Online Learning for Dynamic Control of OpenMP Workloads

Maxime Mirka, Gilles Sassatelli, Abdoulaye Gamatié

Online Learning for Dynamic Control of OpenMP Workloads

Maxime Mirka, Gilles Sassatelli, Abdoulaye Gamatié
LIRMM (CNRS and University of Montpellier)
Montpellier, France
name.surname@lirmm.fr

***Abstract*—Optimizing energy-efficiency of modern multicore compute systems through online control is often regarded as both promising and challenging. In this paper, we propose a dynamic control technique for OpenMP workloads that exploits online energy efficiency measures derived from the OpenMP runtime. The proposed strategy relies on an automatic program phase identification which detects workload execution patterns, used by an online learning back-end. We design a synthetic benchmark template that makes it possible to produce benchmarks with controllable characteristics for mimicking a wide range of workload profiles. Experimental results show improvements on a 20-core server when compared to default Linux governors.**

***Index Terms*—Online learning, energy-efficiency, multicore systems, OpenMP**

I. Introduction

A major challenge of modern compute systems lies in achieving the best possible energy efficiency for both embedded and server-class / HPC systems. While minimizing power consumption is rather easily achieved thanks to widely supported control schemes such as frequency and voltage scaling, directly optimizing energy efficiency requires to relate energy consumption to an amount of compute, or to a fraction of the entire compute job so as derive an efficiency measure.

Existing approaches that aim at optimizing energy-efficiency of compute systems rely on various design techniques as already surveyed in literature [3], [5]. Specifically, dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM) [8] are widely used to decrease the CPU energy consumption. They are both controlled by the OS which usually provides interfaces enabling users to directly control those features (e.g. intel_pstate scaling driver). More recent solutions promote the integration of emerging technologies, e.g. non-volatile memories [21], [22].

Alternatively, performing energy-aware task allocation and mapping optimizations on multicore architectures is also studied in the literature [7], [12]. Among those control methods, we see in recent years a growing interest in using machine learning [6], [15], [24], and more specifically reinforcement learning [14], [16], [20]. The use of reinforcement learning (RL) is proven effective when considering decision making problems. Indeed, it theoretically guarantees overall convergence to the best-performing solution among the available actions, though convergence time is often prohibitive. Hence, it is attractive for non-intuitive decision-making problem, i.e. problems for which no satisfying system model can be built. RL problems require careful definition of not only *state* but also *action* for devising a satisfying controller, as described in [9] and [7] for which action space lies on DVFS and DPM for the former and task allocation for the latter.

This work assumes actions comprising DVFS and active core count. The reward function used in our RL engine is a *relative energy efficiency* metric that is extracted from the OpenMP runtime as described in [4]. This approach has the distinct advantage to require no prior program profiling nor code annotation: it relies on the fundamental concept of *chunks* that OpenMP uses for breaking down job into parallelizable job fragments. A modified OpenMP runtime computes *chunk throughput* referred to as *CpS* (Chunks per second), which is therefore a compute performance measure. From the energy consumption the runtime derives the *CpJ* (Chunks per Joule) metric which expresses for the current workload how many chunks are processed per Joule, i.e. an energy efficiency measure. The same setup as that of the original contribution [4] is here used as a frontend to the RL engine with additional phase detection logic described later on in this paper.

This paper makes the following specific contributions:

- Formalization of the OpenMP metrics i.e. *chunk*, *CpS* and *CpJ*, introduced in [4];
- A demonstration of the importance of managing both frequency and computing resources assignment to a parallelized workload, through a synthetic benchmark analysis;
- An online-learning approach to dynamically control parallelized OpenMP workloads;
- A phase detection module to optimize the previous controller when applied to multiphases application;

Contrary to most recent investigations in the literature that target *Heterogeneous Multicore Systems* (such as ARM big.LITTLE systems) our approach targets server-class SMP systems, i.e. 20-cores Intel Xeon in which no *a priori* assumption can be made. It demonstrates the effective adaptation to workload changes. We obtain an energy efficiency gain of up to 67%, when applied to a synthetic multiphase benchmark, while potential gains for single-phase program are up to 4.7x, compared to default Linux governors.

