Lightning Talk: Creating a Standardised Set of Batched BLAS Routines

Jack Dongarra
Innovative Computing Laboratory,
University of Tennessee,
Oak Ridge National Laboratory,
TN, USA.
School of Computer Science and School of Mathematics,
The University of Manchester,
Manchester, UK.
dongarra@icl.utk.edu

Sven Hammarling, Nicholas J. Higham, Samuel D. Relton,
Pedro Valero-Lara and Mawussi Zounon
School of Mathematics,
The University of Manchester,
Manchester, UK.
sven.hammarling, nick.higham, samuel.relton,
pedro.valero-lara, mawussi.zounon@manchester.ac.uk

***Abstract*—One trend in modern high performance computing is to decompose a large linear algebra problem into thousands of small problems that can be solved independently. For this purpose we are developing a new BLAS standard (Batched BLAS), allowing users to perform thousands of small BLAS operations in parallel and making efficient use of their hardware. We discuss and introduce some details about how we are implementing this new scientific standard as well as some ideas about the upcoming processes that we plan to follow during its development.**

***Index Terms*—BLAS; Scientific Computing; High Performance Computing;**

I. Introduction

While high-performance computing typically aims to solve increasingly large linear algebra problems efficiently, there is a current trend towards splitting these into thousands of smaller linear algebra problems which are solved concurrently [1]. The solutions of these smaller problems are then combined to give the solution to the original large problem.

Examples of applications that can be decomposed in this way include solving separable elliptic equations using the algorithm by Swarztrauber [2], matrix-free finite element methods [3], domain decomposition [4] and image processing [5]. Batched BLAS is already utilized in popular machine learning libraries such as Theano and TensorFlow. Other examples of such applications where the solution of many small problems are required include metabolic networks [6], astrophysics [7], and computational fluid dynamics.

There are many software libraries that provide Basic Linear Algebra Subproblems (BLAS) such as Intel MKL and NVIDIA CuBLAS, but these libraries are optimized for solving large linear algebra problems. This means that applications like the above typically run with suboptimal performance.

The solution to this problem is to create a set of routines for computing linear algebra operations on batches of small matrices, building upon the BLAS. This Batched BLAS (BBLAS) should be able to compute many small matrix operations in parallel. For example, if we consider a GEMM operation over a batch of N matrices then we would like to compute, in parallel,

$$C_i \leftarrow \alpha_i A_i B_i + \beta_i C_i, \qquad i = 1 : N.$$

Currently there is no standard interface for batched BLAS operations and no complete implementation of all batched BLAS routines. Intel MKL has support for batched GEMM computation whilst NVIDIA CuBLAS supports batched GEMM and TRSM (along with a small subset of LAPACK). However, these two libraries do not follow the same API design. The introduction of the BLAS standard was critical to the sustainability, vendors and academics could both focus on obtaining optimal performance and users could easily switch between different implementations of the software. We hope that a standard API for BBLAS will be similarly transformative.

II. Towards a Sustainable Batched BLAS Library

Our medium term goals in this regard are as follows.

1) Propose a standard API for BBLAS routines.
2) Reach a consensus on the BBLAS API among vendors and academics.
3) Have high quality HPC software available for computing BBLAS routines.
4) Increase uptake of BBLAS routines in HPC applications.

Next we explain each point in detail. The lifecycle containing all these points is illustrated in Figure 1. Note that, whilst our description is focused on BBLAS, we believe that a similar process could be used to define any computing standard to be used by the wider community.

–1– We have already proposed an initial draft BBLAS API [8]. For our particular scenario, since BBLAS is closely related to BLAS, we attempt to follow the same conventions as well as the programming style used in the latter. This can ease the transition as the community becomes familiar with the new API and resulting libraries.

–2– When looking to obtain a standard API for BBLAS routines it is vitally important to involve both academics

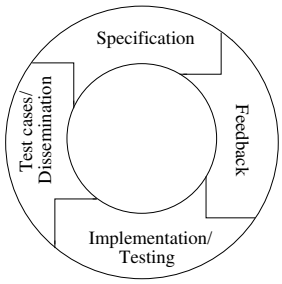

Fig. 1. Lifecycle of the BBLAS implementation process.

and vendors in the discussions. While academics may be the primary users of the software, vendors such as Intel and NVIDIA will likely provide optimized versions of the functions and have considerable experience in creating high-quality mathematical software.

