Towards Energy-efficient Linear Algebra with an ATLAS Library Tuned for Energy Consumption

Jens Lang, Gudula Rünger, and Paul Stöcker
Department of Computer Science
Technische Universität Chemnitz
09107 Chemnitz, Germany
jens.lang@cs.tu-chemnitz.de, ruenger@cs.tu-chemnitz.de, stop@hrz.tu-chemnitz.de

***Abstract*—Autotuning is an established method for adapting the execution of an application to the underlying hardware for minimising the execution time. This article investigates whether autotuning is also suitable for minimising the energy consumption of an application. The investigation is done with the linear algebra library ATLAS. Adaptations for the ATLAS package which enable energy autotuning are proposed. Different tuning parameters are investigated for whether they show a different behaviour when ATLAS is tuned for energy consumption instead for execution time. The results suggest that some tuning parameters have to be set differently when ATLAS is supposed to work with a minimum energy consumption than with a minimum execution time. The results further indicate that tuning the complete ATLAS package for energy consumption leads to a more energy-efficient execution than tuning it for execution time.**

***Keywords*—energy autotuning, energy efficiency, power, basic linear algebra subroutines (BLAS), ATLAS**

I. Introduction

Linear algebra is an important element of many scientific simulations. Over the years, BLAS (Basic Linear Algebra Subroutines) has established as a standard interface for linear algebra operations [1]. For keeping pace with the constantly improving hardware, automated tuning, or autotuning, is necessary for packages implementing the BLAS interface as well as for other software [2]. Until today, autotuning has mainly focused on minimising the execution time of an application [3]. Currently, no autotuning package for linear algebra considers energy as an optimisation goal. However, for future hardware architectures, especially with exascale computing, the energy consumption for executing scientific codes will become an important cost factor.

One of the most popular autotuned BLAS packages is ATLAS (Automatically Tuned Linear Algebra Software) [2]. ATLAS investigates a large number of code variants for different operations and selects those showing the best performance, partly even considering data characteristics. ATLAS finds the optimal values for several tuning parameters, such as block sizes, pre-fetch distances, or crossover points of different implementations. The ATLAS library, however, currently only optimises for execution time and does not consider the energy consumption. Although it has often been found that optimising the execution time also optimises the energy consumption [4], [5], this is not always true [5], [6]. This article presents some approaches to introduce energy as an optimisation goal into the ATLAS autotuner. The aim is to create an autotuned BLAS library which uses as little energy as possible, for example by favouring low-energy implementation variants over fast variants in cases where there is a conflict between the goals of optimising execution time or energy.

This article makes the following contributions. It proposes methods for incorporating energy as an optimisation goal in ATLAS. It discusses a number of ATLAS parameters and their influence on execution time and energy consumption, and it presents an ATLAS variant completely tuned for energy consumption.

The rest of this article is structured as follows: Sec. II gives a summary on the autotuning mechanism of ATLAS and proposes methods for introducing energy awareness into the autotuner. Sec. III investigates several tuning parameters and their behaviour when tuned for execution time and for energy consumption. Sec. IV investigates the effect of tuning the complete package for energy consumption. Sec. V presents related work, and Sec. VI concludes the article.

II. Autotuning in ATLAS

Autotuning is an 'automated process, guided by experiments, of selecting one from among a set of candidate program implementations to achieve some performance goal' [7]. The approach of autotuning usually consists of three steps [7]:

1) Creation of a number of candidate implementations for the operation to be tuned,
2) Experimental evaluation of the performance of the variants,
3) Selection of the best variant for the practical execution of the code.

As the performance of a code normally strongly depends on the hardware on which it is executed, the autotuning procedure is, in most cases, performed during the installation of the software on a new hardware.

A. The ATLAS Autotuning Process

A good review of the autotuning process of ATLAS is given in [2], [8]. Fig. 1 visualises the steps of the ATLAS

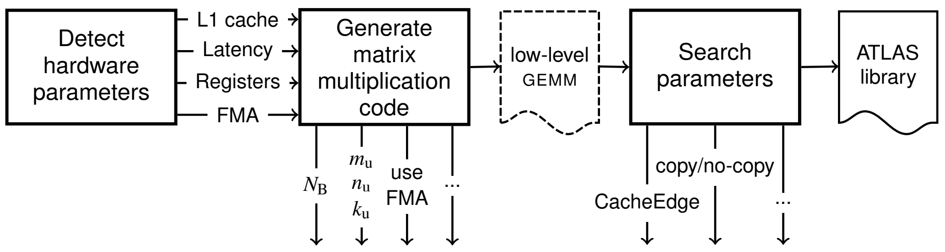

Figure 1. Autotuning process in the ATLAS library

autotuning process, the data transferred between the steps, and the parameters which can be extracted from each step (arrows pointing down). The first step of ATLAS is the detection of certain hardware parameters, including the size of the level-1 cache, the latency of a multiply operation, the number of floating-point registers, and the presence of the fused multiply-add (FMA) hardware instruction. In order to save tuning time, ATLAS provides a library of hardware parameters for the most common architectures. If the library values are used, the hardware detection step is omitted. Hardware detection can, however, be enforced (by giving the parameter `-Si archdef 0` to the configure script).

