Automatically Tuned on Multicore Systems

Murilo Boratto[1], **Leandro Coelho**[2], **Brauliro Leal**[1]

[1] Colegiado de Engenharia da Computação (CECOMP)
Universidade Federal do Vale do São Francisco (UNIVASF)
Av. Antonio Carlos Magalhães, 510, Santo Antonio, 48902-300,
Juazeiro – Bahia – Brazil

[2]Núcleo de Arquitetura de Computadores e Sistemas Operacionais (ACSO)
Universidade do Estado da Bahia (UNEB)
Rua Silveira Martins, 2555, Cabula, 41195-001,
Salvador – Bahia – Brasil

{murilo.boratto, brauliro.leal}@univasf.edu.br, leandrocoelho@uneb.br

Abstract. *Automatic tuning techniques have been used in the design of several programming algorithms in recent years. This paper aims to explore the programming routines that can be automatically adapted to the computational system conditions. Techniques have been developed in different fields, and especially in linear algebra routines. In this work the possibility of applying automatic optimization techniques to solve the triangular system problem is analyzed. The routines are developed along with their theoretical execution time, t(s) = f(s, AP, SP), where s represents the problem size, SP are system parameters, and AP are algorithmic parameters.*

1. Introduction

During the last years different auto-optimization techniques of parallel routines have been developed with the purpose of achieving routines that automatically tune to the characteristics of computer systems and that are able to run efficiently independent of the knowledge of end-users in parallel programming. The potential users of high performance parallel systems are mainly scientists and engineers that require problem solving with an elevated computational cost, but that normally do not have an extended knowledge in parallelism. Auto-optimization techniques have been applied in different fields [Brewer 1994], specially in linear algebra routines [Chen et al. 2003], which make up the basic computing element in the resolution of many scientific problems.

One of the techniques used in the development of routines with auto-optimization capability uses the parametrization of the execution time model [Cuenca et al. 2005]. The models can be used to make some of the decisions that enable the routine to reduce its execution time. The methodology applied consists of identifying algorithm and system parameters to analyze the algorithm, both theoretically and experimentally, with the purpose of determining the influence of system parameter values in the selection of the best algorithm parameter values. The team that has developed this piece of work has experience in the application of automatic optimization techniques in Jacobi methods for the eigenvalue problem [Cuenca et al. 2001], LU and QR factorization [Song et al. 2010], and the problem of least squares in Toeplitz matrices [Alberti et al. 2004]. The application of the previous ideas to different algorithmic schemes is currently being worked on, such as dynamic programming schemes [Martínez et al. 2006], as well as traversal algorithms for solution trees [Giménez et al. 2006]. The present work is mainly concerned with auto-optimization of parallel algorithm schemes knowledge field, studying the application of algorithm optimization techniques in which an execution time parametrization is able to be done, centering on multicore systems.

In this work, analysis and application of automatic optimization techniques were proposed. The basic idea behind of the optimization consists of obtaining routine execution time in the form of: *t(s) = f(s, AP, SP)* [Cuenca et al. 2004], where s represents the size of the problem, SP system parameters, and AP algorithm parameters. The intention is to study techniques that allow the development of parallel routines that automatically tune to the characteristics of the parallel system where they will be executed. This way the users are provided with routines capable of running efficiently in the respective working environment, regardless of the characteristics of the system and the knowledge of the user about parallel programming.

Firstly, to test the developed methodology, the resolution of triangular systems is studied as an example. In these cases, the system parameters are costs of basic arithmetic operations and communication, and algorithmic parameters are block size and number of threads. Some methods used to estimate these parameters are analyzed. In execution time, the value of the algorithm parameters with which the problem is solved is obtained by estimating the values through the theoretical model, where values of system parameters are automatically substituted (number of threads and block size), previously obtained in an installation process.

