Machine Learned Machines: Adaptive Co-optimization of Caches, Cores, and On-chip Network

Rahul Jain
Dept. of Computer Sc. and Engg.
Indian Institute of Technology Delhi
Email: rahuljain@cse.iitd.ac.in

Preeti Ranjan Panda
Dept. of Computer Sc. and Engg.
Indian Institute of Technology Delhi
Email: panda@cse.iitd.ac.in

Sreenivas Subramoney
Microarchitecture Research Lab
Intel Technology India Pvt. Ltd., Bangalore
Email: sreenivas.subramoney@intel.com

***Abstract*—Modern multicore architectures require runtime optimization techniques to address the problem of mismatches between the dynamic resource requirements of different processes and the runtime allocation. Choosing between multiple optimizations at runtime is complex due to the non-additive effects, making the adaptiveness of the machine learning techniques useful. We present a novel method, Machine Learned Machines (MLM), by using Online Reinforcement Learning (RL) to perform dynamic partitioning of the last level cache (LLC), along with dynamic voltage and frequency scaling (DVFS) of the core and uncore (interconnection network and LLC). We show that the co-optimization results in much lower energy-delay product (EDP) than any of the techniques applied individually. The results show an average of 19.6% EDP and 2.6% execution time improvement over the baseline.**

I. Introduction

As the number of available optimization techniques increases, the decision to find the optimal subset becomes difficult because of a non-uniform effect of the optimizations on power and performance. This problem is complicated further on multi-core systems due to global effects on the shared resources. We address this problem of simultaneous application of multiple power optimization techniques on multi-core systems using online reinforcement learning. We show that applying multiple optimization techniques gives superior results.

Runtime resource allocation for optimizing various multi-core system objectives such as reliability, performance, and power [1], [2] has been the focus of research attention in recent years. DVFS and dynamic cache partitioning (DCP) techniques allow adaptive runtime resource allocation with frequency and cache size being the respective resources.

Reinforcement Learning [3] is gaining traction as a popular technique for power management [4]. An initial attempt at online learning based CPU-DVFS used an estimate of the CPU-intensiveness of running applications [5]. Recently, a comparative study showed that co-operative core/uncore DVFS using RL technique gives superior results compared to isolated core and uncore DVFS [6].

Early work on dynamic cache partitioning (DCP) has focused on reducing the shared cache misses [7], [8], while recent work has shown that reducing the off-chip memory bandwidth [9] is more effective. None of these works have considered applying additional power saving techniques such as core or uncore DVFS along with DCP.

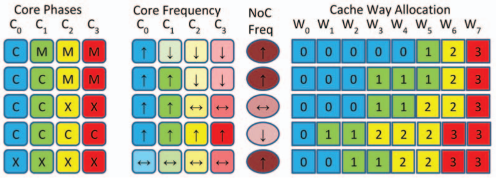
We present the following novel contributions: *1) LLC, Core, and On-chip Network Co-optimization:* We propose to use an Online Reinforcement Learning (RL) based technique, Machine Learned Machines (MLM), for simultaneously applying multiple optimizations through a dynamic partitioning of last level cache (DP-LLC) along with DVFS on the cores and uncore (on-chip network+LLC). We believe this is the first attempt at such a co-optimization problem. *2) Online RL Technique for Dynamic LLC Partitioning:* We propose a novel method using Online RL for partitioning the LLC at runtime. *3) Efficient Metric for Markov Decision Process (MDP) Modeling:* We propose an improved metric for DVFS-MDP modeling as compared to previous work, which results in better EDP and execution time, with a significant decrease in storage overhead. *4) Shared Learning MDP for Cores:* We propose to use a shared Q-Table between the agents running on the cores which further reduces the storage overhead.

II. Motivation for Co-optimzation

Changing phases in applications lead to dynamically varying resource requirements. Figure 1 shows some possible system phases due to the changing program phases along with resource allocation possibilities to the 4-cores. A CMMM system phase has one program in a CPU-intensive phase and other three in memory intensive phases. The system may prefer to run C0 at high frequency, and the other three cores at low frequency, since these cores maybe stalling frequently due to high memory latency. Additionally, the uncore should be run at high frequency due to larger traffic from the three memory-intensive programs. The memory-intensive programs might flush out useful data for the CPU-intensive program, hence, it might be better to exclusively allocate most of the LLC to C0, so that its performance is not degraded. This LLC-way partitioning decision would change dramatically if C1 changes its phase to CPU-intensive (CCMM). Now most of the LLC must be allocated to the two CPU-intensive cores (C0,C1). Similar resource allocation decisions are required for other system phases as shown in Figure 1. This complexity of performing multiple optimizations simultaneously motivates us to use a machine learning based approach to perform the runtime adaptation in response to system phase changes. We

