Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Tuning linear algebra for energy efficiency on multicore machines by adapting the ATLAS library

Thomas Jakobs, Jens Lang*, Gudula Rünger, Paul Stöcker

Department of Computer Science, Technische Universität Chemnitz, 09107 Chemnitz, Germany

HIGHLIGHTS

- We present a version of ATLAS that minimises energy instead of time.
- When ATLAS is tuned for minimal energy consumption, 10% of energy can be saved.
- Some tuning parameters show differences when optimised for time and for energy.
- Being compute-bound, DGEMM profits most from the tuning for energy.

ARTICLE INFO

Article history:
Received 29 February 2016
Received in revised form
21 January 2017
Accepted 7 March 2017
Available online xxxx

Keywords:
Energy autotuning
Energy efficiency
Power
Basic linear algebra subroutines (BLAS)
ATLAS

ABSTRACT

While automated tuning is an established method for minimising the execution time of scientific applications, it has rarely been used for an automated minimisation of the energy consumption. This article presents a study on how to adapt the auto-tuned linear algebra library ATLAS to consider the energy consumption of the execution in its tuning decision. For different tuning parameters of ATLAS, it investigates which differences occur in the tuning results when ATLAS is tuned for a minimal execution time or for a minimal energy consumption. The tuning parameters include the matrix size for the low-level matrix multiplication, loop unrolling factors, crossover points for different matrix-multiplication implementations, the minimum size for matrices to be transposed, or blocking sizes for the last-level cache. Also, parameters for multithreaded execution, such as the number of threads and thread affinity are investigated. The emphasis of this article is on a method proposed with which it is possible to replace a tuning process for execution time by a tuning for energy consumption, especially in the parallel case. ATLAS serves as a prominent example for a tuned library. Furthermore, the article draws conclusions on how to design an energy-optimising autotuning package and how to choose tuning parameters. The article also discusses why the matrix-matrix multiplication has a potential for increasing the energy efficiency while the time efficiency remains constant, whereas other routines have shown to improve their energy efficiency by reducing the execution time.

1. Introduction

Most simulations from the field of scientific computing can be decomposed into subroutines from a set of commonly used numerical and computational methods [1]. One of the most important classes of such subroutines is dense linear algebra. As a standard interface for dense linear algebra libraries, BLAS (Basic Linear Algebra Subroutines) has established over the last decades [2]. Since keeping pace with the constantly improving hardware is important, automated tuning or autotuning methods are widely used and have also been used for packages implementing the BLAS interface [3]. Until today, autotuning has mainly focused on minimising the execution time of an application [4]. However, energy efficiency of scientific codes will become an important design goal in the near future [5]. Currently, there is no autotuning package for linear algebra available that considers energy as an optimisation goal. This article proposes a method for creating a BLAS library which is automatically tuned for a minimum energy consumption.

One of the most popular autotuned BLAS packages is ATLAS (Automatically Tuned Linear Algebra Software) [3]. ATLAS investigates a large number of code variants for different BLAS operations and selects those showing the best performance, partly even considering data characteristics. It finds the optimal values for several tuning parameters, such as block sizes, pre-fetch distances, or

* Corresponding author.
E-mail addresses: thomas.jakobs@cs.tu-chemnitz.de (T. Jakobs), jens.lang@cs.tu-chemnitz.de (J. Lang), ruenger@cs.tu-chemnitz.de (G. Rünger), stop@hrz.tu-chemnitz.de (P. Stöcker).

http://dx.doi.org/10.1016/j.future.2017.03.009
0167-739X/© 2017 Elsevier B.V. All rights reserved.

performance 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 reduces the energy consumption by nearly the same percentage [6–8], this is not always true; in [7,9,10], it is shown that a deviation of up to 7% is possible.

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 in which a conflict between the two goals of optimising execution time or energy occurs. Thus, our approach is a proposal to tackle the problem of a multi-objective optimisation by enhancing the autotuning approach in ATLAS. An important requirement for our approach is that running the adapted ATLAS routines shall not have any side effects that may influence other running processes or non-ATLAS threads. This prevents the use of widely known and effective techniques, such as frequency manipulation or clock gating [5].

The parameters that are evaluated during the autotuning process of ATLAS for execution time are also evaluated by using energy measurements. This enables us to point out specific differences in the effect of e.g. matrix block sizes on timing and energy behaviour. Some parts of this investigation have been presented earlier in [11], which focused on parameters for singlethreaded execution of the BLAS routines. In this article, we extend this work by including parameters that are mainly relevant for a multithreaded execution and on a multi-objective optimisation. The additional contributions of this article are given in the following. It investigates whether using simultaneous multithreading and thread affinity yield a higher performance in terms of time or energy for multithreaded execution. Additionally, the influence of different execution variants on cache usage boundaries is discussed. Thus, the article presents an ATLAS variant for multithreaded execution, which has been completely tuned for energy consumption. Hence, this ATLAS variant should exhibit a minimum energy consumption of all variants producible by the autotuner.

In summary, the contributions of this article are:

- the modification of the autotuning process of ATLAS with the goal to exploit different objectives, i.e. execution time and energy efficiency;
- a concise and thorough empirical study of the differences in execution time and energy behaviour of the ATLAS library with respect to several parameters, such as matrix block size or cache boundaries;
- a first step towards a multi-objective autotuning process concentrating on the two optimisation goals time and energy by picking the implementation variant with the best result for both criteria;
- the evaluation of the autotuning process for multithreaded execution.

The rest of this article is structured as follows: Section 2 gives a summary on the autotuning mechanism of ATLAS and proposes methods for introducing energy awareness into the autotuner. Section 3 investigates several tuning parameters and their behaviour when tuned for execution time and for energy consumption. Section 4 investigates the effect of tuning the complete package for energy consumption. Section 5 presents related work, and Section 6 concludes the article.

2. 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" [12]. Autotuning usually consists of three steps [12]:

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.

2.1. The ATLAS autotuning process

A good review of the autotuning process of ATLAS is given in [3,13]. Fig. 1 visualises the steps of the ATLAS 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 tested. 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 [3], 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., [14,7].