II. Related Work

There exist several machine learning-based approaches for bettering energy efficiency through dynamic control. In [16]

Maeda-Nunez et al. proposed PoGo, an adaptive energy minimization approach used as a Linux governor and tested on an embedded system. A Q-Learning algorithm is implemented as a decision unit, and the overall technique demonstrates energy savings compared to the existing Ondemand Linux governor [23]. This work has been extended in [19] where the proposed method considers both intra- and inter-application changes involving transfer-learning techniques. The method appears promising for multicore systems, even though it assumes one task per core. In [20], a Q-learning based approach is proposed for idle period management of multicore processors showing the efficiency of RL for that application. Finally, Basireddy et al. proposed in [15] a workload-aware runtime management method for improving energy efficiency in HPC systems. They implemented a binning based learning algorithm [13] to design the decision unit, and used hardware performance counters for the workload characterization. In Alessi et al. [11], the authors proposed an OpenMP specific approach that consists in extending the existing OpenMP API with *Energy Savings* features. It allows taking decision directly at runtime for energy minimization.

None of the aforementioned works target optimizations of OpenMP parallel applications through runtime using RL. The present work aims at investigating this opportunity and shows promising results on energy efficiency improvement through a suitable resource allocation at runtime based on RL.

III. Chunks Formalization

A. Description of the chunk metric

An OpenMP chunk is defined as an iteration of a loop flagged with an OpenMP directive, thereby parallelizable as per the Programming Model specification.

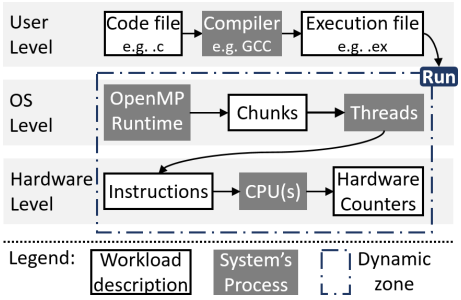


Fig. 1: Execution steps of OpenMP workloads.

Figure 1 shows a conceptual representation of a typical OpenMP use case from user level to hardware level. The flow depicts the process from the source code to the execution on the multicore system equipped with hardware counters. Chunks, being loop iterations, are hardware agnostic. Once the application compiled for a given hardware architecture, chunks are handled by the runtime that distributes chunks to threads for parallel processing. Threads in turn execute a given number of iterations (i.e. chunks) onto processor cores, whose behaviour is monitored by the hardware counters, producing execution statistics.

B. Formalization

We focus the formalization on parallel workloads, and thereby leave aside the serial code which is not subject to parallel execution. Parallel workloads can be described as a finite number of chunks, that can be executed either in series or in parallel. Let us denote by $\mathcal{I}$ the set of instructions supported by the considered execution platforms.

a) Parallel workload: A parallel workload $Par = \{C\}$ is a set of chunks $C = \{i_k\}$. Each i_k represents an instance of an instruction that belongs to $\mathcal{I}$.

b) Architecture: We define an execution architecture $Arc = (Proc, Mem)$ of a compute system as the combination of the processing elements $Proc$ and the associated memory resources Mem. Let $\alpha = \{Arc_k\}$ denote the set of all possible architectures. An architecture can be monitored through hardware counters and energy consumption sensors. Let us consider $V = \langle v_1, v_2, ..., v_n \rangle$ as a vector of values corresponding to measured hardware counters and energy consumption. For instance, these values can be the number of cache misses, hits, memory access such as read and write, etc. The set of all such vectors is denoted by $\mathcal{V}$.

c) Execution Characterization: We define the execution function $Ex : \mathcal{I} \times \alpha \rightarrow \mathcal{V}$, which outputs a vector of metrics, given the execution of an instruction on an architecture.

