Parallel Computing 000 (2015) 1–15

Contents lists available at ScienceDirect

Parallel Computing

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

Recursion based parallelization of exact dense linear algebra routines for Gaussian elimination

Jean-Guillaume Dumas[a], Thierry Gautier[b], Clément Pernet[c,*], Jean-Louis Roch[b], Ziad Sultan[a,b]

[a] Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, CNRS, Inria
[b] Laboratoire d'informatique de Grenoble, Univ. Grenoble Alpes, CNRS, Inria
[c] Univ. Grenoble Alpes, Laboratoire de l'informatique et du parallllisme, Université de Lyon Inria

A R T I C L E I N F O

Article history:
Available online xxx

Keywords:
PLUQ decomposition
Parallel shared memory computation
Finite field
Dataflow task dependencies
NUMA architecture
Rank deficiency

A B S T R A C T

We present block algorithms and their implementation for the parallelization of sub-cubic Gaussian elimination on shared memory architectures. Contrarily to the classical cubic algorithms in parallel numerical linear algebra, we focus here on recursive algorithms and coarse grain parallelization. Indeed, sub-cubic matrix arithmetic can only be achieved through recursive algorithms making coarse grain block algorithms perform more efficiently than fine grain ones. This work is motivated by the design and implementation of dense linear algebra over a finite field, where fast matrix multiplication is used extensively and where costly modular reductions also advocate for coarse grain block decomposition. We incrementally build efficient kernels, for matrix multiplication first, then triangular system solving, on top of which a recursive PLUQ decomposition algorithm is built. We study the parallelization of these kernels using several algorithmic variants: either iterative or recursive and using different splitting strategies. Experiments show that recursive adaptive methods for matrix multiplication, hybrid recursive–iterative methods for triangular system solve and tile recursive versions of the PLUQ decomposition, together with various data mapping policies, provide the best performance on a 32 cores NUMA architecture. Overall, we show that the overhead of modular reductions is more than compensated by the fast linear algebra algorithms and that exact dense linear algebra matches the performance of full rank reference numerical software even in the presence of rank deficiencies.

1. Introduction

Dense Gaussian elimination over a finite field is a main building block in computational linear algebra. Driven by a large range of applications in computational sciences, parallel numerical dense LU factorization has been intensively studied for several decades which results in software of great maturity (e.g., LINPACK is used for benchmarking the efficiency of the top 500 supercomputers). As in numerical linear algebra, exact dense Gaussian elimination is a key building block for problems that are dense by nature but also for large sparse problems, where some intermediate computations also involve dense linear algebra:

* Corresponding author. Tel.: +33426233967; fax: +33472728080.
E-mail address: clement.pernet@imag.fr (C. Pernet).

http://dx.doi.org/10.1016/j.parco.2015.10.003
0167-8191/© 2015 Elsevier B.V. All rights reserved.

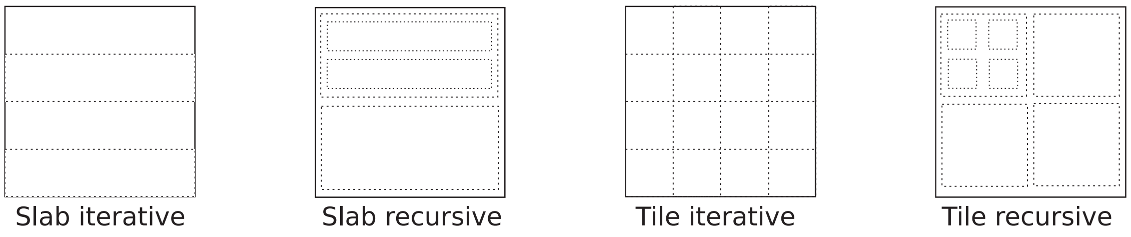

Fig. 1. Main types of block splitting

- sparse direct methods, may switch to dense Gaussian elimination when the fill-in becomes too large [30, Section 10.3];
- block iterative methods (like the block-Wiedemann or block-Lanczos algorithms) also require dense linear algebra to handle the block projections.

