Benchmarking single- and multi-core BLAS implementations and GPUs for use with R

Dirk Eddelbuettel
Debian Project

Abstract

We provide timing results for common linear algebra subroutines across BLAS and GPU-based implementations. Several BLAS (Basic Linear Algebra Subprograms) implementations are compared. The first is the unoptimised reference BLAS which provides a baseline to measure against. Second is the Atlas tuned BLAS, configured for single-threaded mode. Third is the development version of Atlas, configured for multi-threaded mode. Fourth is the optimised and multi-threaded Goto BLAS. Fifth is the multi-threaded BLAS contained in the commercial Intel MKL package. We also measure the performance of the graphics processing unit (GPU) based implementation for R (R Development Core Team 2010a) provided by the package **gputools** (Buckner *et al.* 2010).

Several frequently-used linear algebra computations are compared across BLAS (and LAPACK) implementations and via GPU computing: matrix multiplication as well as QR, SVD and LU decompositions. The tests are performed from an end-user perspective, and 'net' times (including all necessary data transfers) are compared.

While results are by their very nature dependent on the hardware of the test platforms, a few general lessons can be drawn. Unsurprisingly, accelerated BLAS clearly outperform the reference implementation. Similarly, multi-threaded BLAS hold a clear advantage over single-threaded BLAS when used on modern multi-core machines. Between the multi-threaded BLAS implementations, Goto is seen to have a slight advantage over MKL and Atlas. GPU computing is showing promise but requires relatively large matrices to outperform multi-threaded BLAS.

We also provide the framework for computing these benchmark results as the corresponding R package **gcbd**. This enables other researchers to compute similar benchmark results which could be the basis for heuristics helping to select optimial computing strategies for a given platform, library, problem and size combination.

Keywords: BLAS, Atlas, Goto, MKL, GPU, R, Linux.

Detailed comments and suggestions from Roger Bivand, Allan Engelhardt, Bryan Lewis, James MacKinnon, Junji Nakano, Mark Seligman, Brian Smith and Luke Tierney are very gratefully acknowledged. All remaining errors are mine.

This version corresponds to **gcbd** version 0.2.1 and was compiled on September 13, 2010.

1. Introduction

Analysts are often eager to reap the maximum performance from their computing platforms. A popular suggestion in recent years has been to consider optimised basic linear algebra subprograms (BLAS). Optimised BLAS libraries have been included with some (commercial) analysis platforms for a decade (Moler 2000), and have also been available for (at least some) Linux distributions for an equally long time (Maguire 1999). Setting BLAS up can be daunting: the R language and environment devotes a detailed discussion to the topic in its *Installation and Administration* manual (R Development Core Team 2010b, appendix A.3.1).

Among the available BLAS implementations, several popular choices have emerged. Atlas (an acronym for *Automatically Tuned Linear Algebra System*) is popular as it has shown very good performance due to its automated and CPU-specific tuning (Whaley and Dongarra 1999; Whaley and Petitet 2005). It is also licensed in such a way that it permits redistribution leading to fairly wide availability of Atlas.[1] We deploy Atlas in both a single-threaded and a multi-threaded configuration. Another popular BLAS implementation is Goto BLAS which is named after its main developer, Kazushige Goto (Goto and Van De Geijn 2008). While 'free to use', its license does not permit redistribution putting the onus of configuration, compilation and installation on the end-user. Lastly, the Intel Math Kernel Library (MKL), a commercial product, also includes an optimised BLAS library.

A recent addition to the tool chain of high-performance computing are graphical processing units (GPUs). Originally designed for optimised single-precision arithmetic to accelerate computing as performed by graphics cards, these devices are increasingly used in numerical analysis. Earlier criticism of insufficient floating-point precisions or severe performance penalties for double-precision calculation are being addressed by the newest models. Dependence on particular vendors remains a concern with NVidia's CUDA toolkit (NVidia 2010) currently still the preferred development choice whereas the newer OpenCL standard (Khronos Group 2008) may become a more generic alternative that is independent of hardware vendors. Brodtkorb *et al.* (2010) provide an excellent recent survey.

But what has been lacking is a comparison of the effective performance of these alternatives. This paper works towards answering this question. By analysing performance across five different BLAS implementations—as well as two GPU-based solutions—we are able to provide a reasonably broad comparison.

Performance is measured as an end-user would experience it: we record computing times from launching commands in the interactive R environment (R Development Core Team 2010a) to their completion. While implemented in R, these benchmark results are more general and valid beyond the R system as there is only a very thin translation layer between the higher-level commands and the underlying implementations (such as, say, `dgemm` for double-precision matrix multiplications) in the respective libraries. This lack of (strong) dependence on the test environment makes our results more generally applicable. However, R is very useful as an environment to generate test data, execute the benchmarks, and to collect the results which are subsequently analysed and visualized.

The rest of the paper is organised as follows. In the next section, the technical background is briefly discussed. The implementation of our benchmark tests is outlined in section 3. We provide results in section 4, before a summary concludes in section 5.

[1] The Atlas FAQ lists Maple, Matlab and Mathematica as using Atlas, and mentions that GSL, Octave, R, Scientific Python, and Scilab can be used with Atlas.

2. Background

Basic Linear Algebra Subprograms (BLAS) provide an Application Programming Interface (API) for linear algebra. For a given task such as, say, a multiplication of two conformant matrices, an interface is described via a function declaration, in this case `sgemm` for single precision and `dgemm` for double precision. The actual implementation becomes interchangeable thanks to the API definition and can be supplied by different approaches or algorithms. This is one of the fundamental code design features we are using here to benchmark the difference in performance from different implementations.

A second key aspect is the difference between static and shared linking. In static linking, object code is taken from the underlying library and copied into the resulting executable. This has several key implications. First, the executable becomes larger due to the copy of the binary code. Second, it makes it marginally faster as the library code is present and no additional look-up and subsequent redirection has to be performed. The actual amount of this performance penalty is the subject of near-endless debate. We should also note that this usually amounts to only a small load-time penalty combined with a function pointer redirection—the actual computation effort is unchanged as the actual object code is identical. Third, it makes the program more robust as fewer external dependencies are required. However, this last point also has a downside: no changes in the underlying library will be reflected in the binary unless a new build is executed. Shared library builds, on the other hand, result in smaller binaries that may run marginally slower—but which can make use of different libraries without a rebuild.

