Why Energy-Efficient Machine Learning Frameworks require Workload Awareness

Andrew Mary Huet de Barochez*[0009−0000−3094−6372], Stéphan Plassart[0000−0002−9342−8262], and Sébastien Monnet[0000−0002−6036−3060]

LISTIC, Université Savoie-Mont-Blanc (USMB), Annecy, France
{andrew.mary-huet-de-barochez, stephan.plassart, sebastien.monnet}@univ-smb.fr
https://www.univ-smb.fr/listic/en

Abstract. The growth of artificial intelligence raises concerns about electricity consumption, especially for the training of cutting edge deep learning models. However, finding the most energy efficient configuration for a given training problem is not trivial: using current machine learning frameworks, one can change for example the parallelism paradigm, number of workers, backend implementations, and floating-point precision. On top of that, such a configuration is often tailor-made for a specific machine. In this paper, we investigate how task graph computing systems can be used to adapt to such constraints. Indeed, the ability to split training into granular tasks allows the scheduler system to auto-balance the workload among available workers. To this end, we built `DAHL`, an energy-efficient machine learning framework based on the `StarPU` task graph computing system. We compare our framework with `Pytorch`, using three different datasets, one of which was scaled to different sizes. Results, obtained from both software-based and physical wattmeters, show significant energy gains on Convolutional Neural Network training: up to 4.75× less against `Pytorch` using `OpenMP`, and on average 1.55× less.

Keywords: Energy-Efficiency · Machine Learning · Workload-Aware · Scheduling · Task-Graph-Computing-Systems.

1 Introduction

At the end of 2025, seven out of the nine planet boundaries were crossed. In this context, it is clear that attention should be paid to the sustainability of Artificial Intelligence (AI). Numerous studies have been conducted to analyze the impact of electricity, carbon emissions, water consumption, and more recently, on applying Life Cycle Assessment (LCA) methods. As a first step, reducing the electricity consumption of AI systems is a good way to mitigate indirect impacts. To this extent, improving the energy efficiency of AI frameworks seems to be a key factor that could impact both training and inference phases. However, estimating the energy-efficiency of a framework is a complicated task, as many parameters

* Corresponding author.

can be tweaked to influence the execution of the training itself. To illustrate this statement, we performed an experiment showcasing four possible CPU execution scenario. Fig. 1 shows the accumulated energy consumption over time of a simple Convolutional Neural Network (CNN) training on CIFAR-10 [19] and Big-Fashion [1] datasets. The four configurations include: `Base`, a regular configuration with as many threads as CPU cores; `Hyper Threading`, which allows two threads per core to be used on our machine; `Pytorch`'s `Distributed Data Parallel` [21], a data parallelism approach; and `OpenMP`, a backend enabling shared-memory multiprocessing. These results supports observations made in the domain of High Performance Computing (HPC) [12, 20], specifically, that improving performance or reducing runtime, does not strictly implies in lowering energy consumption. For example, training on Fashion-MNIST with `Hyper Threading` is 1.15× slower than `Base`, but the energy consumption is 1.25× lower. Furthermore, not all parallelism paradigms are suitable for all situations. In this scenario, using `Distributed Data Parallel` leads to both higher energy consumption (11.5× higher) and higher runtime (8.7× slower) compared to `Base`'s default intra-operation parallelism. Likewise, `OpenMP` does not lead to any improvements in this experiment. Lastly, an optimal configuration is often specific to a given workload. In this case, `Base` is the best runtime-wise configuration when training on Fashion-MNIST, but on Big-Fashion (13× more pixels), `Hyper Threading` is both the most energy efficient and the fastest configuration.

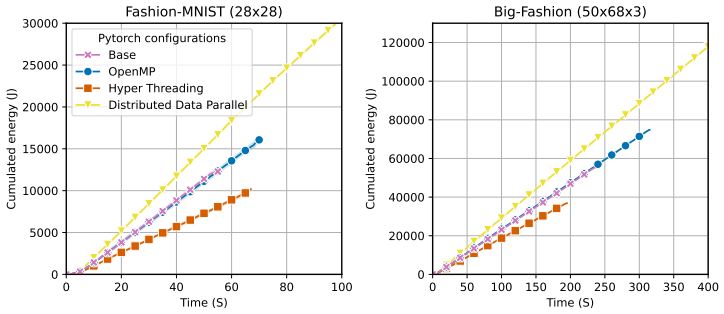

Fig. 1. CNN training on Fashion-MNIST and Big-Fashion datasets, showing accumulated wattmeter-measured energy consumption over time. Training performed over 100 epochs, 8192 samples, with 5 repetitions for each configuration. 95% Confidence intervals appear in light color and is on average 1.59%. `Distributed Data Parallel` lines have been cropped, their runtime were equal to 487 and 3340 seconds respectively.

In summary, finding the most energy-efficient configuration for a given training workload depends on several interdependent factors: the computational load, the parallelism paradigm, and the target architecture. The interdependence of factors leads to tailor-made solutions that fit training workloads to specific machines, which prevents scalability and energy efficiency on heterogeneous clusters. To this end, much work has been accomplished towards training on heterogeneous clusters [8, 11, 25, 34], but they mainly focus on solving synchronization issues and optimizing training time, while ignoring energy consumption.

