**A parallel implementation of the Karmarkar algorithm
using a parallel Linear Algebra library**

Johannes G.G. van de Vorst
Koninklijke/Shell-Laboratorium, Amsterdam (Shell Research BV)
P.O. Box 3003, 1003 AA Amsterdam, The Netherlands

Abstract

An important decision in the construction of a parallel Linear Algebra library is the choice of the process structure. The structure that is presented in this article allows for simple functional specifications of the processes and for their compositionality. In the functional specification of a process, matrices and vectors occur as ordinary variables and nothing is said about the communication behaviour of the instances of a process. The representations of the matrices and vectors are distributed across the process instances. A number of distributions are introduced. All library processes conform to the same representation details and communication conventions. The library processes can be composed sequentially, without requiring global synchronisation between process calls. As an example, a parallel implementation of the Karmarkar algorithm is briefly discussed.

Key words: Parallel computing - Linear Algebra - Library - Interfaces -
Karmarkar

0. Introduction

In many application areas, the solution of Linear Algebra problems constitutes the main computational task. With the aims of speeding up these computations and of bringing within reach the solution of larger problems, a number of parallel Linear Algebra processes have been constructed at KSLA. They are intended to be executed on an MIMD (Multiple Instructions, Multiple Data) machine. Such a machine consists of a number of processors, each with its own

local memory. The processors are interconnected by a communication network, and do not have a global clock.

In order to ease the use of the processes in various applications, the processes are collected in a Linear Algebra library. The library processes are called in a sequential way. Each call gives rise to a number of instances of the process that execute in parallel.

A number of library conventions have been adopted, to which all library processes adhere. Two important conventions concern the ways in which matrices and vectors are distributed across process instances and the ways in which process instances can communicate.

This article deals with full matrices only. A separate effort made at KSLA by R.H. Bisseling, A.F. van der Stappen, A. Sweeney and the author is aimed at the construction of a sparse library, with a structure similar to the one described in this article.

Two of the main design objectives of the Linear Algebra library are: simplicity of the functional specifications and compositionality of the processes.

The functional specifications of the library processes are given in such a way that they do not reflect any details of the parallel implementation. They contain nothing about communication between process instances. A prerequisite in the construction of the parallel library is that the functional specifications of the Linear Algebra processes can be readily understood by a layman in the field of parallel computing. Compositionality is reached by imposing strict and uniform interface conventions on the library processes.

The library assumes a $Q \times Q$ mesh communication network, with $Q > 0$. This is not a strong assumption, because meshes are physically embedded in a large number of networks. Examples are: a completely connected network, a hypercube, and a torus. Further, it is assumed that the order in which messages are received equals the order in which they were sent, and that messages do not get lost.

The examples in this article are given in the programming language occam2 [2].

Section 1 deals with notational issues. In section 2, the process structure is introduced. Section 3 contains a description of the functional specification of the library processes. In section 4, the library constants and functions are given. Section 5 deals with the naming conventions of the variables. Some communication conventions between the instances of any single process are given

in section 6. Section 7 gives an example of the use of the processes in the Karmarkar algorithm for solving Linear Programming problems. The conclusions are given in section 8.

1. Notation

$v: m$	: v is a vector of length m .
$X: m \times n$	: X is a matrix with m rows and n columns.
$\text{rank}(X)$	: the rank of X .
$\text{diag}(d)$	: the diagonal matrix with diagonal d .
P	: the number of processors, $P = Q^2$.
max.B.size	: $P * \text{max.B.size}$ is the maximum number of elements of a matrix.

2. The process structure

Each Linear Algebra operation is performed by the parallel composition of $P = Q^2$ instances of a single Linear Algebra process. Therefore, the operands, i.e. the matrices and the vectors, are distributed across the process instances. In order to avoid unnecessary communication overhead, the representations of the matrices and vectors remain distributed between process calls: they are passed as parameters. Input and output of matrices and vectors to and from a distributed representation is performed explicitly by a set of i/o processes.

Instances of different library processes never communicate. Therefore, two different kinds of process instance interfaces exist. There are interfaces between the instances of a single process which are composed in parallel, and there are interfaces between the instances of (possibly different) processes which are composed sequentially. The first kind deals with communications, the second kind deals with parameter passing. Because instances of different library processes can only pass parameters, the parameter mechanism provides the only kind of interface in the program specifications.

