DNPC: A Dynamic Node-level Power Capping Library for Scientific Applications

Sahil Sharma[a,1], **Zhiling Lan**[a], **Xingfu Wu**[b], **and Valerie Taylor**[b]

[a] Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 ; [b] Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439

As the race to exa-scale computing accelerates, power consumption continues to be a critical challenge. While several technologies are available for power management, balancing energy efficiency and application performance during execution remains an important topic of research. In this study, we develop an open-source library called DNPC for dynamically controlling an application's package-level power consumption during execution. Given a specific performance degradation threshold, DNPC aims to minimize power consumption by following the application's power profile and adjusting the power cap accordingly. Further, DNPC is able to estimate the online performance of an application under capping, and predict its estimated performance degradation relative to an uncapped run. In this paper, we present an overview of the library, followed by a case study to illustrate the use of DNPC with an Exascale Computing Project proxy application on a production supercomputer at Argonne National Laboratory.

High-Performance Computing | Power Management | Online Performance Monitoring

Current high-performance computing (HPC) systems require a large amount of electricity to operate system components. As HPC systems increase in scale and capability, the cost of supplying power to these systems will grow as well. In fact, it is estimated that the energy cost of a large-scale system during its lifetime can surpass the equipment itself (1). This introduces the urgent need for energy efficient computing through controlling power consumption. However, there is not one clear solution to achieve this result. The PowerStack initiative provides a holistic and extensible power management framework (2). It defines power management interfaces at three specific levels: cluster scheduler, job-level runtime systems, and node-level managers. The Global Extensible Open Power Manager (GEOPM) from Intel is a framework for exploring power and energy optimization. It provides an energy profiling and monitoring tool as well as multi-node power balancer to optimize power across nodes based on work imbalance. RAPL exposes a number of model-specific registers (MSRs) that can be used to monitor energy and set power limits on different parts of a chip (3). PoLiMEr provides C functions to leverage RAPL for application-level power monitoring and capping (4).

While the tools above provide useful monitoring or certain power management capabilities, little work has been done to investigate when and how much power capping should be applied during application execution. In particular, an open question is: *given a specific bound for application performance degradation, how can we dynamically adjust the power cap during execution so as to minimize power usage?* PoLiMEr includes functions to set and monitor power caps, but these must be set manually and are limited to functions visible within the application source code. This requires offline analysis of the application to know when and how much to set the power cap to. Additionally, every time an adjustment to the capping scheme is made, the application source code must be modified and recompiled. Gholkar et al. developed UPScavenger, a runtime system to save power by dynamically detecting phase changes and automatically tuning uncore frequency during application execution (5). UPScavenger targets uncore power saving and is not designed for package level power saving.

In this study, we aim to address this problem by developing *DNPC, a Dynamic Node-level Power manager and Capping library for high-performance computing.* A key challenge is to estimate online performance during application execution. Ramesh et al. (6) found that online performance is highly correlated with the figure of merit; however their work is limited to the applications for which a figure of merit is defined. Our work addresses this issue through fine-grained power profiling and active hardware counter analysis. DNPC directly incorporates PoLiMEr and PAPI (7). It includes several adaptive algorithms for runtime performance prediction and power capping. DNPC is also able to detect and follow patterns in the power curve, called power phases, without offline profiling. Another key feature is that DNPC can estimate the online performance of the application based on execution time relative to uncapped performance, and use this estimation to control the predicted amount of performance degradation from power capping. The library can utilize the detection of phases to set the appropriate package power cap to minimize power during execution. DNPC is easy to use: to use the library, a user only needs to insert a couple of lines of code into their application code.

DNPC Overview

DNPC allows easy dynamic package power capping of single node applications written in C or C++. Currently, there are no plans to extend DNPC to monitor multiple nodes at once.

[1] To whom correspondence should be addressed. E-mail: ssharma18@hawk.iit.edu

```c
#include "dnpc.h"
int main(int argc, char *argv[])
{
    /* application code */
    MPI_Init(&argc, &argv);
    dnpc_init();
    /* application code */
    dnpc_finalize();
    MPI_Finalize();
    /* application code */
}
```

Listing 1. An example to illustrate the use of DNPC with an application

It works by combining the energy monitoring and hardware counter monitoring of PoLiMEr and PAPI, respectively. It employs one of any number of built-in or user-supplied algorithms during run-time to periodically and dynamically set a power cap.

