A flexible Numerical Framework for Engineering - a Response Surface Modelling Application

P. Viviani[1,2,], M. Aldinucci[1], R. d'Ippolito[2], J. Lemeire[3], D. Vucinic[3]

[1]Dipartimento di Informatica,

Università degli Studi di Torino, Torino, IT.

[2]Noesis Solutions, Leuven, BE.

[3]Dept. of Electronics and Informatics (ETRO),

Vrije Universiteit Brussel, Brussels, BE.

Abstract

This work presents an innovative approach adopted for the development of a new numerical software framework for accelerating dense linear algebra calculations and its application within an engineering context.

In particular, response surface models (RSM) are a key tool to reduce the computational effort involved in engineering design processes like design optimization. However, RSMs may prove to be too expensive to be computed when the dimensionality of the system and/or the size of the dataset to be synthesized is significantly high or when a large number of different response surfaces has to be calculated in order to improve the overall accuracy (e.g. like when using ensemble modelling techniques).

On the other hand, the potential of modern hybrid hardware (e.g. multicore, GPUs) is not exploited by current engineering tools, while they can lead to a significant performance improvement. To fill this gap, a software framework is being developed that enables the hybrid and scalable acceleration of the linear algebra core for engineering applications and especially of RSMs calculations with a user-friendly syntax that allows good portability between different hardware architectures, with no need of specific expertise in parallel programming and accelerator technology.

The effectiveness of this framework is shown by comparing an accelerated code to a single-core calculation of a radial basis function RSM on some benchmark datasets. This approach is then validated within a real-life engineering application and the achievements are presented and discussed.

Keywords: Response surface modelling, GPU computing, linear algebra, armadillo

1 Introduction

Response surface modelling (RSM) is a key tool when it comes to engineering design optimization: in most real-life use cases the engineer knows its domain of design parameters as a discrete dataset, namely as a black-box model, while most optimization strategies need a certain degree of continuity in order to be applied. Regression (i.e. approximation and interpolation) is then necessary to obtain a continuous or differentiable function that well represents the underlying model. Moreover, computing new points on an analytical regression function is significantly cheaper than obtaining new samples of the dataset, as they come from experimental data or expensive numerical simulations.

In this context, response surface modelling can address both the sparsity of the dataset and the expensiveness of producing new points, since it provides a continuous and analytical function that can be easily evaluated when performing optimization; however, there are a number of situations in which even the RSM can be too expensive computationally-wise to actually represent an advantage compared to numerical simulation. When the dataset is very large and its dimensionality is possibly high (in terms of inputs and outputs), then the time needed to obtain an accurate regression becomes not acceptable, so the response surface is not able anymore to fulfil its purpose of reducing the computational effort.

A possible solution to this issue comes from the recent development of new computational architectures, namely multicore and manycore platforms, which allow to speed up numerical calculation even on off-the-shelf workstation. Unfortunately, it is not straightforward to port existing code like sophisticated regression tools to such architectures, since specific expertise in parallel and GPU computing is needed. The aim

of this work is hence to address this issue by introducing a numerical framework that can accelerate the most numerical intensive parts of well-known regression methodologies and needs no specific low-level coding expertise from the domain expert who uses it; the actual application of this framework within a response surface model is showed and validated, both from the performance point of view and from the usability point of view.
The paper is structured as following: Section 2 explains in detail the use case that drove the development of this work, Section 3 presents the main related work in terms of available and competing tools for high-performance linear algebra computations, Section 4 illustrates the architecture of the framework developed by the authors and Section 5 reports the experimental validation of such framework.

