Auto-tuning for Energy Usage in Scientific Applications

Ananta Tiwari, Michael A. Laurenzano, Laura Carrington, Allan Snavely

Performance Modeling and Characterization Laboratory
San Diego Supercomputer Center

Abstract. The *power wall* has become a dominant impeding factor in the realm of exascale system design. It is therefore important to understand how to most effectively create application software in order to minimize its power usage while maintaining satisfactory levels of performance. In this work, we use existing software and hardware facilities in order to tune applications to minimize for several combinations of power and performance. The tuning is done with respect to software level performance-related tunables (cache tiling factors and loop unrolling factors) as well as for processor clock frequency. These tunable parameters are explored via an offline search in order to find the parameter combinations that are optimal with respect to performance (or delay, D), energy (E), energy×delay ($E \times D$) and energy×delay×delay ($E \times D^2$). These searches are employed on a parallel application that solves Poisson's equation using stencil computations. Stencil (nearest-neighbor) computations are very common operations in today's scientific applications. We show that the parameter configuration that minimizes energy consumption can save, on average, 5.4% energy with a performance loss of 4% when compared to the configuration that minimizes runtime. Furthermore, with the work presented in this paper, we provide evidence for the existence of opportunities to auto-tune for energy in parallel applications.

1 Introduction

As the HPC community prepares to enter the era of exascale systems, a key problem that the community is trying to address is the *power wall* problem. The power wall arises because as compute nodes (consisting of multi/many-cores) become increasingly powerful and dense, they also become increasingly power hungry. This problems this creates are two-fold; it is more expensive to run compute nodes due to the energy they require and it is difficult/expensive to cool them.

Going forward, power-aware computing research in the HPC community will focus in at least two main areas. The first is to develop descriptive and universal ways of describing power usage, either by direct measurement or through explanatory models. Inexpensive, commercially-produced devices such as WattsUp? Pro [3] or more customized frameworks such as PowerMon2 [4] or PowerPack [13] can help measure power and energy consumption. Modeling energy usage through combinations of architectural parameters with performance counters [26] or other resource usage information [21] also fall in this category. The second thrust, which invariably depends on the first, is to attempt to minimize the amount of energy required to solve various scientific problems. This includes the use of Dynamic Voltage Frequency Scaling (DVFS) technique to exploit processor underutilization due to memory stalls [12,19] or MPI inter-task load imbalances in large scale applications [12], improvements in the process and design of hardware, or software-based techniques that change some feature of application behavior in order to lower some energy-related metric [20].

In this work we explore the optimization space of energy usage and performance on a stencil computation, which is an important kernel in many HPC applications [1]. We use a compiler-based methodology to generate and select among alternative mappings of computational kernels that reduce energy consumption. A unique feature of this work is the combined exploration of the effect of code-transformation level tunables and CPU clock frequency scaling[2] on the overall energy consumption of the system during an application run. In other words, clock frequency is considered to be one of the tunables along with code-transformation parameters such as tiling factors and unrolling factors. This approach can be used to help answer the following questions:

1. Can simple loop transformation strategies such as loop blocking, unrolling, data copy etc. have an impact on energy consumption?
2. Can we use search heuristics to find a code-variant that performs better in terms of energy consumption?
3. Is there a trade-off between energy consumption and execution time of an application? In other words, does the execution time of an application suffer when our primary goal is to optimize for energy consumption?

In this paper we take a first concrete step towards answering these questions. The study presented here takes a search-based offline auto-tuning approach. We start by identifying a set of tunable parameters for different potential performance bottlenecks in an application. The feedback driven empirical auto-tuner monitors the application's performance and power consumption and adjusts the values of the tunable parameters in response to them. When the auto-tuner requires a new *code variant* in order to move from one set of parameter values to another, it invokes a code generation framework [8] to generate that code variant. The feedback metric values associated with different parameter configurations are measured by running the target application on the target platform. The methodology is thus offline because the tuning adjustments are made between successive full application runs based on the observed power consumption for code-variants.

[1] The prevalance of stencil computations in DARPA Ubiquitous High Performance Computing (UHPC) challenge applications is documented in [10].

[2] Frequency scaling can be used to reduce energy consumption.