From this definition, the execution of a chunk $C = \{i_k\}$ on an architecture $a \in \alpha$ produces a vector $V = \langle v_1, v_2, ..., v_n \rangle$ as follows: $\forall i_k \in C, Ex(i_k, a) = \langle v_1^k, v_2^k, ..., v_n^k \rangle$ such that

$$v_1 = \sum_{\forall i_k} v_1^k, v_2 = \sum_{\forall i_k} v_2^k, ..., v_n = \sum_{\forall i_k} v_n^k$$

Hence, a chunk is a workload description of coarser granularity than instructions. The result of the execution of a chunk, corresponds to the sum of the execution of all instructions included into the chunk. Therefore, monitoring chunks gives a view of the workload execution as a whole, whereas global tracking using hardware counters can only be done after data fusion, as counters give only specific information about independent events.

d) Parallel execution of a workload: When executed, a parallel workload Par will be dispatched among a number of threads according to the target architecture and user decision. We define $\mathcal{T} = \{th_1, th_2, ...th_n\}$ the set of threads $th_{k,(k\in 1..n)}$ used to execute Par. Let us denote by $q_{k,(k\in 1..n)} \in \mathcal{P}(Par)$ the set of chunks assigned to a thread th_k for execution, such that: $q_1 \cup q_2 \cup ... \cup q_n = Par$

e) Performance and energy-efficiency: Given a parallel workload Par, its execution time δ_{Par} is given by the maximum value of $\delta_{th_{k,(k\in 1..n)}}$, where $\delta_{th_k} = \sum_{C_i \in q_k} \delta_{C_i}$ represents the execution time of all chunks in q_k assigned to thread th_k.

Similarly, the energy consumption ϵ_{Par} of a parallel workload Par is defined by the sum of $\epsilon_{th_{k,(k\in 1..n)}}$, where $\epsilon_{th_k} = \sum_{C_i \in q_k} \epsilon_{C_i}$ represents the execution energy for all chunks in q_k assigned to the thread th_k.

The amount of chunks executed per second can be easily calculated: it is defined as the *Chunk per Second* (CpS). Sim-

ilarly, the amount of chunk executed per Joule consumed can be derived: it is defined as the *Chunk per Joules* (CpJ). These two metrics introduced in [4] enable to describe respectively the performance and energy efficiency of parallel OpenMP workloads. Therefore, we define the next two metrics:

$$Perf(Par) = \frac{1}{\delta_{Par}} * \sum_{q_k \in \mathcal{P}(Par)} |q_k| \quad (1)$$

and

$$EnergyEff(Par) = \frac{1}{\epsilon_{Par}} * \sum_{q_k \in \mathcal{P}(Par)} |q_k| \quad (2)$$

Formula (1) defines the chunk-based performance metric when executing a parallel workload, while formula (2) indicates its energy-efficiency. They are respectively defined in terms of CpS and CpJ.

Discussion: We formalize chunks as a relevant metric to monitor parallel workload execution at runtime. Moreover, for performance and energy efficiency monitoring purpose, CpS and CpJ metrics are defined based on the chunks.

In this work we only consider parallel workloads. However, sequential workloads can be seen as mono-threaded parallel workloads i.e. $\mathcal{T} =$ singleton. Hence, following this formalization, one could easily apply formula (1) and (2) by simply replacing the $|q_k|$ terms by 1. Indeed, a sequential workload is analog to a parallel workload comprised of a single chunk.

IV. Application Characterization and Energy Efficiency Optimization

A. Synthetic benchmark template

In order to assess potential gains in term of energy efficiency on the chosen symmetric dual-socket system we design a synthetic parametrizable benchmark template from which we derive benchmarks with different profiles. The template is built on 2 consecutive and tunable code fragments covering memory-intensive operations and compute-intensive operations. Thus, applications are built to have different behaviours, according to both their CPU intensity and memory intensity. Fig. 2 describes the benchmark template.

```
00  #include <omp.h>
01  // Initialization
02  Some environment definitions
03  // Parallelization
04  omp_set_num_threads(nthreads);
05  // OpenMP directive
06  #pragma omp parallel for schedule(dynamic, 1)
07  for (i=0; i<n; i++){
08      for (j=0; j<100; j++){
09          if (j < coef){
10              MEM-intensive code fragment
11          }
12          else{
13              CPU-intensive code fragment
14          }
15  }
16  return 0;
```

Fig. 2: Benchmark template.