2 Industrial Use Case

The response surface modelling use case that served as benchmark for this work has been provided by Noesis Solutions NV, a simulation innovation partner to manufacturers in automotive, aerospace and other engineering-intense industries. Specialised in simulation process integration and numerical design optimization (PIDO), its flagship software Optimus leverages Noesis' experience in optimization and system integration methodologies to increase the efficiency of engineering practices and processes. Noesis research tracks include process integration, extraction and exploitation of engineering knowledge within multidisciplinary industrial processes, advanced methods for modelling and optimization of the behaviour of large engineering systems in the virtual prototype stage, parallelization of computational effort, and assessment of quality and robustness of the final product. The implementation of RSM in the context of design engineering and optimization is a well-known technique usually referred to as Metamodel Based Design Optimization (Booker et al. 1999).
One of the most significant functionalities of Optimus is indeed the calculation of response surface models related to arbitrarily complex engineering simulation workflows: this feature makes heavy use of linear algebra operations and performing such operations as fast as possible is a paramount in order to achieve the purpose of response surface

modelling. The presented work is focused on a specific RSM: the *radial basis function interpolation* (RBF), but the techniques presented here are flexible enough to be applied to other models as well as completely different engineering fields, given that they require intensive linear algebra computations.

The calculation of the RBF involves the resolution of a linear system, which can be expressed as $Ax = b$ and where A is rank-deficient; this system can be solved by computing the *Moore-Penrose pseudoinverse* of A by means of a *singular value decomposition* (SVD) (Golub and Van Loan 1996). Accelerating the SVD represents the main requirement to the tool here presented since it is the most expensive operation of the whole interpolation process, accounting for more than the 95% of computing time. For this reason, the SVD represents the main benchmark function for the performance of this numerical library.

The main goal of this work is to provide a tool that allows engineers and domain expert to exploit the computing capabilities of modern architectures to perform numerical linear algebra; in this sense, given the use case and the industrial context where the tool is expected to be used, a number of requirements that drove the development of this work have been identified:

- State-of-the-art performance on heterogeneous CPU-GPU platforms.
- Support for advanced linear algebra operations like linear system solving and matrix decompositions.
- Easy incorporation into existing C++ code.
- Simple interface (possibly similar to MATLAB).
- Hidden parallelism and GPU specific operations (i.e. memory transfer).
- Capability to switch from CPU to GPU implementation at runtime.
- Minimal amount of code to be maintained.
- Licensing compatible with commercial use.
- Support for both Linux and Windows.

From here on we will show how the proposed library is compliant with such requirements and how it performs within the use case introduced above, both in terms of usability and performance.

3 Related Work

Given the large spectrum of applications for dense linear algebra, it is not surprising that several tools exist to perform such operations; a significant subset of such tools is also developed with performance in mind, but only recently heterogeneous architectures like CPU/GPU ones are being targeted by such tools. The *de-facto* standard for what concern linear algebra computations is the software stack composed of BLAS (Dongarra et al. 1988) and LAPACK (Anderson et al. 1999). The first library is focused on elementary operations like matrix-vector and matrix-matrix multiplications, while the second implements more complex functions like matrix decompositions (including the SVD), least squares and linear system solving. Both libraries are also implemented by third parties (either commercially or open-source) that preserve the API while they modify the internal mechanisms, possibly targeting different architecture like GPUs. Below we report the most notable implementations of BLAS and LAPACK:

BLAS

- Netlib BLAS, original implementation (Dongarra et al. 1988)
- Intel MKL
- OpenBLAS (Wang et al. 2013)
- NVidia cuBLAS, dedicated to CUDA GPUs (NVIDIA Corporation 2016)
- clBLAS, dedicated to OpenCL GPUs

LAPACK

- Netlib LAPACK, original implementation (Anderson et al. 1999)
- Magma, hybrid CPU-GPU implementation (Agullo et al. 2009; Tomov et al. 2010)
- Plasma, multithreaded implementation (Agullo et al. 2009)

- Intel MKL
- CULA Dense (Humphrey et al. 2010)

While most of these libraries provide very high-performance execution of the linear algebra operations that we are concerned about, there is a main drawback that prevents their application in an engineering environment: their interface is so complex such that specific expertise is needed to be used directly. In this context a number of higher-level libraries have been developed in order to provide access to BLAS and LAPACK functionalities to domain experts and engineers. Such wrappers usually provide a large catalogue of functionalities that goes far beyond linear algebra (e.g. array slicing, sorting, cumulative summing), but for the scope of this work we are going to focus only to BLAS and LAPACK capabilities. A number of these tools have been evaluated for an application to the use case presented above, in particular we considered:

- Armadillo (Sanderson 2010)
- Eigen (Guennebaud and Jacob 2010)
- ArrayFire (Yalamanchili et al. 2015)
- ViennaCL (Tillet et al. 2013)
- LAMA (Kraus et al. 2012)

Armadillo and Eigen are not intended to be used on different platforms other than the CPU, while ArrayFire, ViennaCL and LAMA provide support for different back-end libraries to target both CPU and GPU.