978-3-9815370-6-2/DATE16/ ©2016 EDAA

Fig. 1. Resource Allocation with changing program phases



Core Phases				Core Frequency				NoC Freq	Cache Way Allocation							
C_0	C_1	C_2	C_3	C_0	C_1	C_2	C_3		W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7
C	M	M	M	↑	↓	↓	↓	↑	0	0	0	0	0	1	2	3
C	C	M	M	↑	↑	↓	↓	↑	0	0	0	1	1	1	2	3
C	C	X	X	↑	↑	↔	↔	↔	0	0	0	1	1	2	2	3
C	C	C	C	↑	↑	↑	↑	↓	0	1	1	2	2	2	3	3
X	X	X	X	↔	↔	↔	↔	↑	0	0	1	1	2	2	3	3

C/X/M : CPU/Mixed/Memory Phase; ↑ / ↔ / ↓ : High/Average/Low Frequency

propose to apply online reinforcement learning to address this problem.

III. ONLINE REINFORCEMENT LEARNING

A Markov Decision Process (MDP) is a framework for implementing Reinforcement Learning [3] which allows an autonomous agent to interact with a dynamic environment and learn the optimal control policy based on the rewards earned for various actions by trial and error. MDP is represented by a set of states S, a set of actions A, a state-transition matrix $T(s_t, a_t, s_{t+1})$, and a reward function $R(s_t, a_t, s_{t+1})$, where s_t is the state at interval t, a_t is the currently known optimal action from s_t. In an online RL problem, the agent does not know the values of T and R in advance; that is, it is unaware of the good states or the actions that lead to better rewards. The agent must interact with the environment and learn.

Q-Learning [10] is one of the most popular algorithms in RL used for off-policy online learning. A one-step Q-Learning can be represented by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \times max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where r_{t+1} is the reward received at the start of $t+1$ interval for performing a_t, Q is the Action-Value function, α is the learning rate, and γ is the discount factor. The above equation is free from T and R matrices, and the agent learns Q over time by interacting with the environment. The above methodology approximates the optimal Q and converges to optimality if all state-action pairs are experienced for enough iterations. A typical implementation has Q as a $|S| \times |A|$ table.

Our proposed system, MLM, consists of three types of MDPs: core-MDP, uncore-MDP for performing DVFS, and LLC-MDP for DCP. In the current proposal, the MDP agents do not coordinate but adapt to the changing system states to optimize EDP.

IV. MODELING DYNAMIC CACHE PARTITIONING

A. LLC Fixed Shared Ways Approach

Several researchers have proposed ways to perform DCP of shared caches by allocating cache ways to individual cores [7]–[9]. Figure 2 shows one such configuration where some cache ways are exclusively assigned to cores. This can potentially lead to fragmentation with some cores being allocated more cache than required. To avoid this, we propose a hybrid approach where certain cache ways are always shared and cannot be allocated exclusively to any core. This allows cores with low cache footprint to only use the shared ways. The above approach is implemented by modifying the LRU replacement policy with the search for a core not performed in other cores' exclusive ways.

B. MDP Modeling of Dynamic Way Allocation

We propose to model the cache state as a vector of state of the sharing cores. For a C core shared cache, with each core being in one of S states, the total states for the cache are S^C. The actions of the MDP form a vector representing way allocation/no change/deallocation $(A/N/D)$ request per core. A C-core shared cache would have an action as a C-element vector. For example, in a 4-core system, $[A, N, N, D]$ represents an action where Core C0 should be allocated a way by deallocating from Core C3. With the fixed shared cache ways technique, an additional cache way allocation to a core must be done with a deallocation from another core. Under this constraint, the total number of valid action vectors for a C-core shared cache is given by $\sum_{i=0}^{C/2} \binom{C}{i}\binom{C-i}{i}$.

V. EFFICIENT MODELING OF DVFS

A. TPI-based MDP Modeling

We propose to use Time Per Instruction (TPI) instead of Cycles Per Instruction (CPI), since we are dealing with core-DVFS. We model the MDP states based on the ratio of the TPIs in the current and previous interval. The TPI_{Curr}/TPI_{Prev} ratio represents the degree of performance change due to the last action performed by the MDP agent. This ratio is quantized to obtain a relatively small number of states, in accordance with the modeling feasibility. The MDP has three possible actions from each state: increase/no change/decrease $(I/N/D)$ the frequency.