Therefore, after proposing a draft specification for the BBLAS API, in May 2016 our team organised a workshop where world-leading academics from the linear algebra community and experts from the vendors could discuss changes to the draft. A number of ideas were proposed: from changing the parameters in the function calls to change the layout of the matrices in memory, with initial performance results given in each case.

Based upon this we have recently released a technical report [9] to review and compare these ideas, aiming to conclusively show which combination(s) will perform well in practical applications, paying particular attention to the memory layout and how data is stored in memory. We have realized that even the smallest changes to the API can have a substantial impact on performance, which reflects the importance that these preliminary studies have for the design of scientific standards.

This report has now been circulated to the vendors and the wider scientific community, so that we can receive feedback and any further suggestions they might have, which will then be fed into a second draft specification for the BBLAS API.

It is very important to spend the necessary time on this step as future changes to the specification will force us to reimplement the entire specification, which would require a significant amount of engineering effort and generate frustration for the users.

–3– Once the vendors and the wider community are in agreement on the BBLAS API, work can focus on obtaining highly optimized implementations for different architectures. In particular we hope to see extremely good performance on GPUs and the new Knights Landing Xeon Phi models within the next few years. Early efforts from Intel and NVIDIA in this for batched GEMM operations look extremely promising.

As HPC nodes become increasingly heterogeneous it is interesting to postulate a more general library that can use all the available computational resources concurrently. There are a number of questions to answer in this area, including how best to spread the workload among the available hardware. We have some initial ideas on this which we hope to implement in an open source library. This will allow interested members of the community to obtain the maximal performance possible on any HPC node. A well designed open source library, with good documentation and examples, will allow application experts to make use of our work easily, whilst allowing experts to add optimized code for additional platforms.

–4– Whilst batched GEMM is already used in some HPC applications, we hope to increase the uptake of batched BLAS in other areas. Whether or not this happens hinges upon the availability of high quality and high performance software for BBLAS routines. Allowing users to download our library via a public repository can be very beneficial for increasing the availability and dissemination of our library. Events such as conferences, workshops and journal publications provide another way to involve the community in our work.

ACKNOWLEDGEMENTS

This project is funded in part from the European Unions Horizon 2020 research and innovation programme under the NLAFET grant agreement No 671633.

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," *Journal of Physics: Conference Series*, vol. 180, no. 1, 2009.

[2] P. Valero-Lara, A. Pinelli, and M. Prieto-Matias, "Fast finite difference Poisson solvers on heterogeneous architectures," *Computer Physics Communications*, vol. 185, no. 4, pp. 1265–1272, 2014.

[3] K. Ljungkvist, "Matrix-free finite-element operator application on graphics processing units," in *Euro-Par 2014: Parallel Processing Workshops - Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II*, 2014, pp. 450–461.

[4] E. Agullo, L. Giraud, and M. Zounon, "On the resilience of parallel sparse hybrid solvers," in *22nd IEEE International Conference on High Performance Computing, HiPC 2015, Bengaluru, India, December 16-19, 2015*, 2015, pp. 75–84.

[5] P. Valero-Lara, "Multi-GPU acceleration of DARTEL (early detection of alzheimer)," in *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014*, 2014, pp. 346–354.

[6] A. Khodayari, A. R. Zomorrodi, J. C. Liao, and C. D. Maranas, "A kinetic model of Escherichia coli core metabolism satisfying multiple sets of mutant flux data," *Metabolic Engineering*, vol. 25, pp. 50–62, 2014.

[7] O. E. B. Messer, J. A. Harris, S. Parete-Koon, and M. A. Chertjkow, "Multicore and accelerator development for a leadership-class stellar astrophysics code," in *Proceedings of "PARA 2012: State-of-the-Art in Scientific and Parallel Computing."*, 2012.

[8] J. Dongarra, I. Duff, M. Gates, A. Haidar, S. Hammarling, N. J. Higham, J. Hogg, P. Valero-Lara, S. D. Relton, S. Tomov, and M. Zounon, "A proposed API for batched basic linear algebra subprograms," The University of Manchester, UK, MIMS EPrint 2016.25, April 2016.

[9] S. D. Relton, P. Valero-Lara, and M. Zounon, "A comparison of potential interfaces for batched blas computations," Manchester Institute for Mathematical Sciences, The University of Manchester, UK, MIMS Eprint 2016.42, 2016.