The memory intensive code block performs various operations on large arrays such as additions, copy, permutations. The intensity of the memory demand depends on the arrays' size, dimension, and the randomness of the access patterns to the arrays. The compute-intensive code block is made of linear algebra and combination of arithmetic / trigonometric functions, involving high computational methods such as Fourier serial analysis and various other floating-point operations. The amount of stress applied to the CPU(s) depends on the number of function calls, and their properties. Hence, those blocks are elementary codes representative of the two characteristics we want to emphasize. Finally, to derive a given benchmark, the ratio between CPU and memory intensity can be set by modifying the "coef" value.

B. Synthetic Applications Characterization

From the benchmark template described above, 6 benchmarks are produced: *C100M0*, *C98M2*, *C96M4*, *C90M10*, *C80M20*, *C0M100*. The *CxMy* naming reflects the balance between the two targeted stress modes: *x* is the percentage of compute intensity and *y* that of memory intensity. Therefore, the *C0M100* and *C100M0* benchmarks are purely memory intensive and CPU intensive, respectively. We assign threads to up to 19 cores of the 20-core Xeon server, the 20^{th} being assigned to the metrics collection process. All of the values reported in the sequel are read out from the Intel hardware performance monitoring counters using the Intel PCM tool [1], and depicted through the net charts shown in Fig. 3. Table I lists all of the considered metrics, sorted according to their nature in term of either compute or memory orientation. Fig. 3 shows that each application has a unique profile, and stresses the CPU and the memory, with various impacts on the counters. Indeed, CPU-intensive benchmark *C100M0* has all of the correspondingly flagged counters at their maximum value. Conversely, the memory-intensive benchmark *C0M100* has all of the memory-oriented counters at full scale.

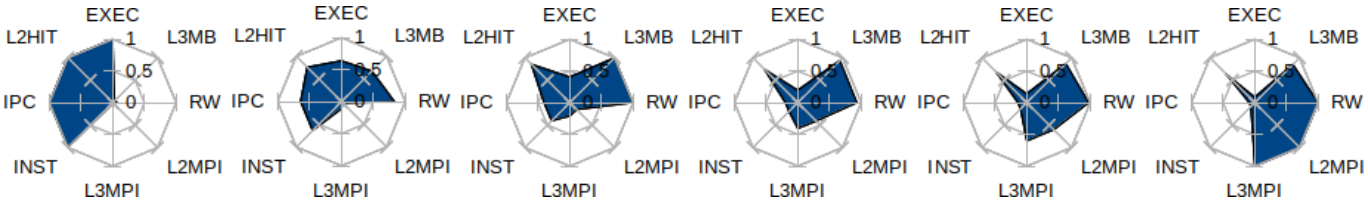
C. Energy Efficiency

For each of the 6 proposed benchmarks we perform an exhaustive characterization, i.e. run the application with each possible configuration (frequency, number of cores) and report the average CpJ over the entire benchmark duration. *Note:* The energy consumption value used to compute CpJ is read out from the Intel's RAPL model specific registers (MSRs) [10].

The results are depicted in Fig. 4, where the best configuration is annotated. Note the variety of optimal configurations among the applications. It emphasizes the duality between

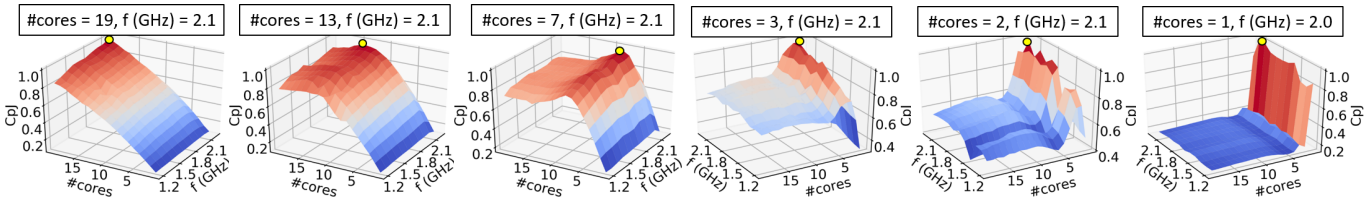
CPU intensity oriented	Memory intensity oriented
IPC = instructions per CPU cycle	RW = MEM Read and Write
EXEC = instructions per nominal CPU cycle	L3MB = L3 cache external memory bandwidth
L3HIT = L3 (read) cache hit ratio	L2MPI = number of L2 (read) cache misses per instruction
INST = Instructions retired	L3MPI = number of L3 (read) cache misses per instruction

TABLE I: Intel PCM counters description.



