Interval Parallel Global Optimization with Charm++

José A. Martínez, Leocadio G. Casado,
José A. Alvarez, and Inmaculada García

Computer Architecture and Electronics Dpt.
University of Almería
04120 Almería, Spain
{jamartin,leo,jaberme,inma}@ace.ual.es

Abstract. Interval Global Optimization based on Branch and Bound (B&B) technique is a standard for searching an optimal solution in the scope of continuous and discrete Global Optimization. It iteratively creates a search tree where each node represents a problem which is decomposed in several subproblems provided that a feasible solution can be found by solving this set of subproblems. The enormous computational power needed to solved most of the B&B Global Optimization problems and their high degree of parallelism make them suitable candidates to be solved in a multiprocessing environment. This work evaluates a parallel version of AMIGO (Advanced Multidimensional Interval Analysis Global Optimization) algorithm. AMIGO makes an efficient use of all the available information in continuous differentiable problems to reduce the search domain and to accelerate the search. Our parallel version takes advantage of the capabilities offered by Charm++. Preliminary results show our proposal as a good candidate to solve very hard global optimization problems.

1 Introduction

The problem of finding the global minimum f^* of a real valued n-dimensional continuously differentiable function $f : S \to \mathbb{R},\ S \subset \mathbb{R}^n$, and the corresponding set S^* of global minimizers is considered, i.e.:

$$f^* = f(s^*) = \min_{s \in S} f(s),\ s^* \in S^*. \tag{1.1}$$

The following notation is used. $\mathbb{I} = \{X = [a,b] \mid a \leq b; a,b \in \mathbb{R}\}$ is the set of all one-dimensional intervals. $X = [\underline{x}, \overline{x}] \in \mathbb{I}$ is a one-dimensional interval. $X = (X_1, \ldots, X_n) \subseteq S, X_i \in \mathbb{I},\ i = 1, \ldots, n$ is an n-dimensional interval, also called box. $\mathbb{I}^n$ is the set of the n-dimensional intervals. $f(X) = \{f(x) \mid x \in X\}$ is the real range of f on $X \subseteq S$. F and $F' = (F'_1, \ldots, F'_n)$ are interval inclusion functions of f and its derivative f', respectively. These interval inclusion functions satisfy that: $f(X) \subseteq F(X)$ and $f'(X) \subseteq F'(X)$ [8].

In those cases where the objective function $f(x)$ is given by a formula, it is possible to use an interval analysis B&B approach to solve problem (1.1) (see [6,8,9,10]). A general interval GO (IGO) algorithm based on this approach is shown in Algorithm 1. An overview on theory and history of the rules of this algorithm can be found, for

J. Dongarra, K. Madsen, and J. Waśniewski (Eds.): PARA 2004, LNCS 3732, pp. 161–168, 2006.

Algorithm 1 : A general interval B&B GO algorithm

Funct IGO(S, f)

1. Set the working list L := $\{S\}$ and the final list Q := $\{\}$
2. **while** ($L \neq \{\}$)
3. Select an interval X from L — **Selection rule**
4. Compute a lower bound of $f(X)$ — **Bounding rule**
5. **if** X cannot be eliminated — **Elimination rule**
6. Divide X into $X^j,\ j = 1, \ldots, p$, subintervals — **Division rule**
7. **if** X^j satisfies the termination criterion — **Termination rule**
8. Store X^j in Q
9. **else**
10. Store X^j in L
11. **return** Q

example, in [6]. Of course, every concrete realization of Algorithm 1 depends on the available information about the objective function $f(x)$. The interval global optimization algorithm used in this article is called AMIGO [7].

AMIGO supposes that interval inclusion functions can be evaluated for $f(x)$ and its first derivative $f'(x)$ on X. Thus, the information about the objective function which can be obtained during the search is:

$$F(x),\ F(X) \text{ and } F'(X). \tag{1.2}$$

When the information stated in (1.2) is available, the rules of a traditional realization of Algorithm 1 can be written more precisely. Below we shortly describe the main characteristics of AMIGO. A detailed description can be found in [7].