In this context, Task Graph Computing System (TGCS) seem promising, as they offer the possibility of dynamically adapting workloads according to each worker's architecture. To the best of our knowledge, only a few preliminary studies have been conducted on the application of TGCS to Machine Learning (ML): [6, 7] covers inference while `NNTile` [24] covers training. The later focuses on training large language models, but suffers from task granularity issues when operating on smaller models, making it less adaptable to various workloads. In this paper, we introduce `DAHL` [1], a novel TGCS-based ML framework, and evaluate its energy efficiency on various ML workloads running on CPUs. This step is a mandatory building block for energy-efficient distributed ML in heterogeneous contexts. Our contributions are the following:

- Development of a robust TGCS-based ML framework, named `DAHL`, following `Pytorch`'s functions results in unit testing.
- Comparison of the energy efficiency of `DAHL` and `Pytorch` during CNN training, using both software-based energy measurements and a wattmeter.
- Study of the impact of workloads and configuration parameters on energy consumption.
- Demonstration that our TGCS-based solution offers better energy efficiency, with energy consumption up to 4.75× lower than that of `Pytorch` using `OpenMP`, and 1.55× lower than that of `Pytorch` using `Hyper Threading`.

2 Background and related works

2.1 Parallelism in Machine Learning

The parallelization of Machine Learning (ML) workloads can be performed on several dimensions: *Sample, Operations, Attribute, Parameter* (*SOAP* [17]). *Sample* dimension is commonly referred to as data parallelism [21], it consists of duplicating the model and training it with different data samples on each version of the model. *Operation* dimension, or inter/intra-operation parallelism [34, 39], consists of parallelizing tensor operations, such as matrix-matrix multiplications, among or across operators. *Attribute* dimension is mainly used in hybrid parallelism [8, 16] (which is useful when combined with multiple parallel dimensions), the attribute refers to the division of the input data shape itself, e.g. parallelizing over channels of an image or even splitting the image into smaller images along the width and height dimensions. Finally, *Parameter* dimension or model parallelism [18] allows the model parameters to be divided and learned in parallel.

2.2 Parallel Computing Frameworks

Several existing parallel computing frameworks supports heterogeneous hardware: `SYCL` [31] is a `C++` high level programming model leveraging different

[1] Distributed, Arbitrary, and Heterogeneous Learning: codeberg.org/PhoqueEberlue/dahl

hardware accelerators. It has many implementations that support various API backends including `CUDA`, `OpenCL`, `OpenMP` and is notably apart of Intel `OneAPI`. `Threading Building Blocks` [28] also belongs to `OneAPI`, but focuses on parallelizing tasks on multi-cores processors. `OpenMP` [10] is a multi-platform `C/C++` API that helps parallelizing loops by adding annotations in the code. While it was originally limited by its fork-join model, which prevented opportunities to progress multiple independent-tasks, `OpenMP 4.0` [23] introduced ways to represent task dependencies, yet it requires manually specifying these dependencies. This can become tough, especially with complex and interrelated graphs of operations, such as in Artificial Intelligence (AI) training. Runtime systems are better alternatives when dealing with such complex task graphs, as they can infer task dependencies with the type of data accesses (read only, read-write, write only). Those systems can be referred as Task Graph Computing System (TGCS), and `StarPU` [4] is one of the leading framework in the High Performance Computing (HPC) world. `Taskflow` [14] is an alternative to `StarPU`, nevertheless the two frameworks can be distinguished by several design choices: compared to `StarPU`, `Taskflow` leaves memory handling to the user, tasks should be explicitly allocated to CPU or GPU, but it does support in-graph control flow. `Specx` [9] uses speculative execution, which consists of evaluating operations in advance in order to improve the degree of parallelism, while taking the risk that the results may subsequently be invalidated. Finally, `Legion` [5] focuses more on data movement in distributed systems.

2.3 Energy consumption

The assessment of the energy consumption of computer systems [26] is divided in two parts: the static part which accounts for the leakage of system components, and the dynamic part which varies depending on the workload to which the system is subjected. Therefore, preliminary estimations can be made by defining energy models using the equation $\textit{static energy} + \textit{dynamic energy} \times \textit{usage}$ [13]. For example, `Ecologits` [32] uses different models to estimate energy consumption and environmental impacts. However, reliability of those estimations is limited by the parameters of the models, such as the theoretical energy consumption of a machine. Therefore, it is preferable to use measurement methods based either on external power-meters or software power-meters to study the actual consumption of systems. Software wattmeters retrieve cumulative energy consumption data via manufacturer interfaces, such as Running Average Power Limit (RAPL) for Intel processors or NVIDIA Management Library (NVML) for NVIDIA graphics processors. There are many software solutions available for measuring energy, including but not limited to: `CarbonTracker` [2], `Scaphandre` [27], `CodeCarbon` [33], `PowerAPI` [35], and more recently, `Alumet` [30], and `Ecofloc` [37].