That last feature is key here: it offers us the ability to use different BLAS implementations in a 'plug and play' mode which facilitates comparisons. So because of both the standardised interface of the BLAS, and the fact that we have several alternative implementations at our disposal, we can switch between these alternatives. To do so easily makes use of a package mechanism used by Ubuntu, the Debian-based Linux distribution we employ. However, a simpler (yet less robust) approach would also be available. This technical aspect is discussed further below.

The first available BLAS implementation stems from the original unoptimised code on Netlib. On Debian-based systems, it is provided by the (source) package **blas** (Blackford *et al.* 2002) and used with the **lapack** package (Anderson *et al.* 1999). This implementation is commonly referred to as 'reference BLAS' (and we will use 'ref' as a shorthand) as it provides a reference implementation.

The second and third BLAS implementations are provided by Atlas (Whaley and Dongarra 1999; Whaley and Petitet 2005) which stands for *Automatically Tuned Linear Algebra Software* as it optimises its performance and parameters (such as cache sizes) during its initial build. Another notable aspect is its liberal licensing which permits wide distribution. Consequently, Atlas is available on most if not all Linux distributions. Windows binaries are also widely available. We used two different Atlas versions. The first one is the (pre-compiled) version in the Ubuntu and Debian releases. It is based on Atlas 3.6.0 and built only for single-threaded operation. The second version is based on the current Atlas development version 3.9.25, built for multi-threaded mode and locally compiled.[2] We will refer to the second variant as 'Atlas39' and 'Atl39' when we report results below.

[2] A critical change enabling multi-threaded mode which we added was also filed as Debian bug report #595326 and will be reflected in future Debian / Ubuntu packages.

The fourth BLAS implementation is provided by the Goto BLAS. Upon the required user registration, these are freely available from the University of Texas, albeit under a license that prohibits redistribution. This prevents inclusion into popular Linux distributions, a clear disadvantage for easy installation and de-installation. However, the contributed Debian repository at the Institute of Statistical Mathematics in Japan provides a 'helper package' (Nakano and Nakama 2009) which, given the required registration information, downloads the source code from the University of Texas site, and then configures and compiles the Goto BLAS in such a manner that a local binary Debian package is produced—which is also optimised for the local installation. This permits us to use the Goto BLAS via the resulting package as a fourth BLAS alternative.

The fifth available BLAS implementation is part of the Intel Math Kernel Library (MKL), a commercial product. However, Ubuntu release 9.10 ("karmic") contains a set of packages sponsored by Revolution Analytics which comprises version 10.2 (dated March 2009) of the Intel MKL in a setup directly usable by R. We use these packages here as a fifth set of optimised BLAS.

The first R extension for graphics-processing units has been implemented by the **gputools** package (Buckner, Seligman, and Wilson 2010). It provides a number of functions which use the GPU instead of the CPU which results in a significant performance increase for 'large enough' problems. Because data has to be transferred from the CPU to the GPU, a fixed cost in communications has to be borne by every invocation of the GPU. For sizable problems, this cost can be outweighed by the benefits of the massively parallel GPU computation. Exactly where the indifference point lies beyond which GPU computing has an advantage is unfortunately dependent on the particular problem and algorithm as well as the given hardware and software combination.

A recent addition to the set of R packages is **magma** (Smith 2010) which interfaces the **Magma** library project of the same name (Tomov *et al.* 2009, 2010) (where we will capitalize the name of the library to distinguish it from the R package). The stated goal of the Magma project it to *develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.* The current release of Magma is version 0.2 which is available for Linux systems with NVidia's CUDA toolkit. It provides LU, QR, and Cholesky factorizations as well as linear solvers based on these along with implementations of several BLAS function including `dgemm`, `dgemv`, and `dsymv` (and well as their single-precision counterparts). Figure 1 illustrates the changes of relative workload between CPU and GPU when using Magma for a (single-precision) QR decomposition. It clearly depicts how the share of the total computing effort that goes to the GPU increases as a function of the matrix size.

A flexible and hybrid approach has the potential to improve upon solutions that use either the CPU or the GPU. However, the current version of the **Magma** library was exhibiting stability problems when used with GPU card employed (a Quadro FX48000). Analysing the performance of **Magma** relative to accelerated BLAS and other GPU-based approached is therefore left as topic for future research.

Finally, an important, and possibly under-appreciated, topic is how to balance explicitly parallel code that distributes load across cores with multi-threaded BLAS. Execution of Code may already be parallelised in a 'coarse-grained' fashion across all local cores or CPUs, maybe via tools that explicitly spawn multiple threads, maybe via compiler-driven directives as for

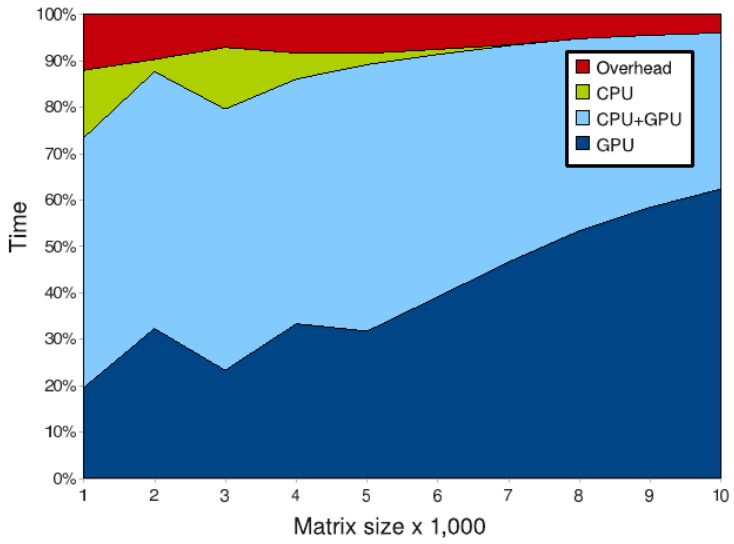

Figure 1: Breakdown of CPU and GPU computing percentages for single-precision QR decomposition using the hybrid Magma approach. (Source: Tomov *et al.* (2009, p.7))

example with Open MP, or maybe via cluster-computing tools across a local network. If such a setup 'booked' all available cores, a key assumption of multi-threaded BLAS no longer holds: other cores are not idle but as busy. In this case contention arises between the explicit parallelism and the implicit parallelism from the multi-threaded BLAS, and performance is bound to suffer. Bivand (2010) discusses this issues and provides several illustrations using examples from data-intensive GIS applications. The simplest remedy is to withdraw one of the mechanisms for multi-threaded use by limiting the number of cores to one. This can be done for Goto BLAS using the `GOTO_NUM_THREADS` environment variable. For the MKL one can use the environment variable `OMP_NUM_THREADS` (of the underlying Open MP implementation), or the R function `setMKLthreads()` of the corresponding package.

