PowerMan: Online Power Capping by Computationally Informed Machine Learning

1st Johannes Gebert†
gebert@hlrs.de

2nd Daniel Barry‡
dbarry@vols.utk.edu

3rd Jack Dongarra‡
dongarra@icl.utk.edu

Abstract—Appropriately adjusting the power draw of computational hardware plays a crucial role in its efficient use. While vendors have already implemented hardware-controlled power management, additional energy savings are available, depending on the state of the machine. We propose the online classification of such states based on computationally informed machine learning algorithms to adjust the power cap of the next time step. This research highlights that the overall energy consumption can be reduced significantly, often without a prohibitive penalty in the runtime of the applications.

Index Terms—Power Capping, Machine Learning, High-Performance Computing

I. Introduction

Climate change considerations [1] require modern and ever faster processors to include the power draw and energy consumption as major factors in the design and use of computing systems [2], [3]. It is also the most important metric used not only by by the TOP500 [4], [5], Green500 [6], and SPEC Power [7] lists, but also by wide range of supercomputing power consumption research [8], [9]. Vendors recognized this situation and early on built power management into their hardware [10]–[13]. However, the energy consumption can be further reduced without a loss of performance by setting an appropriate power cap, e.g., when the system's workload is memory-bound [14], [15].

Manual determination of the current compute load and predicting the next one prevents efficient online interaction with a running application and thus preventing even greater energy savings. Therefore, automating the process of recognizing and predicting compute loads is a key design goal of our offline-online machine learning framework.

In 2017, the powercap component was introduced in the PAPI library [14], which enabled to both read performance counters [16]–[18] and also write to them in order to set power caps that establish either the long- or short-term power draw limits.

It was unexpected that the power cap can be lowered by significant margins without loss of performance [14]. In a first step, we show in Fig. 1 that this phenomena can be reproduced on a modern architecture [19] without

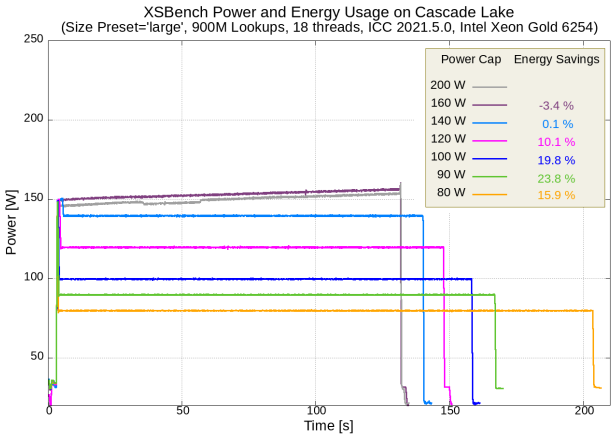

Fig. 1. Power capping can reduce the energy consumption, given as the enclosed surface, of contemporary processors.

High-Bandwidth Memory (HBM) [20] featured on Intel's Knights Landing (KNL) [21].

II. Classification for Online Power Capping

Reducing power draw and, more importantly, saving energy in scientific applications requires online determination of the processor's workload due to the dynamic nature of execution profiles [22].

A. Requirements

Our experiments use x86-64 architecture due to its support for user-controlled power caps, however, the following criteria may apply to GPU accelerators as well [3], [23]:

- Readily available hardware counters,
- Multi-threaded architectures,
- Minimal computational overhead,
- Online classification,
- Socket-wide power capping with on-package classification.

B. Classifiers's Overview

Hardware events are measured with a sampling rate of $f_s = 5$ Hz which implicitly smooths the characteristics of the performance counters as shown in the artifact description (AD).

Tracing events in this way results in a discrete time series with a classification rate r_{cl}:

$$r_{cl} = t_s \cdot n_p, \tag{1}$$

†High-Performance Computing Center Stuttgart; University of Stuttgart; Stuttgart, BW, Germany

‡Innovative Computing Laboratory; University of Tennessee; Knoxville, TN, USA

where the number of (time) points per instance was n_p, each one average over time step t_s.

Performance profiles of software packages vary depending on their algorithms and problem sizes [24]–[26]. Forecasting-based prediction may lack reliability since it assumes recurrence of observed patterns [27], [28]. However, determining changes internal to compute- or memory-bound workloads [29] is itself a classification [30]. To this end, self-supervised learning based on sktime [31] fits the amount and structure of known data and expected characteristics well.

C. Our Computationally Informed Approach

Decisions on setting the power cap must be made as a function of the interaction between the machine and the running code. We propose a computationally informed approach to set the power cap of the subsequent sampling interval. Analyzing power signatures from Fig. 2 suggests a consistent architectural behavior for applications while in their memory-bound portion.