With respect to our requirements, the last three tools listed above are good candidates, but there are drawbacks: LAMA and ViennaCL cannot switch from GPU to CPU at runtime, while ArrayFire, as will be shown in Section 5, presents significant performance issues when considering the SVD.

These limitations lead to a different implementation approach: given that is required to keep the amount of code to be maintained to a minimum, existing state-of-the-art tools have been reused as much as possible in order to build a software stack that actually complied with the industrial requirements. Section 4 will present the components used and will outline the architecture of such software stack.

4 Architecture and Usage

The presented software stack is structured as outlined in Figure 1: a user friendly interface is provided to the domain expert, then the code of such interface is provided with mechanisms so it becomes possible to switch between different computing back-ends at runtime. In this way the state-of-the-art performance of existing BLAS/LAPACK implementations can be leveraged without burdening the user with a cumbersome API or with the need of taking care of the GPU specific mechanisms.

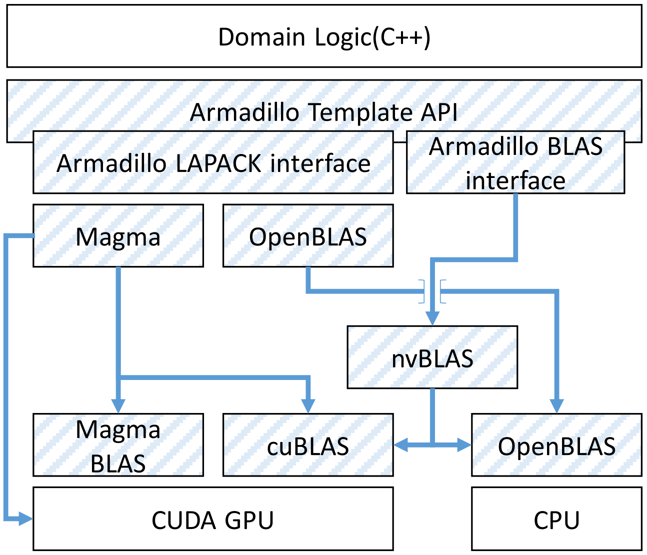


Figure 1 *Proposed software stack*

Below, the individual components are presented, along with the reasons that guided us in the choice.

Armadillo

The interface exposed to the user is Armadillo, a C++ template library for linear algebra with a high-level API, which is deliberately similar to Matlab (Sanderson 2010). It provides a large number of functions to manipulate custom objects representing vectors, matrices and cubes (namely 3rd-order tensors); to perform the most intensive computations it provides an interface to BLAS and LAPACK implementations (hereafter we will refer to such implementation as the back-end). It employs a number of internal layers in order to translate simple function calls like

```
Solve(A,b); //Solves the linear system Ax=b
```

to more complex but equivalent LAPACK syntax

```
dgesv( &n, &nrhs, a, &lda, ipiv, b, &ldb, &info );
```

Armadillo is designed to support whatever library providing an API compliant with BLAS and LAPACK, such as MKL or OpenBLAS. It is also able to perform a few of the operations included in the back-end with its own implementation, but they are not designed for high performance. Its modularity and the simple and user-friendly interface guided our choice towards Armadillo as the API presented to the developer.

OpenBLAS

The default CPU back-end is OpenBLAS: an open-source implementation of BLAS which, given the benchmarks provided by the authors, can be compared to the best-in-class proprietary libraries like Intel MKL (Wang et al. 2013). The standard distribution of OpenBLAS also provides LAPACK functions, some of which are further optimised by the authors. The interface is compatible with the standard distribution of BLAS/LAPACK. The very easy build-and-deploy workflow, along with its solid performance, was the key point for the adoption of OpenBLAS as the reference CPU back-end.