3. Implementation

3.1. Requirements

In order to undertake the automated benchmarking, we need to be able to switch between different implementations of the BLAS API. As discussed above, dynamic libraries are one possible solution that avoids having to rebuild R explicitly for each library. However, this also requires that R itself is built with shared-library support, as well as with support for external BLAS and LAPACK libraries. This requirement is however the default configuration on Debian and Ubuntu systems.

The reference BLAS as well as Atlas have been available for essentially all Debian and Ubuntu releases. Atlas 3.9.25 was packaged locally; the package and helper scripts are available upon request. The Intel MKL is available for Ubuntu following the Revolution R upload for Ubuntu release 9.10.

For Goto BLAS, we are using a helper script provided in the contributed `gotoblas2-helper` package (Nakano and Nakama 2009; Nakama 2010). This package arranges for a download of the Goto BLAS sources (provided the required account and password information for the University of Texas software download center) and an automated Debian package build and installation via the command `sudo /etc/init.d/gotoblas2-helper start`. Note that the initial invocation of this command will trigger a build of the package which may take up to two hours. While designed for Debian, it also works perfectly on the Ubuntu systems used here.[3]

For GPU-based testing we require the R packages **gputools** and **magma** which in turn require support for CUDA and the NVidia SDK (as well as appropriate NVidia hardware). Detailed installation instructions are provided by either package so we will defer to these and assume that the packages are in fact installed. Helper scripts in our package will then verify this availability while the benchmark is executed.

3.2. Benchmark Implementation

The benchmarks described in this paper are produced by the package **gcbd** which is part of the larger **gcb** project (Eddelbuettel, Buckner, and Seligman 2010) on the R-Forge hosting site (Theußl and Zeileis 2009). The **gcbd** package (where the acronym expands to 'GPU/CPU Benchmarking on Deb-based systems') contains a number of R helper functions as well as an actual benchmarking script which is executed.

The helper functions fall into two groups: utilities, and benchmarks. The utilities fall into several categories:

- initial feature detection via a function `requirements()` which asserts that a number of testable features of the host operating system are met;
- feature tests via the function `hasGputools()` allowing the benchmark script to bypass GPU-based tests in systems without a GPU;
- installation and removal functions which interface the package management layer and install (or remove) the Atlas, MKL or Goto BLAS packages, respectively, which helps ensure that at any one point in time only one accelerated BLAS library package is present;
- database creation where the results database (and table schema) is created if not already present;
- recording of simulation results in the database.

The benchmark functions can also be categorized:

- creation of random data for standard **Matrix** or **magma** objects, respectively;
- actual benchmarking code for
  - matrix crossproducts;

[3] The `/etc/init.d/gotoblas2-helper` script required one change from `/bin/sh` to `/bin/bash`.

- SV decomposition;
- QR decomposition;
- LU decomposition;

for BLAS, **gputools** and **magma**, respectively.

For the QR decomposition, we set the flag `LAPACK=TRUE`. For R, the default QR operation is provided by LINPACK which uses level 1 BLAS operations where LAPACK can reap a larger benefit from accelerated or optimised BLAS libraries. For the LU decompositions, we use the function from the **Matrix** package by Bates and Maechler (2010).

3.3. Benchmark script

The benchmark execution can then be triggered by the script `benchmark.r` in the sub-directory `scripts` of the **gcbd** package. It is implemented as an executable R script which uses the **getopt** package (Day 2010) for command-line parsing:

```
$ ./benchmark.r -h
Usage: benchmark.r [-[-verbose|v]] [-[-help|h]] [-[-nobs|n] <integer>]
                   [-[-runs|r] <integer>] [-[-benchmark|b] <character>]
    -v|--verbose      verbose operations, default is false
    -h|--help         help on options
    -n|--nobs         number of rows and columns in matrix, default is 250
    -r|--runs         number of benchmark runs, default is 30
    -b|--benchmark    benchmark to run (matmult, qr, svd, lu), default is matmult
```

Each benchmark experiment consists of r runs for a matrix of size $n \times n$ observations using the chosen benchmark—matrix cross product or one of the QR, LU or SVD decompositions—over all five BLAS implementations and two GPU packages. The run against the GPU packages is optional and dependent on the GPU packages being present.

At the end of each benchmark experiment, the results are appended to a SQLite database.

We use the 'elapsed time' as measured by the R function `system.time()`. This measure is preferable over the sum of system time and user time which adds up total cputime—but without adjusting for multiple threads or cores. Elapsed time correctly measures from start to finish, whether one or multiple threads or cores are involved (but is susceptible to be influenced by system load).

Using this script `benchmark.r`, users can collect benchmark results on their systems. These could be used to aggregate more performance data which could then be used to estimate realistic performance numbers for a much wider variety of hardware configurations. Doing so is beyond the scope of this paper but a possible avenue for future work.

3.4. Alternative implementation

On platforms that do not have access to pre-built BLAS library packages, an alternative approach could consist of locally installing the different libraries into sub-directories of, say, `/opt/blas`. One could then use the environment variable `LD_LIBRARY_PATH` to select one of

these directories at a time. However, such an approach places a larger burden on the user of the benchmarking software as she would have to download, configure, compile and install the BLAS libraries which is typically not a trivial step.

3.5. Hardware and software considerations

For benchmarking linear algebra performance, hardware and software aspects matter greatly for the overall results. Different CPUs, different GPUs, different memory type as well as different compilers (or operating systems) may generate different performance profiles.

We have been running the results presented here on these two platforms:

- a four-core Intel i7 920, 2.67 GHz clock speed, 8192 kb CPU cache, 6 gb RAM, in hyper-threaded mode for eight visible cores, running Ubuntu 10.4 in 64-bit mode;
- a dual-CPU four-core Intel Xeon 5570, 2.96 Ghz clock speed, 8192 kb CPU cache, 16gb RAM, in hyper-threaded mode for sixteen visible cores, running Ubuntu 10.4 in 64-bit mode.

The i7 system also has a NVidia Quadro FX4800 GPU with 192 cores and 1.5 gb of RAM.

