An hybrid linear algebra framework for engineering

An hybrid linear algebra framework for engineering

Paolo Viviani[1,2], Marco Aldinucci[1], and Roberto d'Ippolito[2]

[1]Computer Science Department, University of Torino, Italy
[2]Noesis Solutions NV, Leuven, Belgium

Abstract

The aim of this work is to provide developers and domain experts with simple (Matlab-like) interface for performing linear algebra tasks while retaining state-of-the-art computational speed. To achieve this goal we extend Armadillo C++ library is extended in order to support with multiple LAPACK-compliant back-ends targeting different architectures including CUDA GPUs; moreover our approach involves the possibility of dynamically switching between such back-ends in order to select the one which is most convenient based on the specific problem and hardware configuration. This approach is eventually validated within an industrial environment.

1 Introduction

Dense linear algebra plays a key role in a large number of scientific and industrial applications, spanning from machine learning to computational fluid dynamic, in this context the need for performing such operations as fast as possible is a well known challenge in computer science. The presented work is aimed towards the creation of a software stack that allows the acceleration of dense linear algebra tasks on heterogeneous architectures without any specific expertise in GPU and parallel programming.
In order to understand the design choices involved in the presented approach, it is useful to list a number of requirements derived from a specific industrial use case, which represented the main driving force to the development of this library:

1. State-of-the-art performance on heterogeneous CPU-GPU platforms.
2. Support for advanced linear algebra operations like linear system solving and matrix decompositions.
3. C++ API.

4. Simple interface (possibly similar to MATLAB).
5. Hidden parallelism and GPU specific operations (i.e. memory transfer).
6. Capability to switch from CPU to GPU implementation at runtime.
7. Licensing compatible with commercial use.
8. Support for both Linux and Windows.

The analysis of these requirements and a review of the available tools that would possibly fits such needs highlighted the lack of an off-the-shelf solution in this case, nevertheless one of the targets is to keep the code to be maintained to a minimum, this led to a library approach where existing state-of-the-art components are extended in order to interoperate, being then used as building blocks of a custom implementation.

2 Architecture

The de-facto standard framework for dense linear algebra is the LAPACK-/BLAS stack: it is implemented by many vendors, but it presents a complex API, unsuitable for being employed by a domain expert. The idea behind this work is to identify one or more LAPACK compliant back-ends that allow to perform such operations on CPUs and GPUs in a flexible way, then to wrap them with a convenient high-level interface.

For the presented work, the software stack consits in a front-end library and two back-ends: one dedicated to hybrid platforms with CUDA GPUs and the other dedicated to multicore CPUs. The choice for the wrapper library falls on Armadillo C++ template library [4], which has a clean and textbook-like syntax (deliberately similar to MATLAB) and provides a standard LAPACK-/BLAS interface. For what concern the back-ends we relied on MAGMA [1] (Hybrid/GPU) and OpenBLAS (CPU) [5], which are both well-regarded, state-of-the-art, implementations of the LAPACK and BLAS libraries. The use of Magma allowed us to hide all the GPU specific tasks (like device memory allocation) from the user, since they are completely performed by the library, which can be then used as a drop-in replacement for LAPACK.
To accelerate direct BLAS calls made by Armadillo we leverage the nvBLAS library [3] provided by NVidia which can dynamically offload such operations to cuBLAS when convenient, otherwise it relies on a CPU BLAS implementation, which in this case is OpenBLAS. Figure 1 depicts the components of the presented library and its architecture.

2.1 Dynamic back-end switching

The key feature of this implementation is the ability to switch between the two back-ends dynamically at runtime, it allows to fall-back on a CPU-only