Magma

The development of Magma is aimed to replace LAPACK on heterogeneous architectures, with the typical Multicore+GPU platform as a paradigmatic example (Agullo et al. 2009; Tomov et al. 2010), in this sense we used it as the GPU back-end for the presented work. The Magma library employs Directed Acyclic Graphs (DAGs) in order to dispatch the different tasks related to a given computation to different cores/devices, taking data dependencies into account and aiming for the best exploitation of the available hardware. The motivation that drove interest to Magma is twofold: it does not require the user to take care of data transfer between the host and the device, and, its API is only marginally different from the standard LAPACK interface. These two features make Magma a good candidate for a *drop-in* replacement of LAPACK on heterogeneous platforms.

NVidia nvBLAS

Direct BLAS calls from Armadillo would normally be handled by OpenBLAS, in this case the nvBLAS library (NVIDIA Corporation 2016) provided by NVidia has been leveraged in order to offload the operation to a GPU when available. nvBLAS intercepts standard BLAS calls and, when available, performs the operation on the GPU using the cuBLAS implementation from NVidia. The use of nvBLAS allowed us to leverage the GPU for what concern the BLAS operations, while keeping Armadillo code unchanged.

Usage

As already stated, we provided Armadillo with mechanisms to handle multiple back-ends while keeping the code to be maintained to a minimum, in this sense we also tried to modify Armadillo's interface as little as possible. Given a user code written using the Armadillo syntax, only a few more lines are needed in order to take advantage of the Magma GPU back-end, the typical usage is showed below:

```
// Check supported CUDA device, then initialises Magma
arma::arma_magma_init();
// Set the Magma back-end at runtime
arma::arma_set_backend(1);

// User code
// ...

// Finalises Magma back-end for a clean exit
arma::magma_finalize();
```

While details will be provided in the next section, this architecture satisfied all the requirements listed in Section 2.

5 Validation

As stated in Sect. 2, the goal of this work is to provide a tool that allows developers to easily leverage modern computing architectures to perform intensive linear algebra operations. As a part of the experimental validation, the Optimus Radial Basis Function interpolation has been re-implemented using Armadillo. Both the usability of the framework and its performance in comparison to the original implementation, where the SVD is largely based on (Press et al. 2002), have been assessed. At last, even if the tools implemented are very well-regarded, the numerical accuracy of the framework with respect to the RBF interpolation has been assessed too.

For what concern the usability, positive feedback has been collected from industrial developers, in particular with respect to the very easy implementation of numerical algorithms and the very little effort required to port such algorithm on high-performance architectures; the productivity improvement can be roughly estimated to reduce development time by 50% with respect to writing plain C++ code from scratch.

Performance

On the performance side we carried out several tests to assess that: 1) the computation time required by the RBF significantly benefits from the new implementation; 2) the two back-ends are somehow complementary and there is a significant advantage given by the ability to switch between the two at runtime.

The first benchmark is performed on a variable-sized synthetic dataset produced by an analytical function, in order to identify the evolution of the model building time versus the size of the dataset.

