# Literature review of approaches to improve software energy efficiency: findings and recommendations

**Part of the UKRI Net Zero Digital Research Infrastructure Scoping Project**

Ole Stubben, Wim Vanderbauwhede

School of Computing Science, University of Glasgow

This report presents an overview of the state of the art on several approaches to improve software energy efficiency: code optimisation to speed up execution and to reduce memory access; parallelisation across multiple hardware cores to speed up execution; tuning compiler optimisations; code rewrite to target accelerators. Our recommendations are to invest in experts to assist researchers with those tasks and to invest in funding for research and development into tools to automate energy efficiency optimisation.

## 1 Introduction

### 1.1 Energy Efficiency and Runtime Efficiency

In all papers surveyed so far where both runtime and energy usage were tracked, there is a tight correlation between the 2 metrics [29, 30, 1, 50, 49]. So, reducing the runtime of any given software will improve the energy consumption by roughly the same amount. There is evidence to suggest that reducing runtime using increased parallelism also reduces energy consumption [50, 49, 26]. This can be attributed to the following: (1) When executing a sequential program on one core, other cores still draw power when idling. (2) Some CPUs will increase power consumption for all cores to speed up execution within one core (such as Intel Turbo Boost).

The former does not apply to parallelism is achieved by using more than 1 compute node. For example, [25] shows that speed-up is at best proportional to number of cores on multicore systems. This is a well-known result: because of communication overheads, speed-up tends to saturate when more nodes are used. Even with linear speed-ups, using more nodes does not improve the energy efficiency, because at best, every node will use the same amount of energy, so there will be no gain; at worst, the interconnect network will on the one hand lead to less-than-linear speed-up, and on the other hand, it will consume considerable energy in its own right [13, 4].

Given the above, research which investigates only runtime efficiency (on single compute nodes) can be used to support arguments for energy efficiency. There is considerable literature on techniques to optimise runtime performance, e.g. [6, 31, 20, 36], which show that performance optimisation is still considered of sufficient complexity to warrant academic research.

Much recent research has focused on algorithms for optimising performance, e.g. [47, 8, 11]. This research shows that it is possible to automate the process of optimisation for a wide range of software. However, state-of-the-art production compilers do not include this type of optimisation capability.

### 1.2 Frequently Used Tools

Many of the papers we have surveyed which consider CPU and DRAM energy usage utilise Intel's Running Average Power Limit (RAPL) energy reporting. The accuracy of RAPL for this purpose was generally within ∼20% of physical measurements [10].

## 2 Improving Software Energy Efficiency

### 2.1 Energy Efficiency of Programming Languages

Generally, compiled languages outperform Virtual Machine (VM) languages and interpreted languages. Furthermore, VM languages outperform interpreted languages [29, 30, 1]. There is some overlap between these groups of languages for some specific algorithms and problem sizes [29, 1]. Switching from a solution implemented solely in an interpreted language to a compiled language can result in drastically lower energy consumption [29, 1]. An important question here is: how do we use this information in the real world of digital research infrastructure? Python (an interpreted language) is very popular in many fields, but this is partly because of the available libraries. These libraries are usually written in a compiled language, and are already highly optimised. Therefore, unless most of the run time is not spent in calls to such libraries, energy efficiency gains from rewriting Python code in C, C++, or Rust would be small; a better approach is to identify the bottleneck using profiling tools and rewrite only that part of the code as a compiled library.

### 2.2 Energy Efficiency Gains with Compiler Optimisations

So far, only one of the papers we have surveyed broke down energy efficiency by compiler flags. In that case, applying flags such as O3 when compiling C/C++ code leads to energy consumption reductions of 52-73% on QuickSort and Fast Fourier Transform (FFT) [1]. If there is a lot of C/C++ code being run that has not been compiled with an optimisation flag, this could result in massive savings. This conclusion also holds for Fortran, as it shares the compiler back-end with C/C++. However, there is no evidence in the literature of the degree to which optimisations are applied. From our own practice and experience with students at different levels, it is clear that very often the chosen compiler optimisations are not the most effective ones, and that in general, optimisation during compilation requires either expert knowledge or the use of advanced techniques for iterative compilation [3, 12, 7].