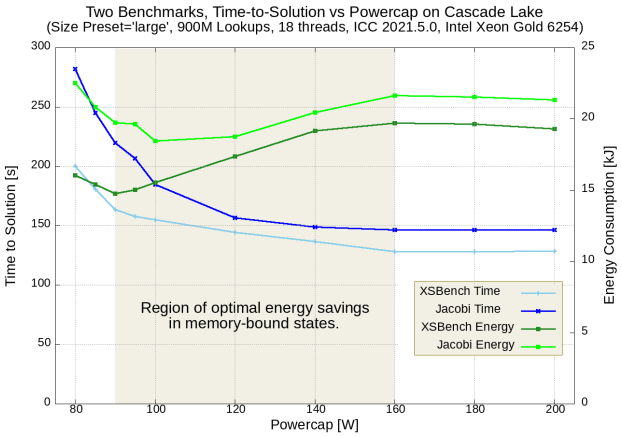

Fig. 2. Energy signatures for memory-bound states.

We observed a latency of power limit enforcement in the regime of $t_{pcap} = 0.15$s depending on the power step ΔP requested. We did not adjust the short term power limit as it was best left to the hardware's power management.

A reasonable compromise of fidelity and adjustment interval of the power cap was found in the $1\frac{1}{2}$s interval of the classification rate r_{cl} where $\frac{1}{2}\text{s} \leq r_{cl} \leq 2\text{s}$.

D. Implementation Remarks

Data for offline training were obtained with PAPI according to the procedure given in AD. The online classification is invoked with the Python's C API [32], NumPy's low-level Array API [33] and sktime's classifier module thus retaining full flexibility of Python's end-to-end ecosystem.

III. Results

We choose XSBench [34], [35] as a lightweight benchmark that closely resembles a real-world scenario. Fig. 3 shows that automated power capping based on computationally informed machine learning algorithms saves energy.

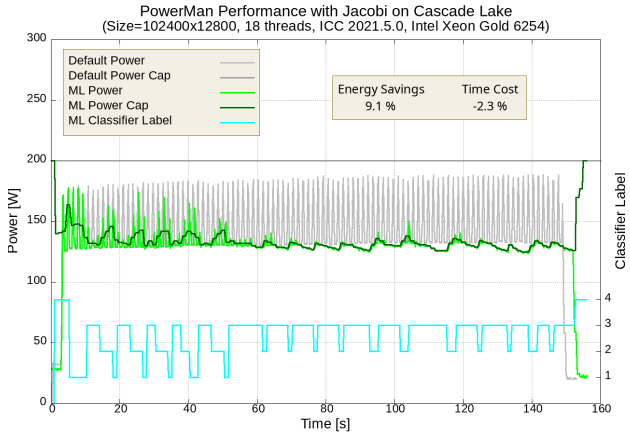

Fig. 3. The energy consumption can be reduced significantly by online power capping.

IV. Limits and Outlook

The approach requires optimizations for reducing computational overhead as well as to meet the demands of other architectures and programs.

We suggest the enumerated use cases for online adjustment of the power of a given program: 1

1) Adjusting the target power of an already running program for computers to meet a changing overall power target.
2) Minimizing energy consumption for temporally uncritical applications.
3) Wall-time aware energy savings by accepting a loss of performance of, for example, no more than 3%.

Computationally aware, software-agnostic approaches based on hardware counters can enable a classifier-firmware for embedding in on-chip power management units. Software-aware [36] classifiers in contrast are considered promising as well, because estimations about the pending runtime of an identified operation lead to even more precise adjustments of the power limit.

V. Conclusion

Online Power Capping by Computationally Informed Machine Learning can be considered feasible.

VI. Acknowledgments

This research was supported by the National Science Foundation Office of Advanced Cyberinfrastructure (OAC) CSE Dir. for Comp. & Info Sci. & Eng. under Grant No. 2004541.

References

[1] W. Marx and R. H. A. T. et al., "Which early works are cited most frequently in climate change research literature? a bibliometric approach based on reference publication year spectroscopy," Scientometrics, vol. 110, p. 335–353, 2017.

[2] W. L. Bircher and K. J. Lizy, "Analysis of dynamic power management on multi-core processors," in ICS '08: Proceedings of the 22nd annual international conference on Supercomputing, 2008, pp. 327–338.

[3] V. M. W. et al., "Measuring energy and power with papi," in 41st International Conference on Parallel Processing Workshops, 2012, 2012, pp. 262–268.

[4] H. W. Meuer, E. Strohmaier, J. J. Dongarra, and H. D. Simon, TOP500 Supercomputer Sites, 32nd ed., November 2008, the report can be downloaded from http://www.netlib.org/benchmark/top500.html).

[5] H. W. Meuer, E. Strohmaier, J. J. Dongarra, H. D. Simon, and M. Meuer, "TOP500 Supercomputing Sites," http://www.top500.org/, 2021.

[6] W.-C. Feng and K. W. Cameron, "The Green500 list: Encouraging sustainable supercomputing," IEEE Computer, vol. 40, no. 12, pp. 50–55, 2007.

[7] SPEC, "The SPEC power benchmark," 2008, see www.spec.org/power_ssj2008/.

[8] K. W. Cameron, R. Ge, and X. Feng, "High-performance, power-aware, distributed computing for scientific applications," IEEE Computer, vol. 38, no. 11, pp. 40–47, 2005.

[9] W.-C. Feng, R. Ge, and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems," in 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 05), Denver, CO, USA, 2005.

