

Challenges in Automatic Software Optimisation: the energy efficiency case

T. Fischbach¹, E. Kieffer¹, and P. Bouvry¹

Luxembourg¹

{tobias.fischbach, emmanuel.kieffer, pascal.bouvry}@uni.lu

1 Introduction

With the advent of the Exascale capability allowing supercomputers to perform at least 10^{18} IEEE 754 Double Precision (64 bits) operations per second, many concerns have been raised regarding the energy consumption of high-performance computing code. Recently, Frontier operated by the Oak Ridge National Laboratory, has become the first supercomputer to break the exascale barrier [1]. In total, *Frontier* contains 9,408 CPUs, 37,632 GPUs, and 8,730,112 cores. This world-leading supercomputer consumes about 21 megawatts which is truly remarkable as *Frontier* was also ranked first on the Green500 list before being recently replaced. The previous top Green500 machine, MN-3 in Japan, provided 39.38 gigaflops per watt, while the *Frontier* delivered 62.68 gigaflops per watt. All these infrastructure and hardware improvements are just the tip of the iceberg. Energy-aware code is now required to minimize the energy consumption of distributed and/or multi-threaded software. For example, the data movement bottleneck is responsible for 35 – 60% of a system’s energy consumption during intra-node communication. In an HPC environment, additional energy is consumed through inter-node communication. This position paper aims to introduce future research directions to enter now in the age of energy-aware software. The paper is organized as follows. First, we introduce related works regarding measurement and energy optimisation. Then we propose to focus on the two different levels of granularity in energy optimisation.

2 Related works

2.1 Measurement

Optimizing software for energy efficiency necessitates thoroughly comprehending the energy consumption of different components (e.g., CPU, DRAM, GPU) at the process level.

Commonly utilized tools, such as IPMI, are either too granular or lack the necessary precision, such as LIKWID and Slurm, for fine-grained measurements, as they are based on the same underlying technology RAPL [2] (a low-level power capping framework that reports fine-grained CPU and RAM energy consumption). Haswell is the last generation that included on-die circuitry for measuring the consumed energy; current CPU generations only estimate the energy consumption. Based on validation experiments, it is necessary to adjust the reported energy consumption by an offset, but the reported values still show large deviations based on utilization as well as a large spread [3][10]. In addition, AMD’s implementation for Ryzen and Epyc lacks certain features (e.g., DRAM energy).

To enhance the accuracy of the energy consumption measurement without altering the motherboard, we propose an experimental setup based on sensible Hall sensors. The Hall sensors are placed strategically to monitor the energy consumption of different CPU packages and socket interconnections, DRAM, and PCIe connections. This approach accurately measures energy consumption without measuring overhead on the experiment itself. Among the challenges that are yet to be solved include the creation of reproducible software (e.g., interfering kernel tasks and binary reproducibility) and hardware environment (e.g., power states and frequency control) using NixOS to reduce noise. Due to the last generation with on-die hardware for power measurement, our results will be validated for Haswell-era CPUs.

2.2 Automatic software optimisation

Up to now, the efforts toward an energy-aware compiler are mainly focused on either single-core or parallel workloads based on MPI. Embecosm employs super-optimisation and machine learning to optimize LLVM-IR to reduce the energy consumption of software [11]. Other methods try to find the most energy-efficient combination of LLVM passes, a time-consuming and non-generalizable approach. Despite many efforts [7], no precise model bridges the gap between coarse and fine-grained and accurately predicts the consumed energy.

3 High-level Energy optimisation

3.1 Surrogate-based algorithms

Automatic software optimisation is an NP-hard optimisation problem that can only be tackled by means of heuristics or metaheuristics(see [13]) to obtain "good enough" approximations. The classical strategies consisting in applying genetic algorithms or, more generally, population-based metaheuristics are likely to fail since code evaluation is time-consuming. This explains why most of the academic research focused only on minimal benchmarks (e.g., matrix multiplication with LINPACK [4]). In automatic software optimisation, the objective function has no explicit representation, and software code have to be simulated in a reproducible environment to get accurate and meaningful results. This is even truer when considering distributed software code on high-performance computing platforms. Therefore, we propose to consider surrogate-based optimisation to minimize the number of simulations needed to evaluate the optimised code. Surrogate optimization attempts to find a global minimum of an objective function using a few objective function evaluations. For this purpose, the algorithm tries to balance the optimization process between two goals the search for a global minimum and the speed to obtain a good solution with few objective function evaluations.

3.2 Inferring decision rules

Our decision to choose surrogate-based optimizers will definitely help us to consider larger problems, but we doubt that there is a significant advantage to applying a complex optimisation framework for every chunk of code that may exist. Why can't we discover a set of rules connecting code patterns with code transformation? We hope to conduct this research to the point where we can automatically identify similarities between high-energy profiles. We could try to build machine learning models connecting LLVM analysis passes to LLVM transformation passes to infer common sense rules improving software energy consumption and change the pass sequence accordingly.

4 Low-level Energy optimisation

The LLVM compiler framework [9] translates, transforms, and optimises source code using an intermediate representation (IR) in human-readable assembly[6]. From the results obtained during the high-level energy optimization, regions that consumed the most energy are identified and prioritized to create an energy-aware compiler.