### 2.3 Energy Efficiency Gains by Refactoring

Similar to how the choice of data structures affect performance, data structures also affect energy efficiency [23, 15]. For Java collections classes, choosing the worst option could increase power usage up to 300%, while choosing the best option could decrease power usage by 38% [17]. However, these results are from looking at collections operations exclusively. When applied to larger code bases, a tool for automatically improving collections choices was shown to reduce energy usage by 2-17% [23]. It should be noted that the automatic optimisation ran between 4-175 wall clock hours, so the expected savings must account for the expense of

the optimisation [23]. Along with other refactoring options, there is good potential here for feasible and concrete recommendations to increase energy efficiency.

### 2.4 Energy Efficiency of Parallelisation Approaches

As discussed above, parallelising code across multiple cores on a manycore processor can improve energy efficiency. However, depending on the approach to parallelisation, the opposite can also be true: different parallelisation frameworks perform differently depending on the type of parallelism in the code (data parallelism, task parallelism). Therefore, a careful choice of the parallelisation framework is crucial for energy efficiency, as demonstrated in [39, 38, 35, 14, 24]. From these papers, it is clear that there is no single framework that has optimal performance across all applications.

### 2.5 Energy Usage of DRAM

Some papers show DRAM energy usage as comprising nearly 50% of system energy usage. However, this is only the case for rare or dated hardware (before 2010) [44, 22], or in highly specific use-cases (memory-intensive computation) [28, 48]. A more general result, performed on somewhat recent consumer-grade hardware (∼2014) and over a range of different problems, shows DRAM energy usage as ∼11-12% (19% in one case, with DDR3 [9]) of system energy usage [29, 5, 2]. Nevertheless, the power consumption of DRAM is not negligible, especially when compared to energy-efficient accelerators instead of CPUs, and therefore there is ongoing research into software approaches to reduce DRAM energy consumption. Because memory is managed by the operating system, this work typically proposes improvements on that level, which would be transparent to the programmer [51, 46, 18]. However, as shown in [21], despite the need for constant refreshing, DRAM power consumption is not constant and reducing reads and writes can significantly affect energy consumption. This implies that techniques to reduce memory usage such as [37] can also lead to significant reductions in energy consumption.

### 2.6 Energy Efficiency of Hardware Accelerators

Hardware accelerators have great promise for energy savings when used for a wide range of applications. In particular, GPUs are increasingly transformative, especially in the field of machine learning, but also in many other fields [49, 42, 16, 27]. FPGAs have, in principle, even higher performance per Watt for a more limited, but still very wide, range of applications, including machine learning and numerical weather simulations [40, 41, 32, 33, 34]. Speed-ups of an order of magnitude compared to reference CPU implementations are routinely reported. Most of the literature on GPUs only reports performance gains, whereas FPGA literature tends to focus more on energy-efficiency gains. The overall picture is that, even compared with modern GPUs, FPGAs can still offer better energy efficiency, but that it remains harder (technically challenging) to achieve the optimal performance of a given program on FPGA than on GPU.

There is also growing interest in compilation for more specialised accelerators such as TPUs [19, 43] and FFT accelerators [45], both of which show similar order-of-magnitude speed-ups and even higher energy efficiency savings, but for much more specialised applications (Neural Networks and Fourier Transforms, respectively).

Overall, from our review, most of the code for accelerators is still written manually by experts, and research into compilation of generic code to accelerators has not yet been implemented in any state-of-the-art production compilers.

## 3 Conclusions

From this literature review, we can conclude that the potential for improving software energy efficiency is considerable. There are several approaches, and for many use cases there are examples in the literature of significant improvements in energy efficiency, from several times to order-of-magnitude, through the use of just a single of the approaches outline below:

- Code optimisation to speed up execution and to reduce memory access.
  This requires experts familiar with the programming language, target platform and application domain. A special case is rewriting code written in inefficient languages (e.g. R, MatLab, Python etc) into more efficient ones such as C/C++ or Julia.
- Parallelisation across multiple hardware cores to speed up execution.
  This requires experts familiar with the programming language, target platform and application domain, as well as parallelisation frameworks.
- Tuning compiler optimisations.
  This requires experts familiar with the programming language and the effect of various compiler optimisations.
- Code rewrite to target accelerators such as GPUs, FPGAs, TPUs etc.
  This is again a very specialised area and requires experts familiar with accelerator architectures and programming models, in addition to the above skills and knowledge.

