Journal of Computational Science 10 (2015) 299–310

Contents lists available at ScienceDirect

Journal of Computational Science

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

Auto-tuning techniques for linear algebra routines on hybrid platforms

Gregorio Bernabé [a], Javier Cuenca [a], Luis-Pedro García [b], Domingo Giménez [a,*]

[a] University of Murcia, Spain
[b] Technical University of Cartagena, Spain

ARTICLE INFO

Article history:
Received 10 October 2014
Received in revised form 25 March 2015
Accepted 22 April 2015
Available online 25 April 2015

Keywords:
Auto-tuning
Models of the execution time
Parallel linear algebra
Hybrid CPU + GPU computing
Intel Xeon Phi

ABSTRACT

This work analyses two techniques for auto-tuning linear algebra routines for hybrid combinations of multicore CPU and manycore coprocessors (single or multiple GPUs and MIC). The first technique is based on basic models of the execution time of the routines, whereas the second one manages only empirical information obtained during the installation of the routines. The final goal in both cases is to obtain a balanced assignation of the work to the computing components in the system. The study is carried out with a basic kernel (matrix–matrix multiplication) and a higher level routine (LU factorization) which uses the auto-tuned basic routine. Satisfactory results are obtained, with experimental execution times close to the lowest experimentally achievable.

1. Introduction

In most scientific and engineering problems, computations are carried out with basic BLAS type matrix routines. Level 3 BLAS collects all the matrix–matrix operations, which are the set of the most computational intensive BLAS routines. Therefore, the improvement in the performance of scientific codes is achieved in many cases by the efficient use of these routines.

Efforts have been devoted to the optimization of linear algebra routines in computational systems of different characteristics [1–5]. The decisions to take depend on the type of the computational system for which the routines are developed. For example, it is necessary to adapt the original ideas developed for homogeneous parallel systems to heterogeneous or dynamic systems [6–9], and the omnipresence today of CPU + coprocessor systems makes the adaptation of previous auto-tuning techniques to these systems compulsory. This paper studies empirical auto-tuning techniques to achieve optimum load balance between a multicore CPU and different possible manycore coprocessors (single or multiple GPUs and MIC [10]) when they perform linear algebra routines. The CPU part can be carried out with a multithread BLAS library. Many BLAS implementations exist, for multicore (vendors implementations: Intel MKL [11], IBM ESSL [12], etc.; or free implementations: ATLAS [5], Goto BLAS [13], etc.) and for GPUs (CULA Tools [14], CUBLAS [15] and MAGMA [16–18]). The CPU and GPU implementations used here are MKL and CUBLAS, but the same techniques can be applied with all other basic libraries. Our study focuses on these two libraries, although preliminary experiments carried out with others provided similar performance results.

In previous works we have considered two different auto-tuning methods to obtain a balanced distribution of the tasks between the computing resources according to the machine characteristics: an experimental method (guided search) and a mixed theoretical-experimental method (empirical modeling). The guided search method was analyzed in [19] and the empirical modeling in [20], in both cases for NUMA platforms. Their extension to CPU + GPU platforms is described in [21] (guided search) and [22] (empirical modeling). In this work, the two methods are adapted to hybrid platforms composed of a multicore CPU plus different co-processors: a single GPU, multiple GPUs or an Intel Xeon Phi.

The rest of the paper is organized as follows. Section 2 comments on some adaptations of linear algebra software to GPU and combinations of CPU + GPUs and CPU + MIC. Section 3 introduces the two auto-tuning methodologies and explains their uses for a basic kernel (matrix–matrix multiplication) in a multicore+GPU system. The experimental results for this kernel are shown in Section 4. Section 5 describes how to use the basic auto-tuned kernel in a LU factorization in order to improve this higher-level routine. The experimental results with the LU factorization are shown in Section 6. The extension of the methodology to other types of hybrid

* Corresponding author. Tel.: +34 606860619.

http://dx.doi.org/10.1016/j.jocs.2015.04.032
1877-7503/© 2015 Elsevier B.V. All rights reserved.

systems (multicore+multiGPU and multicore+MIC) is discussed in Section 7. Finally, Section 8 concludes the paper and outlines possible research directions.

2. Linear algebra on hybrid platforms

Due to the omnipresence of multicore systems with GPU accelerators, efforts are being devoted to the development of software for these systems, and especially to the design of linear algebra routines which manage the heterogeneity of the whole system to obtain the maximum achievable performance. In [23] a strategy is presented to perform matrix–matrix multiplications on hybrid NVIDIA GPU systems. The basic idea is to carry out a matrix multiplication $A = BC$ by splitting the data of matrices B and C between the CPU and a single GPU, and performing the operations simultaneously on both devices. The final result is obtained by aggregation of the results independently obtained in CPU and GPU. A similar approach is used in the core numerical kernels included as part of the NVIDIA LINPACK TOP 500 benchmark suite [24] to rank the fastest heterogeneous supercomputers in the world. The PHIGEMM [25] library uses a workload distribution based on a pre-defined split factor and the latest capabilities of CUDA to efficiently control asynchronous data transfer and overlapping multi-device computations. Users must define this split factor manually according to the ratio of computational power of the CPU and the GPU. In [26] a hybrid programming model combining MPI, OpenMP and streaming computing is described. The LAPACK task, thread and data parallelisms are exploited. The main idea to optimize the load distribution across the CPUs and GPUs is to use a two-level adaptive method to measure the relative performance of GPUs and CPUs at runtime. Additionally, a software pipelining technique is used to bypass the low-bandwidth communication between CPU and GPU. [27] proposes a variable block size auto-tunig scheme on CPU–GPU hybrid systems for the QR factorization in MAGMA. The approach is to fit the CPU and GPU cost via two independent regression models and to define a cost objective function to balance the workloads between CPU and GPU. Moreover, various auto-tuning projects oriented to mutiple GPUs can be found in the literature: The StarPU platform [28] provides numerical kernel designers a way to generate parallel tasks over heterogeneous hardware with multiple cores and GPUs. The proposal of [29] consists of a task-based fine-grained execution scheme that can dynamically balance workload on individual GPUs and among different GPUs located in a single node. In order to do that, a task queue scheme is implemented between the host and the different devices on the machine. Finally, [30] is a proposal for a polyalgorithmic autotuner that combines different algorithmic techniques, determining empirically how to map portions of programs across all types of resources (CPU and GPUs) in a single node.