ATLAS uses two different methods for creating candidate implementations. The first method is the *source code adaptation* (second box in Fig. 1), which is used when different source codes are needed for implementing the candidates. When using source code adaptation, different algorithms for one operation may be evaluated or, if the code is generated automatically, different levels of loop unrolling or different loop nestings can be investigated. The incorporated search engine determines which code variants and which parameter settings exhibit the best performance. The parameters to be tuned include the block size N_B for the blocked matrix multiplication, the loop unrolling factors n_u, m_u and k_u, and a parameter indicating whether fused multiply-add is used. The search engine hands over the parameter set to be investigated to the code generator, which generates the source code considering these parameters and compiles it. During the execution, the performance of the generated routine is measured and returned to the search engine which then, based on the result, refines the parameter set. ATLAS offers various timers for the evaluation of the performance, including wall clock timers and cycle-accurate CPU timers.

The second method for creating candidate implementations is the *parametrised adaptation* (third box in Fig. 1), which means that a piece of code contains a parameter which controls its execution. These parameters are tuned after the source code generation. The parameters include the copy/no-copy threshold, which controls whether the matrices are copied and rearranged in memory before the operation, and the CacheEdge, which optimises the cache usage of higher-level caches.

This article will mainly (but not solely) focus on operations of BLAS level 3, i.e. matrix-matrix operations. For level-3 kernels, ATLAS uses a two-stage optimisation [2], which works as follows: The *high-level stage* splits up the operation into smaller, block-wise matrix-matrix multiply operations. These matrix-matrix multiplications are processed in the *low-level stage*. For the low-level stage, there exists a highly optimised kernel which performs a matrix multiplication of the form $C \leftarrow A^\top B + \beta C$. All matrices are square matrices with their dimensions (N, M and K) being N_B. The value for N_B is chosen in a way such that the operation becomes *cache-contained*, i.e. the data needed fits into the level-1 cache. Furthermore, the article will only deal with tuning parameters that are already present in ATLAS. New parameters, such as the operating frequency of the CPU, are not introduced. Works investigating dynamic voltage and frequency scaling for linear-algebra routines are, e.g., [9], [5].

B. *Energy Awareness for the ATLAS Autotuning*

In order to create an energy-optimised ATLAS library, we use the energy consumption as the performance measure for the candidate implementations. The existing performance measurement, which measures time, can be replaced or complemented by the measurement of the energy consumption. Both strategies have been followed as proposed in [10].

The energy consumption is measured using the Running Average Power Limiting (RAPL) [11] software interface provided by recent Intel CPUs. The RAPL interface has already been used for studies on the energy consumption of linear algebra routines, e.g. [12], [13], [14]. In various works, it has been shown that the values returned by RAPL correspond sufficiently to values obtained by hardware energy meters, e.g. [15], [11], [16]. Only hyperthreading is not captured well by RAPL [16], which has not been used here. For this work, the PAPI [17] package is used for accessing the RAPL interface. The PAPI event PACKAGE_ENERGY:PACKAGE0, which maps to the RAPL register MSR_PKG_ENERGY_STATUS is used. This register measures the energy consumption of the complete CPU package, including core, cache and uncore energy consumption. The measurement value is updated every 1 ms and the register holding it overflows after roughly 60 s if the machine is heavily loaded [18, vol. 3B, ch. 34.7.2].

The performance of codes considering the execution time is called the *time efficiency* in this article. It is measured in ‘flop/s’, i.e. the number of floating point instructions which are executed in one second. The corresponding performance measure for energy consumption is the *energy efficiency*, measured in 'flop/J'. The flop/J value measures how many floating point instructions can be executed using the energy of one joule.

1) Measuring Energy and Execution Time: For some parameters, the ATLAS autotuner obtains a table of different parameter settings and the corresponding performance values. From the table, the autotuner selects the parameter setting with the highest performance, i.e. the highest flop/s value. In these cases, we perform the energy measurement in addition to the time measurement. Both values, flop/s and the flop/J, are then included in the table.

In the source code, the energy measurement is started before the time measurement and stopped after it. The ATLAS version 3.10.2 has been used for these experiments.

2) Measuring Energy Instead of Execution Time: In some cases, e.g. for the evaluation of the generated matrix multiplication routines, the result of the performance evaluation is the input for the next iteration of the optimisation in ATLAS. In this case, measuring time and energy at once is not useful as ATLAS only optimises for one criterion. Thus, we replace the time measurement by an energy measurement. This replacement effects the whole ATLAS package so that ATLAS is only optimised for energy, not for execution time. In detail, we modify the routine `ATL_cputime` in the file `tune/sysinfo/ATL_cputime.c` such that it returns the energy consumption. For the measurement, the PAPI library is used as described above. It has to be ensured that no overflow occurs in the measurement register. However, we found that all measurements need less than one minute to complete so that the overflow will not occur. The ATLAS version 3.10.1 has been used for these experiments.