Although these software use the same interfaces to collect energy measurements, differences lays in their additional running overhead. While the use of software solutions facilitates configuration, total energy consumption is often underestimated compared to physical measurements taken using a wattmeter [15].

This is due to the consumption of fans, storage, and network components, which is not always available on certain machines. Nevertheless, more recent studies have shown that newer machines now indicate the energy consumption of different components by distributing consumption among different domains, notably the *psys* domain, which indicates the consumption of the entire machine [29].

3 Our framework: Distributed, Arbitrary, and Heterogeneous Learning (`DAHL`)

Our AI framework, `DAHL`, uses a TGCS, which aims to reduce energy consumption, by dynamically adjusting the workload. Having auto-parallelism [22], is useful for heterogeneous clusters, since an ideal parallelization solution will be found on each machines. `DAHL` uses also StarPU [4] for several reasons, including: hardware and API supports (CUDA, OpenCL, OpenMP, MPI, SimGrid), the maturity of the project, academic use, and documentation. Using a TGCS, developers specify the parallelization implicitly through data dependencies. Fig. 2 shows a simplified example of a Convolutional Neural Network (CNN) using batch parallelism. The forward pass only accesses dataset, weights and biases in read only. Thus, StarPU is able to automatically create a task Directed Acyclic Graph (DAG), parallelized over samples. Then, the first part of dense backward-pass can still be executed in parallel. However, when weights and biases need to be updated, partial results will be aggregated and parameters modified with write permissions, so that a synchronization point will be created in the DAG. The same applies when tensors are divided, e.g. splitting the image batch leads to parallelization, and grouping it again leads to the synchronization of all related precedent tasks.

Nevertheless, parallelization is also constrained by the task granularity itself. As seen in Sec. 2.1, `NNTile` was also implemented using `StarPU`, but the design choices are very different from `DAHL`. They leveraged tensor parallelism with a high task granularity in order to train large models. Although it offers a very high throughput [24] compared to `Pytorch` on models with more than 50 billion parameters, the task granularity is too high to adapt to smaller models. Indeed, when the workload becomes too low, task execution takes less time than task scheduling, thus resulting in struggles to place tasks on workers correctly, creating overheads. Low task granularity will limit the parallelism degree, since a single task can only be placed on a single worker. Thus, the development of a more versatile framework seemed promising. In `DAHL`, we tried to maintain a reasonable task granularity, compatible with batch parallelism but not limited to it. For instance, the pooling layer consists of trivial matrix traversal operations, hence, sizing tasks to handle samples individually is sufficient. However, in the convolution layer, the convolution operations are much heavier, making it worth to parallelize on the filters dimension too, i.e. parallelism degree is equal to *number of feature maps* $\times$ *batch size*. Lastly, note that `DAHL` was developed following the implementation of `Pytorch`. Unit tests are based on `Pytorch` results, and our precision ranges from 10 to 15 digits after decimal point for *float64* values.

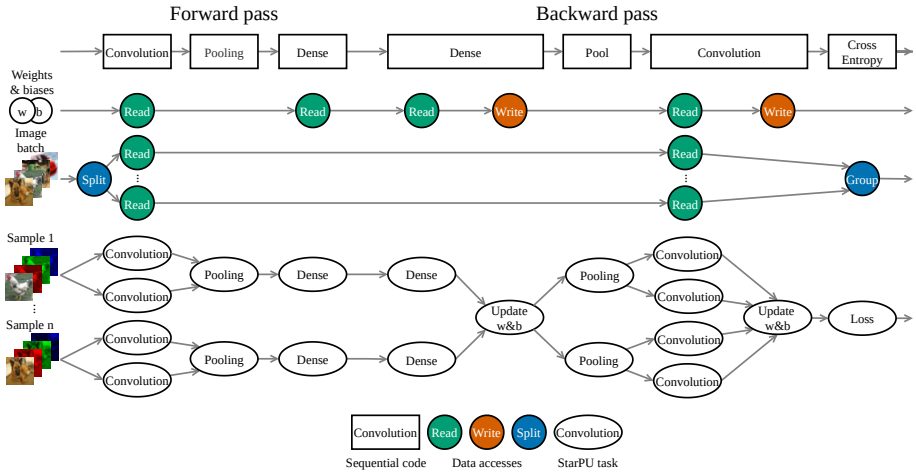

Fig. 2. Simplified example of batch parallelism on a CNN using StarPU. The sequential code shows the chronological order of forward and backward passes, data accesses provide details on the use of the dataset, weights and biases; followed by the resulting task graph built by StarPU.

In this article, the energy comparisons are fair, as all experiments have a similar training accuracy, or even better with `DAHL`.

4 Experimental setup

In order to compare the energy efficiency of our solution with a well-established framework, we created an experimental setup consisting of a CNN to be trained on 3 different image datasets. The same exact CNN architecture was reproduced on `DAHL` and `Pytorch` and consists of: *Convolution, Relu, Max Pooling, Flatten, and Dense*. Image classification was performed on Fashion-MNIST [38], CIFAR-10 [19], and a third dataset with high definition images, derived from the publicly available "Fashion Product Images Dataset" [1], called Big-Fashion in this paper.