(a) C100M0 (b) C98M2 (c) C96M4 (d) C90M10 (e) C80M20 (f) C0M100

Fig. 3: Applications profiling according to PCM counters.



(a) C100M0 (b) C98M2 (c) C96M4 (d) C90M10 (e) C80M20 (f) C0M100

Fig. 4: Applications energy efficiency (i.e. CpJ) characterization and best configuration.

CPU intensiveness and memory intensiveness within the applications. Indeed, the two extreme applications (*C100M0* and *C0M100*) possess two opposed best configurations CpJ-wise regarding the number of compute resources allocated, i.e. 1 core for the memory intensive application and 19 cores for the CPU intensive application (maximum core count available, with 1 core per thread).

To observe the impact of applying the best configuration of an application while running on a compute system, we compare the average CpJ and CpS with executions with Linux Powersave, Performance, Ondemand and Conservative governors. Results are reported on Table II.

We obtain potential gains in energy efficiency from 10% to 469%, compared to usual Linux governors. Performance loss is observed for the most CPU intensive applications, but it remains below 20%, which is reasonable comparing to the significant improvements in energy efficiency. The main issue with Linux governors is the lack of *thread to core* control. It highlights the benefits controlling the resource allocation of a parallel application. Hence the use of online learning described later to dynamically adapt the compute configuration.

Benchmark		C100M0	C98M2	C96M4	C90M10	C80M20	C0M100
Best Conf. i.e. Reference	CpJ	3866	2505	1581	818	517	336
	CpS	303k	170k	90k	40k	25k	12k
vs. Performance	CpJ	10%	25%	29%	95%	60%	442%
	CpS	-12%	-11%	-19%	21%	1%	151%
vs. Powersave	CpJ	16%	24%	33%	44%	75%	469%
	CpS	75%	59%	49%	56%	92%	355%
vs. Ondemand	CpJ	10%	20%	32%	18%	75%	433%
	CpS	-12%	-14%	-17%	-1%	50%	193%
vs. Conservative	CpJ	10%	20%	29%	32%	56%	469%
	CpS	-12%	-14%	-19%	-16%	1%	160%

TABLE II: Energy efficiency (CpJ) and performance (CpS) gains of the benchmarks' best configuration, when compared to Linux governors: Powersave, Performance, Ondemand and Conservative.

V. Control System for Stationary Workloads

Here we describe a dynamic control system that performs both *neural network training* and *configuration control* at runtime. The system is depicted in Fig. 5.

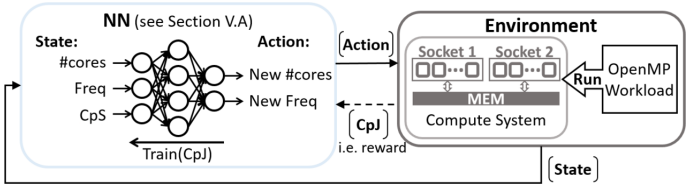


Fig. 5: Control system design.

It is based on the reward concept used in reinforcement learning (RL) among others. Here, we only train our network to perform combinatorial inference, that is to say a purely current state-dependent best action prediction (not function of the previous state). The framework has been developed with genericity in mind such that RL (therefore taking past system states into account) is supported, though this is beyond the scope of this paper. The environment represents the multicore compute system running OpenMP workloads. For each possible state value the quality of each action (i.e. configuration) is evaluated using the reward function that is a direct function of the CpJ metric. Our approach towards obtaining an efficient controller relies on a compact state definition for ensuring quick exploration and convergence. For that reason state is being defined as the compute configuration (i.e. current frequency and active cores count) alongside CpS value.