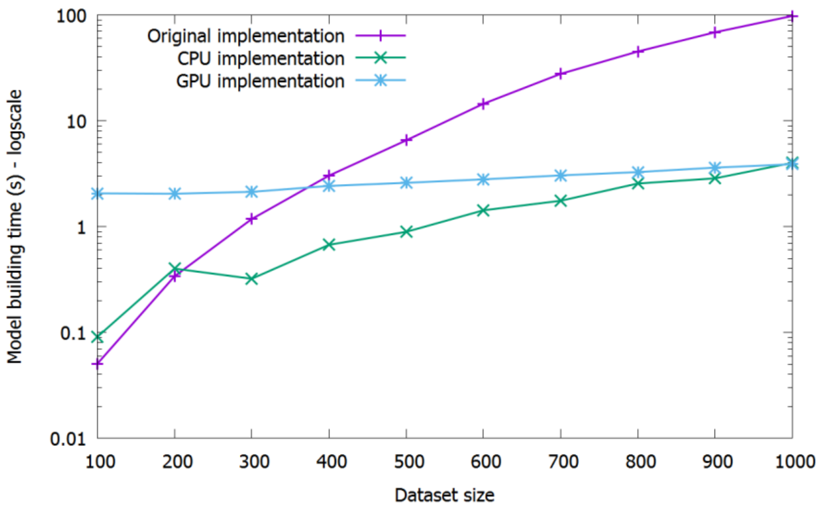


Figure 2 *Comparison of model building times for a variable-sized dataset.*
2x Xeon E7- 4820@2.00GHz, NVidia Tesla K20c

The results presented in Fig. 2 highlight how the Armadillo's implementation becomes almost two orders of magnitude faster than the original one as the size of the dataset grows; let us remark that, for this range of dataset sizes, OpenBLAS outperforms Magma noticeably.

This result becomes significant as we consider a real-life engineering problem: the following test has been performed on a large dataset (4149 samples) provided by a major manufacturer in the aerospace industry. Figure 3 shows, in a logarithmic scale, the model building times for the same three cases considered before.

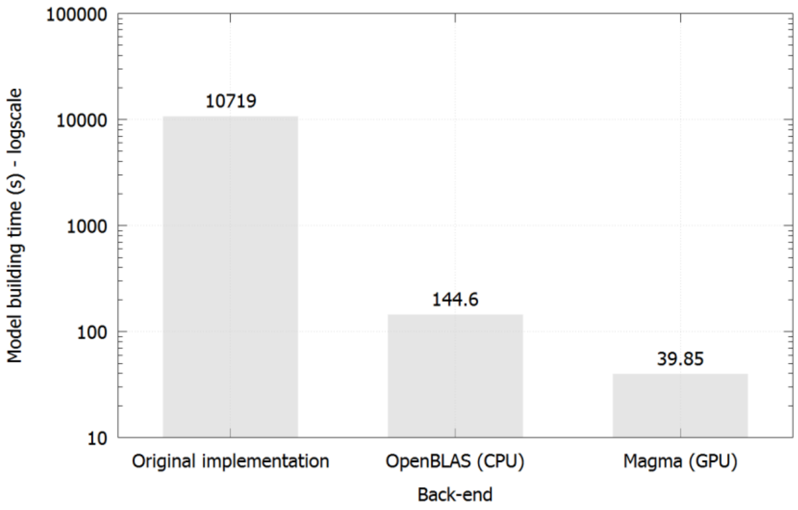


Figure 3 *Model building times for a large manufacturing dataset.*
2x Xeon E7- 4820@2.00GHz, NVidia Tesla K20c

Let us underline the advantage in terms of computing time provided by our framework: this result is consistent with the two orders of magnitude gap showed by the rightmost part of Fig. 2 and reduces the model building time from almost three hours to one and a half minute, restoring the feasibility of a response surface modelling approach. It is also important to note that, for such a large dataset, a visible gain by using the GPU back-end with respect to OpenBLAS has been experienced.

While above the comparison with the original implementation of the RBF has been considered, now the focus is put on comparing the two BLAS/LAPACK back-ends of this framework. The previous experiments showed a substantial advantage in favour of OpenBLAS for datasets of size up to a thousand of samples, while the real-life dataset showed how the GPU takes the edge for very large problems. In this sense it would be useful to understand the behaviour of the back-ends in the region between these two cases: Figure 4 shows the break-even dataset size for which the GPU becomes actually faster.

In fact, the expectation is that the fastest back-end depends on both, the problem size and the hardware configuration. To prove such assumption, experiments on different machines have been performed considering only the most expensive part of the RBF interpolation: solving the rank-deficient linear system $Ax = b$.

