GAUSS ELIMINATION ALGORITHMS FOR MIMD COMPUTERS

M. Cosnard+, M. Marrakchi+, Y. Robert+ and D. Trystram#
+ CNRS, Laboratoire TIM3, BP 68, 38402 St Martin d'Hères Cedex, France
# Ecole Centrale PARIS, 92295 Chatenay Malabry Cedex, France

ABSTRACT: This paper uses a graph-theoretic approach to analyse the performances of several parallel variations of the Gaussian triangularization algorithm on an MIMD computer. Dongarra et al. [DGK] have studied various parallel implementations of this method for a vector pipeline machine. We obtain complexity results permitting to select among these parallel algorithms.

INTRODUCTION

The most commonly used algorithm to solve linear systems of equations on sequential computers is the well-known Gaussian elimination method. Six different versions for a vector pipeline machine have been considered in [DGK]. Three implementations where data are accessed columnwise have been discussed in detail, each of them corresponding to a given permutation of the loop indices i,j,k of the sequential algorithm: namely the SAXPY version (form kji), the GAXPY version (form jki) and the DOT version (form ijk). In this paper, we deal with the design of MIMD versions of these algorithms (see [Fly], [GP], [HI] and [HB] for a classification of parallel computers).

A parallel MIMD version of the Gaussian elimination algorithm with partial pivoting has been discussed in [Kum] and [LKK], and an implementation of the LDL\textsuperscript{T} decomposition algorithm in [KK]. The performance analysis is based on the task graph model presented in [Kum]. Informally, algorithms are splitted into elementary tasks, whose execution ordering is directed by precedence constraints. The task graph model which can be constructed directly from these precedence constraints, is the basic tool of our theoretical analysis. Together with MIMD versions of the [DGK] algorithms, we analyse a modified version of the KJI-SAXPY and the Doolittle algorithm.

We assume a system which is capable of supporting multiple instruction streams executing independently and in parallel on multiple data streams [GP], [HI], [HB] and [Sch], and that there are means to impose temporal precedence constraints between the tasks of the implemented algorithms [KK]. Moreover we suppose that each processor can perform any of the four arithmetic operations in an unit of time and that there are no memory conflicts nor data communication delays. Throughout the paper, p denotes the number of processors, and \( E_p \) is the efficiency of the parallel algorithm under study. When triangularizing a dense \( n \times n \) matrix, we set \( p = \alpha n \), with \( \alpha < 1 \), in order that processors communications costs do not overcome arithmetic [Saa]. Elementary tasks will be of length \( O(n) \), so that synchronization and data communication do not predomine.

A model where communication delays are neglected could appear unrealistic. However, we deal with pointwise methods for which the elements of a given matrix are accessed columnwise or rowwise, depending on the storage mechanism and programming environment. Hence we can assume a stride-one accessing of data as prevalent in vector machines and cache-based architectures, so that data loading and unloading can be pipelined, and overlapped with arithmetic. In the following we assume that data is accessed columnwise, to be close to a FORTRAN programming style environment.

This work has been supported by the CNRS through the GRECO C\textsuperscript{3}. 
If data loading and unloading cannot be overlapped with arithmetic, communications should not be neglected any longer. It is shown in [CMRT] that block methods should be used in such a case.

Most often, pivoting of rows or columns is used for stability reasons. We shall not consider here the overhead due to pivoting, taking into account the remarks of [DGK] who state that the pivoting procedure is just some additional overhead and does not significantly affect the performance. However, partial pivoting could be very easily included in our analysis: simply consider that comparing and interchanging two reals takes one unit of time, which only modifies the granularity of the tasks.

Once the constraints defined, the first step in the parallelization of a method is the definition of the elementary tasks and their precedence graph. This graph shows the temporal dependency of the operations of the algorithm. The tasks are then assigned to the available processors according to the precedence graph. Throughout the paper, the relation $T << T'$ denotes the precedence constraint and means that task $T$ is to be completed before task $T'$ can start its execution [Kum], [LKK].

THE ALGORITHMS

Five sequential algorithms will be considered which are variations of Gaussian elimination. Like this one, they all need the same number of arithmetic operations: $2n^3/3+O(N^2)$

(A) { Generic Gaussian elimination algorithm. Form KJI - SAXPY of [DGK] }

- For $k := 1 \text{ to } n-1$
  - $T_{kk} : < \text{For } i := k+1 \text{ to } n$
    - $a_{ik} := - a_{ik}/a_{kk}$
    - $n-k$ arithmetic operations
  - $T_{kj} : < \text{For } j := k+1 \text{ to } n$
    - $T_{kj} : < \text{For } i := k+1 \text{ to } n$
      - $a_{ij} := a_{ij} + a_{ik}a_{kj}$
      - $2(n-k)$ arithmetic operations

Figure 1: Precedence graph of Gaussian elimination (form KJI)