Ada programming language and specifying, modeling and distributed computing for the implementation of the discrete singularities methods

Mishchenko Victor
V.N. Karazin Kharkov National University
V.O.Mischenko@univer.kharkov.ua

Abstract

An elementary technique of debugging and optimizing Ada programs (using one personal computer (PC)) designed for subsequent distributed computing in a random local area network is given in this report. This is the most available and inexpensive approach to the problem of increasing the available dimensionality limit in computational problems, when we do not need to make the limit greater for more than one order. The testing was conducted by the example of the linear systems generation and solution subsystem that are typical for the application of the discrete singularities methods (DSM) to the modeling of physical processes.

1. Introduction

Computer clusters have become very popular for the past decade [1]. However, many investigators and, quite often, whole research teams do not have access to an appropriate cluster project or unable to get connected to any of them via the Internet. Under these circumstances the problem of a short-term transformation of a local access network having some other purpose into a cluster system is an actual problem from the practical point of view. However, administrators of such networks, for example, academic ones, usually provide machine time for calculations only when the network is not used for its conventional purposes. It is possible to make use of such opportunities only when there are debugged portable programs to be installed on the network computers. One has to manage the efficiency of parallel execution of such programs at the expense of their inherent parameters.

2. The Problem Statement

The objective of this work is to describe and test a simple technique on the basis of the Ada compiler, which can be offered as a tool for the development of portable and adjustable programs divided into parts to be installed on the local network computers and used for computational purposes typical for a certain type of computational methods. The aim of this work was to complete all the stages by a transparent but real example typical for a computational methods type known as DSM. The subsystem for generating and solving linear equations systems was selected for software of numerical experiments in electrodynamics model problems on the basis of discrete singularities methods (DSM). Mathematical aspects of the problem were discussed in [2].

3. Method and Materials

The Author has offered a three-stage technique for the development of portable distributed computational programs. Any computer can serve for development purposes, and regular Ada compiler can be used as a software tool. They make it possible to generate and debug the entire program (stage 1). There is no need to change the program when distributing it among different computers (all we have to change is just formats of main programs consisting of several lines). File data formats and file access path structures will remain the same as well. The optimal (i.e. even) duty of the impromptu cluster is calculated on the basis of several test executions of the distributed program in a specific network (stage 2) using a parallel computing simulation program (stage 3).

This technique was tested by a real example.

An application of DSM leads to one or many tasks of the next character. It needs to process an array which elements may be generated independently. The processing of a part of the array depends on results of processing of some other parts. We illustrate our technique by the real example when array processing is a solving of a linear system.

Stage 1. Firstly to elaborate tools library for the decomposition of a linear equations system. Strips name parts of system matrix:

```
    with Types, Distributed_SLAE;
    use  Types, Distributed_SLAE;
package Matrix_Strips is
    --     …
    type Matrix_Strip is private;
    function Init
    ( P: Line_Index; N: Colon_Index; S:
    Machine_Index:= 0 )
    return Matrix_Strip;
    procedure Change
     (MS: in out Matrix_Strip; S: Machine_Index );
    procedure Strip_Check
     ( MS: Matrix_Strip );
```

```
private
    type M_Strip is tagged
    record
           S: Machine_Index;
           A: Matrix;-- ( Strip_Line_Index, 1..N );
    end record;
    --      …
end Matrix_Strips;
```

(It is Ada specification of the key unit. The whole library contains nine library units and twelve compilation units those are library unit bodies or subunits.)

Secondly to code tasks those realize the matrix generation and linear system solving. We construct solving method as one of numerous modifications of so-called cell version of well-known LR-algorithm:

```
    with  Matrix_Strips;
     use  Matrix_Strips;
package SLAE_Distribution is

   task type Slae_Generator is
   end Slae_Generator;

   M: constant Machine_Index:= Matrix_Strips.M;
   subtype M_Index is Index range 1..M;

   task type Lr_Processor (S: M_Index) is
   end Lr_Processor;

end Slae_Distribution;
```

(This package have the body and two subunits.)
Let design the primary main program:

```
with Slae_Distribution, Text_IO, Ada.Integer_Text_IO;
 use Slae_Distribution, Text_IO, Ada.Integer_Text_IO;
procedure Model_Of_Par5_Program is
   procedure Start (S: M_Index) is -- M=5 in the test
      Lr: Lr_Processor (S);
   begin Put(" #");Put(S,1);
   end;
begin
   Put_Line("Model Of Parallel Compact LR-algorithm”
   &”Rev: v.1. ");
   for I in M_Index
   loop
      Start(I);
   end loop;
end Model_Of_Par5_Program;
```

It remains to debug this program. Note that it is possible mutual tasks blocking when the program is executed on one processor. To eliminate those conflicts we use a modeling of file data.