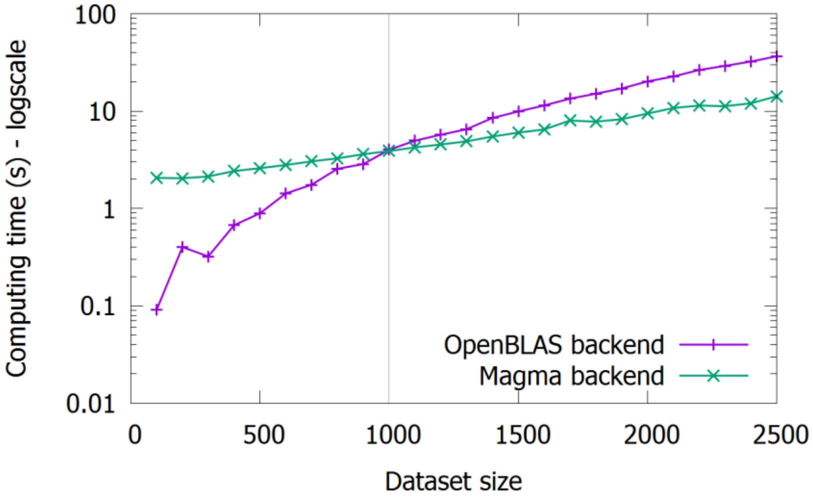


***Figure 4** Model building times for different dataset sizes.*
2x Xeon E7- 4820@2.00GHz, NVidia Tesla K20c

Fig. 5 shows three different hardware configurations and the statement that there is not a back-end that is in principle faster than another holds true: different back-ends behave better based on the hardware configuration and the problem size and, above all, the break-even point for Magma to take the edge gets larger as the GPU gets more low-end, to the extreme case in which there is no break-even at all. In this sense the possibility to switch among the back-ends at runtime opens up the possibility to implement a policy that always chooses the fastest based on the problem and the platform.

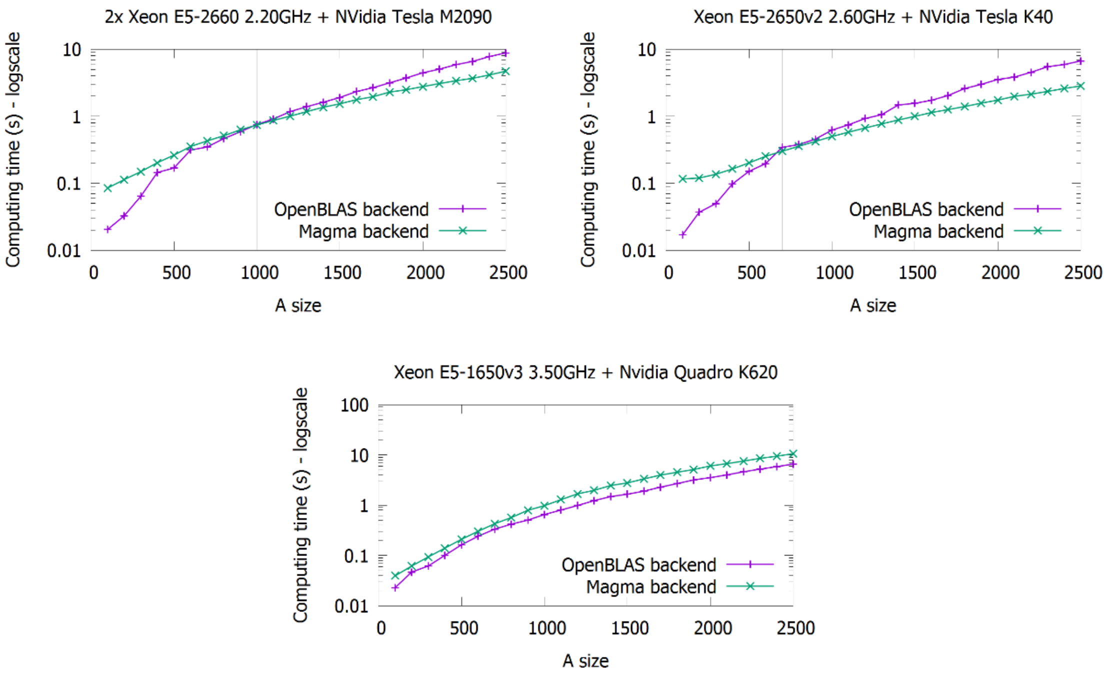
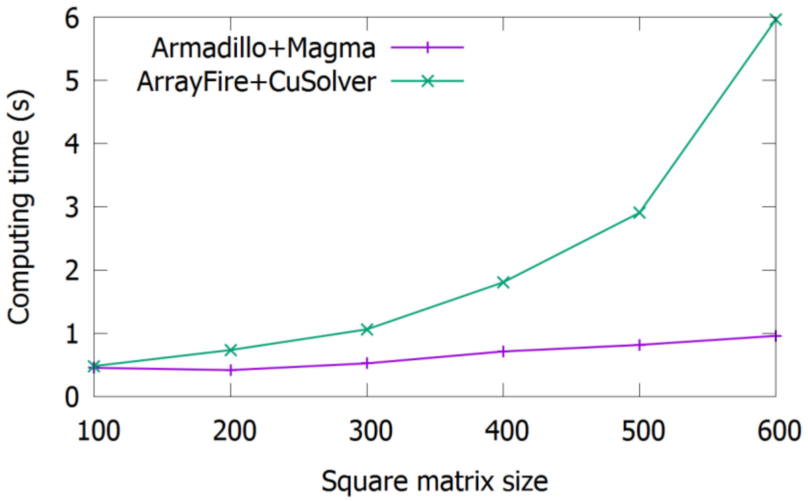