All experiments were done on a DELL Precision 3680, with Intel i7-14700K processor at 5.5GHz, 28 CPUs, 2 threads per core, 125Gi RAM. For reproducibility purposes, the operating system used was NixOS 25.05 with a minimal headless setup, and a Nix environment that specifies dependencies to build `DAHL` and run `Pytorch`. Power consumption was measured using the `Alumet` [30] software, as well as the Aoetec Smart Switch 7, a smart plug that monitors energy consumption [2]. The plug's energy logs were collected on a Raspberry Pi 5 to avoid overloading the consumption. Thus, only `Alumet` adds a slight overhead to energy measurements, as it runs directly on the target machine.

Two types of experiments were conducted to evaluate the energy efficiency of our framework. The first one consists of varying the *batch size* on each of the datasets, in order to find the optimal value for subsequent experiments. It

[2] https://aeotec.com/products/aeotec-smart-switch-7/

should be noted that the Big-Fashion images were scaled down to 270x360x3 due to memory and runtime limitations. The second experiment uses multiple scaled version of Big-Fashion to explore in more details the impact of workload on energy consumption. We also performed this experiment on 3 different numbers of cores to explore the trade-offs associated with training with fewer cores than those available on the machine.

Each experiment uses the following hyper-parameters: 0.001 LR, SGD optimizer, 8192 samples, 5 epochs, with 5 runs per configuration for confidence intervals, and 15 seconds of cool-down between experiments. We also compared 3 types of configurations: `DAHL`, `Pytorch` using `Hyper Threading` (`HT`), and `Pytorch` using `OpenMP` as the backend.

5 Results

5.1 Runtime evaluation

Both software and physical wattmeters take measurements every seconds. Furthermore, we decided to use the same number of training epochs, samples, and CNN architecture, for each dataset in order to obtain comparable results. It should be noted that this choice has a negative impact on training accuracy for small image datasets. As an example, on 5 epochs and *batch size* 32, Fashion-MNIST training accuracy is 30%(`Pytorch`) and 28%(`DAHL`), and is 90%(`Pytorch`) and 91%(`DAHL`) for Big-Fashion. However, Fig. 1 shows that training time and energy consumption are linear, allowing us to extrapolate to a higher number of epochs where the accuracy would be correct.

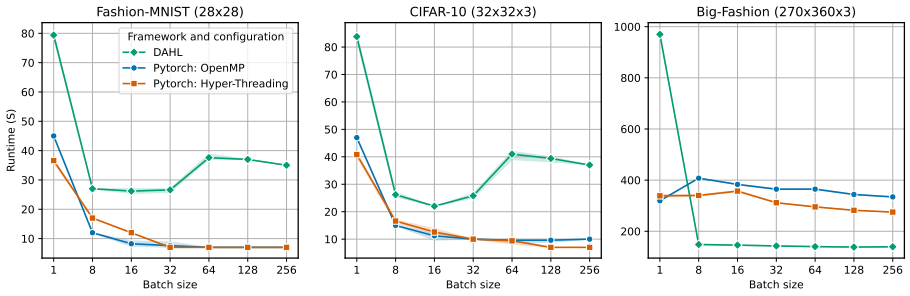

Fig. 3. CNN training on Fashion-MNIST, CIFAR-10, and Big-Fashion, showing runtime in seconds, depending on the *batch size*. Confidence intervals appear in light color, and for each dataset, the error difference averages to 3.16%, 5.57%, and 0.93% respectively.

Fig. 3 shows the training runtime of each datasets as a function of *batch size*. We found that using 5 epochs provided a good compromise in terms of runtime across different dataset sizes, ranging from 10 seconds for the smallest dataset, and to $1,000$ seconds for the worst case with the largest dataset. This keeps the total runtime of each experiment low (since we have to run them several times for the confidence intervals), while providing sufficient measurements on

the small datasets. As expected, increasing the *batch size* reduces the runtime for all datasets and all frameworks. Indeed, increasing the *batch size* reduces data dependency, which facilitates parallelization, particularly on `DAHL` which uses batch parallelism. On CIFAR and Fashion-MNIST, `Pytorch`-based solutions show better runtime regardless of the *batch size*. However, on Big-Fashion, `DAHL` outperforms as soon as it uses batching, while `Pytorch` solutions seem to benefit from it to a lesser extent. Lastly, `DAHL` is the only configuration where increasing the *batch size* beyond 32 leads in overheads, on the Fashion-MNIST and CIFAR-10 datasets.

5.2 Energy comparison

The previous experiment helped us determine the optimal execution time for our configurations: 5 epochs and a *batch size* of 32. It also showed that `DAHL` worked best on datasets containing larger images, while datasets such as Fashion-MNIST and CIFAR-10 were too similar in terms of image sizes to allow us to see significant differences. Thus, the following experiment aims to characterize the point at which `DAHL` becomes interesting in terms of runtime and energy. To do this, we used several down-scaled versions of the Big-Fashion dataset. It was down-scaled 12 times with a factor 2/3, making the largest images (270x360x3) are 110× larger than the images in the smallest dataset (26x34x3). It should be noted that the smallest images are comparable in size to CIFAR-10 images (32x23x3).

