Future Generation Computer Systems 37 (2014) 64–75

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Architectural investigation of matrix data layout on multicore processors

Minwoo Kim, Won Woo Ro *

School of Electrical and Electronic Engineering, Yonsei University, 262 Seongsanno, Seodaemun-gu, Seoul, Republic of Korea

HIGHLIGHTS

- We propose the optimal matrix layout for data-intensive parallel matrix operations.
- We achieve reduced data latency through the optimized canonical data layout.
- Loop transformation using "tiling" is used for efficient parallel algorithms.
- The proposed optimized canonical data layout outperforms the existing block data layout.

ARTICLE INFO

Article history:
Received 16 April 2012
Received in revised form
21 August 2013
Accepted 13 October 2013
Available online 25 October 2013

Keywords:
Matrix data layout
Multicore architecture
Multilevel cache structure
Parallel algorithm
Tiling algorithm

ABSTRACT

Many practical applications include matrix operations as essential procedures. In addition, recent studies of matrix operations rely on parallel processing to reduce any calculation delays. Because these operations are highly data intensive, many studies have investigated work distribution techniques and data access latency to accelerate algorithms. However, previous studies have not considered hardware architectural features adequately, although they greatly affect the performance of matrix operations. Thus, the present study considers the architectural characteristics that affect the performance of matrix operations on real multicore processors. We use matrix multiplication, LU decomposition, and Cholesky factorization as the test applications, which are well-known data-intensive mathematical algorithms in various fields. We argue that applications only access matrices in a particular direction, and we propose that the canonical data layout is the optimal matrix data layout compared with the block data layout. In addition, the tiling algorithm is utilized to increase the temporal data locality in multilevel caches and to balance the workload as evenly as possible in multicore environments. Our experimental results show that applications using the canonical data layout with tiling have an 8.23% faster execution time and 3.91% of last level cache miss rate compared with applications executed with the block data layout.

1. Introduction

A large number of applications in various fields such as scientific calculation [1], signal processing [2], image processing [3], and network coding [4,5] use matrix operations as core procedures. Many matrix operations include data-intensive processes, especially when managing large matrices, and numerous studies have been conducted to optimize various matrix operations [6–17]. However, many of these methods have limitations when utilizing modern multicore environments. Several studies have proposed parallel algorithms but they did not provide experimental results [6,7,18], partly because multicore environments were not available when these studies were conducted. Other parallel algorithms were evaluated on multiprocessor systems [19,20]. Many studies have also focused on a limited number of the features that affect the performance of matrix operations, e.g., the matrix data layout and memory latency [10,21–23], work distribution techniques, and parallel algorithm [16,17,24–26]. In addition, the ideas proposed were verified using a number of theorems and lemmas, with proofs and related experimental results [27]. However, the numerical analyses did not scale well with the experimental results because the numerical analyses assumed that the matrices were accessed evenly in two directions, whereas experimental applications only access the matrices in a specific order, i.e., the row-major or column-major order.

Given the weaknesses of previous studies, we considered the multicore architecture and the application-optimized data layout used in data-intensive matrix operations. A previous study claimed [27] that the use of the tiling algorithm with a *block* data layout is the best way to enhance the translation lookaside buffer (TLB) and cache performance, but we argue that this no longer applies to multicore and multilevel cache architectures. Thus, we determined the best type of matrix layout for use in matrix

* Corresponding author. Tel.: +82 2 2123 5769; fax: +82 2 313 2879.
E-mail addresses: kenstars@yonsei.ac.kr (M. Kim), wro@yonsei.ac.kr (W.W. Ro).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.10.020

operations, such as matrix multiplication, *LU* decomposition, and Cholesky factorization. The individual operands in these matrix operations have unique access directions, so using a canonical data layout to minimize the spatial locality might help to optimize the data access latency.

The tiling algorithm is used to change the matrix access order by loop transformation to enhance the cache performance [28]. The tiling algorithm searches for an improved temporal locality based on frequent data reuse [29]. In the present study, we also exploited the advantages of the tiling algorithm for workload distribution on multiple processors, as well as for improving the cache performance in multicore environments. The efficient utilization of the multicore architecture is considered essential, which means that an effective load distribution is an important issue that affects the overall performance of applications.

Modern multicore processors are equipped with a private L1 cache, which may require specific sizes of tiles for optimum performance. In addition, the optimal tile size leads to a data prefetching effect, which further improves the cache performance. Thus, for multithreaded processing in a multicore environment, we focused on load balancing and optimal tile size to accelerate the large-volume data-intensive matrix operation process.

The main contributions of this study are summarized in the following bullet points.