In this modified version of ATLAS, the hardware parameters *number of registers*, *latency* and *size of level-1 cache* have been set fix to the values obtained by the original ATLAS version. This ensures that all optimisations start with the same conditions and effects arising from differing initial conditions are avoided. As the energy measurement shows a greater variance than the time measurement, the number of the measurements per kernel has been increased from 3 to 5. Furthermore, the number of repetitions per measurement, which is between 40 and 18,500, has been increased by a factor of 100.

III. Evaluation of Single Parameters

A selection of the large number of parameters tuned during the autotuning process of ATLAS is investigated in this section. During the tuning of each parameter, the performance of ATLAS with the different parameter settings is evaluated. This section presents measurements and results for both kinds of tuning, for time and for energy, and attempts to show their relations for each parameter. While this section evaluates the time efficiency versus the energy efficiency of single parameters such as the block size and loop unrolling factors, Sect. IV investigates the differences that arise when the complete package is tuned for energy consumption, i.e. without regarding single parameters.

For the experiments, a machine with a quad-core Intel Core i7-4770K processor has been used. The processor has a clock frequency of 3.5 GHz and cache sizes of: 32 KiB level-1 data cache, 32 KiB level-1 instruction cache, 256 KiB level-2 cache (all per core), and 8 MiB level-3 cache (shared). The architecture offers 256-bit fused multiply-add units [19].

As the detection of the hardware parameters shows differences when optimising for time and when optimising for energy, the following hardware parameters have been set to the values obtained by the time-optimised version: the cache size is 128 KiB, the multiply latency is 5, fused multiply-add is available.

A. Block Size for Low-level Matrix Multiplication

For the low-level matrix multiplication, from which the high-level operations are composed, the optimal size N_B of the block matrices is determined. The low-level matrix multiplication performs operations of the form $C \leftarrow A^\top B + \beta C$ with all dimensions N, M and K being N_B. The operation is cache-contained, which means that N_B is chosen such that the entire matrix A and 2 columns of B fit into the level-1 cache. Additionally, one cache line remains for storing the parts of C currently needed [2].

The differences for N_B resulting from an energy optimisation and a time optimisation of ATLAS are evaluated using the original ATLAS library and the ATLAS version replacing the time measurement by energy measurement according to Sect. II-B2. In both versions, the autotuning process performs two consecutive searches for the best N_B: In the first search, a preliminary value is determined which is used during the optimisation of the values of the loop unrolling factors m_u and n_u. In the second search, which this section deals with, the final value is determined. The second search uses the loop unrolling factors found in the first step. The search itself is performed iteratively, i.e. the next candidate value of N_B is determined based on the performance of the current value. Therefore, the searches for the time-optimal and for the energy-optimal values for N_B may diverge.

The time and energy efficiency for the N_B candidate values (as created by the search algorithm) of the single-precision matrix multiplication are shown in Fig. 2. The horizontal axis shows the performance of the energy-optimised variant for the given block size N_B, the vertical axis shows its performance with the time-optimised variant. So, the rightmost value, i.e. $N_\mathrm{B} = 76$, exhibits the best energy efficiency, the uppermost value, i.e. $N_\mathrm{B} = 80$, exhibits the best time efficiency. It may be questioned, however, whether the differences in the cluster $\{60, 64, 68, 76, 80\}$ are really significant, as they only differ in a range of roughly 2 %. The value $N_\mathrm{B} = 72$ deserves special attention: Its time efficiency is less than 1 % worse than the best of $N_\mathrm{B} = 80$. But its energy efficiency is roughly 9 % worse than for the energy-optimal value $N_\mathrm{B} = 76$. So, under

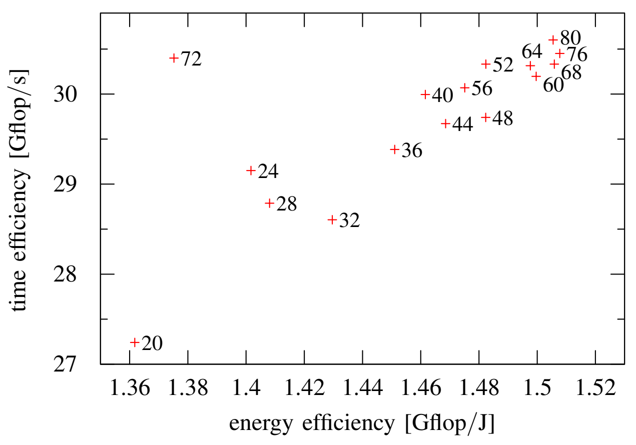

Figure 2. Efficiency for different values of the low-level matrix multiplication block size N_B of the energy-optimised variant (horizontal axis) and the time-optimised variant (vertical axis), outlier $N_B = 16$ omitted (data type: *float*)