Fig. 4, shows the runtime, energy consumption and average CPU usage for each scaled dataset. The experiment was repeated using different amount of cores. For example, using 8 cores, `OpenMP` and `DAHL` are only allowed to use the exact same 8 cores, the others remaining unused, with no particular mechanism being used to make them sleep or save energy. For `Hyper Threading`, we specify the number of threads instead, here, the machine used for the experiment supports running 2 threads per core. Benchmarks validated the intuition that using two times the number of cores leads to better performances.

Analysis of the results reveals that each framework exhibits specific behavior. For `OpenMP`, allocating more cores for training leads to a decrease in runtime, an increase in average CPU usage and higher energy consumption. In contrast, for `HT`, we observe more modest gains in execution time, a decrease in average CPU utilization, but significantly lower power consumption. `DAHL` exhibits behavior comparable to that of `OpenMP`: increasing the number of cores leads to an increase in power consumption, but only for small datasets. However, we observe small gains in runtime on larger datasets, and average CPU usage is slightly higher. Comparing `OpenMP` and `HT` on 8 cores, although the former runs 1.3× longer, it consumes 1.2× less energy and its CPU usage averages 25% compared to 50%. This trend reverses completely at 18 and 28 cores, as `OpenMP` significantly increases its average CPU usage (from 25%, 50%, to 75%) and energy consumption: 1.2× less, 1.28× more, and 1.32× more than `HT`, which follows the opposite trend. `DAHL` results seems sparser as training on 28 cores is 3× times slower on small datasets, but, up to 2× faster on larger datasets. Nevertheless,

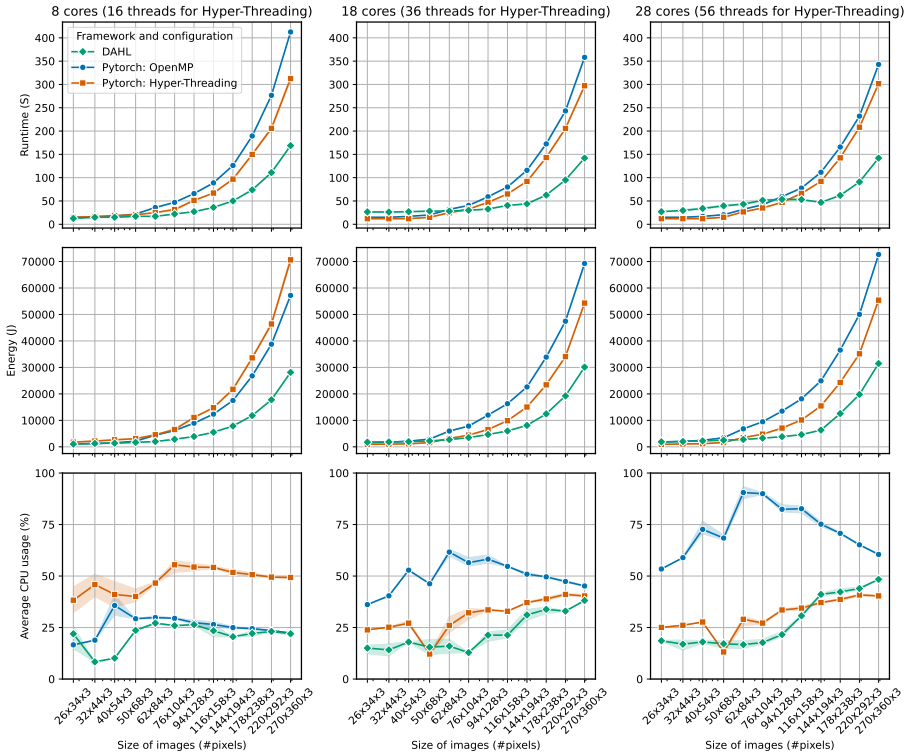

Fig. 4. CNN training on multiple down-scaled versions of Big-Fashion. Three different setups evaluate the runtime, energy consumption, and average CPU usage, depending on the amount of cores used. For `Hyper Threading`, we set the number of threads to the number of CPU cores multiplied by two. Size of images are represented on a logarithmic axis. Confidence intervals appear in transparency, the error difference averages 2.61%, 2.91%, and 7.71% for runtime, energy, and average CPU usage respectively.

average CPU usage and energy consumption are always lower than or equal to those of `Pytorch` solutions.