- We performed a mathematical analysis of the cache performance with various matrix data layouts. We argue that the layout transformation improves the cache performance compared with the row-major data layout. We analyzed the layout transformation overheads and determined the optimal type of matrix data layout.
- We demonstrated the effect of the tiling algorithm, which transforms loop iterations to increase the cache performance based on a high data reuse rate. We tried to improve the performance of large-scale matrix operations with a canonical data layout based on an efficient tile access order.
- The multicore processor architecture and parallel processing can accelerate the speed of matrix operations using an efficient work distribution technique. The effect of data prefetching on the multilevel caches of multicore processors is also considered to be important for canonical data layouts.

Three matrix operations were tested in this study, which are widely used in scientific and engineering applications and as test benches: matrix multiplication (MM), *LU* decomposition (*LU*), and Cholesky factorization (CF) [7–17,27]. The processing of these operations is highly data-intensive because the operand matrices are large and the data are accessed repeatedly during multiple iterations. The execution time and multilevel cache performance were evaluated on multicore systems and the canonical layout was effective for all three applications. The benefits of multicore systems were also analyzed based on the first and intermediate level cache miss rates, as well as the last level cache miss rates.

2. Related work

Park et al. first showed that combining the block data layout with the tiling algorithm enhances the cache performance and reduces TLB misses [27]. They also showed the effectiveness of this approach by performing simulations of several applications with matrix operations. The SimpleScalar simulator was used to evaluate the cache performance and several actual single-core platforms were also used to measure the real-world execution time.

The data locality problem was first discussed in the 1990s by Wolf et al. in [30]. They introduced several loop transformation algorithms, including tiling, which was utilized in the present study. They also presented blocked algorithms, including matrix multiplication, and discussed their relationship with the cache performance in [31]. Later, variations of loop tiling were introduced, such as that in [29] by Parsa et al. and by Panda et al. [28].

In addition, some studies have tried to optimize the data layouts, including array layouts to enhance the cache performance [22,23,32–34]. In [23], Rivera et al. developed padding techniques, including inter- and intra-variable padding, for performing data layout transformations. Temam et al. developed a copying technique where the data in an array was copied to a temporary array, which delivered superior cache performance [34]. Manjikian et al. performed cache partitioning to logically divide the cache into several parts where each part contained a single array [32]. Kandemir et al. proposed the utilization of reuse vectors to restructure data [22]. This technique is used mainly for data layout transformations of arrays. Rather than utilizing a linear data layout, which is familiar from the user's perspective, nonlinear matrix layouts were proposed by Chatterjee et al. [33]. This form of data restructuring also enhances the memory performance.

Various studies have been conducted to accelerate matrix operations, such as matrix multiplication, *LU* decomposition, and Cholesky factorization [6–14,16,17]. A three-dimensional algorithm was proposed by Agarwal et al., which distributed the matrices in an efficient manner to achieve a balance between the loads on each processor [16]. A task-parallel algorithm was introduced by Hunold et al. [8], which separated work in a task-based manner to reduce the communication overheads between processors. Li et al. showed that parallel matrix multiplication can be accelerated by using a linear array with a reconfigurable pipelined bus system (LARPBS), which enhanced the communication speed between processors [9]. Chatterjee et al. proposed a recursive array layout to replace the row-major or column-major layout, which facilitated rapid matrix multiplication [10]. They presented five recursive layout functions for three parallel matrix multiplication algorithms. Van de Geijn et al. proposed a scalable implementation of matrix multiplication, which was also simpler than previous methods [11]. This method accelerated the speed of computation and it required less work space. Song et al. proposed a runtime dynamic task scheduling method for linear algebra algorithms, such as *LU* decomposition and Cholesky factorization [35], where they implemented a task-based library to automate the scheduling process of linear algebra algorithms using tiling. On the other hand, Ltaief et al. implemented parallel tile algorithms for two-sided linear algebra transformations such as Hessenberg and bidiagonal reductions [36]. Three scheduling methods, i.e., static, hand-coded dynamic, and SMP Superscalar, were implemented to achieve a parallel algorithm. However, this tile algorithm for two-sided transformations was unable to provide full reduction in a single step.

Stone proposed a parallel algorithm that solved a tridiagonal linear system of equations (including *LU* decomposition and Cholesky factorization) [6]. Later, van de Vorst introduced a parallel *LU* decomposition program based on a multiple instruction, multiple data (MIMD) machine with message passing [7]. Agarwal et al. proposed block algorithms for Cholesky factorization and a parallel scheme [13], where the performance was evaluated using an IBM 3090 Vector Facility. Buttari et al. exploited the parallelism of Cholesky factorization, *LU* decomposition, and QR factorization on multicore processors using the tiling algorithm [37]. Rothberg et al. used a heuristic remapping of the matrix blocks to improve the load distribution on each processor [17]. Ng et al. also overcame memory bottlenecks by implementing supernodes using a parallel algorithm for Cholesky factorization [14]. Recently, Venetis et al. presented an *LU* decomposition algorithm, which was solved using a many-core architecture with register tiling [12].

Kurzak et al. implemented parallel linear algebra operations on several parallel programming frameworks [24], where they