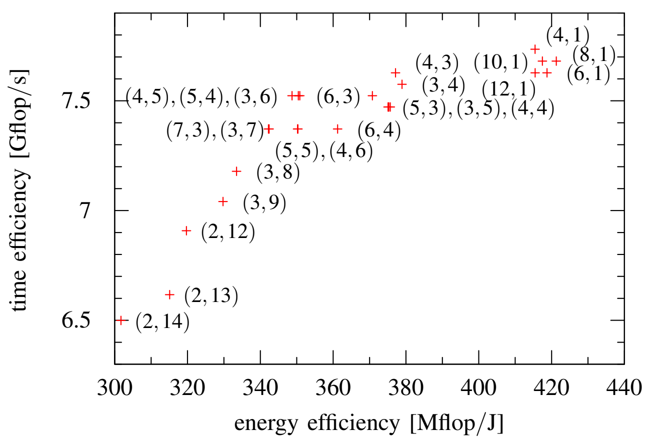

Figure 3. Efficiency for the (m_u, n_u) loop unrolling factors for the energy-optimised (horizontal axis) and the time-optimised (vertical axis) variants (data type: *double*, $N_B = 56$)

slightly different conditions, e.g. by using a processor with a slightly higher or lower frequency, the search for the time-optimal N_B could yield an $N_B = 72$ as the optimum. That would substantially degrade the energy efficiency. These large differences in the energy efficiency for parameter settings exhibiting a similar time efficiency show that a dedicated optimisation for energy efficiency is necessary.

For obtaining values comparable between the time-optimising and the energy-optimising variant, besides the hardware parameters, the following parameters have been set fix in these experiments: The loop unrolling factors are $m_u = 4$, $n_u = 1$, $k_u = 1$, and fused multiply-add is used. These values have been obtained by the time-optimising variant.

B. *Loop Unrolling Factors for Low-level Matrix Multiplication*

The cache-contained matrix multiplication uses a triple-nested loop running over the dimensions N, M and K of the matrices. The K loop is the outermost loop, the M and N loops are the inner loops. The inner loops are unrolled by the factors m_u and n_u, such that all local variables can be stored in the registers available. Different settings for (m_u, n_u) are probed, and k_u is chosen in a subsequent step. As with N_B, the search for optimal unrolling factors is performed twice. This section deals with the values obtained in the second search, i.e. after the final value for N_B has been obtained.

Figure 3 shows the time efficiency and the energy efficiency for different settings of (m_u, n_u) for the double-precision matrix multiplication. The axes indicate the energy efficiency and the the time efficiency for the given values using the ATLAS variants optimised for the respective criterion. The figure shows that there is again a cluster of parameter settings which are efficient in both, time and energy. This cluster comprises the values $\{(4,1),(10,1),(8,1),(12,1),(6,1)\}$, i.e. all values with $n_u = 1$. There is an outlier from this cluster, $(m_u, n_u) = (4,3)$, which has a slightly worse time efficiency (1 %), but a clearly worse energy efficiency (12 %). So if, due slightly changed conditions, the N loop is unrolled by a factor of 3 instead of by a factor of 1 when optimising for time, the energy efficiency degrades by 12 %. This again shows the need for an energy-aware tuning process. This finding is supported by [6], where it was also found that loop unrolling factors have to be set differently for time and energy optimisation.

C. *Copy/no-copy Crossover*

Before the execution of a GEMM-based operation, ATLAS copies and rearranges large matrices in memory for improving the efficiency. Alongside the copying, matrices are transposed (if necessary), the scalar factor α is applied, the data is broken up into contiguous blocks of size $N_B \times N_B$, and reordered into a block-major format. With the copied matrices, the low-level (cache-contained) matrix multiplication is used. The copy operation has costs of $O(n^2)$, compared to the costs of $O(n^3)$ for the multiplication operation for large matrices. Thus, for large matrices, the copy operation does not have a substantial impact on the execution time, whereas for small matrices, the copy operation has a relevant impact.

There is a specific matrix size, called the *crossover point*, which reflects the boundary between the copy and the no-copy strategy: If the matrices are larger than the crossover point, they are copied. If the matrices are smaller, they are not copied. According to [2], this crossover point ‘strongly depends’ on the hardware architecture, thus making a tuning of this parameter necessary. The crossover points not only depend on the hardware architecture, but also on the shape of the matrices being processed. Table I indicates under which conditions the matrices are copied (as coded in src/blas/gemm/ATL_-gemmXX.c). For example, if K is smaller than $3 \cdot N_B$, then the matrices are copied if and only if the product $M \cdot N \cdot K$ is not greater than the crossover constant *MNK_K*. If all dimensions are greater than or equal to $3 \cdot N_B$, then the matrices are copied if and only if their product is not greater than the crossover constant *MNK_GE*. These crossover constants are determined

matrix shapes K	M	N	copy matrices iff $M \cdot N \cdot K \dots$
$K < 3N_B$			$\leq MNK\_K$
$K \geq 3N_B$	$M < 3N_B$	$N < 3N_B$	$\leq MNK\_MN$
$K \geq 3N_B$	$M < 3N_B$	$N \geq 3N_B$	$\leq MNK\_N$
$K \geq 3N_B$	$M \geq 3N_B$	$N < 3N_B$	$\leq MNK\_M$
$K \geq 3N_B$	$M \geq 3N_B$	$N \geq 3N_B$	$\leq MNK\_GE$

