A comparison between the usage of free and proprietary solutions for General-purpose computing on GPUs (GPGPU)

Isabella B. do Amaral[1], **Alfredo G. vel Lejbman**[2]

[1]University of São Paulo (USP)
São Paulo, SP – Brazil

isabellabdoamaral@usp.br

[2]Mathematics and Statistics Institute (IME) – University of São Paulo (USP)
São Paulo, SP – Brazil

gold@ime.usp.br

Abstract. *In this research project, we explore the usage of open-source and proprietary solutions for performing general-purpose computing on graphics processing units (GPGPU). We begin by categorizing projects them according to their descriptions using unsupervised machine learning. Then, we compare the performance and usability of the solutions in each category, proposing enhancements to better support the development of open-source GPGPU tooling.*

1. Introduction

1.1. Motivation

Computers have become invaluable assets to research, mainly with respect to numerical problems – those that are largely unsolvable through analytical methods. As needs evolve, not only software but also hardware has become increasingly complex, giving rise to **heterogeneous systems**: these are composed of multiple processing units, each with its own architecture, which will dictate performance characteristics for different workloads.

Graphics Processing Units (GPUs) are optimized for parallel computing, whereby taking advantage of uncoupled data we can cut execution times proportionally to the number of computational units available. Linear algebra routines, which can be thoroughly optimized for parallelism, sit at the core of many scientific applications, such as machine learning and numerical simulations. We can clearly see that GPUs outperform CPUs in such tasks, as shown in [Buber and Banu 2018].

Unfortunately, software for research does not seem to be held to standards as high as more traditional content (see [Sufi et al. 2014]). It is also often developed by researchers that are inexperienced in real world development practices, leading to unmaintainable code (see [Carver et al. 2022]).

One particular development strategy that appeals to modern scientific standards is that of open source, in which the code is fully available to the public. Notably, as open-source software is auditable, it becomes easier to verify the reproducibility of scientific results created with it (see [Barba 2022]), but this also allows for early collaboration between researchers and developers, which can lead to better software design and performance (see [Wilson et al. 2014]).

Building on the practice of open source we also have *free software* (commonly denoted by FLOSS or FOSS): a development ideology that focuses around volunteer work and donations, criticizing corporate interests, and is usually accompanied by permissive or *copyleft* licenses. There is emerging work on the role of FLOSS in science, such as [Fortunato and Galassi 2021], and some initiatives which praise a similar approach (see [Katz et al. 2018, Barker et al. 2022]).

1.2. Proposal

As the GPU is a separate piece of hardware, it is necessary to use specialized APIs to communicate with it. Implementation of such APIs is usually provided by the hardware vendor, as it depends on the specific architecture of the GPU. Graphics APIs provide much-needed abstractions to the programmer, as they are made to orchestrate the entire pipeline, making the choice of API a crucial decision for the robustness of a given application.

In the context of GPGPU, **CUDA** – a proprietary API projected and maintained by NVIDIA – is historically significantly more widespread than any other API, proprietary or not. On the one hand, most open-source alternatives that provide the same functionality are either in their infancy or have not provided what was needed to replace CUDA, examples are, respectively, HIP by AMD, and OpenCL by the Khronos Group[1]. On the other hand, OpenCL-related tooling has recently gained more focus amongst open-source software consultancies, as we can see in [**?**], so it is possible that the situation is changing.

Thus, in this research project, we aim to:

1. Get a more precise figure of the usage of graphics-accelerated open-source software used in scientific applications.
2. Find out how CUDA stands out from open-source alternatives.
3. Find ways to overcome such differences and make it easier for maintainers to add support for open-source APIs.

We choose to analyze **Vulkan** and **OpenCL** as they are the most prominent open-source alternatives to CUDA, and we will also take a look at **HIP** as it is the most recent addition to the list. Both Vulkan and OpenCL are Khronos Group standards, of which more than 170 companies are members, including AMD, ARM, Intel, NVIDIA, and Qualcomm. Their API specifications are open and have ample documentation and adoption, which makes them a good choice for our analysis. HIP (Heterogeneous-Compute Interface for Portability) provides a solution to ease the transition from CUDA to more portable code, thus making it interesting to us as well.