2 Motivation

In this section, we demonstrate that there are opportunities for power and energy consumption auto-tuning. We use an implementation of the Poisson's equation solver (described in more detail in Section 3.4) as a test application. One of the computational hotspots on this application is the relaxation function, which uses a 7-point stencil operation within a triply nested loop. The two outer-most (`i` and `j`) loops of this function are blocked (using blocking factors `TI` and `TJ` respectively) for better cache usage following Rivera et. al.'s tiling scheme [24]. Better cache utilization can reduce overall power usage because it reduces data movement costs.

We run the solver with different combinations of `TI` and `TJ` and measure the overall power usage for each combination. The experiment was conducted on an Intel Xeon E5530 workstation (more description in Section 3). We ran the application on 8 cores for different combinations of the blocking factors. Clock frequencies for each of the cores were kept fixed at the highest available level, 2.4GHz. We normalized the measured energy consumption for each combination of `TI` and `TJ` with respect to the energy consumed by compiler optimized original implementation (non-blocked version compiled with the `-O3`). Figure 1 shows these results. The interaction between energy consumption and tiling optimization is interesting and complex. From the energy consumption point of view, long slender tiles (with `TI`>`TJ`) are preferable. The best tiling configuration uses 30% less energy than the compiler optimized original implementation. Moreover, there exists a fairly large "good" energy consumption area in the figure. These results imply that relatively naive tuning does not result in anything near optimal energy usage and that something is needed to guide application execution toward the optimal solution. We show in the remainder of this work that auto-tuning is a practical and useful methodology for doing this.

3 Experiments

To drive the tuning process we use Active Harmony [9, 27], which is a search-based auto-tuner. Active Harmony treats each tunable parameter as a variable in an independent dimension in the search (or tuning) space. Parameter configurations (admissible values for tunable parameters) serve as points in the search space. The objective function values (feedback metrics) associated with points in the search space are gathered by running and measuring the application on the target platform. The objective function values are consumed by the Active Harmony server to make tuning decisions.

For tunable parameters that require new code (e.g. unroll factors), Active Harmony utilizes code-transformation frameworks to generate code. The experiments reported in this paper use CHiLL [8], a polyhedral loop transformation and code generation framework. CHiLL provides a high-level script interface that auto-tuners can leverage to describe a set of loop transformation strategies

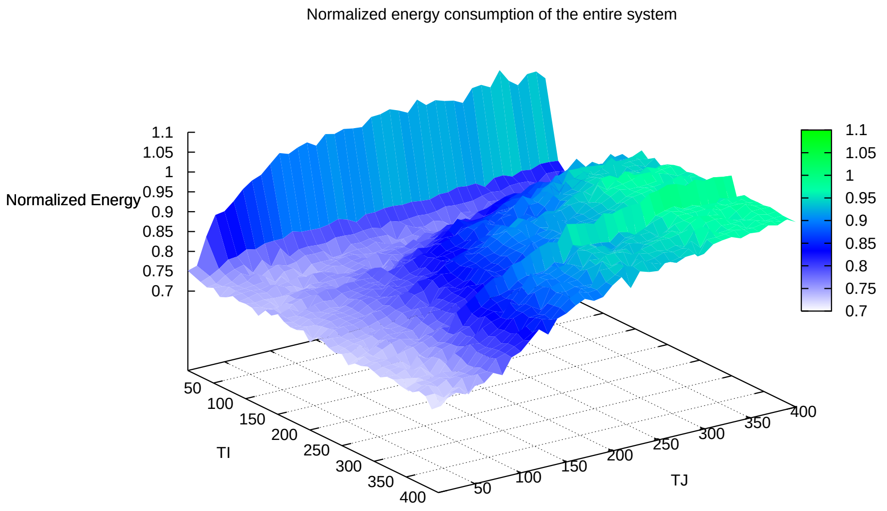

Fig. 1. Normalized energy consumption of the entire system for 8-core experiment. Figure is easier to see in color.

for a given piece of code. More details on offline auto-tuning using Active Harmony and CHiLL are described in [27]. Both Active Harmony and CHiLL are open-source projects.

3.1 Power/Energy Measurement

We measure the energy consumption of a system using the WattsUp? Pro power meter [3]. The power meter is a fairly inexpensive device and, at the time of this writing, costs less than $150. This device measures the AC power being consumed by the entire system. We have implemented a command line interface on top of the wattsup driver to monitor and calculate the overall energy usage of an application.