B. Motivation for Shared Learning MDP

As discussed in Section II, the optimization decisions depend on the program phase and the system phase (other program phases). Since this feature is independent of the program functionality, the MDP-agents running on individual cores can share their learning by using a common Q-Table. This accelerates the learning and keeps the memory overhead independent of the number of cores. A shared Q-Table has communication overheads due to state, action and reward information flow between the cores and the Q-Table. This overhead is very minimal as this communication is done only once per reconfiguration interval.

VI. CO-OPTIMIZATION SYSTEM BUILDING

We proceed to the architecture for co-optimizing the overall system based on the state of different components.

A. Exploration of Reward Function for LLC Partitioning

The MDP-based reinforcement learning allows us to explore various reward functions for DP-LLC in order to select the best setup for the co-optimization approach. We explore the following reward functions: *1) Time-per-instruction (TPI):* Optimizing the DP-LLC for faster execution, i.e., reducing the

Fig. 2. Co-optimization with DVFS Domains, LLC partitions and MDP Agents

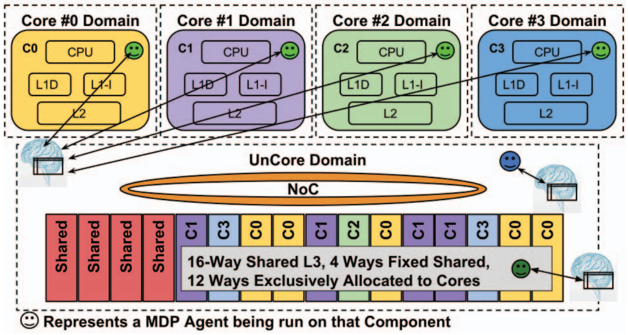


TABLE I
BASELINE NORMALIZED EDP AND EXECUTION TIME RESULTS

	Core DVFS				CU	DP-LLC			MLM
Group	[6]	$[6]^{SL}$	TPI	TPI^{SL}	DVFS	TPI	EPI	Offchip	Co-opt
EDP									
MC	0.944	0.949	1.035	0.922	0.869	0.905	0.908	0.772	0.758
CC	0.981	0.967	0.992	0.98	0.968	1	1.046	1.04	0.97
MX	0.961	0.936	0.863	0.968	0.933	0.769	0.882	0.893	0.793
XX	1.040	0.986	0.993	0.961	0.952	0.972	0.730	0.800	0.629
CX	0.866	0.869	0.864	0.831	0.811	0.854	0.900	0.925	0.868
Avg	**0.958**	**0.941**	**0.949**	**0.932**	**0.907**	**0.900**	**0.893**	**0.886**	**0.804**
EXECUTION TIME									
MC	1.159	1.07	1.157	1.02	1.027	0.951	0.951	0.867	0.947
CC	1.028	1.09	1.075	1.026	1.031	1	1.032	1.03	1.042
MX	1.138	1.058	0.984	1.017	1.086	0.872	0.94	0.951	1.002
XX	1.132	1.104	1.068	1.002	1.072	0.995	0.848	0.884	0.865
CX	0.959	1.062	0.957	0.941	0.949	0.91	0.95	0.969	1.014
Avg	**1.083**	**1.077**	**1.048**	**1.001**	**1.033**	**0.946**	**0.944**	**0.940**	**0.974**

SL: Shared Learning, CU: Core+Uncore DVFS

TPI. *2) Energy-per-instruction (EPI):* Optimizing for lower system energy. *3) Off-chip/DRAM Bandwidth:* Reducing the off-chip bandwidth is an effective metric to perform DP-LLC [9]. Based on this metric, we can optimize for minimal DRAM bandwidth by selecting the appropriate reward function.

L3 has an independent LLC-MDP, which performs DCP among the 4 cores sharing the cache. We use TPI-based core-states, as discussed in Section V-A by quantizing the TPI_{Curr}/TPI_{Prev} ratio into 2 states per core, effectively leading to a 16-state LLC model for a 4-core system, with 19 valid action vectors (Section IV-B). Columns 7-9 in Table I show the EDP and execution time values for the three reward functions. The off-chip bandwidth based reward function gives the best results with improvement of 11.4% and 6% in EDP and execution time respectively and chosen for Machine Learned Machines setup.