The Bounding rule lets to get a lower and upper bounds of $f(X)$. Interval arithmetic provides a natural and rigorous way to compute these bounds by changing the real function to its interval version $F(X)$. Better approximations are obtained using derivative information. The Selection rule chooses among all the intervals X^j stored in the working list L, the interval X with a better lower bound of $\underline{f}(X)$. Most of the research of interval global optimization algorithm was done in the elimination rule to reduce as much as possible the search space. AMIGO incorporates the traditional elimination rules (Cutoff and Monotonicity tests) and additionally can reduce the size of an interval using the information given in (1.2). Cutoff test: An interval X is rejected when $\underline{F}(X) > \overline{f^{\sim}}$, where $\overline{f^{\sim}}$ is the best known upper bound of f^*. The value of $f^{\sim} = [\underline{f^{\sim}}, \overline{f^{\sim}}]$ is usually updated by evaluating F at the middle point of X. Monotonicity test: If condition $0 \notin F'(X)$ for an interval X is fulfilled, then this means that the interval X does not contain any minimum and, therefore, can be rejected. The easier division rule usually generates two subintervals using bisection on the direction of the wider coordinate. The termination rule determines the desired accuracy of the problem solution. Therefore, intervals X with a width smaller or equal to a value ε, are moved to the final list Q.

This deterministic global optimization algorithm exhibits a strong computational cost, mainly for hard to solve problems. Nevertheless, it exhibits a high degree of par-

allelism which can be exploited by using a multicomputer system. It also exhibits a high degree of irregularity. This means that special attention has to be paid to the load balancing and communication cost problems.

In this work we are using a SPMD parallel programing model. In our proposals, the set of subproblems generated by the B&B procedure is distributed among processors following a random strategy, which is appropriate when the number of generated subproblems is huge.

This paper is outlined as follows: Section 2 describes some issues related to the general framework of parallel B&B algorithms. Section 3 is a brief description of the Charm++ environment and of the parallel version of AMIGO algorithm. Finally, in Section 4 experimental results and evaluations of our parallel implementation on a distributed system are shown.

2 Parallel Issues in Interval B&B Global Optimization

The verified global optimization method considered in this paper belongs to the B&B class of methods, where the given initial problem is successively subdivided into smaller subproblems. Some of these subproblems are again subdivided while other do not need to be considered anymore because it is known they cannot contain the solution of the problem. Parallelizing B&B methods mainly consists of distributing among processors the set of independent subproblems being dynamically created. In order to achieve an efficient parallel method one should be concerned with the following issues:

1. All processors should always be busy handling subproblems;
2. The total cost of handling all the subproblems should not be greater than the cost of the serial method;
3. The overhead due to communications among processors should be small.

More precisely, issue 2 means that all processors should not be just busy but doing useful work. It is necessary to point out that a B&B algorithm executed in parallel does not process the subproblems in the same order that a sequential program does, so the number of created (and eliminated) subproblems will depend on the number of processors used in a particular execution. The resulting effect is that a specific parallel execution will create more (or sometime less) subproblems depending on the specific function and the number of processors.

Here we shall investigate the parallelization of a B&B global optimization algorithm (AMIGO) on a distributed memory multicomputers. In this case it is difficult to fulfill all the above described issues (1-3), simultaneously. For issue 2, it is important that the current best bounding criterion (in our case the smallest value of $\tilde{f}$ on all processors) is sent to every processor as soon as possible. Additionally, one must try to distribute among processors all the subproblems created thus far. This will contribute to keep all processors equally loaded or at least to keep them all busy. In addition one should try to fulfill all three issues to get an efficient parallel method.

When parallelizing the B&B method there are two possibilities for managing subproblems. The first is to store subproblems on one central processor. The other is to distribute them among processors. In our context of verified global optimization this

means either to store boxes in a list on one processor or to store them in several lists each created on every processor. The first case has a disadvantage: the maximal length of a list is limited by the amount of memory of one processor, whereas in the second case the memory of all processors can be used.