This restriction provides the programmer with the possibility to give simple functional specifications, and the possibility to understand and verify

the library processes in isolation.

The parallel Linear Algebra processes only require that the order in which messages are received equals the order in which they were sent and that no messages get lost. The communication mechanism can be either synchronous or asynchronous. Hence, the library can be implemented both on synchronous and asynchronous message passing machines.

Another consequence of the preservation of messages and their order is that processes can be composed in a sequential way without requiring a global synchronisation prior to the activation of the process instances.

Example: P instances of the Linear Algebra process `Lin.sol` are composed in parallel. Each instance of the process `Lin.sol` activates three subprocess instances: first `LU.deco`, then `Back.subst.L`, and lastly `Back.subst.U`. The instances of `Lin.sol` do not synchronise prior to each activation of the subprocess instances.

(end of example)

The processors in the mesh network have Cartesian coordinates (p_i, p_j) , with $0 \leq p_i, p_j < Q$. Each call of a Linear Algebra process gives rise to $P = Q^2$ instances of that process. The instances will execute in parallel on different processors. They inherit the coordinates from the processors.

3. Functional specifications

In the library, the matrices and vectors are called 'objects'. An object can be thought of as an ordinary matrix or vector variable, as one uses them in Linear Algebra or other parts of mathematics. In correctness arguments, the objects are used as auxiliary variables. The objects are not declared in the program text, because they are represented in a distributed way.

The functional specification of a Linear Algebra process is given in terms of a precondition and a postcondition, describing the effect of the parallel composition of its P instances. The objects mentioned above are used in them.

All variables which are declared in the program text are called 'instance variables'. A collection of instance variables may represent an object. Examples of other instance variables are: process instance coordinates and scalars.

Instance variables used for the distributed representation of an object never occur in a functional specification. In such a specification, an abstraction is made of distributed representations. Only the kind of distribution is mentioned. This is sufficient to indicate which processes are compatible. Other instance variables may occur in a functional specification.

Since instances of different processes do not communicate, and the way in which the instances of a single process can be composed in parallel has been prescribed (section 2), the communication behaviour of a process instance does not play a rôle in the functional specification of the processes.

4. Library constants and functions

The two library constants that are of importance are the sets Distributions and Objects. Distributions is the set of all matrix and vector distributions, Distributions = {Grid, Diag}. Objects denotes the set of matrix and vector variables that occur in the library.

The function `distr: Objects -> Distributions` is used in the functional specification of a Linear Algebra process. It gives the distribution of an object. When passing parameters, it must be ensured that in both processes involved the same assumptions on the distributions of the matrix and vector parameters are made.

5. Naming conventions for variables

<code>pi, pj:</code>	process instance coordinates.
<code>b, c, v:</code>	vectors.
<code>b.part, c.part, v.part:</code>	instance variables, used in the distributed representation of <code>b, c, v</code> .
<code>B, C, X:</code>	matrices.
<code>B.part, C.part, X.part:</code>	instance variables, used in the distributed representation of <code>B, C, X</code> .

Note that `b, c, v, B, C`, and `X` are objects in the sense of section 3.

6. Communication conventions

In accordance with the assumption that the library assumes a mesh communication network, the process instances only perform nearest neighbour communications. Each Linear Algebra process can have up to eight channel parameters, two for each nearest neighbour, in opposite directions. The software communication network is a mesh of size $Q \times Q$.

In a mesh process instance network, the standard (north, east, south, west) orientation is used. The channel parameters of the corresponding process are named: from.north, to.south, from.south, to.north, from.west, to.east, from.east, to.west. The process instances have coordinates (p_i, p_j) , with $0 \leq p_i, p_j < Q$.

Example of the functional specification of a process whose instances communicate in a mesh; the process has eight channel parameters:

```
PROC B.mult.BT
  (VAL INT pi, pj, m, n,
   CHAN OF ANY from.north, to.south, from.south, to.north,
               from.west, to.east, from.east, to.west,
   [ ]REAL32 B.part, C.part)

pre:  B: m x n  ^  distr (B) = Grid.

post: C: m x m  ^  C = B * BT ^  distr (C) = Grid.

(end of example)
```

7. Example: the Karmarkar algorithm

The Karmarkar algorithm [1] provides a way to solve Linear Programming problems. In this section it is made clear how a parallel implementation of the Karmarkar algorithm can be made, using the Linear Algebra library. First a brief explanation of some relevant aspects of the algorithm will be given, and after that it is shown how the most important process Next.dir is constructed from the library processes.