Different hardware platforms could be reflected by other users installing the **gcbd** package on their system and reporting results back by supplying the resulting SQLite database files.

The software configuration was held constant by running on 64-bit Ubuntu 10.4 in both cases.

It should be noted that hyper-threading is a potential distraction. While it was not possible to reconfigure the machines used here, it could be informative to compare results on identical hardware with hyper-threading turned on and off, respectively.

Lastly, comparing across operating system would also be of interest. While the test platform used here makes extensive use of the packaging system available on Debian / Ubuntu, it would of course be possible to set up something similar on, say, OS X to measure the performance of the vecLib system.

4. Results

4.1. BLAS Comparison

We present the benchmark results (which are also included in the **gcbd** package in the file `sql/gcbd.sqlite`). We will show one type of benchmark per test architecture (currently one of i7 or xeon as detailed in section 3.5) with times first shown on a regular scale (with the matrix dimension on the horizontal axis and the elapsed time in seconds on the vertical axis), and also in a log/log plot with both matrix dimension and elapsed time on a logarithmic scale in order to better differentiate between alternatives.[4]

[4] I am particularly grateful to Allan Engelhardt for suggesting to switch from a plot with logged y-axis to a log-log plot.

Matrix Multiplication: i7 with GPU

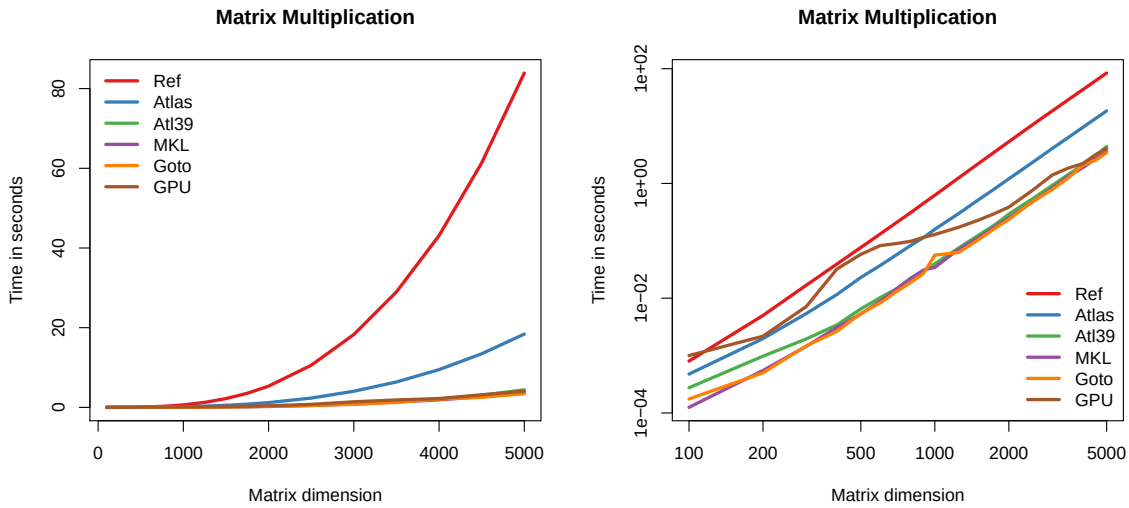

Figure 2: Matrix multiplications on i7 with GPU

It is immediately apparent that the reference BLAS underperform very significantly. Similarly, the single-threaded Atlas performance is also dominated by the multi-threaded BLAS (Atlas39, Goto, MKL) and the GPU-based gputools.

We also see (especially in the log/log plot) that the GPU-based solution bears a clear penalty for smaller dimensions. It only crosses below the single-threaded Atlas around $N = 1000$ and approaches the best performing multi-threaded BLAS at the very end for $N = 5000$. On the other hand, it clearly exhibits a much slower slope implying a lesser time penalty as matrix sizes increases.

The three multi-threaded BLAS implementations are essentially indistinguishable for this test and hardware platform.

Matrix Multiplication: Xeon

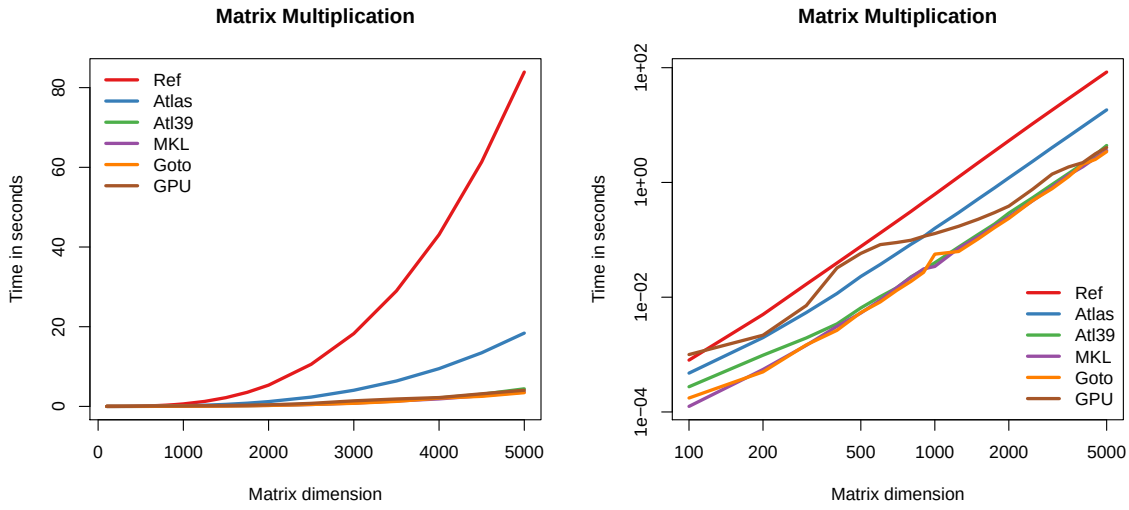

Figure 3: Matrix multiplications on xeon

As above, the reference BLAS underperform significantly, and single-threaded Atlas is dominated by all multi-threaded BLAS implementation. On the other hand, on this platform the log/log plot offers a better differentiation between the multi-threaded BLAS implementations. Goto is clearly ahead of its competitors. MKL shows a lower slope and higher times for small sizes hinting at some suboptimal configuration. Atlas39 beats MKL for essentially all sizes but cannot catch Goto BLAS' performances.