Table I
CHOICE OF COPY/NO-COPY PARAMETER FOR GEMM DEPENDING ON THE MATRIX SHAPES DEFINED BY N, M AND K AND THEIR SIZE DEFINED BY $M \cdot N \cdot K$

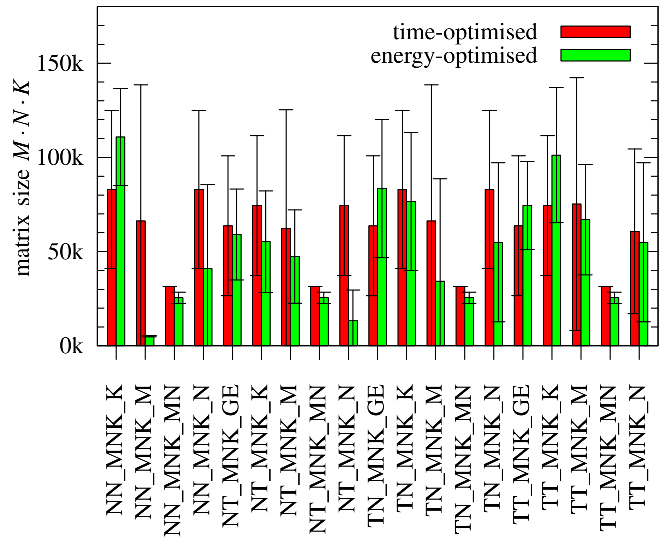

Figure 4. Copy/no-copy crossover points for double-precision operations with different matrix shapes (90 % confidence intervals given)

empirically. Each of them receives a prefix indicating whether the matrices involved are transposed (T) or not transposed (N).

The copy/no-copy crossover points are determined in tune/blas/gemm/mmcuncpsearch.c. In the file bin/gemmtst.c, the time measurement has been replaced by energy measurement resulting in the energy consumption being used as optimisation goal. No precautions are taken that the other tuning parameters are identical in the time-optimising and the energy-optimising variants as the copy/no-copy search is hard to isolate.

The results of the experiment are shown in Fig. 4. The bars indicate the mean values of the crossover points for the different matrix shapes as obtained in 5 runs. Additionally, the 90 % confidence intervals are given. For each matrix shape, there is one crossover point for the time-optimised ATLAS and one crossover point for the energy-optimised ATLAS. There is tendency towards smaller crossover points for the energy-optimised variant. Due to the large error intervals, this tendency is, however, not significant in most cases. The reason for large error intervals is the wide variation in the results for each matrix shape. Only for the *MNK_MN* matrix shapes, the crossover points for the energy-optimised variant are significantly smaller than for the time-optimised variant.

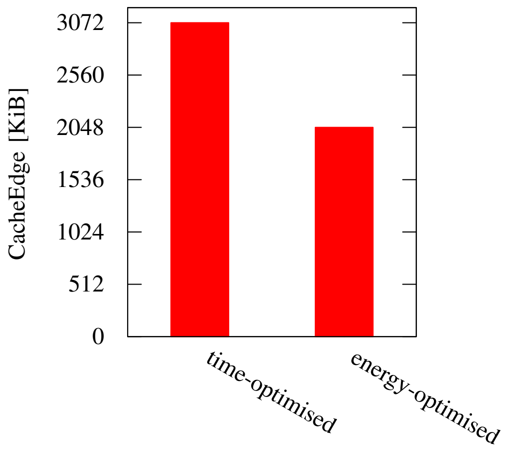

Figure 5. CacheEdge for the time-optimised and the energy-optimised version of ATLAS

This means that, for an energy-optimised operation, sometimes a relatively small matrix is copied which would not have been copied for a time-optimised operation.

The results indicate that the energy efficiency makes more profit of a regular execution scheme, which is ensured by the matrix copying, than the time efficiency. As a consequence more matrices are copied if ATLAS is optimsed for energy consumption. This partly contradicts findings that intra-node data movement has a large impact on the energy consumption [13], [20]. This effect, however, might not be visible here as the energy consumed by the RAM is not considered by the RAPL register used.

D. CacheEdge

Besides the cache-contained low-level matrix multiplication, which optimises for the level-1 cache, ATLAS also offers an optimisation for higher levels of cache, i.e. level-2 or level-3 cache. For this optimisation, the parameter *CacheEdge* is tuned. This parameter is determined empirically and it 'represents the amount of the cache that is usable by ATLAS for its particular kind of blocking' [2]. The values are taken from the file include/atlas_cacheedge.h. The values obtained by the ATLAS versions replacing time by energy measurement according to Sect. II-B2 are shown in Fig. 5. The chart shows that for an energy-efficient operation, only a smaller portion of the higher-level cache should be used, compared to a time-efficient operation. Using a smaller amount of cache leads to smaller block sizes for the operations. This improves data locality, which might be more efficient. It furthermore leaves more space in the cache for data not taken into account for the block size calculation, which avoids energy-intensive data movement to some extend.