4.1 Energy-aware compiler

A heterogeneous set of benchmarks is selected as optimization targets to reflect HPC's highly parallel and diverse nature. Finally, the energy consumption of performance and energy-optimised software is compared to identify the LLVM passes that reduce the energy requirements. The LLVM-IR is inspected concerning code regions that can be used to reduce data movement, increase cache hits and prioritize energy-efficient instructions. Before enhancing instructions, candidates must be identified, and suitable replacements must be selected. Creating metrics correlating the measured energy consumption with IR code and the corresponding instructions should aid us in this process. While we intend to infer rules based on a consistent energy consumption model, this challenging task results in the large degree of freedom and non-linearity of interactions between instructions, hardware implementation, and compiler optimization. Several critical factors related to communication are to be considered for optimisation. The communication is considered by inter- and intra-node communication and cache misses [8]. The existing MPI implementations [12, 14] are commonly used to implement intra- and inter-node communication effectively. On an intra-node communication scale, the energy consumption of communication is driven by inter-socket communication, core-to-core communication, and accelerator communication. The energy consumption of inter-node communication is affected by data movement [5] and synchronization.

4.2 Energy-aware language: a perspective

The two proposed methods represent a "good enough" approach and try to enhance energy efficiency with languages that prioritize performance. We propose purpose-built languages that focus on energy efficiency instead. Expressing algorithms in more coarse language allows the usage of grammar evolution to automatic software creation. After sampling a sufficiently large state space, our proposed methods can optimize the vocabulary and grammar for energy efficiency. This purpose-built language can be extended to support more algorithms over time. Communication-heavy algorithms, like molecular dynamics simulations or deep learning, are ideal candidates due to their popularity and high-performance requirements.

5 Conclusion and perspectives

Improving the energy efficiency of software and algorithms is necessary for the era of exascale computing and IoT devices. However, current energy measurements are too coarse, especially for communication level and instruction level optimizations. This paper proposes a precise experiment based on sensitive hall sensors to measure the energy consumption of computation. We propose high-level energy optimizations based on surrogate algorithms to infer decision rules if and when LLVM performs specific transformations. Based on the high-level results, the most energy-consuming computation is identified, and low-level energy optimizations concerning intra- and inter-node communication are performed on an instruction level. Nevertheless, those methods try to transform performance-optimised languages to be more energy efficient. Lastly, we propose a purpose-built language with energy efficiency in mind.

Acknowledgement

Tobias Fischbach acknowledges financial support of the Institute for Advanced Studies of the University of Luxembourg through a YOUNG ACADEMICS Grant (YOUNG ACADEMICS-2022-NETCOM).

References

1. The beating heart of the world's first exascale supercomputer. <https://spectrum.ieee.org/frontier-exascale-supercomputer>. Accessed: 2023-01-24.
2. Intel® 64 and IA-32 Architectures Software Developer Manuals.
3. Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. A Validation of DRAM RAPL Power Measurements. In *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pages 455–470. Association for Computing Machinery, October 2016. Place: New York, NY, USA.
4. Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK benchmark: past, present and future. *Concurr. Comput. Pract. Exp.*, 15(9):803–820, 2003.
5. Graham Gobieski, Souradip Ghosh, Marijn Heule, Todd Mowry, Tony Nowatzki, Nathan Beckmann, and Brandon Lucia. RipTide: A Programmable, Energy-Minimal Dataflow Compiler and Architecture. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 546–564, October 2022.
6. Neville Grech, Kyriakos Georgiou, James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. Static analysis of energy consumption for LLVM IR programs, July 2015. arXiv:1405.4565 [cs].
7. Franz Christian Heinrich, Tom Cornebize, Augustin Degomme, Arnaud Legrand, Alexandra Carpen-Amarie, Sascha Hunold, Anne-Cécile Orgerie, and Martin Quinson. Predicting the Energy-Consumption of MPI Applications at Scale Using Only a Single Node. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 92–102, September 2017. ISSN: 2168-9253 tex.eventtitle: 2017 IEEE International Conference on Cluster Computing (CLUSTER).
8. Alla G. Kravets and Vitaly Egunov. The Software Cache Optimization-Based Method for Decreasing Energy Consumption of Computational Clusters. *Energies*, 15(20):7509, January 2022. Publisher: Multidisciplinary Digital Publishing Institute.
9. C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86, San Jose, CA, USA, 2004. IEEE.
10. Unai Lopez-Novoa. Exploring Performance and Energy Consumption Differences between Recent Intel Processors. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 263–267, August 2019.
11. James Pallister, Simon J. Hollis, and Jeremy Bennett. Identifying Compiler Options to Minimize Energy Consumption for Embedded Platforms. *The Computer Journal*, 58(1):95–109, January 2015.
12. Henrik Valter, Axel Karlsson, and Miquel Pericàs. Energy-Efficiency Evaluation of OpenMP Loop Transformations and Runtime Constructs, September 2022. arXiv: 2209.04317 [cs] Number: arXiv:2209.04317.
13. Sébastien Varrette, Frédéric Pinel, Emmanuel Kieffer, Grégoire Danoy, and Pascal Bouvry. Automatic software tuning of parallel programs for energy-aware executions. In Roman Wyrzykowski, Ewa Deelman, Jack Dongarra, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics*, pages 144–155, Cham, 2020. Springer International Publishing.
14. Akshay Venkatesh, Abhinav Vishnu, Khaled Hamidouche, Nathan Tallent, Dhabaleswar Panda, Darren Kerbyson, and Adolphy Hoisie. A case for application-oblivious energy-efficient MPI runtime. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, November 2015. ISSN: 2167-4337 tex.eventtitle: SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.