All these approaches can in principle be automated, but the tools to do so are still at the academic research stage.

## 4 Recommendations

The above conclusions allow us to formulate the following recommendations:

- Because all approaches to code optimisation for energy efficiency require experts:
  - Researchers should have access to software energy efficiency experts.
  - Experts should be trained specifically in the approaches outline above.

As noted above: in principle, all approaches to code optimisation for energy efficiency could be automated and integrated into existing compilers. However, there is a gap in funding for this kind of activity: it is not considered academically relevant, and thus not eligible for UKRI funding; it is also not economically viable for start-ups; and it is to far from the market to be adopted by developers of production compilers.
Nevertheless, availability of automated tools for improving energy efficiency of existing code could make a significant difference to the UKRI DRI energy consumption.

- Therefore:
  - UKRI should encourage reseach into novel approaches to improve energy efficiency of scientific software.
  - UKRI should provide specific funding schemes to develop proof-of-concept tools for improving energy efficiency into software products usable by non-expert researchers.

## References


[1] Abdulsalam, S., Lakomski, D., Gu, Q., Jin, T., and Zong, Z. Program energy efficiency: The impact of language, compiler and implementation choices. In *International Green Computing Conference* (2014), IEEE, pp. 1–6.

[2] Acar, H., Alptekin, G. I., Gelas, J.-P., and Ghodous, P. Beyond cpu: Considering memory power consumption of software. In *2016 5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)* (2016), IEEE, pp. 1–8.

[3] Agakov, F., Bonilla, E., Cavazos, J., Franke, B., Fursin, G., O'Boyle, M. F., Thomson, J., Toussaint, M., and Williams, C. K. Using machine learning to focus iterative optimization. In *International Symposium on Code Generation and Optimization (CGO'06)* (2006), IEEE, pp. 11–pp.

[4] Andújar, F. J., Coll, S., Alonso, M., Martínez, J.-M., López, P., Sánchez, J. L., and Alfaro, F. J. Energy efficient hpc network topologies with on/off links. *Future Generation Computer Systems 139* (2023), 126–138.

[5] Barroso, L. A., Clidaras, J., and Hölzle, U. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture 8*, 3 (2013), 1–154.

[6] Baruffa, F., Iapichino, L., Hammer, N. J., and Karakasis, V. Performance optimisation of smoothed particle hydrodynamics algorithms for multi/many-core architectures. In *2017 International Conference on High Performance Computing & Simulation (HPCS)* (2017), IEEE, pp. 381–388.

[7] Blackmore, C., Ray, O., and Eder, K. Automatically tuning the gcc compiler to optimize the performance of applications running on embedded systems. *arXiv preprint arXiv:1703.08228* (2017).

[8] Cortellessa, V., Di Pompeo, D., Stoico, V., and Tucci, M. On the impact of performance antipatterns in multi-objective software model refactoring optimization. In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2021), IEEE, pp. 224–233.

[9] David, H., Fallin, C., Gorbatov, E., Hanebutte, U. R., and Mutlu, O. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing* (2011), pp. 31–40.

[10] Desrochers, S., Paradis, C., and Weaver, V. M. A validation of dram rapl power measurements. In *Proceedings of the Second International Symposium on Memory Systems* (2016), pp. 455–470.

[11] Di Pompeo, D., and Tucci, M. Search budget in multi-objective refactoring optimization: a model-based empirical study. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2022), IEEE, pp. 406–413.

[12] Dubach, C., Cavazos, J., Franke, B., Fursin, G., O'Boyle, M. F., and Temam, O. Fast compiler optimisation evaluation using code-feature based performance prediction. In *Proceedings of the 4th international conference on Computing frontiers* (2007), pp. 131–142.

[13] Faisal, F. A., Rahman, M. H., and Inoguchi, Y. 3d-ttn: A power efficient cost effective high performance hierarchical interconnection network for next generation green supercomputer. *Cluster Computing 24*, 4 (2021), 2897–2908.