B. Core and Uncore DVFS Model

As discussed in Section V, we use a 3-state MDP model for core and uncore DVFS by quantizing the TPI_{Curr}/TPI_{Prev} ratio into 3 states with three possible actions from each state: increase/no change/decrease of the frequency. Figure 2 shows the co-optimization architecture with each core's MDP agent updating the shared Q-Table. The uncore (NoC and L3), has an independent DVFS-MDP running.

We performed experiments (summarized in Columns 2-5 of Table I) to compare the shared learning and isolated MDP approaches for our TPI-based model (Section V-A) and [6]. The results show that the SL is better for both [6] and the TPI-based model. Overall the SL TPI-based core-MDP model shows an average system EDP saving of 6.8% as compared to 4.2% for [6]. The average execution time penalty of TPI-based model is just 0.1% compared to 8.3% for [6]. Additionally, the proposed TPI-based model consumes 87% less storage due to lesser number of states. The core+uncore DVFS is found to be performing better with 9.3% average EDP saving with only 3.3% average execution time penalty (Column 6 of Table I).

VII. EXPERIMENTS

A. Simulation Setup

We used the Sniper Simulator [11] integrated with McPAT [12] and tuned the various parameters using SPEC2006 benchmarks to power/performance data from a commercial processor with a similar configuration. We have used the *pybrain* [13] python library for MDP implementation.

The experiments are performed on a 4-core system with private 32KB L1-I, L1-D, private unified 128KB L2, 1MB shared L3. The core and uncore operate on the following V/F levels: 1.2V/3.2GHz, 1.1/3.0, 1.0/2.7, 0.9/2.4, 0.8/2.0. The system simulation is divided into fixed-time intervals of 500μs, for a total of 1000 million instructions executed, with a 2000 CPU cycle reconfiguration penalty at the interval boundary. At the end of each time interval, the system computes the next-state and reward, which is read by the MDPs whose subsequent actions may result in a reconfiguration for the next interval. Spiliopoulos et al. [14] have categorized the SPEC2006 benchmarks into 3 categories: CPU-bound, Mixed, and Memory-bound. We created benchmarks groups as follows: 4-CPU bound (CC:*gromacs, calculix, h264ref, gamess*), 4-Mixed (XX:*mcf, leslie3d, GemsFDTD, lbm*), 2-Mem, 2-CPU bound (MC:*milc, libquantum, calculix, h264ref*), 2-Mem bound , 2-Mixed (MX: *milc, libquantum, leslie3d, GemsFDTD*), 2-CPU bound, 2-Mixed (CX:*gamess, gromacs, lbm, mcf*).

All the results presented in Table I are normalized to the EDP and executing time metric of the baseline system. The baseline system is a no-optimization system configuration with cores and uncore at the maximum frequency of 3.2 GHz and no exclusive LLC-way allocation.

B. Analysis

Table I is divided into two parts corresponding to EDP and execution time. Column 1 shows the benchmark groups. Columns 2-5 show the EDP and Time values for the four strategies applying only Core DVFS: [6], [6] with Shared Learning(SL), TPI-model with isolated learning, and TPI-model with SL. Column 6 shows the results for core+uncore

Fig. 3. Benchmark Group MC: Core 2,3 Instruction Execution Profile and LLC-Way Assignments

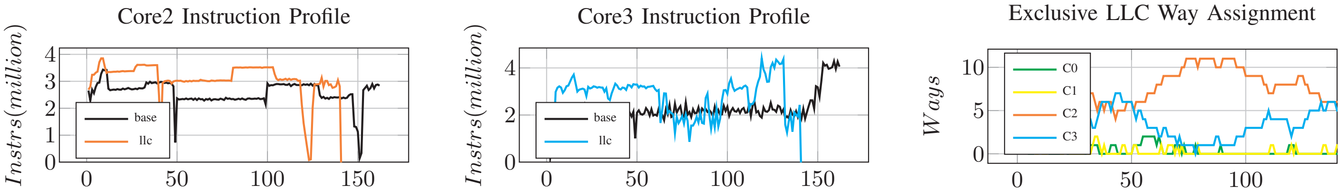
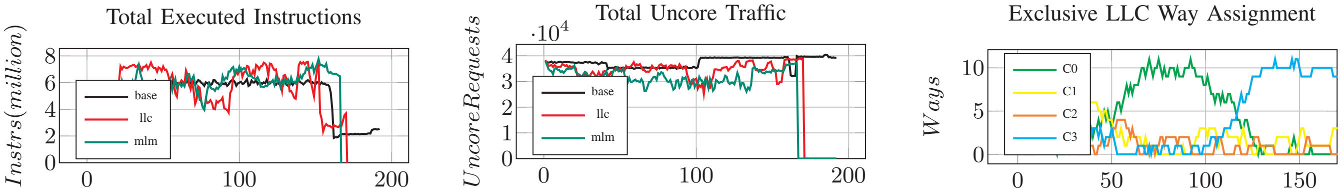