Fig. 5 shows the energy consumption when using 8 and 28 cores with the energy represented on a logarithmic axis, where dashed lines represent software measurements. Software-based energy measurements consistently underestimated physical wattmeter ones. `DAHL` had the largest bias (34.8%, CI-95±0.04%), followed by `OpenMP` (28.3%, CI-95±1%), and `HT` (26.7%, CI-95±1.3%). Such biases are explained by the fact that on our server, RAPL only measures the CPU package, as the whole package *psys*, seems to be mostly supported on laptops. On both 8 and 28 cores, `DAHL` and `OpenMP` share equivalent energy consumption for the 3 smallest datasets, but `DAHL` does scale better on bigger datasets (from 1.3× up to 4× less energy). Conversely, `OpenMP` experiences a sudden increase in energy consumption for the 62x84x3 dataset. `HT` exhibits the highest energy consumption on 8 cores compared to the other solutions (from 1.17× to 2.3× more). When using all cores, it achieves the best energy consumption for

the smaller datasets (2× less), but undergo the same increase for the 62x84x3 dataset. `DAHL`'s energy increase only occurs with 28 cores for the 178x238x3 dataset, making ot the most energy-efficient solution for datasets containing images larger than 62x84x3, i.e. from 1.2× up to 2.5× lower than `HT`, and 2.6× up to 4.75× lower than `OpenMP`. Furthermore, note that `DAHL`'s runtime on 28 cores only outperforms `Pytorch` runtime after dataset 116x158x3, which means that between this dataset and dataset 62x84x3, energy is lower while runtime is higher. Lastly, training on 8 cores using `DAHL` shows the best overall energy consumption, on average 1.55× less than the second best solution `HT` on 28 cores. As an exception, this last obtains a lower energy for the three first datasets (1.15× less), and `DAHL` on 28 cores also gets a slightly lower energy consumption, on images between 76x104x3 and 116x158x3 (1.1× less).

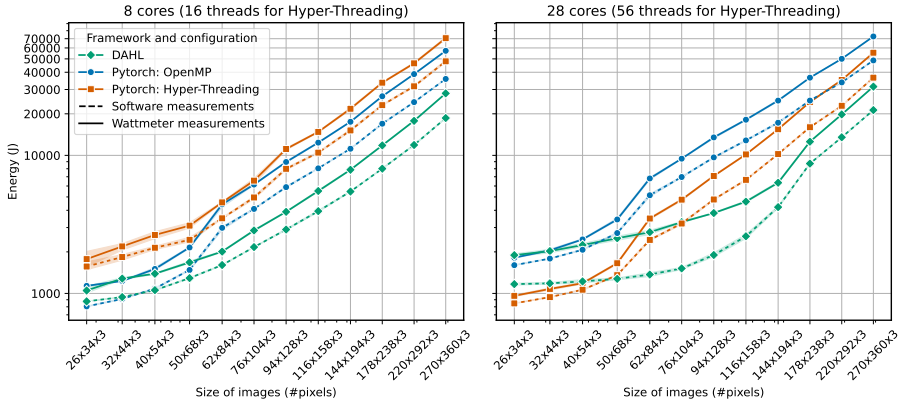

Fig. 5. Logarithmic view of Fig. 4 energy consumption using 8 and 28 cores. Energy consumption of the machine is displayed using both a software-based wattmeter (dashed lines), and a physical wattmeter (solid lines). Confidence intervals are displayed in light color and range from 3.87% to 2.48% for wattmeter, and from 2.50% to 2.18% for software measurements, on 8 and 28 cores respectively.

5.3 Discussions

Previous results highlight the trade-off between performance and energy consumption as well as the sizing of tasks based on workload. [36] notes that the most energy-efficient solution is not always the one with the shortest runtime. Indeed, we also observe in some configurations that `OpenMP` and `DAHL` have higher runtimes than `HT` but lower energy consumption. The common factor leading to such scenario being that the average CPU usage is always lower, in consequence, power consumption is lower, mitigating the cost of training for a longer period of time. In addition, previous works show that in some cases, parallel applications running on more nodes at a slower speed (instead of running on fewer nodes at a higher speed) can help saving energy [12]. This idea is clearly demonstrated with `HT`, as increasing the amount of cores results in lower average CPU usage and energy consumption.

Furthermore, task dimensioning is another key problem, as different workload sizes will perform very differently depending on the implementation used. As an example, `NNTile` [24] was specifically developed for models up to 50 billion parameters. This solution does indeed offer better performance than `Pytorch` for very large models; however, in other cases, it is less efficient due to the low task granularity, which only benefits large workloads. One might assume that finer task granularity offers more parallelization opportunities, but depending on the workload, parallelization logic and synchronization can take longer than the work itself. In our experiments, we observed the same problem with `DAHL` when increasing the number of cores on a small dataset.

For example, `StarPU` profiling tools indicates that the parallelism degree is too high for the images 26x34x3, as running on 28 and 8 cores respectively reports 1.75% and 7.55% of the time spent executing tasks. The remainder is spent on sleep, callbacks, synchronization and scheduling. In contrast, training on images 270x360x3 shows 58.54% and 84.39% for resp. 28 and 8 cores. These figures reveal obvious limitations of our framework but also highlight the significant opportunities for optimizing its performance on smaller workloads. Moreover, it shows encouraging results towards the usage of TGCS in the field of AI. Firstly, because we obtain better energy efficiency in most of the cases. But more importantly because TGCS could tackle task dimensioning issues by using adaptative task sizes, task composability as in `Taskflow` [14], or using novel techniques that adapts scheduling depending on granularity overheads [3].