[14] Ferrer, R., Planas, J., Bellens, P., Duran, A., Gonzalez, M., Martorell, X., Badia, R. M., Ayguade, E., and Labarta, J. Optimizing the exploitation of multicore processors and gpus with openmp and opencl. In *Languages and Compilers for Parallel Computing: 23rd International Workshop, LCPC 2010, Houston, TX, USA, October 7-9, 2010. Revised Selected Papers 23* (2011), Springer, pp. 215–229.

[15] Georgiou, S., Rizou, S., and Spinellis, D. Software development lifecycle for energy efficiency: techniques and tools. *ACM Computing Surveys (CSUR) 52*, 4 (2019), 1–33.

[16] Harvey, P., Hameed, S., and Vanderbauwhede, W. Accelerating lagrangian particle dispersion in the atmosphere with opencl across multiple platforms. In *Proceedings of the International Workshop on OpenCL 2013 & 2014* (2014), pp. 1–8.

[17] Hasan, S., King, Z., Hafiz, M., Sayagh, M., Adams, B., and Hindle, A. Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering* (2016), pp. 225–236.

[18] Hassan, A., Vandierendonck, H., and Nikolopoulos, D. S. Software-managed energy-efficient hybrid dram/nvm main memory. In *Proceedings of the 12th ACM International Conference on Computing Frontiers* (2015), pp. 1–8.

[19] Hu, P., Lu, M., Wang, L., and Jiang, G. Tpu-mlir: A compiler for tpu using mlir. *arXiv preprint arXiv:2210.15016* (2022).

[20] Kaeslin, A. E. Performance optimisation of discrete-event simulation software on multicore computers, 2016.

[21] Lee, Y., Kim, S., Hong, S., and Lee, J. Skinflint dram system: Minimizing dram chip writes for low power. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)* (2013), IEEE, pp. 25–34.

[22] Lefurgy, C., Rajamani, K., Rawson, F., Felter, W., Kistler, M., and Keller, T. W. Energy management for commercial servers. *Computer 36*, 12 (2003), 39–48.

[23] Manotas, I., Pollock, L., and Clause, J. Seeds: A software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering* (2014), pp. 503–514.

[24] Memeti, S., Li, L., Pllana, S., Kołodziej, J., and Kessler, C. Benchmarking opencl, openacc, openmp, and cuda: programming productivity, performance, and energy consumption. In *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing* (2017), pp. 1–6.

[25] Ouro, P., Fraga, B., Lopez-Novoa, U., and Stoesser, T. Scalability of an eulerian-lagrangian large-eddy simulation solver with hybrid mpi/openmp parallelisation. *Computers & fluids 179* (2019), 123–136.

[26] Padoin, E. L., Pilla, L. L., Castro, M., Boito, F. Z., Alexandre Navaux, P. O., and Méhaut, J.-F. Performance/energy trade-off in scientific computing: the case of arm big. little and intel sandy bridge. *IET Computers & Digital Techniques 9*, 1 (2015), 27–35.

[27] Pandey, M., Fernandez, M., Gentile, F., Isayev, O., Tropsha, A., Stern, A. C., and Cherkasov, A. The transformational role of gpu computing and deep learning in drug discovery. *Nature Machine Intelligence 4*, 3 (2022), 211–221.

[28] Paul, I., Huang, W., Arora, M., and Yalamanchili, S. Harmonia: Balancing compute and memory power in high-performance gpus. *ACM SIGARCH Computer Architecture News 43*, 3S (2015), 54–65.

[29] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., and Saraiva, J. Energy efficiency across programming languages: how do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering* (2017), pp. 256–267.

[30] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., and Saraiva, J. Ranking programming languages by energy efficiency. *Science of Computer Programming 205* (2021), 102609.

[31] Sakai, Y., Mendez, S., Strandenes, H., Ohlerich, M., Pasichnyk, I., Allalen, M., and Manhart, M. Performance optimisation of the parallel cfd code mglet across different hpc platforms. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (2019), pp. 1–13.

[32] Shawahna, A., Sait, S. M., and El-Maleh, A. Fpga-based accelerators of deep learning networks for learning and classification: A review. *ieee Access 7* (2018), 7823–7859.