2. Experimental Results

This section shows experimental results of the implemented algorithms for the auto-optimized to solve the triangular system problem. Experiments were done in one type of architecture, machine with 2 processors with 4 physical cores per processor. Including architecture Intel®Core 2 Quad, 3 GHz clock rate and 16 GB. An icc compiler and MKL library for LAPACK [MKL 2009] were used, are Intel proprietary providing support for the OpenMP API. Previously, a methodology to self-optimized parallel implementation has been presented. The code was run several times with different problem sizes (n) for different block sizes (sb) and varying the number of threads (nth) take this off through an executable code. A significant improvement in execution time was obtained and search is done to find the best SP parameters. A self-optimized parallel implementation has been taken as parameter to build the theoretical modeling of the execution time with the following:

$$t(n) = f(n, AP, g(n, AP)) = f(n, sb, nth, g(n, sb, nth)). \tag{1}$$

The optimum block size and number of threads are not a constant value. It depends on the platform and on the problem size. Thus, a good selection for each specific case of this block size and number of threads is important. For the system the routine has been executed for different reasonable block sizes (16, 32, 64, 128, 512 and 1024) and number of threads (2, 3, 4, 5, 6, 7 and 8) comparing the experimental with theoretical model.

In charts of the Fig. 1 (1)(2) it is a comparison performance between sequential implementation corresponding a call **DTRSV** BLAS routine and self-optimized parallel implementation with the best parameters. Basing on execution times observed, has been obtained automatically optimum number of threads and block size, finding the best combination of parameters for this platform. After varying the amount of threads in experiments, it has been observed that performance was improved by using a block size of 1024. This is justified by the cache-level exploring nature of multicore architecture which perfectly adjusts to the block size found. The charts in the Fig. 1 (3)(4) shows the comparison between the performance of self-optimized parallel and sequential implementation using block decomposition, having the same trend as the previous case, but with a potential increase due to the number of processors and cores of the tested node. When looking at the table for optimal experimented block size, it can be seen that task calculation time decreases with the number of used threads. This reduction in time occurs until we reach 8 threads when the number of cores is 8. Moreover, asynchrony provides algorithms with super scalability, unlike distributed memory, having in the management of performed tasks, a workload balance between thread execution.