Stage 2. Firstly it needs to elaborate command files for physical distribution of executed files and for running them on computers of used net. Then let us substitute for each copy of *Model_Of_Par5_Program* by one of two variants of main program:

```
with Slae_Distribution, Text_IO, Ada.Integer_Text_IO;
 use Slae_Distribution, Text_IO, Ada.Integer_Text_IO;
procedure Parallel_Generator_Program is
   procedure Start (S: M_Index) is
      Lr: Slae_Generator;
   begin Put(" #");Put(S,1);
   end;
begin
   Put_Line("Matrix for Parallel Compact LR-algorithm”
   &”Rev: v.2. ");
      Start (S_Value); -- S_Value is 0 or –1, -2, ..
end Parallel_Generator_Program;
```

or

```
with Slae_Distribution, Text_IO, Ada.Integer_Text_IO;
 use Slae_Distribution, Text_IO, Ada.Integer_Text_IO;
procedure LR_Gauss_Parallel_Program is
   procedure Start (S: M_Index) is
      Lr: Slae_Generator;
   begin Put(" #");Put(S,1);
   end;
begin
   Put_Line("Parallel Compact LR-algorithm Rev: v.2.
");
      Start (S_Value); -- S_Value is 1 or 2, .. M
end LR_Gauss_Parallel_Program;
```

Parallel execution of our distributed program provides us with time data. It makes possible to determine coefficients of the execution mathematical model.

Stage 3. We may optimize parallel execution of our distributed program for time by means of simulation. In our case simulation implies substitution of bodies of calculating program units for calls to a monitor for a time counting. Variants of structure for simulation programs written in Ada are well known [3]. Yet the book [3] uses the language Ada-83. Now for the actual language Ada-95 we have to replace the task mechanism of simulation for the protected unit one. The target of our optimization is to load computers uniformly. In common case the number of *Parallel_Generator_Program*-s is one of the mains optimizing parameters.

Let take a real example. Let system dimension N equals 2400. The time of the system solving by 7 computers has been roundly two times greater then the time of the solving by 4 computers. But the time of the system solving by 11 computers was the same (6 computers of the type of *LR_Gauss_Parallel_Program* were in long waiting for the *Parallel_Generator_Program* computer). How many computers of *Parallel_Generator_Program* type may be added to eliminate this waiting? The answer is hot evident. We can obtain it by simulation.

We obtain the simulation program from our old good *Model_Of_Par5_Program* by substitution for Read/Write operations. New ones call protected operations of the monitor which model current time for all tasks of our program so, as they are executed on different computers. So we don't need for results of computing apart from modeling time the actual dimension has been changed. Therefor the time of simulated program execution can be done available.

4. Conclusions

An 'economical' version of the cluster approach to labor-intensive computations available to all investigators which develop their own software, including those who can have only short-term and restricted access to computer networks, has been offered.

This work was aimed at and will be used for the easing of resource restrictions with which many students and employees of Karazin Kharkov National University had to face when conducting computer-oriented experiments on the basis of DSM. At the same time, the work can be of help for those who need to increase the available dimensionality limit for about one order in computational problems at a low price.

It is evident that it is possible to use the reported results for training purposes since the suggested approach to the implementation of parallel computations enables us to demonstrate their particular features in computational clusters visually, using a common computer classroom.

Wide potentialities of the Ada programming language have been used. Its advantages lying outside the generally recognized fields of its application (namely, large-scale projects, built-in systems, real-time software, etc.) have been demonstrated.

It is possible to extend this approach into other types of parallel computations. A study of whether it is possible to use this technique to promote the software development for special-purpose cluster systems (for example, MPI-operated ones) is of importance.

References

[1] Voyevodin V.V., Voyevodin Vl.V. *Parallel computations, BHV-Petersburg, Saint Petersburg, 2002.*

[2] Mishchenko V.O. About modeling of electromagnetism on the basis of discrete singularities methods for solving hyper-singular equations. *Transactions of the International Conference on Computational Mathematics (ICCM-2004). Part II.* Novosibirsk: IVMiMG Publishing House, Russian Academy of Sciences, 2004, p. 555-560.

[3] Vasilesku, Eugen N. *Ada Programming with applications, Allyn and Bacon Inc., Boston-London-Sydney-Toronto, 1987 (Russian translation: "Mir", Moskow, 1990).*