Parallel Implementation of Interval Analysis for Equations Solving

Yves Papegay, David Daney, and Jean-Pierre Merlet

INRIA Sophia Antipolis – COPRIN Team,
2004 route des Lucioles, F-06902 Sophia Antipolis, France,
Yves.Papegay@sophia.inria.fr

Abstract. ALIAS is a C++ library implementing interval analysis and constraint programming methods for studying and solving set of equation over the reals. In this paper, we describes a distributed implementation of the library.

1 Introduction

ALIAS[1] is a C++ library of algorithms for solving set of equations and related problems, e.g finding an approximation of the real roots of a 0-dimensional system, giving bounds of the roots of an univariate polynomial, or performing global optimization of a function with interval coefficients. Most of this algorithms are based on interval analysis and constraint programming methods and their scope is not restricted to algebraic expressions.

Although if most of the algorithms of the library have a structure that allows a parallel implementation, we will focus in this paper on the systems solving algorithms.

2 Interval Analysis and Systems Solving

Interval Analysis is based on *interval arithmetics* which is a well-known extension of classical arithmetics. Even if there are numerous ways to calculate the mapping Y of an interval X by a function f (see [1,2]), interval arithmetics guarantee that $\forall x \in X, f(x) \in Y$ and allows to take into account round-off errors[3].

The basic solving algorithm of interval analysis is based on interval evaluation and bissection:

Let $B = \{X_1, \ldots, X_n\}$ be a box and $\{F_i(X_1, \ldots, X_n) = 0\}$ a set of equations to be solved within B. Let L be a list of boxes initially restricted to $\{B\}$. Let ϵ be a threshold for width of intervals. The algorithm proceed as follows:

boxes loop: while L is not empty, let select a box b from L and
– if $\forall i, 0 \in F_i(b)$ and size of b is less than *epsilon*,

[1] http://www.inria-sop.fr/coprin/logiciel/ALIAS/ALIAS.html

- then store b in the solution list and remove b from L
- else, if it exists at least one $F_j(b)$ not containing 0, then remove b from L
- else, select one of the direction i and bisect B among this direction, creating 2 new boxes. Store them in L.

This basic method may be drastically improved by *filtering* i.e. decreasing the width of the current box "in place". ALIAS implements namely the 2B method [4] which is a *local method* as it proceeds equation by equation without looking at the whole system but *global methods* also exist, such as the classical Newton interval method [5]. A second type of improvement relies on the use of *unicity operators* whose purpose is to determine eventually a box, called a *unicity box*, that contains a unique solution of the system, that furthermore can be numerically computed in a certified manner. The most classical unicity operator is based on Kantorovitch theorem [6].

3 Parallel Implementation of ALIAS

A simple distributed implementation scheme based on a master computer and a set of slave computers may be used:

- The master will maintains a list of unprocessed boxes. These boxes will be dispatched to the slave computers.
- The slave computers run the solving algorithm with as initial search space the box received from the master. This slave program performs iterations within the boxes loop until either some stop criteria is fulfilled or if the boxes list has been exhausted. The list of unprocessed boxes (that may be empty) and, eventually, the solutions that have been found is sent back to the master.
- The master will periodically check if any slave computer has terminated its job and may eventually process a box in the meantime.

We use different stop criterion to stop an ongoing slave process:

1. The number of boxes still to be processed is greater than a given threshold N: this allows to limit the size of the message that is sent back to the master. However it may be acceptable to have more than N boxes in the boxes list of the slave process if we may assume that the slave will be able to process all the boxes in a reasonable amount of time. Indeed not stopping the slave will avoid a large number of message exchanges between the master and slave processes. For that purpose if the width of the box received by the slave is lower than a given threshold the slave process is allowed to perform at most M iterations of the solving algorithm. If the process is not completed after this number of iteration a FAIL message is sent to the master process.
2. the computation time has exceeded a fixed amount of time, in which case a FAIL message is sent to the master process: this indicates that it is necessary to distribute the treatment of the processed box among different slaves.

3.1 Efficiency

To obtain an efficient parallelization of the procedure it is necessary to choose carefully the threshold values for the stopping criteria and the values of the solving parameters for the slave process.

Indeed if the time necessary for the slave process to produce the N returned boxes is very small, then most of the computation time will be devoted to message passing, thus leading to a poor efficiency. On the other hand if this time is too large for a given box it may happen that the computer receiving this box will do most of the job, while the other slaves are free.

There is no general rule of a thumb for finding the right compromise but in our experience a first run with standard value for the parameters and then eventually a few additional run for fine tuning the parameters allows to determine near optimal values for the parameters, that are valid not only for the current problem but for a class of problems.

3.2 Implementation

A particularity of the ALIAS library is that it is mostly interfaced with Maple. Without going into the details the system of equations may be written in Maple and can be solved directly within Maple. The procedures invoqued create the necessary C++ code specific of the equations for solving the system, compile it and then execute it, returning the results to the Maple session (furthermore the symbolic treatment of the equations allows also for a better efficiency of the solving algorithm). The purpose of our implementation is to allow the use of a distributed implementation within Maple with a minimal modification of the Maple code.

For a parallel implementation it is hence necessary to have a message passing mechanism that enable to send a box to a slave program on another computer and to receive data from the slave computers. For the parallel implementation we use the message passing mechanism PVM10.1.

Simple master programs using PVM may be found in the ALIAS distribution along with its corresponding slave programs: these programs are used by Maple for creating the distributed implementation of the general solving procedures.