A. Decision Making Process

The online decision making process is based on learning by experience i.e. learning-by-doing. In this terminology, decisions are called *actions* and the environment is defined by *states*. Decisions start to be taken after the end of the learning process, known as *exploration phase*. During the exploration phase random actions are taken, in order to find the best action for each state. Actions are rewarded according to their benefits.

The advantage of using a neural network (NN) here lies in the implicit ability to perform interpolation i.e. ability to infer an action for an unknown state based on the knowledge acquired during the exploration phase.

In our context, the state is defined by the configuration of the system coupled with the CpS value i.e. the state space is not discrete. The action is a new configuration to apply, and the reward is the resulting CpJ. The Tensorflow API [17] is used to build the neural network, using Keras [2] as frontend.

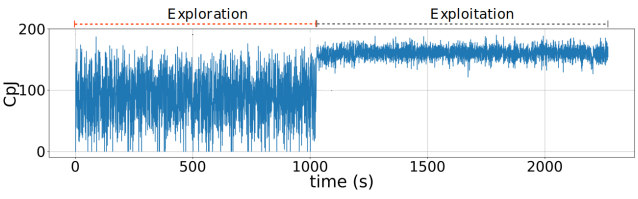
B. Results and Discussion

The design is evaluated with the DGEMM benchmark [18], on the same Xeon server. An ad-hoc exaustive search has revealed the best configuration for this workload to be 18 cores at 2.1GHz with a mean CpJ = 166. The sampling period is set to 500ms. The action set is defined as follows:

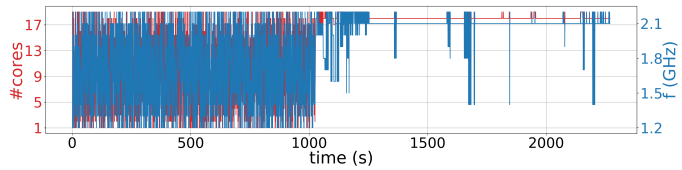
- Number of cores (i.e. #core): 19 cores available, allocated equally between the two sockets.
- Frequency: from 1.2GHz to 2.2GHz.

In our experiments, the exploration phase is arbitrarily set to 2048 iterations (i.e. 1024s) which happens to be relatively short in comparison to the usual amount of data used to train neural networks in general. No data-augmentation technique has been used for bettering the quality of the dataset. We nevertheless observe on Fig. 6 that it is enough for our system to reach near-optimal performance. Fig. 6a shows a plot *CpJ* over the execution. Note the rather stable CpJ, within 3% of the known best configuration. Some fluctuations are visible, incurring transient sporadic modifications of the chosen configuration.

Gains: When compared to the default Linux governors available on our Intel Xeon server, our method shows the following gains: 10%, 17%, 11% and 10%, respectively compared to the *Performance, Powersave, Ondemand, Conservative* Linux governors. Those results are similar to the *C100M0* benchmark, as DGEMM is a compute-intensive workload.



(a) Energy efficiency evolution.



(b) Online learning actions i.e. configuration state.

Fig. 6: Online learning variables evolution, when running DGEMM benchmark.

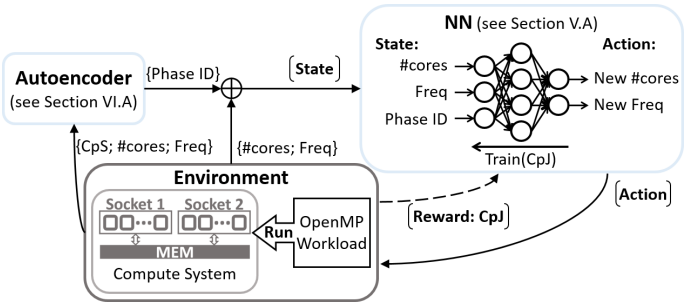


Fig. 7: Control system design.