[33] Singh, G., Alser, M., Cali, D. S., Diamantopoulos, D., Gómez-Luna, J., Corporaal, H., and Mutlu, O. Fpga-based near-memory acceleration of modern data-intensive applications. *IEEE Micro 41*, 4 (2021), 39–48.

[34] Singh, G., Diamantopoulos, D., Hagleitner, C., Gómez-Luna, J., Stuijk, S., Mutlu, O., and Corporaal, H. Nero: A near high-bandwidth memory stencil accelerator for weather prediction modeling. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)* (2020), IEEE, pp. 9–17.

[35] Stpiczyński, P. Language-based vectorization and parallelization using intrinsics, openmp, tbb and cilk plus. *The Journal of Supercomputing 74*, 4 (2018), 1461–1472.

[36] Sun, T. Abstractions and performance optimisations for finite element methods.

[37] Szafarczyk, R., Nabi, S. W., and Vanderbauwhede, W. Reducing fpga memory footprint of stencil codes through automatic extraction of memory patterns. In *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)* (2022), IEEE, pp. 148–152.

[38] Tousimojarad, A., and Vanderbauwhede, W. Number of tasks, not threads, is key. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (2015), IEEE, pp. 128–136.

[39] Tousimojarad, A., and Vanderbauwhede, W. Steal locally, share globally: A strategy for multiprogramming in the manycore era. *International Journal of Parallel Programming 43* (2015), 894–917.

[40] Vanderbauwhede, W., and Benkrid, K. *High-performance computing using FPGAs*, vol. 3. Springer, 2013.

[41] Vanderbauwhede, W., Scholz, S.-B., and Margala, M. Fpgas for domain experts. *International Journal of Reconfigurable Computing 2020* (2020), 1–2.

[42] Vanderbauwhede, W., and Takemi, T. An investigation into the feasibility and benefits of gpu/multicore acceleration of the weather research and forecasting model. In *2013 International Conference on High Performance Computing & Simulation (HPCS)* (2013), IEEE, pp. 482–489.

[43] Wang, Y. E., Wei, G.-Y., and Brooks, D. Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701* (2019).

[44] Ware, M., Rajamani, K., Floyd, M., Brock, B., Rubio, J. C., Rawson, F., and Carter, J. B. Architecting for power management: The ibm® power7™ approach. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture* (2010), IEEE, pp. 1–11.

[45] Woodruff, J., Armengol-Estapé, J., Ainsworth, S., and O'Boyle, M. F. Bind the gap: Compiling real software to hardware fft accelerators. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (2022), pp. 687–702.

[46] Xie, M., Tong, D., Feng, Y., Huang, K., and Cheng, X. Page policy control with memory partitioning for dram performance and power efficiency. In *International Symposium on Low Power Electronics and Design (ISLPED)* (2013), IEEE, pp. 298–303.

[47] Ye, P., Ni, Y., and Du, X. An algorithm for multi-objective software performance optimisation at the architecture level. In *Proceedings of the 2020 4th International Conference on Electronic Information Technology and Computer Engineering* (2020), pp. 1113–1119.

[48] Yoon, D. H., Chang, J., Muralimanohar, N., and Ranganathan, P. Boom: Enabling mobile memory based low-power server dimms. *ACM SIGARCH Computer Architecture News 40*, 3 (2012), 25–36.

[49] Zecena, I., Burtscher, M., Jin, T., and Zong, Z. Evaluating the performance and energy efficiency of n-body codes on multi-core cpus and gpus. In *2013 IEEE 32nd International Performance Computing and Communications Conference (IPCCC)* (2013), IEEE, pp. 1–8.

[50] Zecena, I., Zong, Z., Ge, R., Jin, T., Chen, Z., and Qiu, M. Energy consumption analysis of parallel sorting algorithms running on multicore systems. In *2012 International Green Computing Conference (IGCC)* (2012), IEEE, pp. 1–6.

[51] Zhan, J., Zhang, Y., Jiang, W., Yang, J., Li, L., and Li, Y. Energy-aware page replacement and consistency guarantee for hybrid nvm–dram memory systems. *Journal of Systems Architecture 89* (2018), 60–72.