QR Decomposition: i7 with GPU

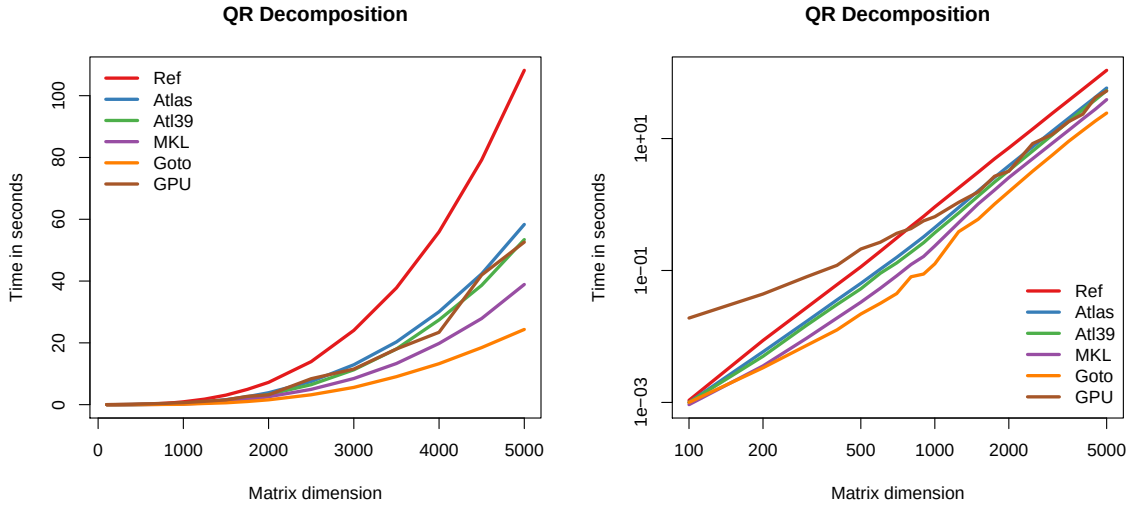

Figure 4: QR decomposition on i7 with GPU

As before, reference BLAS are behind all other implementations. The single-threaded Atlas is on-par with the multi-threaded Atlas39—and the GPU solution. Goto and MKL beat all others, with Goto dominating overall. The log/log plot once again illustrates the higher 'startup cost' of the GPU solution due to the communications cost being high (in relatively terms) for small matrices where the computations are still very fast.

QR Decomposition: Xeon

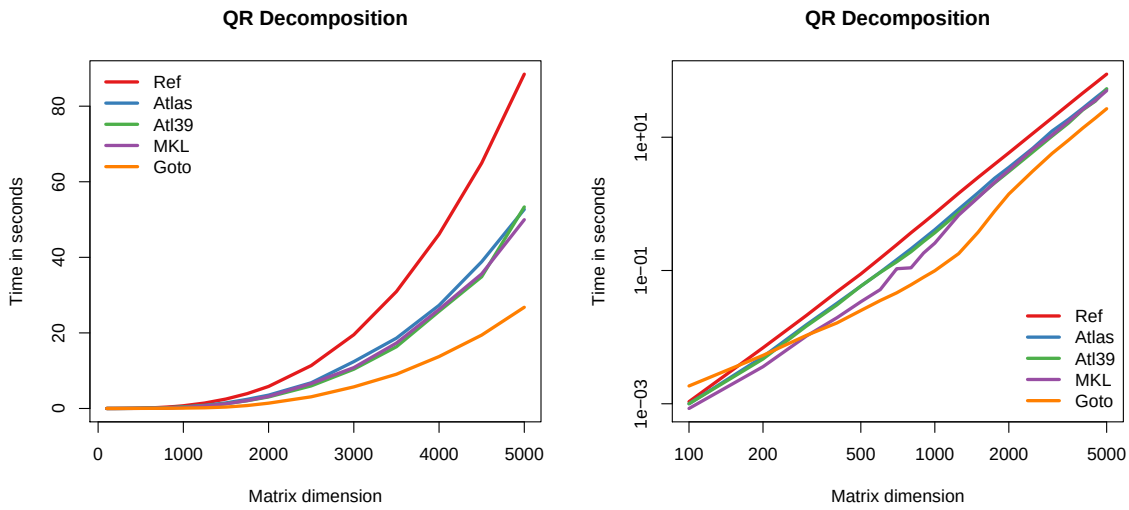

Figure 5: QR decomposition on xeon

Results fall basically into three groups. The Reference BLAS are behind everybody. A second group of both Atlas variants and MKL forms the middle with MKL having a slight edge over the Atlas libraries. However, Goto is ahead of everybody (yet shows a somewhat surprising non-linearity in the log/log plot).

SVD Decomposition: i7 with GPU

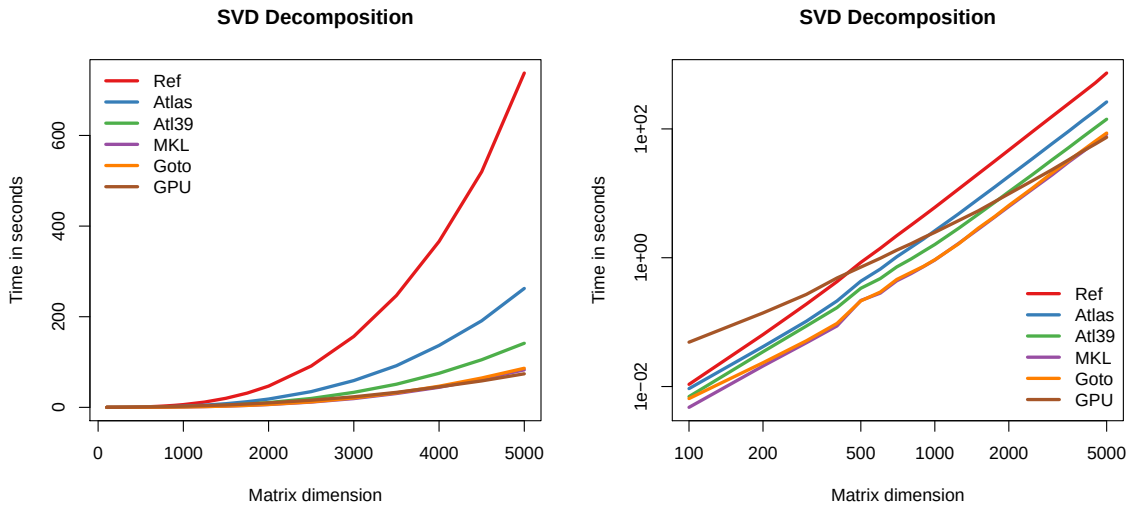

Figure 6: SVD on i7 with GPU

For the SVD, we have the familiar ordering of Reference BLAS behind single-threaded Atlas which is behind multi-threaded Atlas. MKL, Goto and GPU are all competitive, with the GPU lagging for small sizes but beating all competitors for the largest matrix size tests. In the log/log chart, the GPU performance also demonstrates a much slower slope, yet along with the much higher intercept.