Algorithms. While DNPC uses PoLiMEr and PAPI to monitor an application's behavior, its main benefit comes from using the data from these libraries every polling cycle as inputs to a run-time algorithm that outputs a power cap to set. Only one algorithm can be used for an application's execution. While the user can easily add their own algorithm, four different dynamic capping algorithms are already implemented within DNPC. More will be added as we continue investigating power management. Their designs are based on state machines in order to be easy to interpret and to have minimal overhead. These four algorithms are briefly explained here.

1. The first algorithm examines the current frequency reading to decide if power capping is needed, and looks at the change in instructions per cycle (IPC) to determine phase changes before adjusting the power cap.

2. The second algorithm builds upon the first by estimating the current online performance and adjusting the amount to set the cap based on that estimate and the degradation threshold.

3. The third algorithm is similar to the second, but exchanges the change in IPC for the change in the ratio of micro-operations to normal instructions to determine phase changes.

4. The last algorithm is an iteration of the third, switching the order in which the micro-operation ratio and current frequency are checked in determining phase changes and power capping candidates.

Using DNPC. DNPC is implemented in C and contains about 1000 lines of code. The core interface only consists of two functions: one to initialize the library, and the other to finalize it. Before modifying the application code, PoLiMEr, PAPI, and an implementation of MPI should be compiled and available on the system. PoLiMEr should be compiled with the `TIMER_OFF` flag, as signal-based timers will conflict with DNPC's internal timer. While DNPC is a user-level library, PoLiMEr internally uses RAPL for power capping, so the user should contact their system administrator for privileges to read/write certain MSRs as outlined in PoLiMEr's documentation. After the appropriate setup, the only modifications to the source code required are shown in Listing 1. After the application is run, it will output four files with the energy, performance, and counter data recorded.

The flexibility of DNPC comes in its configuration through environment variables. This allows different environment settings within a batch submission script to be set without having to recompile the library or the application. Some important environment variables that can be set are the dynamic capping algorithm, the performance degradation threshold, the polling interval, PAPI counters to collect, and more.

Adding a new power capping algorithm is simple as well. The only steps required are to first write a new function within the `dnpc.c` file, add the function to the `dnpc.h` file, and finally add it as case within the polling function's algorithm `switch` statement.

We plan to release the library as open source on GitHub after we extensively evaluate the library with more applications.

Application Case Study

We now show a case study of applying DNPC to an application to illustrate the benefits of the library. The application is an Exa-scale Computing Project (ECP) proxy app (8) called MiniQMC. MiniQMC is a "mini" version of QMCPACK (9) designed to be used for benchmarks and optimization testing, and it is available freely on GitHub or from the ECP website. MiniQMC uses different quantum Monte Carlo algorithms in order to calculate the total energy of a quantum mechanical system. Different algorithms and kernels present opportunities within the power profile for power cap adjustments.

MiniQMC was configured with DNPC, MPI, 64 OpenMP threads, and a problem size of 128 atoms and 1536 electrons. We then performed the following experiment on a single node of the Cray XC40 supercomputer Theta at Argonne National Laboratory. Each Theta node is equipped with 192 GiB of DRAM, 16 GiB of MCDRAM, and an Intel KNL 7230 processor with 64 cores. On the same node, we first ran MiniQMC without any power capping, then again using dynamic power capping based on Algorithm 3 from DNPC. Each configuration had three trials taken, and the results of one of the trials for each is shown in Figure 1. In the evaluation, the median value from the trials was taken for each metric.