Recently, efficient sequential exact linear algebra routines have been developed [12]. The kernel routines run over small finite fields and are usually lifted over $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{Z}[X]$. They are used in algebraic cryptanalysis [3,15], computational number theory [27], or integer linear programming [18] and they benefit from the experience in numerical linear algebra. In particular, a key point there is to embed the finite field elements in integers stored as floating point numbers, and then rely on the efficiency of the floating point matrix multiplication `dgemm` of the BLAS. The conversion back to the finite field, done by costly modular reductions, is delayed as much as possible.

Hence a natural ingredient in the design of efficient dense linear algebra routines is the use of block algorithms that results in gathering arithmetic operations in matrix-matrix multiplications [7]. Those can take full advantage of vectorized SIMD instructions and have a high computation per memory access rate, allowing to almost fully overlap the data accesses by computations and hence deliver close to peak performance efficiency. A key feature of exact dense linear algebra, is that fast matrix multiplication algorithms, like Strassen [28] and Strassen–Winograd algorithms [12,16] can be used with no concern of numerical instability. The complexity improvement of these algorithms also gives a significant speed-up in practice [12]. In order to benefit from these sub-cubic time matrix multiplication algorithms, all other linear algebra computations, including Gaussian elimination need to reduce to it by block recursive algorithms. Hence, among the many variants of block algorithms, we only focus on the recursive ones.

In a previous work [11], we presented our first investigations on the parallelization of exact dense Gaussian elimination for multicore computers. The focus there was on exploring the variants of parallel exact Gaussian elimination algorithms (block iterative or recursive, using slabs or tiles, etc) and showed how and why the tile recursive variant outperforms the others.

We now focus in this manuscript on the building blocks on which these elimination algorithms rely. Our approach is to also apply recursive algorithms with a task based parallelization, for these building blocks. We explore six variants for the parallelization of the matrix multiplication (five recursive variants compared to the block iterative algorithm of [11]) and introduce a new hybrid parallel algorithm for the triangular system solve with matrix right hand side. We then recall the slab and tile recursive Gaussian elimination algorithms and present their performance using these new building blocks.

The scope of this study extends more generally to the problem of parallelizing any set linear algebra routines based on sub-cubic time matrix arithmetic, such as Strassen's $O(n^{2.81})$ algorithm. In particular, numerical linear algebra based on Strassen's algorithm (if numerical stability issues have been considered acceptable) should clearly benefit from most of its results. Related work on the parallelization of the sub-cubic numerical linear algebra include [1,2,6,24,25].

Our focus is on parallel implementations using various pivoting strategies that will reveal the echelon form, or the rank profile of the matrix [13,21]. The latter is a key invariant used in many applications such as Gröbner basis computations [15] and computational number theory [27].

As the PLUQ decomposition reduces to matrix-matrix multiplication and triangular matrix solve, we thus study several variants of the latter sub-routines as single computations or composed in the higher level decomposition.

The sub-routines used for the computation of parallel PLUQ decomposition are mainly:

- The `fgemm` routine that stands for Finite field General Matrix Multiplication and computes: $C \leftarrow \beta C + \alpha A \times B$ where A, B, C are dense matrices.
- The `ftrsm` routine that stands for Finite field Triangular Solving Matrix and computes: $A \leftarrow BU^{-1}$ where U is an upper triangular matrix, and B a dense matrix (or $A \leftarrow L^{-1}B$ where L is a lower triangular matrix, and B a dense matrix).
- The `PLUQ` routine that computes the triangular factorization $P, L, U, Q = A$, where P and Q are permutation matrices, U is upper triangular and L is unit invertible lower triangular.

Several schemes are used to design block linear algebra algorithms: the splitting in blocks can occur on one dimension only, producing row or column slabs [23], or both dimensions, producing tiles [5].

Algorithms processing blocks can also be either iterative or recursive. Fig. 1 summarizes some of the various existing block splitting obtained by combining these two aspects.