2. Methodology

So far, we have made some progress on objective 1 by analyzing popular graphics-accelerated projects using topic modeling[2].

[1]More generally speaking, there are also competing APIs in the heterogeneous computing space, such as OpenMP, OpenACC, and SYCL, but we focus primarily on GPGPU in this work.

[2]All code related to this work is available at isinyaaa/foss-gpgpu-stack.

2.1. Data collection

We assume that most relevant scientific applications have support for CUDA. NVIDIA's list of accelerated apps provides a non-exhaustive list of applications that use it, and by filtering by relevant categories and manual sorting, we got a list of 141 projects that are open source and use `git` as their version control system. After that, following [Zheng et al. 2018]'s methodology, we gathered the repositories' `README` files and removed sections that were not relevant to our analysis.

2.2. Data analysis

The gathered data was cleansed further, removing stop words, symbols (such as punctuation), standalone numbers, and very short words. We then break up the data into sizeable chunks by tokenizing and lemmatizing it. This reduces the number of words to a more manageable amount, and also allows us to compare words that are similar in meaning.

We then store the results of preprocessing in a count vector, which abstracts away from the order of the words and makes it easier to compare documents. In this step, we also remove words that appear in less than 10% of documents, or in more than 90% of them, as they are either too common or too rare to be relevant to the analysis.

After that, we use Latent Dirichlet Allocation (LDA) to group the documents into N topics. We use the *coherence score* for tuning the hyperparameters of our model, randomizing samples of the data, so we do not overfit the model to the data while testing for different ranges of hyperparameters with varying precision.

We were able to generate clear visualizations for a tuned `gensim` model by using the *t-distributed stochastic neighbor embedding* (t-SNE) method for `pyLDAvis`, as we can see in figure 1. T-SNE works well with sparse data, as it is better able to preserve local structure, but it needs manual tuning.

3. Discussion

By taking the most likely topic for each document, we were able to plot figure 2. Looking at the words composing each topic, it is still non-trivial to label them. Thus, we are still unable to make any conclusions about the repositories' actual subjects.

4. Next steps

4.1. Statistics

As suggested by one of the reviewers, using Named Entity Recognition (NER) to help cluster the repositories might be promising.

4.2. Graphics APIs

As we refine our statistical analyses, we then proceed to profile GPU performance of the projects we have selected, giving special attention to those that implement another graphics API besides CUDA. We will then compare the performance between top projects in each category, and analyze different implementations of similar routines. This requires understanding the different APIs, and how they work. It will also require finding useful benchmarks and tools and possibly implementing our own.