[10] E. Rotem and A. Naveh, "Power-management architecture of the Intel microarchitecture code-named Sandy Bridge," IEEE Micro, vol. 32, pp. 20–27, 2012.

[11] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar, "Power and thermal management in the intel core duo processor," Intel Technology Journal, pp. 109–122, 2006.

[12] W. L. Bircher and S. Naffziger, "Amd soc power management: Improving performance/watt using run-time feedback," in Proceedings of the IEEE 2014 Custom Integrated Circuits Conference, 2014, pp. 1–4.

[13] M. W. et al., "Architecting for power management: The IBM® POWER7™ approach," in HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, 2010, pp. 1–11.

[14] A. Haidar, H. Jagode, P. Vaccaro, A. YarKhan, S. Tomov, and J. Dongarra, "Investigating power capping toward energy–efficient scientific applications," Concurrency and Computation: Practice and Experience, vol. 31, 2019.

[15] A. Haidar, H. Jagode, A. YarKhan, P. Vaccaro, S. Tomov, and J. Dongarra, "Power-aware computing: Measurement, control, and performance analysis for Intel Xeon Phi," Concurrency and Computation: Practice and Experience, pp. 1–7.

[16] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," Tools for High Performance Computing, pp. 157–173, 2009.

[17] H. Jagode, A. YarKhan, A. Danalis, and J. Dongarra, "Power management and event verification in papi," Tools for High Performance Computing, pp. 41–52, 2015.

[18] H. Jagode, A. Danalis, H. Anzt, and J. Dongarra, "Papi software-defined events for in-depth performance analysis," The International Journal of High Performance Computing Applications, vol. 33, no. 6, pp. 1113–1127, 2019.

[19] M. A. et al., "Cascade lake: Next generation Intel Xeon scalable processor," pp. 29–36, 1 March-April 2019.

[20] S. Salehian and Y. Yan, "Evaluation of Knight Landing High Bandwidth Memory for HPC workloads," pp. 1–4, 2017.

[21] A. Sodani, R. Gramunt, J. Corbal, H. Kim, K. V. andS. Chinthamani, S. Hutsell, R. Agarwal, and Y. Liu, "Knights landing: Second-generation Intel Xeon Phi product," IEEE Micro, vol. 36, pp. 34–46, 2016.

[22] R. E. Grant, J. H. Laros, M. Levenhagen, S. L. Olivier, K. Pedretti, L. Ward, and A. J. Younge, "Evaluating energy and power profiling techniques for HPC workloads," in Eighth International Green and Sustainable Computing Conference (IGSC), Orlando, FL, USA, 2017, pp. 1–8.

[23] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding GPU power: A survey of profiling, modeling, and simulation methods," ACM Comput. Surv., vol. 49, no. 3, pp. 1–27, Sept. 2017, art. 49.

[24] E. Strohmaier, "Supercomputer benchmarks: A comparison of HPL, HPCG, and HPGMG and their utility for the TOP500," in Workshop on Sustained Simulation Performance 24, High-Performance Computing Center Stuttgart, 2016.

[25] R. C. Agarwal, F. G. Gustavson, and M. Zubair, "Exploiting functional parallelism of POWER2 to design high-performance numerical algorithms," IBM Journal of Research and Development, vol. 38, no. 5, pp. 563–576, Sept. 1994.

[26] B. Kågström, P. Ling, and C. V. Loan, "GEMM-based Level 3 BLAS: high-performance model implementations and performance evaluation benchmark," ACM Trans. Math. Softw., vol. 24, pp. 268–302, 1998.

[27] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. El-Shishiny, "An empirical comparison of machine learning models for time series forecasting," Econometric Reviews, vol. 29, pp. 594–621, 2010.

[28] G. Bontempi, S. B. Taieb, and Y. L. Borgne, "Machine learning strategies for time series forecasting," Business Intelligence, vol. 138, 2013.

[29] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," IEEE Micro, vol. 23, no. 6, pp. 84–93, Nov.-Dec. 2003.

[30] S. Kotsiantis, I. Zaharakis, and P. Pintelas, "Machine learning: A review of classification and combining techniques," Artificial Intelligence Review, vol. 26, pp. 159–190, Nov. 2006.

[31] F. Király, "sktime," Zenodo, vol. v0.13.0.

[32] M. Hu and Y. Zhang, "The Python/C API: evolution, usage statistics, and bug patterns," in 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 532–536.

[33] C. R. Harris, K. J. Millman, and S. J. van der Walt et al., "Array programming with NumPy," Nature, vol. 585, pp. 357–362, 2020.

[34] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "XSBench - the development and verification of a performance abstraction for Monte Carlo reactor analysis," in PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future, Kyoto, Japan, 2014.

[35] J. R. Tramm and A. R. Siegel, "Memory bottlenecks and memory contention in multi-core Monte Carlo transport codes," Annals of Nuclear Energy, vol. 82, pp. 195–202, 2015.

[36] A. Elhabbash, M. Salama, R. Bahsoon, and P. Tino, "Self-awareness in software engineering: A systematic literature review," ACM Trans. Auton. Adapt. Syst., vol. 14, pp. 1–42, June 2019, art. 5.