Fig. 4. Benchmark Group XX: Total Instruction Execution Profile, Uncore Traffic Profile and LLC-Way Assignments



DVFS. Columns 7-9 show the results when only LLC partitioning is performed, using three different rewards: TPI, EPI, and off-chip traffic. The last column shows the results for our proposed co-optimization technique, MLM, using SL TPI-model DVFS for cores, TPI-model DVFS for uncore, and off-chip traffic based reward for DP-LLC. MLM gives much better EDP and latency metric, with the average system EDP reduced by 19.6% and 2.6% improvement in time. Due to limited space we present an analysis of only three out of the five benchmark groups: MC (best EDP with DP-LLC), XX (best EDP with MLM), and CC (worst EDP with DP-LLC and MLM).

The MC benchmark group shows almost 23% better EDP and 13.3% better execution time with DP-LLC. The MDP is able to allocate LLC in an effective way to the two CPU-intensive applications. The first two graphs of Figure 3 show how the instruction throughput increased due to DP-LLC for the two applications. The third graph shows the runtime LLC ways allocation to the cores. The XX benchmark group shows 20% and 37% better EDP with LLC (DP-LLC only) and MLM respectively. Figure 4 shows the total instruction execution profile for baseline, LLC and MLM configuration. DP-LLC leads to higher throughput for both LLC and MLM cases. The second graph shows that the uncore traffic is reduced for LLC and MLM case as compared to the baseline case, which allows the MLM case to reduce additional power through uncore DVFS. The CC benchmark group had the maximum IPC, as all four applications are CPU-intensive. This did not allow much core-DVFS optimization opportunities. Also, this group had less uncore traffic, hence there was not much opportunity for dynamic power optimization by uncore-DVFS or uncore traffic reduction by DP-LLC. MLM could only save static power on uncore using DVFS. This benchmark group showed the least improvement in EDP or execution time with all optimization configurations, as shown in Table I.

VIII. Conclusion and Future Work

We proposed and evaluated a machine learning based technique, MLM, and showed that the adaptiveness of the machine learning based approach is effective in handling the complexity of applying multiple optimizations on multi-core systems. Our strategy results in an average improvement of 19.6% EDP and 2.6% execution time. In the future, we plan to study the effect of coordinating multiple agents on the optimizations.

Acknowlegment

We would like to thank Intel, CII and SERB for partial sponsorship of this research.

References

[1] J. Henkel, V. Narayanan, S. Parameswaran, and J. Teich, “Run-time adaption for highly-complex multi-core systems,” in *CODES+ISSS*, 2013.

[2] S. Wildermann, M. Glaß, and J. Teich, “Multi-objective distributed run-time resource management for many-cores,” in *DATE*, 2014.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[4] Y. Wang, Q. Xie, A. Ammari, and M. Pedram, “Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification,” in *DAC-48*, 2011.

[5] G. Dhiman and T. S. Rosing, “Dynamic voltage frequency scaling for multi-tasking systems using online learning,” in *ISLPED-2007*.

[6] D.-C. Juan and D. Marculescu, “Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors,” in *ISLPED-2012*.

[7] M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” in *International Symposium on Microarchitecture*, 2006.

[8] A. Gordon-Ross and F. Vahid, “A self-tuning configurable cache,” in *DAC-44*, 2007.

[9] C. Yu and P. Petrov, “Off-chip memory bandwidth minimization through cache partitioning for multi-core platforms,” in *DAC-47*, 2010.

[10] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[11] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation,” in *SC-2011*.

[12] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *MICRO-2009*.

[13] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, “PyBrain,” *Journal of Machine Learning Research*, vol. 11, pp. 743–746, 2010.

[14] V. Spiliopoulos, G. Keramidas, S. Kaxiras, and K. Efstathiou, “Power-performance adaptation in Intel core i7,” 2011.