J. Parallel Distrib. Comput. 74 (2014) 2548–2560

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Looking back at dense linear algebra software

Piotr Luszczek[a], Jakub Kurzak[a], Jack Dongarra[a,b,c,*]

[a] University of Tennessee Knoxville, United States
[b] Oak Ridge National Laboratory, United States
[c] University of Manchester, United Kingdom

HIGHLIGHTS

- Growing gap of processor–memory communication affects linear algebra software.
- CPU/GPU hybridization creates challenging design combinations for legacy libraries.
- Loop restructuring methods include vectorization, chaining, blocking, and tiling.
- Scheduling techniques cover bulk-synchronous, SPMD, asynchronous, and dataflow.
- The effects of increased parallelism on numerical properties are briefly discussed.

ARTICLE INFO

Article history:
Received 17 July 2013
Accepted 28 October 2013
Available online 4 December 2013

Keywords:
Decompositional approach
Parallel algorithms
Dense linear algebra

ABSTRACT

Over the years, computational physics and chemistry served as an ongoing source of problems that demanded the ever increasing performance from hardware as well as the software that ran on top of it. Most of these problems could be translated into solutions for systems of linear equations: the very topic of numerical linear algebra. Seemingly then, a set of efficient linear solvers could be solving important scientific problems for years to come. We argue that dramatic changes in hardware designs precipitated by the shifting nature of the marketplace of computer hardware had a continuous effect on the software for numerical linear algebra. The extraction of high percentages of peak performance continues to require adaptation of software. If the past history of this adaptive nature of linear algebra software is any guide then the future theme will feature changes as well – changes aimed at harnessing the incredible advances of the evolving hardware infrastructure.

Published by Elsevier Inc.

1. Introduction

Over the decades, dense linear algebra has been an indispensable component of science and engineering. While the mathematical foundations and application methodology has changed little, the hardware has undergone a tumultuous transition. The latter precipitated a number of paradigm shifts in the way the linear algebra software is implemented. Indeed, the ever evolving hardware would quickly render old code inadequate in terms of performance. The external interfaces to the numerical software routines have undergone only minor adjustment which is in line with the unchanged mathematical formulation of the problem of solving a system of linear equations. The internal implementation of these interfaces was changing to accommodate drastic redesign of the underlying hardware technology. The internals have been modularized to ease the implementation process. These modules over time have become the building blocks of new generations of the numerical linear algebra libraries and made the effort more manageable in the long run. Over time, the number and functionality of these building blocks have increased but the delegation of responsibilities between various modules inside the software stack has been retained. The most recent increase in hardware parallelism further altered the established composition process of the modules by necessitating the use of explicit scheduling mechanism which needed to be handled manually in the composing code or externally with a software scheduler. The rapid transformation of computer hardware has not been ongoing and is expected to be continuing into the future. With it, the software will evolve further and our hope is that the design decision made in the past will allow for a smooth transition by reusing the tested and optimized libraries we have become accustomed to.

2. Motivation: plasma physics and electronic structure calculation

Computational experiments of self-sustaining fusion reactions could give us an informed perspective on how to build a device capable of producing and controlling the high performance [3].

* Corresponding author at: University of Tennessee Knoxville, United States.
E-mail addresses: luszczek@eecs.utk.edu (P. Luszczek), kurzak@eecs.utk.edu (J. Kurzak), dongarra@cs.utk.edu, julie@cs.utk, dongarra@eecs.utk.edu (J. Dongarra).

0743-7315/$ – see front matter. Published by Elsevier Inc.
http://dx.doi.org/10.1016/j.jpdc.2013.10.005

Modeling the heating response of plasma due to radio frequency (RF) waves in the fast wave time scale leads to solving the generalized Helmholtz equation. The time harmonic terms of effective approximations of the electric field, magnetic field, and distribution function as a time-averaged equilibrium satisfy the equation. The Scientific Discovery through Advanced Computing project (SciDAC) Numerical Computation of Wave Plasma-Interactions in Multi-dimensional Systems developed and implemented a simulation code that gives insight into how electromagnetic waves can be used for driving current flow, heating and controlling instabilities in the plasma. The code is called AORSA [21–23,19,5] and stands for All ORders Spectral Algorithm. The resulting computation requires a solution of a system of linear equations exceeding half a million unknowns [4].