SVD: Xeon

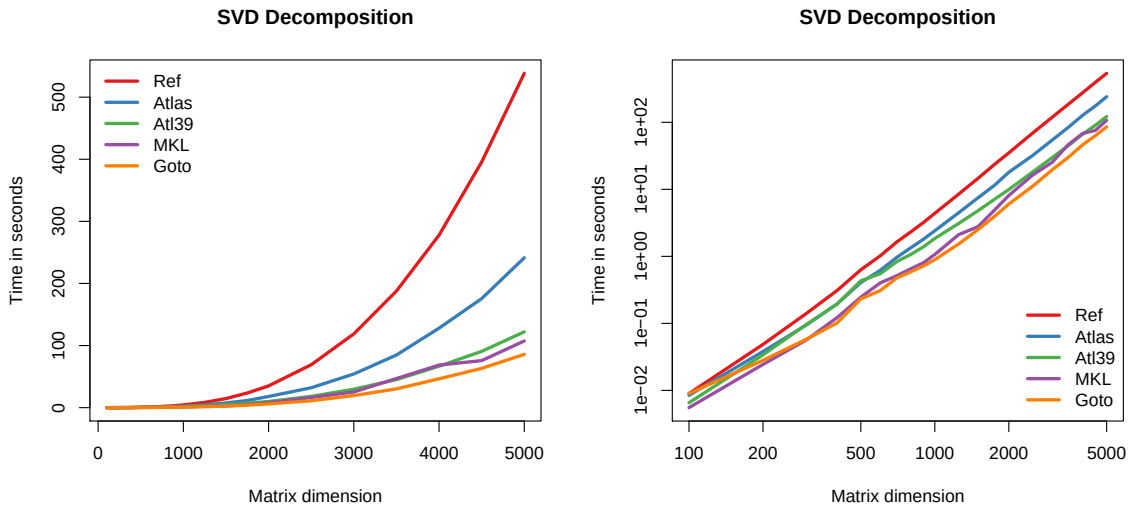

Figure 7: SVD on xeon

On the Xeon, results are similar to the i7 with the main difference that the multi-threaded Atlas39 is closer to its competitors, in particular the MKL. Goto BLAS are once again ahead.

LU Decomposition: i7 with GPU

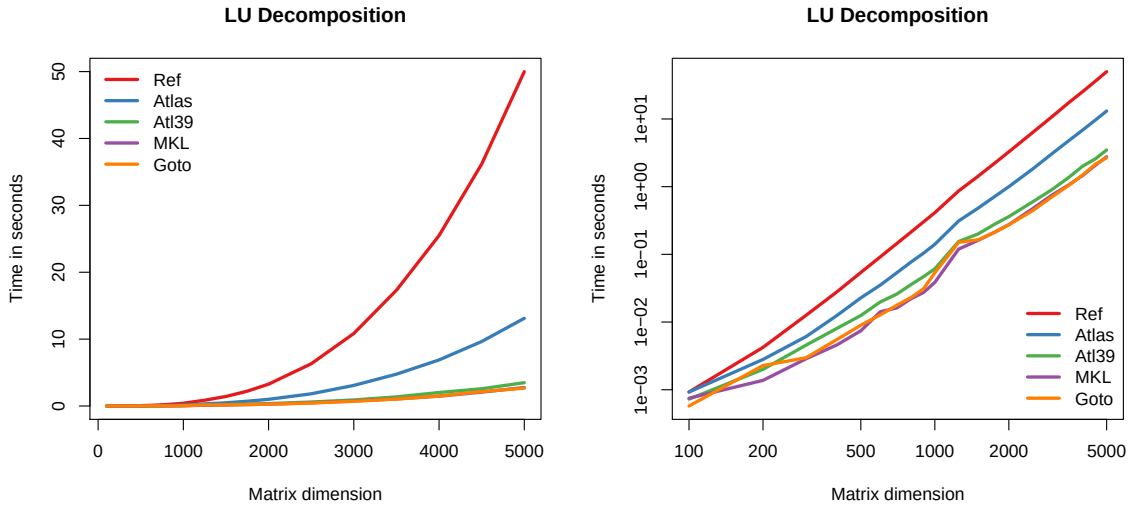

Figure 8: LU decomposition on i7 with GPU

LU decomposition results, using the function from the **Matrix** package (Bates and Maechler 2010), are similar to earlier results. Reference BLAS are by far the slowest, single-threaded Atlas beats them clearly and loses equally clearly to the multi-threaded BLAS. Multi-threaded Atlas is very close to Goto and MKL, which are ever so slightly ahead and essentially indistinguishable. For this algorithm, no GPU-based performance numbers are available as the current version of the **gputools** package does not provide a LU decomposition.