3.2 Auto-tuning Feedback Metric

The most common feedback metric used by auto-tuners is application execution time, which can also be expressed as runtime delay with respect to some baseline. For energy auto-tuning, however, we need a feedback metric that combines power usage with the execution time of a given program. There has been a lot of debate about the appropriateness of different combinations of power and performance [5, 7, 14] in investigating energy consumption reducing techniques in today's architectures. All of them hinge on how much the delay in execution time should be penalized in return for lower energy.

In this work, we use four different feedback metrics: E (total energy), ED (energy×delay), ED^2 (energy×delay×delay) and T (execution time). Total energy (E) is derived by multiplying the average power usage by the application execution time. E does not penalize execution time delay at all. T penalizes only execution time delay with no credit for saving energy. Between these extremes, ED and ED^2 metrics put more emphasis on the total application execution time than the total energy metric. The appropriateness of which metric to use depends on the overall goal of the tuning exercise and one could certainly consider a user-set set delay penalty per job. We think these 4 are enough to characterize our methods and and the optimization space.

3.3 Experimental Platform

The experiments were conducted on an Intel Xeon E5530 workstation. The E5530 has 2 quad-core processors. Each core has its own 32KB L1 cache and 256KB L2 cache. Each of the quad-core processors has a shared 8MB L3 cache (for a total of 16MB of L3 for the 8 cores). Each of the 8 cores can be independently clocked at 1.60GHz, 1.73GHz, 1.86GHz, 2.00GHz, 2.13GHz, 2.26GHz, 2.39GHz or 2.40GHz. Processor clock frequency is changed using the `cpufreq-utils` package [1] that is available with many popular Linux distributions.

3.4 Results for Poisson's Equation Solver (PES)

Poisson's equation is a partial differential equation that is used to characterize many processes in electrostatics, engineering, fluid dynamics, and theoretical physics. To solve for Poisson's equation on a three-dimensional grid, we use a modified version of the parallel implementation provided in the KeLP-1.4 [2] distribution. The application is written in C++ and Fortran. The implementation uses the redblack successive over relaxation method to solve the equation. The core of the computational time is spent on two kernels: the relaxation function, which uses the stencil computation (described in Section 2), and the error calculation function, which calculates the sum of squares of the residual over the 3D grid. The error calculation portion of the code is optional and can be turned on or off using a command line parameter. These functions are tuned separately in offline modes.

For the relaxation function, we use the tiling optimization scheme that we described in Section 2. Active Harmony determines the dimension of the tiles and the appropriate CPU frequency — a three dimensional search space. The error function optimization requires new code generation. For this function, we tile all three loops and the innermost loop is unrolled. Thus, the search space for error function optimization is five dimensional — four code transformation parameters and the CPU frequency.

Table 1 shows the results for relaxation function tuning. For each of the feedback metric, we conducted three auto-tuning runs. The table shows the results for the best parameter configurations and also the averages across three runs. The data provided in the table are normalized with respect to the timing

Table 1. PES-Relaxation kernel results (400^3 grid, 8-cores)

	$E(\mu)$	$ED(\mu)$	$ED^2(\mu)$	$T(\mu)$
Best Configurations				
speedup (time)	2.18 (2.13)	2.25 (2.39)	2.24 (2.39)	2.27 (2.40)
norm. ener. usage	0.42	0.43	0.43	0.44
Averages				
norm. speedup (time)	2.16	2.22	2.22	2.24
norm. ener. usage	0.42	0.44	0.44	0.45
μ (*clock frequency in GHz*)				

and energy usage of the original program (compiled with the `-O3`) when run at the highest available frequency.

Empirical tuning via automatic generation of code alternatives and/or careful selection of parameters that govern the application of different optimization strategies has proven its merit in last few decades [24, 27, 28]. Needless to say, the technique delivers its promise in our experiments as well. However, we are more interested in the energy side of the application execution behavior. Active Harmony's search favors a lower clock frequency for better energy usage. On average, energy conscious parameter configurations save 58% energy and run 2.16× faster compared to the baseline compiler optimized code. The auto-tuning runs that use ED, ED^2 and T metrics all favor high clock frequency and show similar performance and energy characteristics. In terms of the best runtime improvement, auto-tuning runs done with delay as the feedback metric achieves 2.24× improvement along with energy saving of 55%. This confirms the popular belief that auto-tuning for runtime in scientific applications leads to better system-wide energy usage.