IV. COMPLETE-PACKAGE ENERGY AUTOTUNING

This section presents the results of the complete adaptation of ATLAS for energy optimisation. The function measuring the time is replaced by a function measuring the energy as discussed in Sect. II-B2. So the optimisations minimise the energy consumption instead of the execution time.

A selection of four routines representing the BLAS levels 1, 2, and 3 has been investigated with the complete-package energy autotuning. Figure 6 shows the results of the measurements. The functions investigated are the GEMM general matrix-matrix multiplication, the GEMV general matrix-vector multiplication, the SPMV symmetric packed matrix-vector multiplication, and the AXPY vector dot product. The charts show the execution time and the energy consumption for one call of the respective routine with different input data sizes, as well as the resulting electrical power of the CPU. The results have been taken for both, operations with single precision and with double precision floating point numbers.

The results show that the tuning for energy compared to tuning for time often improves both, the time efficiency as well as the energy efficiency. For instance, for the GEMM routine with double precision, the energy consumption decreases by roughly 6 % while the execution time decreases by roughly 4 %. This results in a decreased power of roughly 6 %. With single precision, time and energy consumption decrease by roughly 4 %. The biggest reduction in the energy consumption is observed for the GEMV routine, where there is a decrease by roughly 10 % to 15 % for both data types.

A relatively low decrease by roughly 2 % is observed for the SPMV routine, which might be caused by irregularities in the memory access due to the packed storage format of the matrix. In contrast, the GEMM and GEMV routines are routines with very regular memory access patterns, which can be optimised more easily. A flucutating behaviour between an increase and a decrease in energy consumption of up to 9 % is observed for the AXPY routine. The AXPY routine is memory-bound which might explain the behaviour differing from the other routines.

In some cases, e. g. for the single-precision GEMM, the energy-optimised variant resulted in a lower execution time than the time-optimised variant. The reason for that might be sought in the more precise measurement due to the increased number of repetitions employed for the energy optimisation. However, also in these cases, the energy consumption decreases by a higher factor than the execution time. This is shown by the fact that a lower power can be observed (which is the ratio between energy and time).

V. Related Work

Autotuning methods for minimising the energy consumption of an application have for example been investigated in [20], [3], [4], [6]: [20] minimises the energy consumption of an LU factorisation by an automated minimisation of the memory traffic based on a model for the number of memory accesses. [3] minimises the energy consumption of the Fast Multipole Method, a particle simulation, by selecting the optimal depth of the particle tree dividing the interactions into near-field and far-field interactions. [4] presents an autotuning approach using polyhedral optimisation. Three scientific kernels from the optimiser's test suite are tuned for minimal energy consumption on the Xeon Phi. [6] autotunes the unrolling factors for the two loops of a stencil operation within a Poisson's equation solver. The objective function is the energy-delay product. None of the above works deals with tuning BLAS operations for energy consumption as the present work does.

Multi-objective optimisation, viz optimisation for energy consumption and a further criterion, is dealt with in [5], [21], [22]. In [5], such a multi-objective autotuner is presented. The article investigates the behaviour of three different hardware architectures, a Xeon Phi, a multi-core Xeon, and a Blue Gene, when executing a number of scientific kernels with dynamic voltage and frequency scaling. Voltage and frequency scaling has not been investigated in the present work as we restricted ourselves to tuning parameters which are already available in ATLAS. [21] presents an autotuner with multi-objective optimisation for time, energy and resource usage. A number of scientific kernels, including GEMM and SYR2K are investigated. A detailed analysis of the parameters is only given for GEMM. The article mainly focuses of the the number of CPU cores and the frequency as parameters, which has not been investigated in the present article. In [22], the GEMM, GEMV and GER BLAS routines are investigated. Different orders of exploring the search space, i.e. the order of applying optimisations such as blocking or parallelisation, are investigated.

A comprehensive survey on the recent developments in the field of energy-aware and energy-efficient linear algebra methods is given in [23].

VI. Conclusion and Future Work

This article has evaluated the possibilities to autotune the ATLAS linear algebra package for a minimum energy consumption. The influence of a choice of tuning parameters on the energy consumption has been investigated. Some tuning parameters show a strong influence, some only a show small influence on the overall energy consumption. A more thorough study is needed for giving a detailed analysis on which parameters are important for optimising the energy consumption of a linear algebra library.

The study proves that tuning for energy consumption is necessary for obtaining an energy-optimised ATLAS library. For several parameters, including the block size N_B for the matrix multiplication and the unrolling factors, there are parameter settings which yield similar execution times, but cause large differences (up to 10 %) in the energy consumption. Therefore, an explicit analysis of the energy consumption is needed in order to avoid that the time-optimising variant selects a non-optimal parameter setting regarding the energy consumption.