Finally, extending these principles to distributed training in heterogeneous clusters could offer significant advantages in terms of energy-efficiency and sustainability. Indeed, current techniques used to train large deep learning models require the use of entirely homogeneous clusters, as it is easier to maximize throughput on such platforms. Nevertheless, it may not be the most environmentally friendly solution, as it requires large investments in cutting-edge GPUs, while neglecting existing hardware and infrastructure. Therefore, exploring AI systems capable of handling heterogeneity and dynamically adapting to the workload on each machines seems promising.

6 Conclusion

This work shows that AI frameworks, their backends, and their configurations are not equivalent in terms of energy efficiency. By conducting experiments on several datasets of varying sizes, we demonstrate that their energy efficiency do scale differently depending on the input workload. We show that runtime is not a sufficient metric to hint for energy improvements, but that both average CPU usage and runtime should be taken into account. This questions optimizations that consists solely of improving throughput or using computing units at full load. The solution we propose demonstrates advantageous energy efficiency in most scenarios. This is made possible by the scheduling capabilities of TGCS, which shows encouraging results in terms of workload adaptability. Limitations relate to supporting of a wide range of workloads, as small ones may incur

scheduling overhead, and larger ones miss opportunities for parallelization due to fixed task sizes.

For future works, similar experiments could be performed on GPU to determine whether comparable observations can be made about trade-offs between runtime and energy consumption. In addition, this would allow `StarPU` to schedule tasks on both CPUs and GPUs, making it possible to study the energy overhead of data transfers between these two computing units. In addition to GPU support, adding multi-node capabilities to `DAHL` could provide a cutting-edge solution for training on heterogeneous clusters in an energy efficient way. Lastly, the workload-awareness evaluation could be extended to different hardware. Indeed, frameworks should adapt to workloads, but also to computing capabilities. Exploring solutions that dynamically resize tasks based on computing power, could help mitigating scheduling and synchronization overheads, thereby improving energy efficiency across a wide range of machines.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aggarwal, P.: Fashion product images dataset. https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset (2019), accessed: 2026-02-17
2. Anthony, L.F.W., Kanding, B., Selvan, R.: Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. arXiv preprint arXiv:2007.03051 (2020). https://doi.org/10.48550/arXiv.2007.03051
3. Anvar, S.T., Kaeli, D.: A granularity characterization of task scheduling effectiveness (2026). https://doi.org/10.48550/arXiv.2602.20561
4. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. In: European Conference on Parallel Processing. pp. 863–874. Springer (2009). https://doi.org/10.1007/978-3-642-03869-3_80
5. Bauer, M., Treichler, S., Slaughter, E., Aiken, A.: Legion: Expressing locality and independence with logical regions. In: SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 1–11 (2012). https://doi.org/10.1109/SC.2012.71
6. Beaumont, O., David, J.F., Eyraud-Dubois, L., Thibault, S.: Staronnx: a dynamic scheduler for low latency and high throughput inference on heterogeneous resources. In: European Conference on Parallel Processing. pp. 375–386. Springer (2024). https://doi.org/10.1007/978-3-031-90200-0_30
7. Beaumont, O., David, J.F., Eyraud-Dubois, L., Thibault, S.: Exploiting processor heterogeneity to improve throughput and reduce latency for deep neural network inference. In: 2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). pp. 37–48 (2024). https://doi.org/10.1109/SBAC-PAD63648.2024.00012
8. Cai, Z., Yan, X., Ma, K., Wu, Y., Huang, Y., Cheng, J., Su, T., Yu, F.: Tensoropt: Exploring the tradeoffs in distributed dnn training with auto-parallelism. IEEE Transactions on Parallel and Distributed Systems **33**, 1967–1981 (2022). https://doi.org/10.1109/TPDS.2021.3132413

