead and is not suitable for real-time applications.

Because of their dynamic behaviors, caches present several challenges in WCET analysis. Several studies have focused on addressing this issue. Li et al. [19] build an Integer Linear Programming solution for WCET estimation problem for direct mapped and set-associative caches, while Ferdinand et al. [20] use abstract interpretation to model the instruction cache behavior for WCET analysis.

Energy Estimation: Dhouib et al. [21] propose a multi-layer power and energy estimation approach for embedded systems. Their approach works by first estimating energy and power consumption of standalone tasks and then adding energy overheads of operating system services such as timer interrupt, inter process communications etc. Zhao et al. [22] present a microarchitectural approach to estimate the energy consumption of embedded operating systems by taking into account the energy spent in system calls and kernel execution paths etc. Our approach is different from these, since we estimate memory sub-system energy under many configurations in a single run.

VIII. Conclusion and Future Work

In this paper, we presented ESTO, a dynamic profiling based approach for estimating application performance and energy consumption under different LLC configurations. We have shown the utility of ESTO for the case when the LLC is an L2 cache, although our approach can can also be applied to an L3 cache. Our future work will focus on making more accurate prediction of impact of cache miss on execution time. This will improve the accuracy of execution time and energy estimation.

References

[1] M. Raju and S. Khaitan, "High performance computing using out-of-core sparse direct solvers," *International Journal of Mathematical, Physical and Engineering Sciences*, vol. 3, no. 2, 2009.

[2] S. Mittal *et al.*, "System generator: The state-of-art FPGA design tool for dsp applications," in *ICEMC2*. Global Education Center, India, 2008.

[3] S. Khaitan *et al.*, "Dynamic load balancing and scheduling for parallel power system dynamic contingency analysis," *High Performance Computing in Power and Energy Systems*, pp. 189–209, 2012.

[4] M. Raju *et al.*, "High performance computing of three-dimensional finite element codes on a 64-bit machine," *Journal of Applied Fluid Mechanics*, vol. 5, no. 2, pp. 123–132, 2012.

[5] A. Pande *et al.*, "Efficient translation of algorithmic kernels on large-scale multi-cores," in *CSE*, vol. 2. IEEE, 2009.

[6] S. Mittal *et al.*, "Design exploration and implementation of simplex algorithm over reconfigurable computing platforms," in *IEEE International Conference on Digital Convergence*, 2011, pp. 204–209.

[7] A. Pande *et al.*, "Efficient mapping and acceleration of AES on custom multi-core architectures," *Concurrency and Computation: Practice and Experience*, 2011.

[8] T. Puzak *et al.*, "Pipeline spectroscopy," in *Workshop on Experimental computer science*. ACM, 2007.

[9] Y. Chou *et al.*, "Microarchitecture optimizations for exploiting memory-level parallelism," in *ACM SIGARCH CAN*, 2004.

[10] S. Mittal *et al.*, "EnCache: Improving cache energy efficiency using a software-controlled profiling cache," in *IEEE EIT*, 2012.

[11] T. E. Carlson *et al.*, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *SC*, Nov. 2011.

[12] X. Li *et al.*, "Accurate timing analysis by modeling caches, speculation and their interaction," in *DAC*, 2003.

[13] S. Mittal *et al.*, "CASHIER: A Cache Energy Saving Technique for QoS Systems," *IEEE VLSID*, 2013.

[14] D. K. Tam *et al.*, "RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations," in *ASPLOS*, 2009.

[15] M. K. Qureshi *et al.*, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO*, 2006.

[16] S. Khaitan and J. McCalley, "High performance computing for power system dynamic simulation," *High Performance Computing in Power and Energy Systems*, pp. 43–69, 2012.

[17] A. Agrawal *et al.*, "A new heuristic for multiple sequence alignment," in *IEEE International Conference on Electro/Information Technology,*, 2008, pp. 215–217.

[18] K. Yamamoto, Y. Ishikawa, and T. Matsui, "Portable execution time analysis method," in *RTCSA*, 2006.

[19] Y. Li *et al.*, "Cache modeling for real-time software: beyond direct mapped instruction caches," in *IEEE RTSS*, ser. RTSS '96, 1996.

[20] C. Ferdinand *et al.*, "Cache behavior prediction by abstract interpretation," *Sci. Comput. Program.*, 1999.

[21] S. Dhouib *et al.*, "Modeling and estimating the energy consumption of embedded applications and operating systems," in *ISIC*, 2009.

[22] X. Zhao *et al.*, "Fine-grained energy estimation and optimization of embedded operating systems," in *IEEE ICESS*, 2008.