Basically a message sent through PVM is composed of a few control characters and a set of floating point numbers. Each box sent to a slave machine is saved in a backup box. The slave process uses the same coding for returning the boxes to process (if any): if there is no return box the slave process will return the keyword `N`, while solutions are returned with a message starting with the keyword `S`.

Within the master program an array of integers is used to determine the state of the slave machines: 0 if the machine is free, 1 if the machine is processing a box and has not returned, 2 if the machine is out of order. If a machine has status 1 the master process checks periodically if the machine has returned a message (using `pvm_nrecv`) or is not responding (using `pvm_mstat`): if this the case the master process will use the backup box of the machine, performs a bisection of this box and add the result in its list.

Both master and slave process use the same C++ solving code but a flag allows to determine if the algorithm is run by a slave or by the master. In the first case at each iteration of the algorithm the slave process will check if a stop criteria is verified while in the second case the master process checks at some key points if a slave has returned a message, in which case the master program stop the processing of the current box. The master process stops and the result is returned in a file as soon as all the boxes have been processed and all the machines are free.

4 Experiments

Our first experiment with the distributed implementation was to solve a very difficult problem in mechanism theory: the synthesis of a spatial RRR manipulator [7]. A RRR manipulator has three links connected by revolute joints and the geometry of the mechanism is defined by 15 parameters. The problem is to determine the possible values of these parameters under the constraints that the robot should be able to reach a set of 5 defined positions. For each such position we have new unknowns, namely the three angles of the revolute joints and the full problem is to solve a set of 30 non-linear equations in 30 unknowns. A first approach to solve this problem was to use a continuation method on a parallel computer with 64 processors, but after one month of calculation the problem was not solved. We then use the distributed implementation of ALIAS on a cluster of 20 PC's and get the solutions in about 5 days.

4.1 Chebyquad Function

For this experiments we compare the performances of the non-parallel version running on an EVO 410 C (1.2 Ghz) and a cluster of 11 medium-level PC's (850 MHz) with the master process running on a Sun Blade.

This system [8] is a system of n equations $f_i = 0$ with

$$f_i = \frac{1}{n}\sum_{j=1}^{j=n} T_i(x_j) + a_i$$

where T_i is the ith Chebyshev polynomial and a_i =0 if i is odd and $a_i = -1/(i^2 - 1)$ if i is even. This system has 24 solutions for $n = 4$ and 0 for $n = 5, 6$. The computation times for a search space of [-100,100] are respectively 25s, 279s and 11286s with the sequential implementation and 27s, 84s and 1523s with the parallel one.

This example allows to establish a rough model for the gain of a distributed implementation. For a sequential implementation the computation time t_s is roughly proportional to the number of boxes processed during the algorithm which is $2^{n\log(w/\epsilon)}$ where w is the initial width of the range in the search space and ϵ the mean value of the width of a box that is either deleted or in which a solution is found.

$$t_s = a2^{n\log(w/\epsilon)} + b \qquad (1)$$

Using the first line of the table we find $a = 0.17e^{-5}, b = 19, \epsilon = 0.87$. For the distributed implementation the computation time t_d should be t_s divided by the number m of slaves, to which should be added a communication time which is proportional to n:

$$t_d = t_s/m + a_1 n + b_1 \tag{2}$$

Using the 2 first values of the second line of the table we get $a_1 = 24.418, b_1 = -73.88$. Using these values for $n = 6$ we get $t_d = 1520.44$ which is coherent with the experimental data. The theoretical maximal ratio t_s/t_d in that case is about 7.8.

5 Conclusion

Interval analysis algorithms have a structure that is highly appropriate for a distributed implementation. We have shown the use of PVM in the distributed implementation of the ALIAS C++ library and presented some examples.

Prospectives for this work are:

– an adaptive control of the solving parameters to improve the load distribution among the slaves.
– a possible modification of the parallel scheme in which expensive global filtering methods will benefit of a distributed implementation while the core of the solving algorithm will be run on a master computer (or a mix between the current scheme and the proposed one).

References

1. Hansen E., *Global Optimization using Interval Analysis*. Marcel Dekker, 1992.
2. Moore R.E., *Methods and Apllications of Interval Analysis*. SIAM Studies in Applied Mathematics, 1979.
3. Revol N. and Rouillier F., *Motivations for an Arbitrary Precision Interval Arithmetics and the MPFI Library*. In *Validated Computing Conference*, Toronto, 2002.
4. Collavizza, H. Deloble, F. and Rueher M., *Comparing Partial Consistencies*. Reliable Computing, 5:1–16, 1999.
5. Ratscheck H. and Rockne J., *Interval Methods*. In Horst R. and Pardalos P.M. editors, *Handbook of Global Optimization*, pages 751–819. Kluwer, 1995
6. Tapia R.A., *The Kantorovitch Theorem for Newton's Method*. american Mathematic Monthly, 78(1.ea):389–392, 1971.
7. Lee E., Mavroidis C. and Merlet J-P., *Five Precision Points Synthesis of Spatial RRR Manipulators using Interval Analysis*. In *ASME 27th Biennal Mechanisms and Robotics Conf.* Montreal, 2002.
8. Moré J.J., Garbow B.S. and Hillstrom K.E., *Testing Unconstrained Optimization Software*. ACM Trans. Math. Software, 7(1):136–140, March 1981.