Given is a matrix $A: m \times n$, with $n \leq m$, of full column rank. Also given are two vectors, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The problem is to determine x , $x \in \mathbb{R}^n$ such that

$$Ax \leq b$$

$$\wedge c^T x = (\text{MAX } y: y \in \mathbb{R}^n \wedge Ay \leq b: c^T y).$$

Note: inequalities of vectors, such as $Ax \leq b$, have to be interpreted element-wise.

The Karmarkar algorithm solves this problem in an iterative way, where in each iteration a given current iterate u , $Au < b$, is used to calculate a new one. Each iteration consists of two steps: the major one is the computation of the direction e , $|e| = 1$, in which the new iterate will lie, and the minor one is the computation of the distance α between the current and the new iterate, $v = u + \alpha e$. We focus on the computation of the vector e , defined by $c = (DA)^T D A e$. D is a diagonal matrix that depends upon u . The computation of e is performed by the parallel process `Next.dir`.

The process heading and the functional specification of `Next.dir` are:

```
PROC Next.dir (VAL INT pi, pj, m, n,
               CHAN OF ANY from.north, to.south, from.south, to.north,
               from.east, to.west, from.west, to.east,
               [ ]REAL32 X.part, d.part, c.part, e.part)

pre:      A: m x n  ^  n ≤ m  ^  rank (A) = n  ^  X = A
      ^  D: m x m  ^  D = diag (d)  ^  rank (D) = m
      ^  c: n
      ^  distr (X) = Grid  ^  distr (d) = Diag  ^  distr (c) = Diag.

post:     e: n  ^  c = (DA)^T D A e  ^  distr (e) = Diag.
```

The process text of `Next.dir` is given below (the process instance coordinates `pi`, `pj` and the channel parameters have been omitted):

```
PROC Next.dir (... , X.part, d.part, c.part, e.part)
```

```
  [max.B.size]REAL32 XTX.part:
```

```
  {X = A}
```

```
  SEQ
```

```
    D.mult.X (... , d.part, X.part)
```

```
    {X = DA}
```

```
  XT.mult.X (... , X.part, XTX.part)
```

```
  {XTX = (DA)TDA}
```

```
  Lin.sol (... , XTX.part, c.part, e.part)
```

```
  {c = (DA)TDAe}
```

```
:
```

8. Conclusions

The Linear Algebra library discussed in this article has been implemented at KSLA by R.H. Bisseling and the author; the Karmarkar algorithm was implemented by P. Rossi. The programming language is occam2 (see [2]) and the library is executed on a mesh network of Inmos T800 transputers. Among others, the library contains processes for matrix multiplication, the solution of a set of linear equations, the projection of a point onto a hyperplane, and the solution of the least squares problem. It is expected that more processes will be added in the future. The library still has to be made robust with respect to singular matrices.

An example of the use of the Linear Algebra library has been given in this article. The processes have simple functional specifications and can be composed sequentially.

The functional specifications of the Linear Algebra processes can be readily understood by a layman in the field of parallel computing. Such a specification does not show details of a parallel implementation or of communication behaviours.

Matrices and vectors occur in the functional specifications as ordinary mathematical variables. Their representations are distributed across a number

of process instances. A number of distributions have been introduced for reasons of efficiency.

The compositionality of the processes has proved to be essential in the construction of the library itself. Many processes include calls to other processes in the library. It is expected that this compositionality will also facilitate the use of the library in various application areas. The envisaged application areas include the solution of Linear Programming problems and the solution of fluid flow problems.

The software communication graphs that are used by the library processes can be mapped directly onto a $Q \times Q$ mesh communication network. The communication mechanism (synchronous or asynchronous) does not influence the semantics of the library processes. These two features enhance the portability of the library.

References

- [1] Adler I, Resende MGC, Veiga G (1986) An implementation of Karmarkar's algorithm for linear programming. Operations Research Centre, ORC 86-8
- [2] Burns A (1988) Programming in occam2. Addison-Wesley, Wokingham