Often the best performing code is nearly the most energy efficient as short runtimes shorten one component of the $Power \times T$ product (Energy). So finally, we compared the performance and energy consumption measurements between configurations that give best timing and best energy usage respectively. The configuration that provides best energy usage suffers a delay of 4.1%; however, the energy usage saving is 5.8%. The search heuristic used in these experiments does not guarantee a globally best configuration with respect to timing or energy consumption, which means there can be other configurations in the search space that can possibly demonstrate different behavior. However, the result indicates that there are some non-trivial interactions between compiler performance optimization strategies and energy usage.

Table 2 shows the results for L2-Error function. The results for this function follows a similar pattern to that of the relaxation function. We then compared the performance and energy consumption measurements between configurations that give best timing [3] and best energy usage respectively. The configuration

[3] Note that for this kernel, ED^2 tuning runs gives the best timing, which is what we use for the comparison with the best energy usage configuration.

Table 2. PES-L2Error kernel results (400^3 grid, 8-cores)

	$E(\mu)$	$ED(\mu)$	$ED^2(\mu)$	$T(\mu)$
Best Configurations				
norm. speedup (time)	1.15 (2.13)	1.17 (2.39)	1.20 (2.40)	1.18 (2.40)
norm. ener. usage	0.80	0.82	0.85	0.86
Averages				
norm. speedup (time)	1.15	1.15	1.18	1.17
norm. ener. usage	0.82	0.83	0.86	0.87
μ (*clock frequency in GHz*)				

that provides best energy usage suffers a performance loss of 3.9% and the energy usage savings is 5%. This result further strengthens our earlier argument about the need to investigate the interactions between compiler optimization strategies and energy consumption.

4 Future Work

In this paper, we have relied exclusively on the Wattsup Pro? to gather AC power measurements at a 1 second granularity. Going forward, we would like to utilize more fine-grained DC power measurement tools such as PowerMon2 [4] that allow for measurements of individual system components at much higher sampling rates. Doing so would allow us to be able to attribute fractions of the total energy consumed to various components of the compute systems, such as memory subsystem energy usage and CPU energy usage. This association will allow us to target specific optimization techniques to reduce energy usage of various components. Finally, we would like to extend this work to use online tuning. That is, the application will be energy-tuned while it runs.

5 Related Work

Reducing power consumption has long been of great interest to embedded and mobile systems architects [11, 18, 22]. The architectural properties of these systems are fundamentally different from those of the HPC systems, so the strategies proposed for for them generally fail to translate into reducing energy consumption for HPC systems and applications [16]. As such, power optimization has received a fair amount of attention from the HPC community. Most previous research on power optimization uses architectural simulation to estimate power or energy usage by different components of the compute system [6]. More recently, direct power and energy measurement hardware and software have been developed [4,13,15]. Bedard et. al. [4] developed PowerMon2, a framework designed to obtain fine-grained current and voltage measurements for different components of a target platform such as CPU, memory subsystem, disk I/O etc. Power profiling frameworks can be integrated within our power auto-tuning framework to obtain greater understanding of the impact of different optimization techniques on individual components of the target architecture.

Power or energy usage modeling and benchmarking is another relevant area. The Energy-Aware Compilation (EAC) [17] framework uses a high-level energy estimation model to predict the energy consumption of a given piece of code. The model utilizes architectural parameters and energy/performance constraints. The overall idea is to use the model to decide the profitability of different compiler optimization techniques. Singh et. al. [26] derive an analytic, workload-independent piece-wise linear power model that maps performance counters and temperature to power consumption. Laurenzano et. al. [19] use a benchmark-based approach to determining how system power consumption and performance is affected by various demand regimens on the system, then use this to select processor clock frequency.

Seng et. al. [25] examine the effect of compiler optimization levels and a few specific compiler optimization flags on the energy usage and power consumption of the Intel Pentium 4 processor. Rather than relying on compiler optimization levels, we exercise a greater control over how different code transformation strategies are applied. Moreover, our technique is general purpose and uses a fairly inexpensive power measurement hardware to guide the exploration of the parameter search space.

Rahman et. al. [23] use a model-based approach to estimate power consumption of chip multiprocessors and use that information to guide the application of different compiler optimization techniques. This work is most closely related to the work that we have presented here. Power estimations for different code-variants are obtained using the model described by Singh et. al. [26]. Our work uses power measurements rather than models and we simultaneously treat clock frequency as a tunable parameter alongside the generation and evaluation of different code variants.

6 Conclusion

In this paper, we showed that there are non-trivial interactions between compiler performance optimization strategies and energy usage. We used a fairly inexpensive power meter and leveraged open source projects to explore energy and performance optimization space for computation intensive kernels.