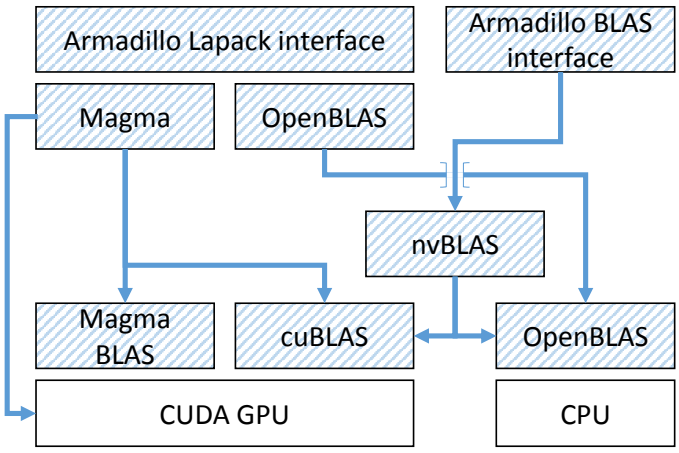

Figure 1: Architecture of the presented library.

implementation when a CUDA GPU is not available and it is possible to always choose the most convenient back-end for a given hardware configuration and matrix size, moreover the user code does not need to be modified when porting it between different platforms.
This result has been achieved by modifying Armadillo's LAPACK interface by means of C++11 function pointers, this approach allowed to keep the code to be maintained internally to a minimum and it is flexible enough to make the integration of additional back-ends straightforward.
Listing 1 shows the typical usage of the presented framework, which includes a very small number of additional lines of code with respect to standard Armadillo code, in this sense it cuts down the development time and the learning curve for a domain expert who needs to implement a complex business logic.

```
1 // Check if supported CUDA driver and device are present, then initializes
      Magma
2 arma::arma_magma_init();
3 // Set the Magma back-end at runtime
4 arma::arma_set_back-end(1);
5 arma::arma_set_device(0);
6 // Domain logic
7 // ...
8 // Finalizes Magma back-end for a clean exit
9 arma::magma_finalize();
```

Listing 1: Typical usage.

3 Results and future work

The presented library has been thoroughly tested also within an industrial environment and it proved itself fully compliant to the requirements listed above.

It allowed the domain experts to achieve significantly better performance with respect to the previous internal implementations, while speeding up the development thanks to its textbook-like API.
We will also present different comparison between the two back-ends, that will show the existence of a break-even point for the two back-ends: for matrices smaller than a certain threshold OpenBLAS is faster, while the situation is reversed for larger matrices. Moreover, the size that represents this threshold varies depending on the specific hardware configuration, to the extreme case where there is no break-even for Magma due to a very low-end GPU. In this sense there is no preferred back-end in principle and this justifies the need for the runtime switching capability, as the back-end of choice can be defined based on the platform.

As a further development, from the point of view of the industrial application, the main concern is to extend the support for multiple hardware architecture, namely to support GPUs from different vendors; this is expected to be achieved via the use of clMagma [2], but its early development stage prevented us from making a working implementation. Whether a LAPACK replacement supporting OpenCL would be available in the future, the integration in the presented library would be straightforward.
On the performance side it would be interesting to test if other implementations of LAPACK, like PLASMA [1], can eventually achieve faster execution times when compared to the back-ends presented here. Also in this case, it would be trivial to add a different computing back-end to the framework, given that it provides a LAPACK-like API.
At last, we envisioned the possibility to exploit the thread safety of all the components (with some caveats) in order to perform computations on different back-ends in a concurrent way, hence providing an additional layer of parallelism.

4 Acknowledgments

This work has been partially supported by ITEA2 project 12002 MACH, the EU FP7 REPARA project (no. 609666), and the NVidia "GPU research center" programme.

References

[1] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *J. Phys.: Conf. Ser.*, 180:012037, 2009.

[2] C. Cao, J. J. Dongarra, P. Du, M. Gates, P. Luszczek, and S. Tomov. clmagma: high performance dense linear algebra with opencl. In

S. McIntosh-Smith and B. Bergen, editors, *IWOCL*, pages 1:1–1:9. ACM, 2014.

[3] NVIDIA Corporation. Cuda toolkit documentation. `http://docs.nvidia.com/cuda/eula/index.html`.

[4] C. Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. In *NICTA*, Australia, 2010.

[5] Q. Wang, X. Zhang, Y. Zhang, and Q. Yi. Augem: Automatically generate high performance dense linear algebra kernels on x86 cpus. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 25:1–25:12, New York, NY, USA, 2013. ACM.