In a final evaluation, the time measurement in ATLAS has been replaced by an energy measurement in order to obtain a variant of the library completely tuned for energy consumption. This energy-tuned ATLAS produces linear algebra routines

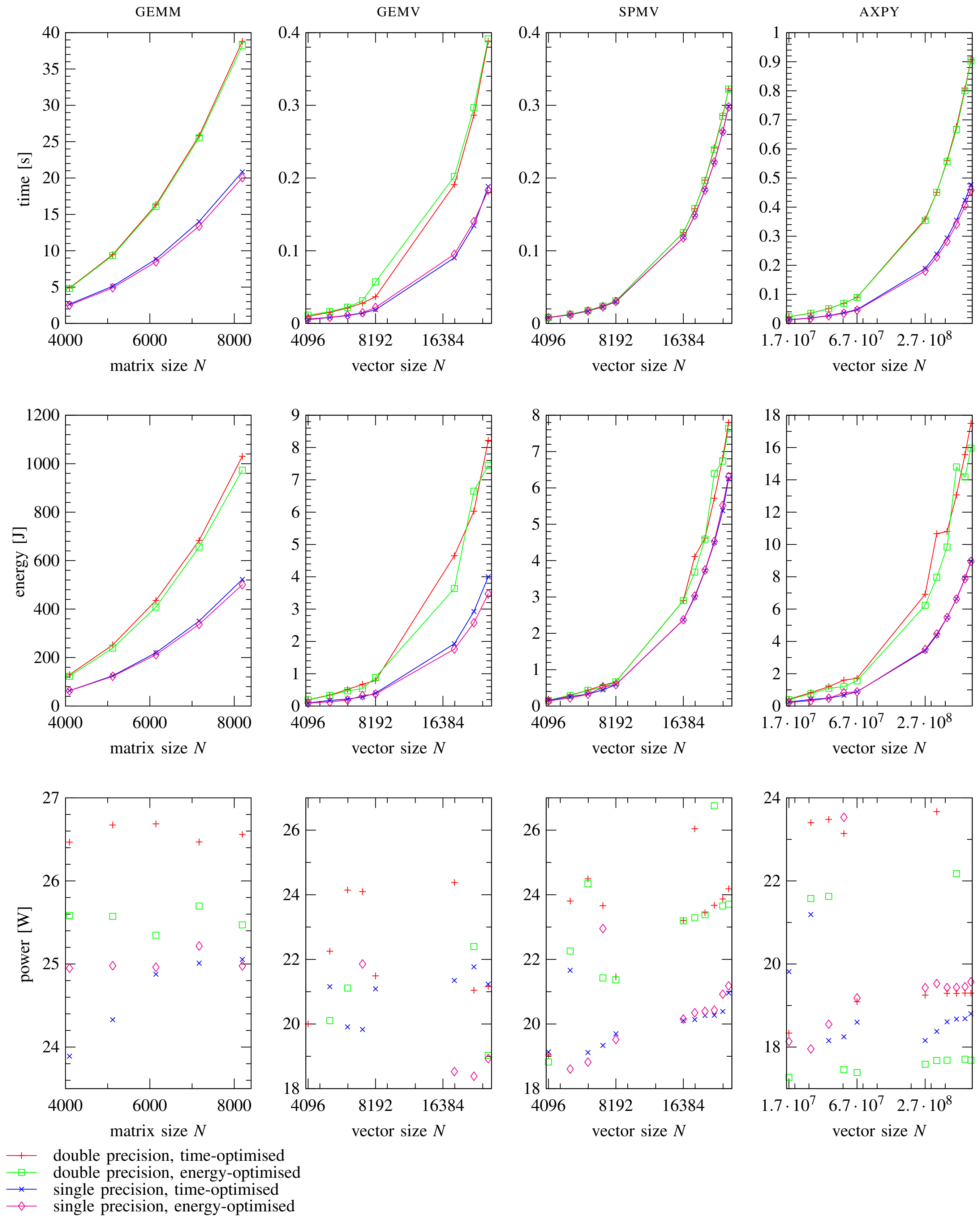

Figure 6. Execution time, energy consumption and power for the BLAS routines GEMM, GEMV, SPMV and AXPY for the time-optimised and the energy-optimised variants of ATLAS

which need from a few percent up to 10 percent less energy than the routines of an ATLAS tuned for time.

In this study, the energy consumption was the only optimisation criterion for ATLAS. The results can be used as a base for applying compound scalar metrics such as energy-delay products or for multi-criteria optimisations. However, some fundamental changes would have to be made to ATLAS for a multi-criteria optimisation. The results may of this study also serve as a base for a version of ATLAS considering a more comprehensive set of parameters that have a greater impact on the energy consumption, such as frequency scaling, the number of cores used, or the amount of memory traffic. Including these parameters into the autotuning decision might result in an even higher saving in energy consumption.

Future work includes also an investigation of other autotuned linear-algebra packages, such as MAGMA [24]. Research is needed for assessing whether the same energy tuning methods as here can be used for MAGMA. Especially the heterogeneity of machines which is considered by MAGMA will need special attention. From comparing different autotuned linear-algebra packages, it might be possible to set up general, evidence-based rules for energy optimisation if parameters such as unrolling factors or cache usage show a similar behaviour in these packages.

Acknowledgement

This work has been supported by Deutsche Forschungsgemeinschaft, grant number RU 591/10-2.

References

[1] R. van de Geijn and K. Goto, "BLAS (Basic Linear Algebra Subprograms)," in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Springer, 2011, pp. 157–164.