VI. Control System for Multi-Phases Workloads

Stationary workloads such as those presented previously are not representative of real-life applications such as multimedia decoders, or cloud applications which usually go through many types of workloads, hence different execution phases. To adapt our control system to multiphases workloads, we propose a phase detection engine. Phase information is retrieved using a specific autoencoder (AE) design described below. The resulting system is depicted on Fig. 7.

A. Autoencoder

Autoencoders are specific neural network architectures used to produce data encodings in an unsupervised way. They are trained in supervised mode, such that the produced output is identical to the input. The key concept of this type of neural networks relies on reducing the dimensionality of the input, through successive layers of neurons until the internal "bottleneck" layer that marks the boundary between the encoder and the decoder. This leads to a compact internal representation of the input (lossy compression) that is then expanded in the decoder until the output. Intuitively, backpropagation algorithm (training) attempt to find the most prominent features in the input vector across the training dataset, such that loss (reflecting the difference between input and output) is minimized.

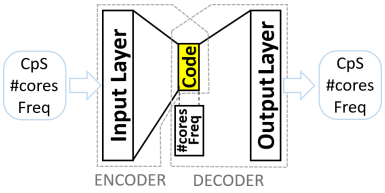


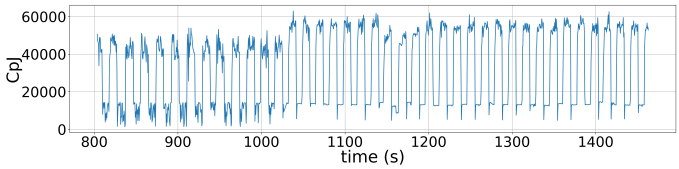
Fig. 8: Designed autoencoder.

Here, we use a specific autoencoder architecture proposed in [4], illustrated in Fig. 8. It is trained offline to reproduce the CpS value and the system's configuration data (i.e. the number of cores and the frequency). Its internal layer is expected to extract the information about the phase of the running application, referred to as *code* in Fig. 8 .

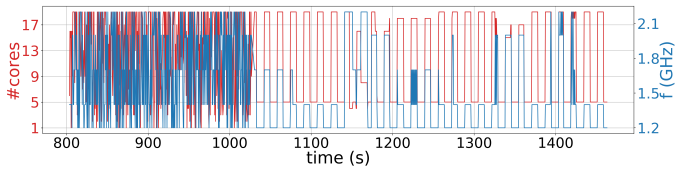
B. Results and Discussion

The overall design is evaluated with a two-phases synthetic application. It is derived from two synthetic workloads different from those of the benchmark suite. One phase is

CPU intensive similar to *C100M0* and its best configuration is 19 cores, f=2.2GHz. The second phase is memory intensive, with the following best configuration: 4 cores, f=1.3GHz. Experimental setup is same as for V-B.



(a) Energy efficiency evolution.



(b) Online learning actions.

Fig. 9: Online learning plot for the 2-phase benchmark.

As shown in Fig. 9a, the 2 phases are clearly visible through their respective distinct CpJ levels, resulting from the different natures of the performed computations. Though not visible, the autoencoder already produces phase information from the early beginning of the execution. The online controller neural network is thereby trained with this information, until time 1024s. We observe from that point on that the system operates as expected, uniquely identifying the 2 phases and adapting the configuration. The online controller selected 19 and 5 cores for the 2 phases, which matches the known best configuration for each. Frequency however oscillates between 1.2GHz and 1.4GHz which is significantly off compared to the known best configuration, suggesting a longer training time would help further improve energy efficiency.

Gains: When compared to the default Linux governors, our method shows the following global gains: 51%, 7%, 51% and 50% respectively compared to the *Performance, Powersave, Ondemand and Conservative* Linux governors. Potential gains of the individual phases have been assessed through offline characterization and extending the learning could result in gains ranging 34% and 136% compared to the default governors.

VII. Conclusion and Perspectives

In this paper, we extend the work introduced in [4], where we proposed energy efficiency metrics coupled with preliminary work on online workloads analysis. We propose a dynamic control method of parallel OpenMP workloads based on online learning, with an autoencoder as back-end for workload phase detection. The resulting system shows excellent capabilities in finding the best configuration, without prior knowledge of the running workload. Experimental results reveal potential energy savings by up to 4-fold for some synthetic benchmarks compared to default Linux governors. Similar results can be obtained on multi-phase workloads.