In [12], where parallelization of different methods for verified global optimization was investigated, one can find a very simple strategy where boxes are stored in a central list which can be handled by all processors. Similar parallelizations of this master-slave principle were proposed in [5] and [1]. Contrarily, Eriksson manages processors in a ring where each processor has its own list of not processed subproblems [4].

3 Parallel Implementation in Charm++

The main characteristics of Charm++, an object oriented portable parallel programming language based on C++, are introduced here to describe our parallel algorithm.

What sets Charm++ apart from traditional programming models such as message passing and shared variable programming is that the execution model of Charm++ is message-driven. Therefore, computations in Charm++ are triggered based on arrival of associated messages. These computations in turn can fire off more messages to other (possibly remote) processors that trigger more computations on those processors. Some of the programmer-visible entities in a Charm++ program are:

Concurrent Objects (Chares): A chare is a Charm++ object that can be created on any available processor and can be accessed from remote processors. A chare is similar to a process. Chares are created dynamically, and many chares may be active simultaneously.

Communication Objects (Messages): Chares send messages to one another to invoke methods asynchronously. Conceptually, the system maintains a "work-pool" consisting of seeds for new chares, and messages for existing chares.

Every Charm++ program must have at least one mainchare. Charm++ starts a program creating a single instance of each mainchare on processor 0, and invokes constructor methods of these chares. Typically, these chares then create a number of other chares, possibly on other processors, which can simultaneously work to solve the problem at hand.

Each chare contains a number of entry methods, which are methods that can be invoked from remote processors. In Charm++, the only communication method among processors is an invocation to an asynchronous entry method on remote chares. For this purpose, Charm Kernel needs to know the types of chares in the user program, the methods that can be invoked on these chares from remote processors, the arguments these methods take as input, etc.

Charm++ also permits prioritizing executions (associating priorities with method invocations), conditional message packing and unpacking (for reducing messaging overhead), quiescence detection (for detecting completion of some phase of the program), and dynamic load balancing (during remote object creation).

In our parallel program, a chare has two entry methods:

Process-Box: It execute one iteration of AMIGO algorithm over the box received in the given message.
Update-$\tilde{f}$: It upgrade the current $\tilde{f}$ value in the current chare.

We use a static mapping of chares on processors and there is only one chare in one processor. Therefore, we do not use the Charm++ dynamic load balancing capability. Each chare will be triggered when receives a message which invokes its entry methods. A chare also can generate messages for other chares (or itself). For instance, when a message with a Process-Box entry method arrives to a chare in one processor, the entry method can:

- Reject the box by some elimination rule: No messages are generated.
- Divide the box: Two new sub-boxes are generated and two new messages with a Process-Box entry method are generated. The receiver (chare) of these new messages are randomly selected.
- A solution box was found: A message with the solution box is generated and sent to the mainchare.

The possibility of associating priorities to entry method invocations is very important in our model. For instance if one chare obtains a better value of $\tilde{f}$, this value has to be broad-casted to all the chares, then they will apply the Cutoff test as soon as possible. Therefore, the Update-$\tilde{f}$ messages will have more priority that the Process-Box ones. If a Process-Box message arrives and its associated box satisfies the Cutoff test then it will not be processed. A priority also has been established in the Process-Box messages to first process the more promising boxes, i.e, those which let to obtain a better $\tilde{f}$ value. This tries to minimize the possibility that the search region visited by the parallel version will be larger than the visited by the sequential one.

4 Performance Results

The speed-up and work load imbalance for our parallel implementation of AMIGO algorithm are shown in Figures 1 and 2, respectively. All the data where obtained from executions carried out on a cluster of workstations composed of 16 nodes with two Pentium XEON 3Ghz, with Hyper-threading running Linux. The nodes are connected by a Gigabit Ethernet network.

