Applied Numerical Mathematics 104 (2016) 62–80

Contents lists available at ScienceDirect

Applied Numerical Mathematics

www.elsevier.com/locate/apnum

Scientific computations on multi-core systems using different programming frameworks

Panagiotis D. Michailidis [a,*], Konstantinos G. Margaritis [b]

[a] Department of Balkan, Slavic and Oriental Studies, University of Macedonia, 156 Egnatia Str., 54636 Thessaloniki, Greece
[b] Department of Applied Informatics, University of Macedonia, 156 Egnatia Str., 54636 Thessaloniki, Greece

ARTICLE INFO

Article history:
Available online 7 January 2015

Keywords:
Scientific computations
Linear algebra
Parallel computing
Multi-core
Parallel programming

ABSTRACT

Numerical linear algebra is one of the most important forms of scientific computation. The basic computations in numerical linear algebra are matrix computations and linear systems solution. These computations are used as kernels in many computational problems. This study demonstrates the parallelisation of these scientific computations using multi-core programming frameworks. Specifically, the frameworks examined here are Pthreads, OpenMP, Intel Cilk Plus, Intel TBB, SWARM, and FastFlow. A unified and exploratory performance evaluation and a qualitative study of these frameworks are also presented for parallel scientific computations with several parameters. The OpenMP and SWARM models produce good results running in parallel with compiler optimisation when implementing matrix operations at large and medium scales, whereas the remaining models do not perform as well for some matrix operations. The qualitative results show that the OpenMP, Cilk Plus, TBB, and SWARM frameworks require minimal programming effort, whereas the other models require advanced programming skills and experience. Finally, based on an extended study, general conclusions regarding the programming models and matrix operations for some parameters were obtained.

1. Introduction

Scientific computing is a collection of quantitative methods and tools used to develop and solve mathematical models of a variety of scientific problems using a computer system [65]. Numerical linear algebra is one of the most important quantitative methods for scientific computing. The basic computations of numerical linear algebra are matrix computations (such as vector and matrix addition, dot product, outer product, matrix transpose, matrix–vector product, and matrix product) and solutions of linear systems (such as the direct Gaussian elimination method and the iterative Jacobi method), which are used as kernels in many computational problems such as computational statistics [27] and combinatorial optimisation [34,67,70]. To satisfy the heavy computational requirements of these methods, large-scale matrix computations use towers of software infrastructure such as BLAS/LAPACK [7], ScaLAPACK [13], and PLAPACK [69] running on cluster computing platforms. Recently, modern high-performance computer systems have been introduced, which can be classified into three categories. The first, multi-core platforms, integrate a few cores (from two to ten) on the same integrated circuit chip (die) in an effort to speed up execution of computationally intensive methods. The second, many-core platforms or general-purpose graphics processing units (GPUs), which consist of a large number of cores (as many as several hundred), are specifically

* Corresponding author.
E-mail addresses: pmichailidis@uom.gr (P.D. Michailidis), kmarg@uom.gr (K.G. Margaritis).

http://dx.doi.org/10.1016/j.apnum.2014.12.008
0168-9274/

oriented to maximizing execution throughput for parallel applications [23]. The third, reconfigurable platforms based on field-programmable gate arrays, are becoming important, especially when higher performance/power computation ratios are desired. Along with the introduction of these platforms, software projects such as Parallel Linear Algebra for Multi-core Architectures (PLASMA) [1] have been developed for multi-core machines and Matrix Algebra on GPU and Multicore Architectures (MAGMA) [1] for GPU platforms. Recent comparisons of these platforms have shown substantial architectural and performance differences for several application areas [11,29,33,43,71]. Therefore, this paper focusses on the multi-core approach because it is the easiest way to speed up an application from the perspective of non-specialised developer. However, sequential implementation of linear algebra computations on a multi-core architecture will not improve performance because it cannot exploit the other cores that are available. Successful implementation of scientific computations on a multi-core machine can be achieved only through parallel (or multi-core) programming.