On the other hand, the launch of the Intel Many Integrated Core architecture (Intel MIC architecture) [10] combining many Intel CPU cores onto a single chip gives developers a key advantage: they run on standard, existing programming tools and methods. The same program source code written for Intel MIC products can be compiled and run on a standard Intel Xeon processor. Intel says its coprocessors have a few advantages over GPGPUs: they operate independently of CPUs and they do not require special code to program. Several auto-tuning projects are under way for this platform, beginning with the Intel Math Kernel Library (MKL) [11], which when their functions are called on the Intel MIC architecture make best use of SIMD parallelism. Another auto-tuned library for this platform is MAGMA MIC [31], which uses a hybridization methodology, where the algorithms are split into computational tasks of varying granularity and, then, they are properly scheduled over the heterogeneous hardware.

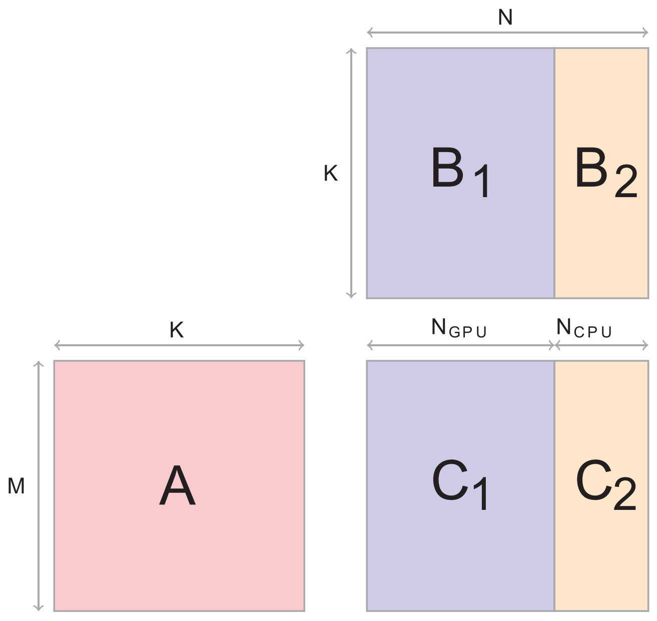

Fig. 1. Distribution of the computation of a matrix multiplication on a CPU + GPU system. The blue portion (left part) is performed in the GPU and the orange portion (right part) in the CPU. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

In [32] an auto-tuning method for the LU factorization is described. In this proposal, the initial amount of work to assign to both the CPU and GPU is determined before execution by means of an analytical model of the execution time, and the workload is dynamically balanced during the execution. In our proposal we use a static approach to decide the split of the matrices between the components of the heterogeneous computing system. The dynamic selection is discarded because it would suppose high overheads when the matrix multiplication is used inside higher level codes. Two techniques previously used for tuning linear algebra routines in NUMA systems according to the machine's characteristics can be applied for this new computing system: an experimental method (guided search) and a mixed theoretical-experimental method (empirical modeling). These techniques are used to tune a matrix–matrix multiplication kernel which is then included inside a matrix factorization by reusing the tuned information obtained for the kernel to improve the higher level routine.

3. Auto-tuning a multi-device matrix multiplication

The matrix multiplication is computed simultaneously on both GPU and CPU cores. The multiplication $C = \alpha AB + \beta C$ can be expressed as $C = \alpha(AB_1 + AB_2) + \beta(C_1 + C_2)$, and $AB_1 + \beta C_1$ can be performed in the GPU and $AB_2 + \beta C_2$ in the CPU[1] (Fig. 1). In the experiments, the CUBLAS library (`cublasDgemm` routine) is used for the computation in the GPU, and the MKL library (`dgemm` routine) in the CPU.

An optimum split of the matrix would keep the time consumed by the GPU and CPU balanced [23,33]. The multi-device (GPU and CPU) computations are overlapped and the data transfers between GPU and CPU are performed asynchronously in order to achieve the maximum performance. To reduce the data transfer time between

[1] Preliminary implementations with different workload splittings of the dgemm were considered, also using CUDA streams in several ways for different communication-computation overlapping situations. Finally, a simple splitting scheme has been considered for this work in order to pay attention to the self optimization process more than the algorithm design.