7 Acknowledgements

This work was supported in part by the DOE Office of Science through the SciDAC2 award entitled Performance Engineering Research Institute.

References

1. CPU Frequency Scaling. https://wiki.archlinux.org/index.php/Cpufrequtils.
2. KeLP. http://cseweb.ucsd.edu/groups/hpcl/scg/KeLP1.4/.
3. WattsUp? Meters. https://www.wattsupmeters.com/secure/products.php?pn=0.
4. D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. PowerMon: Fine-grained and integrated power monitoring for commodity computer systems. In *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, pages 479 –484, 2010.
5. C. Bekas and A. Curioni. A new energy aware performance metric. *Computer Science - Research and Development*, 25:187–195, 2010.
6. D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 83–94, New York, NY, USA, 2000. ACM.
7. D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20:26–44, November 2000.
8. C. Chen. *Model-Guided Empirical Optimization for Memory Hierarchy.* PhD thesis, University of Southern California, 2007.
9. I.-H. Chung and J. Hollingsworth. A case study using automatic performance tuning for large-scale scientific programs. In *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 45 –56, 0-0 2006.
10. P. Ciccotti et. al. Characterization of the DARPA Ubiquitous High Performance Computing (UHPC) Challenge Applications. In Submission to International Symposium on Workload Characterization (IIWSC) 2011.
11. J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 48–63, New York, NY, USA, 1999. ACM.
12. V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch. Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. *J. Parallel Distrib. Comput.*, 68:1175–1185, September 2008.
13. R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658 –671, may 2010.
14. M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, pages 8 –11, oct 1994.
15. Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi. Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 298–298, Washington, DC, USA, 2006. IEEE Computer Society.
16. C.-h. Hsu and W.-c. Feng. A Power-Aware Run-Time System for High-Performance Computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 1–, Washington, DC, USA, 2005. IEEE Computer Society.
17. I. Kadayif, M. Kandemir, N. Vijaykrishnan, M. Irwin, and A. Sivasubramaniam. Eac: a compiler framework for high-level energy estimation and optimization. In *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 436 –442, 2002.

18. M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of compiler optimizations on system power. *IEEE Trans. Very Large Scale Integr. Syst.*, 9:801–804, December 2001.
19. M. A. Laurenzano, M. Meswani, L. Carrington, A. Snavely, M. M. Tikir, and S. Poole. Reducing Energy Usage with Memory and Computation-Aware Dynamic Frequency Scaling. In *Proceedings of the 17th international Euro-Par conference on Parallel processing*, EuroPar'11, Bordeaux, France, 2011. To Appear.
20. D. Li, B. de Supinski, M. Schulz, K. Cameron, and D. Nikolopoulos. Hybrid MPI/OpenMP power-aware computing. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1 –12, april 2010.
21. C. Olschanowsky, L. Carrington, M. Tikir, M. Laurenzano, T. S. Rosing, and A. Snavely. Fine-grained energy consumption characterization and modeling. In *DOD High Performance Computing Modernization Program User Group Conference*, 2010.
22. P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35:89–102, October 2001.
23. S. F. Rahman, J. Guo, and Q. Yi. Automated empirical tuning of scientific codes for performance and power consumption. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 107–116, New York, NY, USA, 2011. ACM.
24. G. Rivera and C.-W. Tseng. Tiling optimizations for 3D scientific computations. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '00, Washington, DC, USA, 2000. IEEE Computer Society.
25. J. S. Seng and D. M. Tullsen. The Effect of Compiler Optimizations on Pentium 4 Power Consumption. In *Proceedings of the Seventh Workshop on Interaction between Compilers and Computer Architectures*, INTERACT '03, pages 51–, Washington, DC, USA, 2003. IEEE Computer Society.
26. K. Singh, M. Bhadauria, and S. A. McKee. Prediction-based power estimation and scheduling for cmps. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 501–502, New York, NY, USA, 2009. ACM.
27. A. Tiwari, C. Chen, J. Chame, M. Hall, and J. Hollingsworth. A Scalable Auto-Tuning Framework for Compiler Optimization. In *23rd IEEE International Parallel & Distributed Processing Symposium*, Rome, Italy, May 2009.
28. R. Vuduc, J. W. Demmel, and K. A. Yelick. Oski: A library of automatically tuned sparse matrix kernels. *Journal of Physics: Conference Series*, 16:521–530, June 2005.