9. Cardosi, P., Bramas, B.: Specx: a c++ task-based runtime system for heterogeneous distributed architectures. PeerJ Computer Science **11**, e2966 (2025). https://doi.org/10.7717/peerj-cs.2966
10. Dagum, L., Menon, R.: Openmp: an industry standard api for shared-memory programming. IEEE Computational Science and Engineering **5**(1), 46–55 (1998). https://doi.org/10.1109/99.660313
11. Ding, Y., Botzer, N., Weninger, T.: Hetseq: Distributed gpu training on heterogeneous infrastructure. Proceedings of the AAAI Conference on Artificial Intelligence **35**(17), 15432–15438 (2021). https://doi.org/10.1609/aaai.v35i17.17813
12. Freeh, V., Pan, F., Kappiah, N., Lowenthal, D., Springer, R.: Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster. In: 19th IEEE International Parallel and Distributed Processing Symposium. pp. 10 pp.– (2005). https://doi.org/10.1109/IPDPS.2005.214
13. Heinrich, F.C., Cornebize, T., Degomme, A., Legrand, A., Carpen-Amarie, A., Hunold, S., Orgerie, A.C., Quinson, M.: Predicting the energy-consumption of mpi applications at scale using only a single node. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 92–102 (2017). https://doi.org/10.1109/CLUSTER.2017.66
14. Huang, T.W., Lin, D.L., Lin, C.X., Lin, Y.: Taskflow: A lightweight parallel and heterogeneous task graph computing system. IEEE Transactions on Parallel and Distributed Systems **33**(6), 1303–1320 (2022). https://doi.org/10.1109/TPDS.2021.3104255
15. Jay, M., Ostapenco, V., Lefevre, L., Trystram, D., Orgerie, A.C., Fichel, B.: An experimental comparison of software-based power meters: focus on cpu and gpu. In: 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid). pp. 106–118 (2023). https://doi.org/10.1109/CCGrid57682.2023.00020
16. Jia, Z., Lin, S., Qi, C.R., Aiken, A.: Exploring hidden dimensions in accelerating convolutional neural networks. In: International Conference on Machine Learning. pp. 2274–2283. PMLR (2018)
17. Jia, Z., Zaharia, M., Aiken, A.: Beyond data and model parallelism for deep neural networks. Proceedings of Machine Learning and Systems **1**, 1–13 (2019). https://doi.org/10.48550/arXiv.1807.05358
18. Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks. arXiv preprint arXiv:1404.5997 (2014). https://doi.org/10.48550/arXiv.1404.5997
19. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
20. de Langen, P., Juurlink, B.: Leakage-aware multiprocessor scheduling for low power. In: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. pp. 8 pp.– (2006). https://doi.org/10.1109/IPDPS.2006.1639317
21. Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., et al.: Pytorch distributed: Experiences on accelerating data parallel training. arXiv preprint arXiv:2006.15704 (2020). https://doi.org/10.48550/arXiv.2006.15704
22. Liang, P., Tang, Y., Zhang, X., Bai, Y., Su, T., Lai, Z., Li, D., et al.: A survey on auto-parallelism of neural networks training. Authorea Preprints (2022). https://doi.org/10.36227/techrxiv.19522414.v1
23. Martineau, M., McIntosh-Smith, S., Gaudin, W.: Evaluating openmp 4.0's effectiveness as a heterogeneous parallel programming model. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 338–347 (2016). https://doi.org/10.1109/IPDPSW.2016.70

24. Mikhalev, A., Katrutsa, A., Sozykin, K., Oseledets, I.: Nntile: a machine learning framework capable of training extremely large gpt language models on a single node (2025). https://doi.org/10.48550/arXiv.2504.13236
25. Nie, C., Maghakian, J., Liu, Z.: Cannikin: Optimal adaptive distributed dnn training over heterogeneous clusters. In: Proceedings of the 25th International Middleware Conference. p. 299–312. Middleware '24, Association for Computing Machinery (2024). https://doi.org/10.1145/3652892.3700767
26. Orgerie, A.C., Assuncao, M.D.d., Lefevre, L.: A survey on techniques for improving the energy efficiency of large-scale distributed systems. ACM Comput. Surv. **46**(4) (2014). https://doi.org/10.1145/2532637
27. Petit, B.: Scaphandre, https://hubblo-org.github.io/scaphandre/
28. Pheatt, C.: Intel® threading building blocks. J. Comput. Sci. Coll. **23**(4), 298 (2008)
29. Raffin, G., Trystram, D.: Dissecting the software-based measurement of cpu energy consumption: A comparative analysis. IEEE Transactions on Parallel and Distributed Systems **36**, 96–107 (2025). https://doi.org/10.1109/TPDS.2024.3492336
30. Raffin, G., Trystram, D., Richard, O.: Alumet: a modular framework to standardize the measurement of energy consumption. In: Workshop on Performance and Energy Efficiency in Concurrent and Distributed Systems. Springer (2025)
31. Reyes, R., Lomüller, V.: Sycl: Single-source c++ accelerator programming. In: Parallel Computing: On the Road to Exascale, pp. 673–682. IOS Press (2016). https://doi.org/10.3233/978-1-61499-621-7-673
32. Rincé, S., Banse, A.: Ecologits: Evaluating the environmental impacts of generative ai. Journal of Open Source Software **10**(111), 7471 (2025). https://doi.org/10.21105/joss.07471
33. Schmidt, V., Goyal, K., Joshi, A., Feld, B., Conell, L., Laskaris, N., Blank, D., Wilson, J., Friedler, S., Luccioni, S.: Codecarbon: estimate and track carbon emissions from machine learning computing. Cited on **20** (2021)
34. Song, L., Chen, F., Zhuo, Y., Qian, X., Li, H., Chen, Y.: Accpar: Tensor partitioning for heterogeneous deep learning accelerators. In: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 342–355 (2020). https://doi.org/10.1109/HPCA47549.2020.00036
35. SPIRALS, I.: Powerapi, https://powerapi.org/
36. Subramaniam, B., Feng, W.c.: Statistical power and performance modeling for optimizing the energy efficiency of scientific computing. In: 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing. pp. 139–146 (2010). https://doi.org/10.1109/GreenCom-CPSCom.2010.138
37. Valera, H., Ravat, F., Roose, P., Song, J., Vallès-Parlangeau, N.: Ecofloc: outil de mesure énergétique multi-composants. In: CNRIUT 2025 Bayonne-Pays Basque (2025)
38. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017). https://doi.org/10.48550/arXiv.1708.07747
39. Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E.P., et al.: Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). pp. 559–578 (2022). https://doi.org/10.48550/arXiv.2201.12023