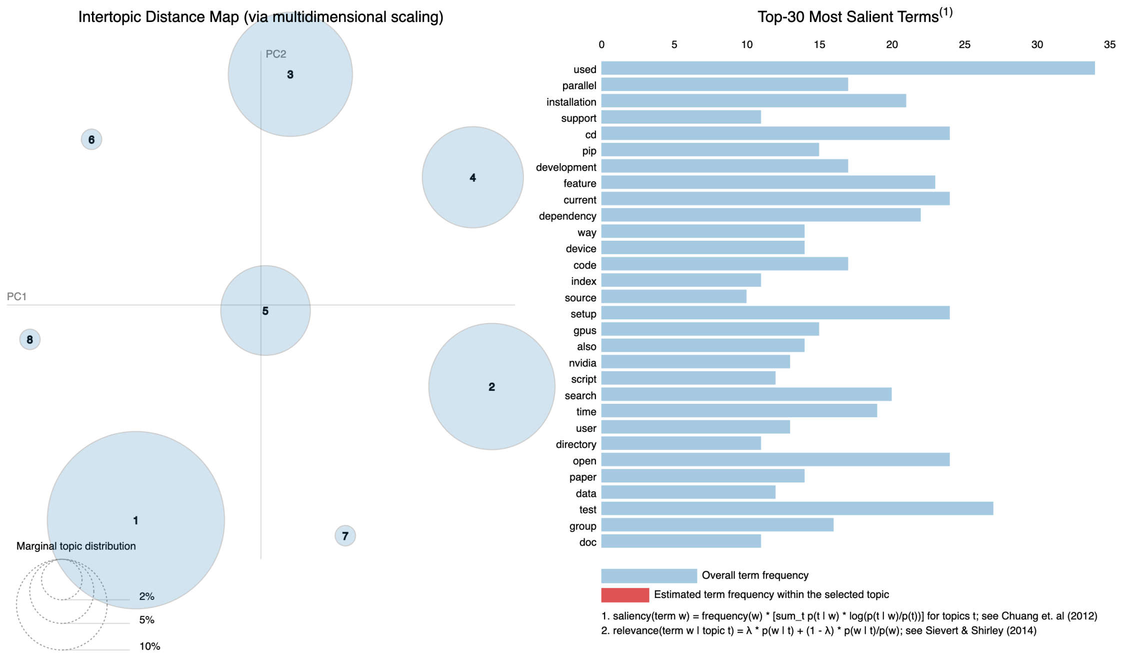

Figure 1. Visualization of gensim's LDA results (trained model with t-SNE dimensionality reduction method).

References

Barba, L. A. (2022). Defining the role of open source software in research reproducibility. *arXiv preprint arXiv:2204.12564*.

Barker, M., Chue Hong, N. P., Katz, D. S., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., et al. (2022). Introducing the fair principles for research software. *Scientific Data*, 9(1):1–6.

Buber, E. and Banu, D. (2018). Performance analysis and cpu vs gpu comparison for deep learning. In *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, pages 1–6. IEEE.

Carver, J. C., Weber, N., Ram, K., Gesing, S., and Katz, D. S. (2022). A survey of the state of the practice for research software in the united states. *PeerJ Computer Science*, 8:e963.

Fortunato, L. and Galassi, M. (2021). The case for free and open source software in research and scholarship. *Philosophical Transactions of the Royal Society A*, 379(2197):20200079.

Katz, D. S., McInnes, L. C., Bernholdt, D. E., Mayes, A. C., Hong, N. P. C., Duckles, J., Gesing, S., Heroux, M. A., Hettrick, S., Jimenez, R. C., et al. (2018). Community organizations: Changing the culture in which research software is developed and sustained. *Computing in Science & Engineering*, 21(2):8–24.

Sufi, S., Hong, N. C., Hettrick, S., Antonioletti, M., Crouch, S., Hay, A., Inupakutika, D., Jackson, M., Pawlik, A., Peru, G., et al. (2014). Software in reproducible research: advice and best practice collected from experiences at the collaborations workshop. In

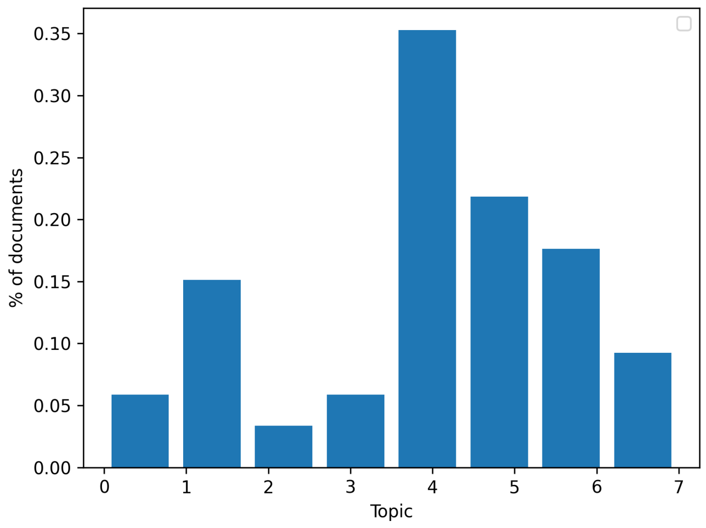

Figure 2. Topic probability distribution of the documents in the corpus (note that topics are numbered from 0).

Proceedings of the 1st ACM sigplan workshop on reproducible research methodologies and new publication models in computer engineering, pages 1–4.

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H., Huff, K. D., Mitchell, I. M., Plumbley, M. D., et al. (2014). Best practices for scientific computing. *PLoS biology*, 12(1):e1001745.

Zheng, Z., Wang, L., Xu, J., Wu, T., Wu, S., and Tao, X. (2018). Measuring and predicting the relevance ratings between floss projects using topic features. In *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*, pages 1–10.