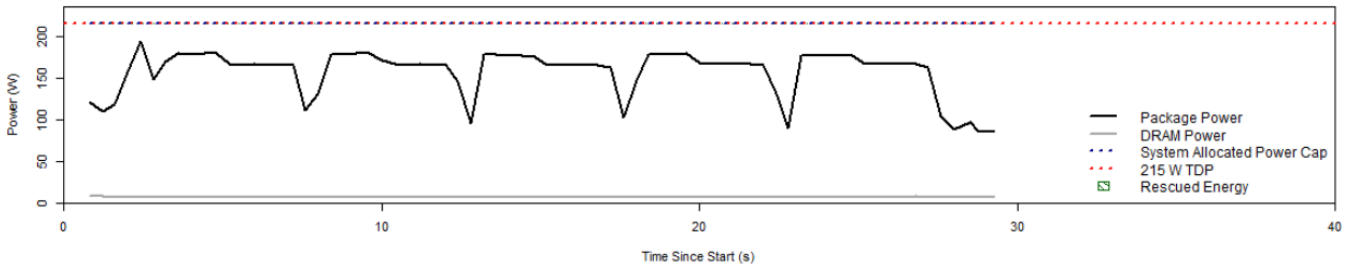
Looking at Figure 1b, it can be seen that the dynamic power cap curve follows the phases of the power curve and adapts accordingly. Also important to note is that the power cap curve becomes more conservative as time goes on due to the estimated performance approaching the degradation threshold of 5%. Three metrics are used in our evaluation:

1. **Execution Time**
   It is defined as how much time the application runs for.

2. **Total Energy**
   It is defined as the total amount of energy consumed by the package and the DRAM during the execution of the application with a dynamic package power cap.

3. **Rescued Energy**
   It is defined as the area between the power given by the

(a) MiniQMC Without Power Capping



(b) MiniQMC With Dynamic Power Capping (Alg. 3), and Degradation Threshold of 5%

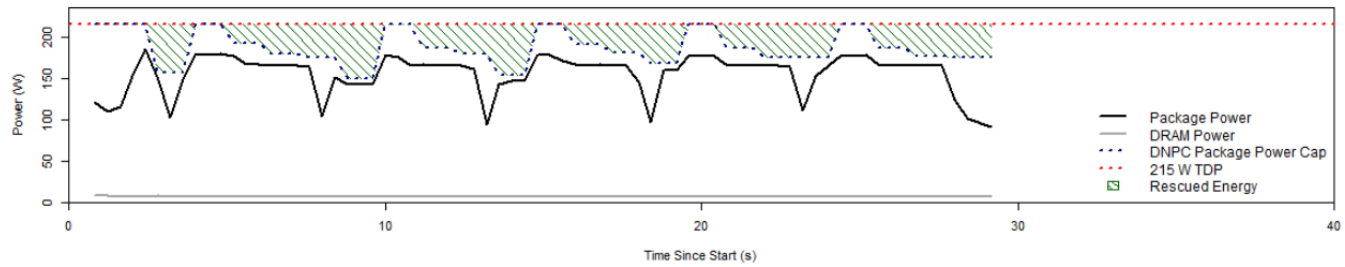


Fig. 1. Power profiles of executions of MiniQMC. (1a) Trial without power capping. (1b) Trial with dynamic power capping from DNPC. In 1a, we assume the system-level power manager has given the node a power cap set to the TDP of the processor. In 1b, where there is a dynamic power cap that is typically below TDP, the system-level power manager can reclaim the rescued energy and distribute to other jobs.

system-level power manager and the power cap set by DNPC. In our experiment, the Thermal Design Power (TDP) of 215 Watts is used as the default power cap given by the system-level power manager. This metric is similar to the stranded power metric used in the context of power grids (10).

The first two metrics are calculated using the median value of the dynamic power cap configuration normalized by the median value of the configuration without a power cap. The third metric is different in that it is normalized by the total energy from the configuration without a power cap rather than the rescued energy (since it is zero for the baseline configuration).

While not directly an evaluation metric, the accuracy between the (normalized) execution time predicted by DNPC and the actual execution time is important to consider. Predicting the total execution time at run-time, or online performance, is difficult as many factors can affect the runtime. DNPC simplifies this prediction by not predicting the actual time, but by predicting the estimated percent of performance degradation from using a power cap compared to not using one. It updates this prediction every polling cycle, and a user can supply a constraint that DNPC will try to stay under while power capping.