LU Decomposition: Xeon

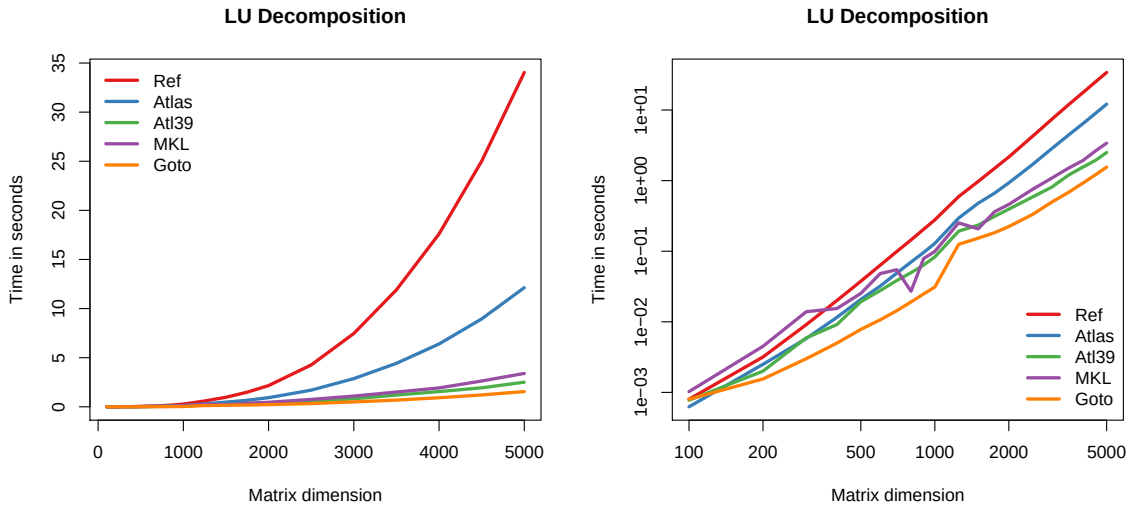

Figure 9: LU decomposition on xeon

On the xeon chip we see once again a clearer separation between the three accelerated BLAS implementations on the one hand and the single-threaded Atlas and the Reference BLAS on the other hand. Goto has a clear edge over MKL, which is also overtaken by Atlas39 for medium-to-large size matrices. MKL also exhibits some irregularity for small-to-medium sized matrices.

Comparison

The charts shown above suggest a further analysis: a regression of the logarithm of the elapsed times on the logarithm of the matrix dimension. These are shown in Table 1 below. For these results, a higher slope coefficient implies a higher increase in elapsed time per increase in matrix dimension, and moreover in a non-linear fashion as we have taken logarithms.

Platform	BLAS	Mat.Mult		QR		SVD		LU	
		α	β	α	β	α	β	α	β
i7	Ref	-21.19	3.00	-20.45	2.95	-18.03	2.88	-20.66	2.87
	Atlas	-21.06	2.80	-20.43	2.86	-17.50	2.69	-19.65	2.59
	Atlas39	-20.89	2.59	-20.57	2.86	-17.04	2.56	-18.30	2.27
	Goto	-21.65	2.67	-20.83	2.79	-17.19	2.51	-18.55	2.27
	MKL	-21.84	2.70	-21.03	2.88	-17.49	2.55	-18.84	2.30
	GPU	-16.42	2.08	-14.64	2.12	-12.15	1.91		
xeon	Ref	-21.31	3.01	-20.58	2.94	-18.28	2.87	-20.75	2.83
	Atlas	-21.06	2.79	-20.49	2.86	-17.60	2.69	-20.02	2.62
	Atlas39	-18.54	2.23	-20.41	2.83	-16.58	2.50	-17.30	2.15
	Goto	-21.39	2.55	-19.82	2.64	-16.62	2.43	-17.80	2.13
	MKL	-18.66	2.33	-21.56	2.97	-17.56	2.59	-16.37	2.05

Table 1: Comparison of slopes and intercepts in log/log analysis of BLAS performance

The results in Table 1 as well as the visualization of different slope estimates in Figure 10, formalise the rank-ordering of BLAS implementation seen in the analysis of the individual charts above.

- Unsurprisingly, Reference BLAS are clearly dominated by all other alternatives and exhibits the increase in elapsed time per increase in matrix dimension.

- Single-threaded Atlas is clearly improving on the Reference BLAS but is dominated by the other multi-threaded libraries and the GPU-based solution; the QR decompostion is an exception where both Atlas variants and the MKL are comparable and just behind Goto.

- Multi-threaded Atlas39 is roughly comparable to the MKL (and ahead for matrix multiplication) and just behind Goto BLAS.

- The MKL Blas perform well, generally on-par with or ahead of Atlas39 but trailing Goto BLAS for two of the four algorithms.

- Goto BLAS are ahead for the QR and SVD tests, and on par or behind for matrix multiplication and LU decompositions.

- GPU computing is seen to have the lowest slope corresponding to the lowest cost per additional matrix dimension increases—but this has to be balanced with the cost for smaller to medium sizes which can be seen from the corresponding analysis for intercepts (not shown but available in the package).

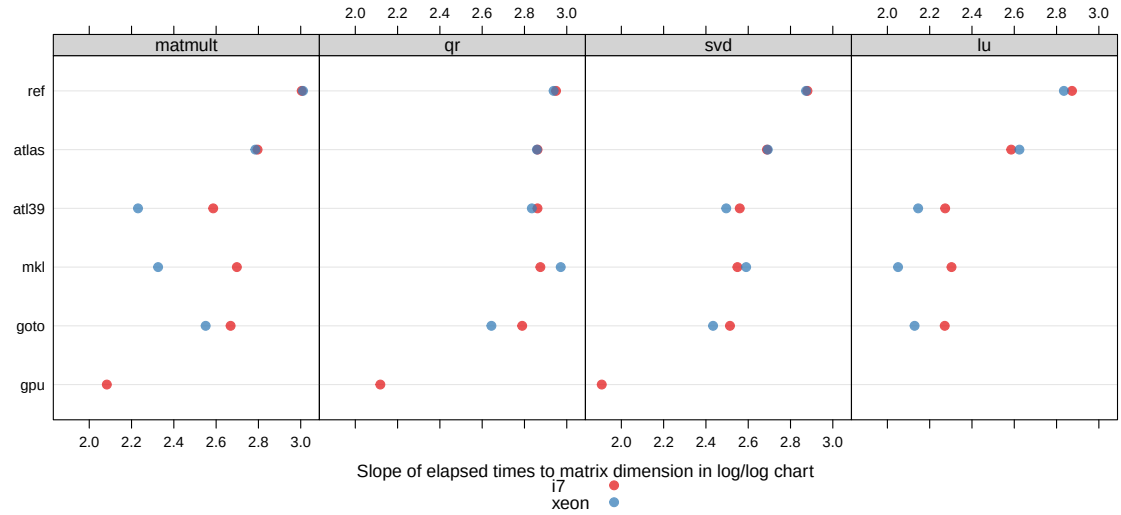

Figure 10: Comparison of slopes in log/log analysis of BLAS performance

5. Summary

We present benchmarking results comparing five different BLAS implementations for four different matrix computations on two different hardware platforms. We find reference BLAS to be dominated in all cases. Single-threaded Atlas BLAS improves on the reference BLAS but loses to multi-threaded BLAS.