Figure 5 *Linear system $Ax = b$ solved on different machines with different hardware configuration.*

As a final performance benchmark it is interesting to show how this framework significantly outperforms the only competing tool that possibly meets the requirements, when considering the calculation of the SVD on GPU. It has been stated in Section 3 that ArrayFire revealed a significant performance issue, so it was discarded despite being a good candidate. Fig. 6 shows the comparison of computing times for ArrayFire and Armadillo+Magma. Let us note that ArrayFire leverages the cuSolver LAPACK replacement provided by NVidia in order to perform the SVD.



p

Figure 6 *Computing time for the SVD of a square matrix.*

It is noticeable how Magma significantly outperforms cuSolver as the dimension of the dataset increases.

Accuracy

In order to verify the numerical accuracy of the presented software stack, the interpolation results obtained on a test dataset by the original implementation and by both the back-ends leveraged by Armadillo have been compared. To produce a measure of accuracy we measured the errors on a different and dense validation dataset and we expected those errors to be at most equal in all the cases. Fig. 7 shows how the Armadillo implementation produces a slightly more accurate implementation compared to the original one: this implies that the faster calculation does not affect final accuracy, which is possibly better than expected. Tuning of the tolerance parameter for the SVD is possible with Armadillo, but we did not consider such intervention as relevant, since under both the performance and accuracy point of view the framework proved itself to be fully satisfactory for the scope of this work.

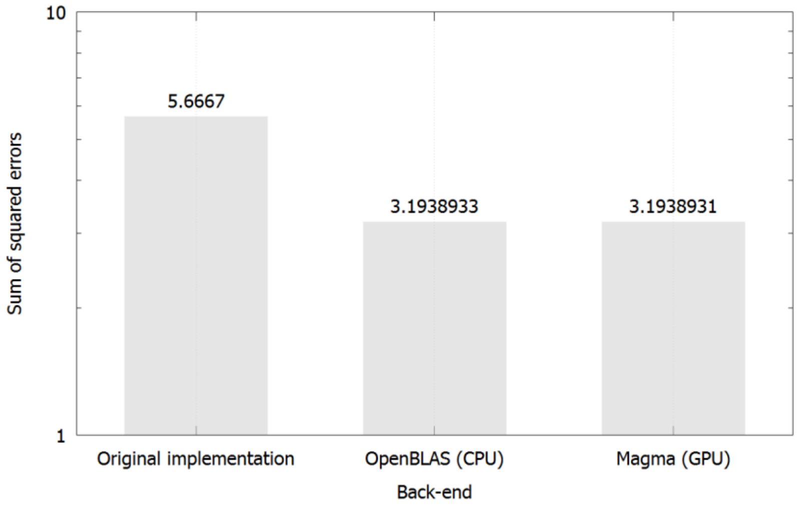


Figure 7 *Sum of squared errors calculated on a 1000 points validation dataset. RBF interpolation built on a 500 points synthetic dataset.*

6 Conclusion

Starting from the specific use case of response surface modelling, in this work we introduced a software stack designed to allow domain experts and engineers to exploit high-performance architectures in a transparent way while developing linear algebra intensive applications. To achieve such result, we carefully extended the Armadillo library, which provides a Matlab-like interface for linear algebra objects and operations, in order to both integrate it with different high-performance computing back-ends and to allow the user to transparently switch between such back-ends at runtime.