In quantum chemistry, most of the scientific simulation codes result in a numerical linear algebra problem that may readily be solved with the ScaLAPACK library [6,8]. For example, early versions of ParaGauss [32,27,31,24] relied on diagonalization of the Kohn–Sham matrix and the parallelization method of choice relied on the irreducible representations of the point group. The submatrices diagonalize in parallel and the number of them depended on the symmetry group. When using one of ScaLAPACK's parallel eigensolvers it is possible to achieve speedup even for a Kohn–Sham matrix with only one block. A different use of the BLAS library occurs in UTChem [36]—an application code that collects a number of methods that allow for accurate and efficient calculations for computational chemistry of electronic structure problems. Both the ground and excited states of molecular systems are covered. In supporting a number of single-reference many-electron theories such as configuration-interaction theory, coupled-cluster theory, and Møller–Plesset perturbation theory, UTChem derives working equations using a symbolic manipulation program called Tensor Contraction Engine (TCE) [25]. It automates the process of deriving final formulas and generation of the execution program. The contraction of creation and annihilation operators according to Wick's theorem, consolidation of identical terms, and reduction of the expressions into the form of tensor contractions controlled by permutation operators are all done automatically by TCE. If tensor contractions are treated as a collection of multi-dimensional summations of the product of a few input arrays then the commutative, associative, and distributive properties of the summation allow for a number of execution orders, each of which having different execution rates when mapped to a particular hardware architecture. Also, some of the execution orders would result in calls to BLAS, which provides a substantial increase in floating-point execution rate. The current TCE implementation generates many-electron theories that are limited to non-relativistic Hartree–Fock formulation with reference wave functions but it is possible to extend it to relativistic 2- and 4-component reference wave functions.

3. Problem statement in matrix terms

Most dense linear systems solvers rely on a decompositional approach [33]. The general idea is the following: given a problem involving a matrix A, one factors or decomposes A into a product of simpler matrices from which the problem can easily be solved. This divides the computational problem into two parts: first determine an appropriate decomposition, and then use it in solving the problem at hand. Consider the problem of solving the linear system:

$$Ax = b \tag{1}$$

where A is a non-singular matrix of order n. The decompositional approach begins with the observation that it is possible to factor A in the form:

$$A = LU \tag{2}$$

where L is a lower triangular matrix (a matrix that has only zeros above the diagonal) with ones on the diagonal, and U is upper triangular (with only zeros below the diagonal). During the decomposition process, diagonal elements of A (called pivots) are used to divide the elements below the diagonal. If matrix A has a zero pivot, the process will break with division-by-zero error. Also, small values of the pivots excessively amplify the numerical errors of the process. So for numerical stability, the method needs to interchange rows of the matrix or make sure pivots are as large (in absolute value) as possible. This observation leads to a row permutation matrix P and modifies the factored form to:

$$PA = LU. \tag{3}$$

The solution can then be written in the form:

$$x = A^{-1}Pb \tag{4}$$

which then suggests the following algorithm for solving the system of equations:

- Factor A according to Eq. (3)
- Solve the system $Ly = Pb$
- Solve the system $Ux = y$.

This approach to matrix computations through decomposition has proven very useful for several reasons. First, the approach separates the computation into two stages: the computation of a decomposition, followed by the use of the decomposition to solve the problem at hand. This can be important, for example, if different right hand sides are present and need to be solved at different points in the process. The matrix needs to be factored only once and reused for the different right hand sides. This is particularly important because the factorization of A, step 1, requires $O(n^3)$ operations, whereas the solutions, steps 2 and 3, require only $O(n^2)$ operations. Another aspect of the algorithm's strength is in storage: the L and U factors do not require extra storage, but can take over the space occupied initially by A. For the discussion of coding this algorithm, we present only the computationally intensive part of the process, which is step 1, the factorization of the matrix.

Decompositional technique can be applied to many different matrix types:

$$A_1 = LL^T \qquad A_2 = LDL^T \qquad PA_3 = LU \qquad A_4 = QR \tag{5}$$

such as symmetric positive definite (A_1), symmetric indefinite (A_2), square non-singular (A_3), and general rectangular matrices (A_4). Each matrix type will require a different algorithm: Cholesky factorization, Cholesky factorization with pivoting, LU factorization, and QR factorization, respectively.

4. Introducing *LU*: a simple implementation

For the first version, we present a straightforward implementation of LU factorization. It consists of n-1 steps, where each step introduces more zeros below the diagonal, as shown in Fig. 1.

Tools often used to teach Gaussian elimination include MATLAB and Python. They are scripting languages that make developing matrix algorithms very simple. The notation might seem very unusual to people familiar with other scripting languages because it is oriented to process multi-dimensional arrays. The unique features of the language that we use in the example code are:

- Transposition operator for vectors and matrices: ′ (single quote)
- Matrix indexing specified as:
  - Simple integer values: A(m, k)
  - Ranges: A(k:n, k)
  - Other matrices: A([k m], :)