For multi-threaded BLAS we find the Goto BLAS dominate the Intel MKL, with a single exception of the QR decomposition on the xeon-based system which may reveal an error. The development version of Atlas, when compiled in multi-threaded mode is competitive with both Goto BLAS and the MKL. GPU computing, when used in isolation, is found to be compelling only for very large matrix sizes.

Our benchmarking framework in the **gcbd** package can be employed by others through the R packaging system which could lead to a wider set of benchmark results. These results could be helpful for next-generation systems which may need to make heuristic choices about when to compute on the CPU and when to compute on the GPU. A strong empirical basis may make these heuristics more robust.

Computational details

The results in this paper were obtained using R 2.11.1 with the packages **gputools** 0.21, **magma** 0.2.2–1, **RSQLite** 0.9–2, **DBI** 0.2–5 and **getopt** 1.15. R itself and all packages used are available from CRAN at `http://CRAN.R-project.org/`.

The following Ubuntu packages were used to provide the different BLAS implementations: **libblas** 1.2-build1, **liblapack3gf** 3.2.1-2, **libatlas3gf-base** 3.6.0-24ubuntu, **revolution-mkl** 3.0.0-1ubuntu1, (containing MKL 10.2 dated March 2009) and **gotoblas2** 1.13-1 which was built using **gotoblas2-helper** 0.1-12 by Nakama (2010). Apart from **gotoblas2-helper** and **gotoblas2**,

all these package are available via every Ubuntu mirror.

Disclaimers

NVidia provided the Quadro FX 4800 GPU card for evaluation / research purposes. REvolution Computing (now Revolution Analytics) employed the author as a paid consultant for the creation of the **revolution-mkl** package and integration of REvolution R into Ubuntu 9.10.

References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999). *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition. ISBN 0-89871-447-8 (paperback).

Bates D, Maechler M (2010). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 0.999375-43, URL http://cran.r-project.org/package=Matrix.

Bivand R (2010). "More from less: Using R to analyse spatial data." Presentation at *Open Source Tools & Standards in GISc Research and Education*, 2010 AAG Annual Meeting, April 14-18, Washington, DC.

Blackford LS, Demmel J, Dongarra J, Duff I, Hammarling S, Henry G, Heroux M, Kaufman L, Lumsdaine A, Petitet A, Pozo R, Remington K, Whaley RC (2002). "An Updated Set of Basic Linear Algebra Subprograms (BLAS)." *ACM Transactions on Mathematical Software*, **28**(2), 135–151.

Brodtkorb AR, Dyken C, Hagen TR, Hjelmervik JM, Storaasli OO (2010). "State-of-the-art in heterogenous computing." *Scientific Programming*, **18**, 1–33. URL http://babrodtk.at.ifi.uio.no/files/publications/brodtkorb_etal_star_heterocomp_final.pdf.

Buckner J, Seligman M, Wilson J (2010). *gputools: A few GPU enabled functions*. R package version 0.21, URL http://cran.r-project.org/package=gputools.

Day A (2010). *getopt: C-like getopt behavior*. R package version 1.15, URL http://cran.r-project.org/package=getopt.

Eddelbuettel D, Buckner J, Seligman M (2010). *gcb: GPU/CPU Benchmarks*. R-Forge project repository, URL https://r-forge.r-project.org/R/?group_id=789.

Goto K, Van De Geijn R (2008). "High-performance implementation of the level-3 BLAS." *ACM Trans. Math. Softw.*, **35**(1), 1–14. ISSN 0098-3500.

Khronos Group (2008). "OpenCL: The Open Standard for Heterogeneous Parallel Programming." Overview presentation. URL http://www.khronos.org/developers/library/overview/opencl_overview.pdf.

Maguire C (1999). "Atlas 2.0-1." Email confirming initial upload to Debian repositories. URL http://lists.debian.org/debian-devel-changes/1999/10/msg03984.html.

Moler C (2000). "MATLAB Incorporates LAPACK: Increasing the speed and capabilities of matrix computation." MATLAB News & Notes. URL http://www.mathworks.com/company/newsletters/news_notes/clevescorner/winter2000.cleve.html.

Nakama E (2010). *gotoblas2-helper: Automatic GotoBLAS2 updater*. Debian package version 0.1-12, URL http://prs.ism.ac.jp/~nakama/debian/lenny-ism.

Nakano J, Nakama E (2009). "Parallel computing environment for R on personal cluster systems." Poster presented at SC09, Portland, OR. URL http://prs.ism.ac.jp/~nakama/debian/SC09-poster.pdf.

NVidia (2010). *NVidia Compute Unified Device Architecture (CUDA) Toolkit, Version 3.1*. NVidia. URL http://developer.nvidia.com/object/cuda_3_1_downloads.html.

R Development Core Team (2010a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

R Development Core Team (2010b). *R: Installation and Administration*. R Foundation for Statistical Computing, Vienna, Austria. ISBN ISBN 3-900051-09-7, URL http://www.R-project.org.

Smith BJ (2010). *magma: Matrix Algebra on GPU and Multicore Architectures*. R package version 0.2.1-2, URL http://cran.r-project.org/package=magma.

Theußl S, Zeileis A (2009). "Collaborative Software Development Using R-Forge." *The R Journal*, **1**(1), 9–14. URL http://journal.r-project.org/2009-1/RJournal_2009-1_Theussl+Zeileis.pdf.

Tomov S, Nath R, Du P, Dongarra J (2009). *MAGMA User's Guide, Version 0.2*. University of Tennesse, Knoxville, TN. URL http://icl.cs.utk.edu/magma.

Tomov S, Nath R, Ltaief H, Dongarra J (2010). "Dense Linear Algebra Solvers for Multicore with GPU Accelerators." In "IPDPS 2010: 24th IEEE International Parallel and Distributed Processing Symposium," Atlanta, GA. URL http://www.netlib.org/netlib/utk/people/JackDongarra/PAPERS/lawn225.pdf.

Whaley RC, Dongarra J (1999). "Automatically Tuned Linear Algebra Software." In "Ninth SIAM Conference on Parallel Processing for Scientific Computing," CD-ROM Proceedings.

Whaley RC, Petitet A (2005). "Minimizing development and maintenance costs in supporting persistently optimized BLAS." *Software: Practice and Experience*, **35**(2), 101–121. URL http://www.cs.utsa.edu/~whaley/papers/spercw04.ps.

Affiliation:

Dirk Eddelbuettel
Debian Project
River Forest, IL, USA
E-mail: edd@debian.org
URL: http://dirk.eddelbuettel.com