[2] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," *Parallel Computing*, vol. 27, no. 1-2, p. 3–35, 2001.

[3] J. Lang, "Energie- und Ausführungszeitmodelle zur effizienten Ausführung wissenschaftlicher Simulationen," Ph.D. dissertation, Technische Universität Chemnitz, Department of Computer Science, Chemnitz, Germany, 2014.

[4] W. Wang, J. Cavazos, and A. Porterfield, "Energy auto-tuning using the polyhedral approach," in *Proc. International Workshop on Polyhedral Compilation Techniques at HiPEAC 2014 conference*, 2014.

[5] P. Balaprakash, A. Tiwari, and S. Wild, "Multi objective optimization of hpc kernels for performance, power, and energy," in *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, ser. Lecture Notes in Computer Science, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Springer, 2014, pp. 239–260.

[6] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, "Auto-tuning for energy usage in scientific applications," in *Proceedings of the 2011 international conference on Parallel Processing - Volume 2*, ser. Euro-Par'11. Springer, 2012, pp. 178–187.

[7] R. W. Vuduc, "Autotuning," in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Springer, 2011, p. 102–105.

[8] K. Yotov, X. Li, G. Ren, M. Garzaran, D. Padua, K. Pingali, and P. Stodghill, "Is search really necessary to generate high-performance BLAS?" *Proceedings of the IEEE*, vol. 93, no. 2, p. 358–386, 2005.

[9] J. I. Aliaga, M. Barreda, M. F. Dolz, and E. S. Quintana-Ortí, "Are our dense linear algebra libraries energy-friendly?" *Computer Science - Research and Development*, pp. 1–10, 2014.

[10] P. Stöcker, "Automatisierte Optimierung des Energieverbrauchs für Routinen der Linearen Algebra," Bachelor's thesis, Technische Universität Chemnitz, Department of Computer Science, Chemnitz, Germany, 2014.

[11] E. Rotem, A. Naveh, A. Ananthakrishnan, D. Rajwan, and E. Weissmann, "Power-management architecture of the intel microarchitecture code-named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, p. 20–27, 2012.

[12] J. Lang and G. Rünger, "An execution time and energy model for an energy-aware execution of a conjugate gradient method with CPU/GPU collaboration," *Journal of Parallel and Distributed Computing*, 2014.

[13] J. Demmel and A. Gearhart, "Instrumenting linear algebra energy consumption via on-chip energy counters," EECS Dept., University of California, Berkeley, Tech. Rep. UCB/EECS-2012-168, 2012.

[14] G. Bosilca, H. Ltaief, and J. Dongarra, "Power profiling of Cholesky and QR factorizations on distributed memory systems," *Computer Science - Research and Development*, vol. 29, no. 2, pp. 139–147, 2014.

[15] T. Rauber, G. Rünger, M. Schwind, H. Xu, and S. Melzner, "Energy measurement, modeling, and prediction for processors with frequency scaling," *The Journal of Supercomputing*, vol. 70, no. 3, pp. 1451–1476, 2014.

[16] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, vol. 0, pp. 194–204, 2013.

[17] S. Browne, J. Dongarra, G. H. Nathan Garner, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *International Journal of High Performance Computing Applications*, vol. 14, no. 3, p. 189–204, 2000.

[18] *Intel 64 and IA-32 Architectures Software Developer's Manual*, Intel, May 2012.

[19] C. Angelini. (2013, Jun.) The Core i7-4770K review: Haswell is faster; desktop enthusiasts yawn. Tom's Hardware. [accessed 2014-12-18]. [Online]. Available: http://www.tomshardware.com/reviews/core-i7-4770k-haswell-review,3521.html

[20] E. Garcia, J. Arteaga, R. Pavel, and G. R. Gao, "Optimizing the LU factorization for energy efficiency on a many-core architecture," in *Languages and Compilers for Parallel Computing*, ser. Lecture Notes in Computer Science, C. Caşcaval and P. Montesinos, Eds. Springer, 2014, pp. 237–251.

[21] P. Gschwandtner, J. J. Durillo, and T. Fahringer, "Multi-objective auto-tuning with Insieme: Optimization and trade-off analysis for time, energy and resource usage," in *Euro-Par 2014 Parallel Processing*, ser. Lecture Notes in Computer Science, F. Silva, I. Dutra, and V. Santos Costa, Eds. Springer, 2014, vol. 8632, pp. 87–98.

[22] S. F. Rahman, J. Guo, and Q. Yi, "Automated empirical tuning of scientific codes for performance and power consumption," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, ser. HiPEAC '11. ACM, 2011, pp. 107–116.

[23] L. Tan, S. Kothapalli, L. Chen, O. Hussaini, R. Bissiri, and Z. Chen, "A survey of power and energy efficient techniques for high performance numerical linear algebra operations," *Parallel Computing*, vol. 40, no. 10, pp. 559 – 573, 2014.

[24] Y. Li, J. Dongarra, and S. Tomov, "A note on auto-tuning GEMM for GPUs," in *Proceedings of the 9th International Conference on Computational Science: Part I*, ser. ICCS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 884–892.