The resulting framework has been validated within the industrial context of the use-case provider, considering either the computing performance as well as the usability: the latter allowed for a code development time decrease of roughly 50%, while the former has largely outperformed the original implementation, as well a competing tool. In this sense we can state that the requirements listed during the analysis of the use-case are completely met by this framework, while providing fully satisfactory performance in both synthetic test cases as well as actual aerospace manufacturing problems.

Future Development

Since the main purpose of this work is to provide a general tool for developers, we expect that in the near future more Response Surface Models other than the RBF interpolation will benefit from the acceleration provided by this approach. Moreover, we also expect that new models will be implemented using Armadillo, significantly accelerating the coding process.

For what concern the architecture of the framework, we propose an improvement based on the results showed by Fig. 4 and 5: we observed that the performance is significantly dependent on the hardware configuration and the problem size, in this sense the idea is to implement a set on policies that, based on early performance evaluation, automatically selects the fastest back-end on which the given dataset should be processed.

At last, a possible development concerns the identification of a suitable BLAS/LAPACK back-end for OpenCL in order to target a wider range of accelerators like AMD and Intel GPUs.

Acknowledgments

This work has been partially supported by ITEA2 project 12002 MACH, the EU FP7 REPARA project (no. 609666), the EU H2020 Rephrase project (no.644235) and the “NVidia GPU research centre” programme.

References

Agullo E, Demmel J, Dongarra J, et al (2009) Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. J Phys Conf Ser 180:12037.

Anderson E, Bai Z, Bischof C, et al (1999) LAPACK Users’ Guide, Third. Society for Industrial and Applied Mathematics, Philadelphia, PA

Booker AJ, Dennis JE, Frank PD, et al (1999) A rigorous framework for optimization of expensive functions by surrogates. Struct Multidiscip Optim 17:1–13.

Dongarra J, Croz J Du, Hammarling S, Hanson RJ (1988) An Extended Set of FORTRAN

Basic Linear Algebra Subprograms. ACM Trans Math Softw 14:1–17.

Golub GH, Van Loan CF (1996) Matrix Computations (3rd Ed.). Johns Hopkins University Press, Baltimore, MD, USA

Guennebaud G, Jacob B (2010) Eigen v3.

Humphrey JR, Price DK, Spagnoli KE, et al (2010) CULA: hybrid GPU accelerated linear algebra routines. In: Modeling and Simulation for Defense Systems and Applications V. {SPIE}-Intl Soc Optical Eng,

Kraus J, Förster M, Brandes T, Soddemann T (2012) Using LAMA for efficient AMG on hybrid clusters. Comput Sci - Res Dev 28:211–220.

NVIDIA Corporation (2016) CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/eula/index.html. Accessed 25 Nov 2016

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2002) Numerical Recipes in C.

Sanderson C (2010) Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. In: NICTA. Australia,

Tillet P, Rupp K, Selberherr S, Lin C-T (2013) Towards Performance-Portable, Scalable, and Convenient Linear Algebra. In: 5th USENIX Workshop on Hot Topics in Parallelism. USENIX, Berkeley, CA,

Tomov S, Dongarra J, Baboulin M (2010) Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Comput 36:232–240.

Wang Q, Zhang X, Zhang Y, Yi Q (2013) 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. ACM, New York, NY, USA, p 25:1--25:12

Yalamanchili P, Arshad U, Mohammed Z, et al (2015) ArrayFire - A high performance software library for parallel computing with an easy-to-use API. AccelerEyes, Atlanta

List of figures