The fundamental programming question on a multi-core platform is how to decompose a problem into several subproblems and how to map these to cores with the goal of increasing performance. Moreover, programmers need to study and understand the hardware characteristics of the multi-core platform to write efficient parallel programs. For this reason, programming on multi-core processors is a more complex procedure than programming on sequential processors because application data in memory can be accessed by several entities called threads, which belong to the same program. For this reason some synchronisation between threads is necessary. Therefore, there is a need to bridge the gap between hardware and software applications to hide the hardware details from the programmers and enable them to write parallel programs with minimal programming effort. This has resulted in the introduction of several parallel programming models [23,41] which simplify the parallelisation of linear algebra computations and other related applications on multi-core computers. However, these models differ significantly in their parallel design principles, abstraction levels, semantics, and syntax. Some popular models are POSIX threads (Pthreads for short) [18], OpenMP [68], Intel Cilk Plus [37], Intel Threading Building Blocks (TBB for short) [40], SoftWare and Algorithms for Running on Multi-core (SWARM for short) [9], and FastFlow [2,3]. These models are based on a small set of extensions to the C programming language and involve a relatively simple compilation phase and a potentially much more complex runtime system.

Based on the various features and qualities of programming models, the question posed by programmers becomes: what is the appropriate parallel programming framework for implementing linear algebra computations on a multi-core system to achieve a balance between high programmer productivity (i.e., minimal programming effort) and high performance? Therefore, the contribution of this paper is a unified and systematic quantitative (i.e., performance-based) and qualitative (i.e., related to the ease of programming effort) comparison of all multi-core programming frameworks for implementing linear algebra computations based on simple parallelisation techniques. Finally, the authors believe that this work is important and interesting because this comparison may turn out to be very helpful for other programmers and scientists who are often faced with a variety of options for implementing projects.

The rest of the paper is organised as follows. In Section 2, related work is discussed. In Section 3, an abstract multi-core system architecture is presented along with all the reviewed parallel programming frameworks, and in Section 4, parallelisation issues in implementing linear algebra computations are considered. In Section 5, a performance and qualitative evaluation of the reviewed parallel programming models for parallelising linear algebra operations is described. Finally, Section 6 presents conclusions.

2. Related work

In the research literature, several studies have evaluated various parallel programming models on multi-core platforms. Most of this research has compared parallel programming models from the performance point of view. However, there has been little related work on comparing parallel programming models from the qualitative or productivity points of view.

The research studies based on performance evaluation of different multi-core programming models can be organised into four groups:

1. *Evaluation of a programming model to parallelise a specific problem*. Research studies [50] and [52] evaluated the parallelisation performance of the Gaussian elimination and LU factorisation algorithms using the OpenMP programming model. The parallelisation of Gaussian elimination [50] was based on the data parallel approach, whereas the parallel implementation of the LU decomposition [52] was based on the pipeline approach. Zuckerman et al. [72] evaluated the performance of the parallel matrix multiplication kernel using a high-performance M:N threading library, Microthread, and showed its efficiency with regard to the well-known Intel Math Kernel Library (IMKL). Runger et al. [62] presented a parallel optimised library for dense matrix multiplication on a multi-core platform. This implementation was based on a recursive approach and was compared with efficient libraries such as GotoBLAS, IMKL, and AMD Core Math Library (AMCL). Finally, Michailidis et al. [54] studied the performance of the pipeline approach in the OpenMP model for parallelisation of the Gauss–Jordan algorithm for solving systems of linear equations. The performance results were compared to the performance of two other naive parallel approaches, row block and row cyclic distribution.
2. *Evaluation of a programming model to parallelise a set of problems*. Michailidis et al. [53] presented performance results for the OpenMP model in the parallelisation of two important matrix computations, matrix–vector product and matrix multiplication. These parallelisations were based on a simple parallel data technique. Buttari et al. [16,17] developed the PLASMA library for implementing certain linear algebra operations on a multi-core platform using the title algorithms.