Future works: In this work we consider only HPC oriented benchmark, with runtimes in hours. Hence, the neural network training periods has not been discussed. This aspect would certainly require investigations in future works. Other future works rely on demonstrating the effectiveness of the system on various real-world applications w.r.t. constraints such as quality of service. Further studies will consist in investigating other compute system architectures such as heterogeneous multicore platforms, as a proof of versatility. Finally, the impact of concurrent workloads should be considered to address multi-programming issue.

References

[1] Intel performance counter monitor. https://software.intel.com/en-us/articles/intel-performance-counter-monitor. Accessed: 2019-09-15.

[2] François Chollet et al. Keras. https://keras.io, 2015.

[3] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *IJCAET*, 6(4):440–459, 2014.

[4] M. Mirka et al. Automatic Energy-Efficiency Monitoring of OpenMP Workloads. In *ReCoSoC, York, UK, July 1-3*, pages 43–50, 2019.

[5] A.-C. Orgerie *et al.* A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, March 2014.

[6] A. Gamatié *et al.* Empirical model-based performance prediction for application mapping on multicore architectures. *J. Syst. Archit.*, 98:1–16, 2019.

[7] B. Donyanavard *et al.* Sparta: Runtime task allocation for energy efficient heterogeneous manycores. In *2016 Int. Conf. on Hardware/Softw. Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, Oct 2016.

[8] B. K. Reddy *et al.* Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores. *IEEE Transactions on Multi-Scale Computing Systems*, 4(3):369–382, July 2018.

[9] Bhatti *et al.* Hybrid power management in real time embedded systems: an interplay of DVFS and DPM techniques. *Real-Time Sys.*, 2011.

[10] D. Hackenberg *et al.* An energy efficiency feature survey of the intel haswell processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 896–904, May 2015.

[11] F. Alessi *et al.* Application-level energy awareness for openmp. 2015.

[12] G. Georgakoudis *et al.* Scalo: Scalability-aware parallelism orchestration for multi-threaded workloads. *ACM Trans. Archit. Code Optim.*, 14(4):54:1–54:25, 2017.

[13] G. Liu *et al.* Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems. In *IEEE Int. Conf. on Computer Design (ICCD)*, pages 54–61, Oct 2013.

[14] H. Shen *et al.* Learning based dvfs for simultaneous temperature, performance and energy management. In *ISQED*, 2012.

[15] K. R. Basireddy *et al.* Workload-aware runtime energy management for hpc systems. In *Int. Workshop on Optimization of Energy Efficient HPC & Distributed Systems (OPTIM 2018)*, May 2018.

[16] Luis Alfonso *et al.* Pogo: an application-specific adaptive energy minimisation approach for embedded systems. In *HiPEAC Workshop on Energy Eff. with Heterogenous Comp. (EEHCO)*, 2015.

[17] Martín Abadi *et al.* Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on OSDI*, 2016.

[18] Piotr Luszczek *et al.* Introduction to the HPC Challenge Benchmark Suite. Dec 2004.

[19] R. A. Shafik *et al.* Learning transfer-based adaptive energy minimization in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(6):877–890, June 2016.

[20] Rong Ye *et al.* Learning-based power management for multi-core processors via idle period manipulation. In *17th Asia and South Pacific Design Automation Conference*, pages 115–120, Jan 2012.

[21] S. Senni *et al.* Emerging non-volatile memory technologies exploration flow for processor architecture. In *IEEE Computer Society Annual Symposium on VLSI, Montpellier, France*, page 460, 2015.

[22] S. Senni *et al.* Exploring MRAM technologies for energy efficient systems-on-chip. *IEEE J. Emerg. Sel. Topics Circuits Syst.*, 6(3):279–292, 2016.

[23] V. Pallipadi *et al.* The ondemand governor: past, present and future. In *Proceedings of Linux Symposium, vol. 2, pp. 223-238*, 2006.

[24] Yen-Lin Chen *et al.* Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems. In *Sensors*, 2018.