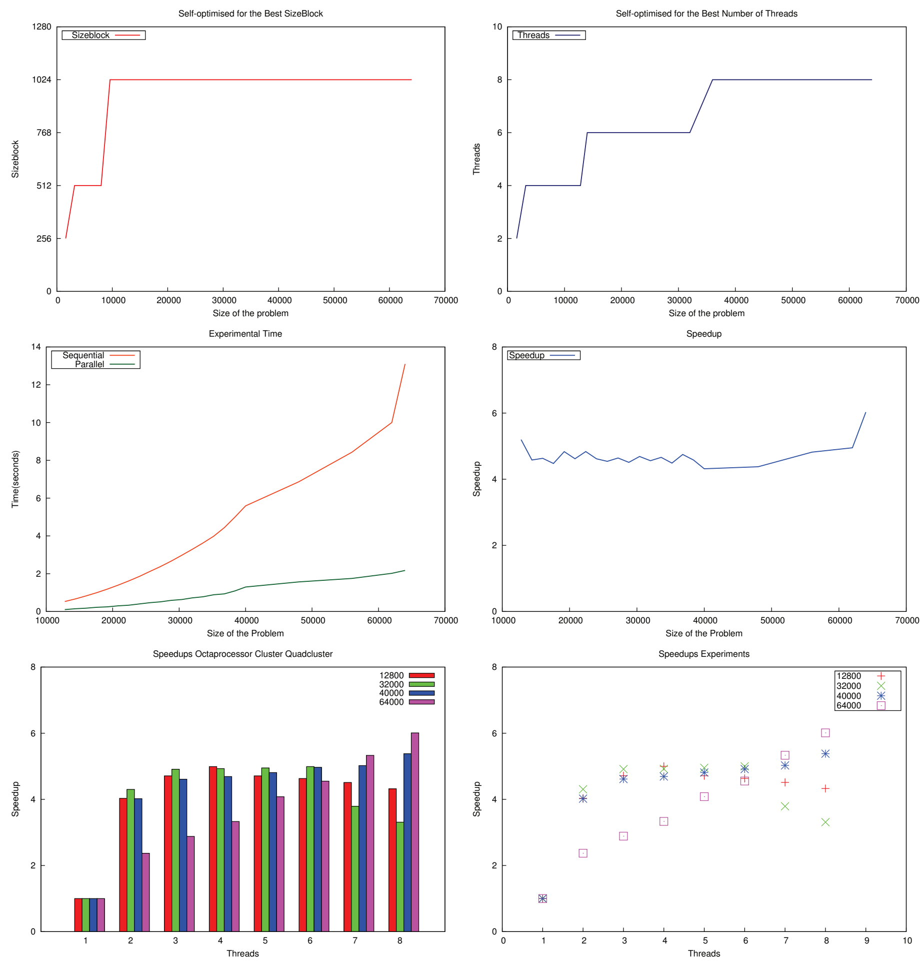

Figure 1. (1)(2) Self-Optimized for search the best block size and number of threads. (3)(4) Comparison between the performance the self-optimized parallel versus sequential routine. (5)(6) Comparison speedups with several cores.

3. Conclusions

This work has studied the usage of auto-optimization techniques in multicore systems using as example the resolution of triangular systems. For instance, the resolution of tridiagonal systems of equations have shown better performance through block decomposition. Thus, the intention is to validate the auto-optimization methodology for a wide range of parallel schemes.

However, it is know that the presented technique (i.e., the one used for the development of libraries with the capability of automatic optimization) could be adapted to other heterogeneous schemes, by simply varying the strategy of workload to processors. On the other hand, it could be interesting the usage previously acquired knowledge in parallel skeleton schemes. Moreover, users will be able to easily generate specific functions of the skeleton, just programming in sequential way. Thus, the parallel execution becomes transparent to the user, neither being necessary parallel programming nor knowledge in parallelism to be able to obtain reduced execution times.

References

Alberti, P., Alonso, P., Vidal, A., Cuenca, J., and Giménez, D. (2004). Designing polylibraries to speed up parallel computations. *International Journal of High Performance Computing Applications*, 1(1/2/3):75–84.

Brewer, E. (1994). *Portable High Performance Supercomputing: High Level Platform Dependent Optimization*. PhD thesis, Massachusetts Institute of Technology.

Chen, Z., Dongarra, J., Luszczek, P., and Roche, K. (2003). Self adapting software for numerical linear algebra and LAPACK for clusters. *Parallel Computing*, 29(11-12):1723–1743.

Cuenca, J., Giménez, D., and González, J. (2001). Modelling the behaviour of linear algebra algorithms with message passing. pages 282–289. IEEE Computer Society.

Cuenca, J., Giménez, D., and González, J. (2004). Architecture of an automatically tuned linear algebra library. *Parallel Computing*, 30(2):187–210.

Cuenca, J., Giménez, D., and Martínez, J. P. (2005). Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems. *Parallel Comput.*, 31(7):711–735.

Giménez, D., Cuenca, J., Martínez, J. P., Beltrán, J. M., Boratto, M., and Vidal, A. M. (2006). Parametrización de esquemas algorítmicos paralelos para autooptimización. In *XVII Jornadas de Paralelismo*.

Martínez, J. P., Almeida, F., and Giménez, D. (2006). Mapping in heterogeneous systems with heuristical methods. Workshop on state-of-the-art in Scientific and Parallel Computing, Sweden. Lecture Notes in Computing Science (LNCS).

MKL (2009). Intel Math Kernel Library for Linux. User's Guide. Available in: http://developer.intel.com.

Song, F., Ltaief, H., Hadri, B., and Dongarra, J. (2010). Scalable tile communication-avoid QR factorization on multicore cluster systems. *SC'10, ACM SIGARCH/IEEE Computer Society*, pages 13–19.