Table 1 shows the set of test problems used to evaluate the algorithms. T_1 correspond with the execution time in seconds of AMIGO algorithm and ε_1 with the precision reached by AMIGO to obtain a solution in less than one hour of running time [7]. T_2 and ε_2 are analogous but for the parallel algorithm running in one processor. Our experiments were done in such a way that executions spend less that one hour. It means that increasing the accuracy of ε_1 and ε_2 one order of magnitude, algorithms do not provide any solution after running one hour. From results in Table 1 is clear that the parallel version can reach better precision. A possible reason is that the parallel version do not need to handle the storage of the pending boxes, just to process the entry methods.

Figure 1 clearly shows that, for this set of very hard to solve problems, an average linear speed-up was obtained.

Table 1. Comparison between sequential AMIGO (T_1, ε_1) and AMIGO-Charm++ (T_2, ε_2) algorithms

Name	Ref	n	$T_1(sec.)$	ε_1	$T_2(sec.)$	ε_2
Schwefel 2.14 (Powell)	[11]	4	15,85	10^{-5}	1898,97	10^{-7}
Schwefel 3.1p	[11]	3	1302,67	10^{-4}	24,14	10^{-4}
Ratz 5	[11]	3	667,35	10^{-3}	2245,77	10^{-5}
Ratz 6	[11]	5	823,49	10^{-3}	1007,58	10^{-4}
Ratz 7	[11]	7	903,75	10^{-3}	2459,16	10^{-4}
Schwefel 2.10 (Kowalik)	[14]	4	405,54	10^{-2}	3116,71	10^{-9}
Griewank 10	[13]	10	334,04	10^{-2}	1114,49	10^{-14}
Rosenbrock 10	[3]	10	199,19	10^{-2}	1357,70	10^{-14}
Neumaier 2	[9]	4	84,41	10^{-2}	1615,26	10^{-14}
EX2	[2]	5	44,38	10^{-2}	3302,40	10^{-10}
Ratz 8	[11]	9	10,65	10^{-2}	685,48	10^{-3}

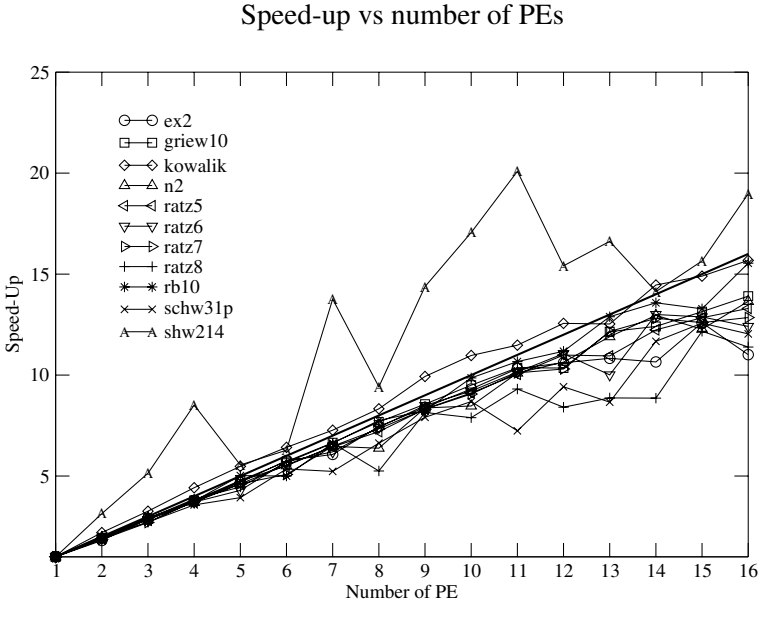

Fig. 1. Speed-up

The workload imbalance has been obtained as $(L_{max} - L_{av})/L_{av} \in [0, p-1]$; where L_{max} is the maximum workload and L_{av} is the average workload in a set of p processors. Notice (see Figure 2) that the workload imbalance in all the cases is negligible.