Looking at the results of MiniQMC under a dynamic power cap, there are two main takeaways. The first being that the dynamic package power cap may not always bring energy savings. The (normalized) total energy for MiniQMC under Algorithm 3 was 1.00, which means the total energy used under a dynamic power cap was the same as the typical case without one. In this case, the increased execution time from the power capping caused the total energy to be the same despite the lower average power usage. This could mean that this MiniQMC specifically is less receptive to power capping, or that this algorithm performs poorly in terms of energy savings. In either case, more tests on different applications will have to be taken in order to verify these findings. However, the rescued energy presents a great benefit from dynamic power capping. The baseline configuration of MiniQMC does not have a power cap, so we assume that the power cap seen by the power manager above the node to be the TDP of the processor. Therefore, the configuration without a power cap in this case has no rescued energy. The rescued energy for MiniQMC under Algorithm 3 was 0.19, or an amount of energy equivalent to 19% of the total energy used in the configuration without a dynamic power cap was reclaimed. Although the total energy used did not decrease, in a multi-node system the rescued energy could be allocated to other nodes.

The second takeaway being that DNPC can estimate the online performance with close accuracy. MiniQMC's (normalized) execution time under Algorithm 3 was 1.05, while its online estimation for that metric was 1.06. Further, since it was given a performance degradation threshold of 5%, this shows it was also able to enable the dynamic power cap while staying within that constraint.

ACKNOWLEDGMENTS. This work is supported in part by US National Science Foundation grants CCF-1618776 and CCF-1801856. We acknowledge Argonne Leadership Computing Facility for use of the Cray XC40 Theta machine.

References

1. Peter Kogge, S. Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, and Robert Lucas. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Techinal Representative*, 15, 01 2008.
2. Christopher Cantalupo, Jonathan Eastep, Siddhartha Jana, Masaaki Kondo, Matthias Maiter, Aniruddha Marathe, Tapasya Patki, Barry Rountree, Ryuichi Sakamoto, Martin Schulz, and Carsten Trinitis. A strawman for an hpc powerstack. Technical report, 2018.
3. Howard David, Eugene Gorbatov, Ulf Hanebutte, Rahul Khanna, and Christian Le. Rapl: Memory power estimation and capping. In *ISLPED*, pages 189–194, 01 2010. .
4. Ivana Marincic, Venkatram Vishwanath, and Henry Hoffmann. Polimer: An energy monitoring and power limiting interface for hpc applications. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, E2SC'17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351324. . URL https://doi.org/10.1145/3149412.3149419.
5. Neha Gholkar, Frank Mueller, and Barry Rountree. Uncore power scavenger: A runtime for uncore power conservation on hpc systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362290. . URL https://doi.org/10.1145/3295500.3356150.
6. S. Ramesh, S. Perarnau, S. Bhalachandra, A. D. Malony, and P. Beckman. Understanding the impact of dynamic power capping on application progress. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 793–804, 2019. .
7. Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-11261-4.
8. Ecp proxy apps suite, 2021. URL https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/.
9. Jeongnim Kim, Andrew D Baczewski, Todd D Beaudet, Anouar Benali, M Chandler Bennett, Mark A Berrill, Nick S Blunt, Edgar Josué Landinez Borda, Michele Casula, David M Ceperley, Simone Chiesa, Bryan K Clark, Raymond C Clay, Kris T Delaney, Mark Dewing, Kenneth P Esler, Hongxia Hao, Olle Heinonen, Paul R C Kent, Jaron T Krogel, Ilkka Kylänpää, Ying Wai Li, M Graham Lopez, Ye Luo, Fionn D Malone, Richard M Martin, Amrita Mathuriya, Jeremy McMinis, Cody A Melton, Lubos Mitas, Miguel A Morales, Eric Neuscamman, William D Parker, Sergio D Pineda Flores, Nichols A Romero, Brenda M Rubenstein, Jacqueline A R Shea, Hyeondeok Shin, Luke Shulenburger, Andreas F Tillack, Joshua P Townsend, Norm M Tubman, Brett Van Der Goetz, Jordan E Vincent, D ChangMo Yang, Yubo Yang, Shuai Zhang, and Luning Zhao. QMCPACK: an open sourceab initioquantum monte carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter*, 30(19):195901, apr 2018. . URL https://doi.org/10.1088/1361-648x/aab9c3.
10. K. Kim, F. Yang, V. M. Zavala, and A. A. Chien. Data centers as dispatchable loads to harness stranded power. *IEEE Transactions on Sustainable Energy*, 8(1):208–218, 2017. .