As the numerical results show, the parallel implementation of AMIGO using Charm++ is suitable to obtain solutions with near linear and sometimes with super speed-ups.

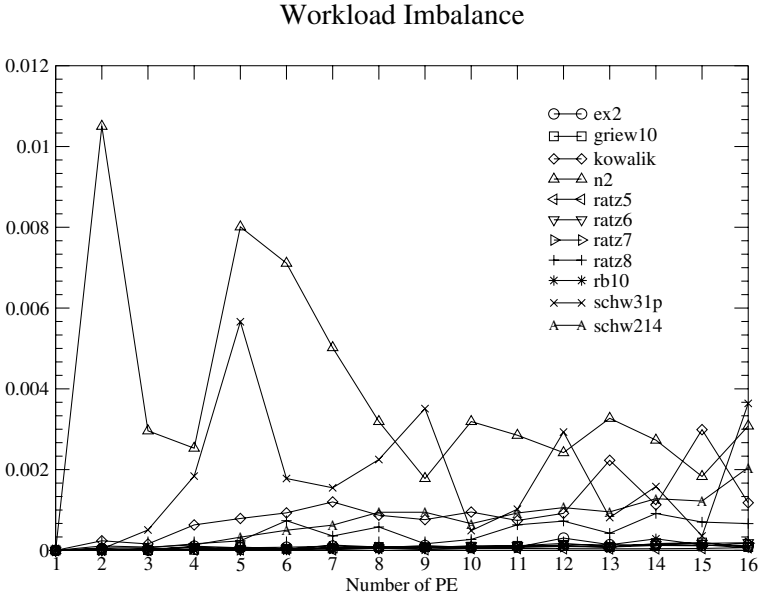

Fig. 2. Workload imbalance

Acknowledgement

This work was supported by the Ministry of Education of Spain(CICYT TIC2002-00228).

References

1. Berner, S. Parallel methods for verified global optimization, practice and theory. *Journal of Global Optimization* 9:1–22, 1996.
2. Csendes, T. and D. Ratz: 1997, 'Subdivision Direction Selection in Interval Methods for Global Optimization'. *SIAM Journal of Numerical Analysis* **34**, 922–938.
3. Dixon, L.W.C. and G.P. Szego (eds.): 1975, *Towards Global Optimization*. North-Holland Publishing Company.
4. Eriksson, J., Lindström, P.: A parallel interval method implementation for global optimization using dynamic load balancing. *Reliable Computing* 1:77-91, 1995.
5. Henriksen, T., Madsen, K. Use of a depth-first strategy in parallel Global Optimization. *Technical Report 92-10*, Institute for Numerical Analysis, Technical University of Denmark, 1992.
6. Kearfott, R.B. Rigorous Global Search: Continuous Problems. Kluwer Academic Publishers, Dordrecht, Holland, 1996.
7. Martínez, J.A., Casado, L.G., García, I., Tóth, B.: AMIGO: Advanced Multidimensional Interval Analysis Global Optimization Algorithm. In Floudas, C., Pardalos, P., eds.: *Nonconvex Optimization and Applications Series. Frontiers in Global Optimization.* 74:313-326. Kluwer Academic Publishers, 2004.

8. Moore, R.: Interval analysis. Prentice-Hall, 1966.
9. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, 1990.
10. Ratschek, H., Rokne, J.: New Computer Methods for Global Optimization. Ellis Horwood Ltd., 1988.
11. Ratz, D., Csendes, T.: On the selection of subdivision directions in interval branch and bound methods for global optimization. *Journal of Global Optimization* 7:183-207, 1995.
12. Suiunbek, I.: A New Parallel Method for Verified Global Optimization. *PhD thesis*, University of Wuppertal, 2002.
13. Törn, A. and A. Žilinskas: 1989, *Global Optimization*, Vol. 3350. Berlin, Germany: Springer-Verlag.
14. Walster, G., E. Hansen, and S. Sengupta: 1985, 'Test results for global optimization algorithm'. *SIAM Numerical Optimization 1984* pp. 272–287.